

# 【TWEvb-IMX6UL 评估板】

## 用户手册

版本：V1.0.0

版本记录						
版本号	修改状态	修改日期	修改摘要	撰稿人	校对	审核
V1.0.0	初次修订	2017-04-10		Jason		

# 目 录

第 1 章 产品介绍.....	7
1.1 产品简介.....	7
1.2 硬件参数.....	7
1.3 软件参数.....	9
1.4 产品接口布局.....	10
1.5 跳线器与指示灯说明.....	11
1.6 软件特性设备管理.....	12
第 2 章 产品快速使用说明.....	13
2.1 启动选择.....	13
2.2 串口登陆.....	13
2.2.1 串口硬件连接.....	13
2.2.2 使用串口终端登录.....	14
2.3 关机和重启.....	16
2.4 查看系统信息.....	17
2.4.1 查看内核版本.....	17
2.4.2 查看内存使用情况.....	17
2.4.3 查看磁盘使用情况.....	17
2.4.4 查看磁盘分区信息.....	17
2.4.5 查看 CPU 信息.....	18
2.5 网络设置.....	19
2.5.1 获取网络信息.....	19
2.5.2 设置 IP 与子网掩码.....	19

2.5.3 设置默认网关.....	20
2.5.4 关闭/启动网卡.....	21
2.5.5 设置 DNS.....	21
2.5.6 开机自动设置网络参数.....	21
2.5.7 注意事项.....	22
2.6 网络登录.....	23
2.7 USB 鼠标与 USB 键盘使用.....	24
2.8 TF 卡使用.....	25
2.9 U 盘使用.....	26
2.10 与 PC 互传文件.....	27
2.11 LED 测试.....	29
2.12 蜂鸣器测试.....	29
2.13 串口测试.....	29
2.14 液晶背光设置.....	33
2.15 音频播放测试.....	33
2.16 WIFI 测试.....	34
2.16.1 配置 WIFI 网络.....	34
2.16.2 使 WIFI 网络配置生效.....	34
2.16.3 测试 WIFI 网络配置.....	35
<b>第 3 章 Linux 应用程序开发.....</b>	<b>36</b>
3.1 安装 Linux 操作系统.....	36
3.1.1 VMware 软件.....	37
3.1.2 创建和配置虚拟机.....	40
3.1.3 安装 Ubuntu.....	46
3.1.4 安装 VMware Tool.....	52

3.1.5 虚拟机和主机之间传输文件.....	53
3.1.6 Ubuntu 操作简介.....	56
3.2 嵌入式 Linux 开发简介.....	60
3.3 安装交叉编译器.....	61
3.3.1 安装编译内核的交叉编译器.....	61
3.3.2 安装编译应用程序的交叉编译器.....	62
3.3.3 设置 PATH 环境变量.....	67
3.4 Hello, World!.....	68
3.4.1 编写 HelloWorld 源程序.....	68
3.4.2 编译 helloworld 程序.....	69
3.4.3 下载程序.....	69
3.4.4 运行程序.....	69
3.5 QT 编程.....	69
3.5.1 QT 介绍.....	69
3.5.2 编译 QT.....	70
3.5.3 安装和配置 Qt Creator.....	72
3.5.4 Tslib—触摸屏校准.....	78
3.5.5 Hello, World!.....	78
3.6 示例程序介绍.....	83
3.6.1 Hello World.....	84
3.6.2 LED 示例.....	84
3.6.3 蜂鸣器示例.....	84
3.6.4 串口编程示例.....	84
3.6.5 CAN 编程示例.....	84
3.6.6 网络编程示例.....	85

3.6.7 数据库编程示例.....	87
3.7 时钟设置.....	87
3.8 停止示例工程运行.....	88
<b>第 4 章 Web 控制系统.....</b>	<b>89</b>
4.1 嵌入式 Web 开发简介.....	89
4.1.1 嵌入式 Web 服务器.....	89
4.1.2 CGI.....	89
4.2 Boa 服务器.....	90
4.2.1 Boa 服务器简介.....	90
4.2.2 编译 Boa 服务器.....	90
4.2.3 Boa 服务器配置.....	90
4.3 Web 开发环境搭建.....	91
4.4 Web 开发实例.....	92
4.4.1 静态网页.....	92
4.4.2 使用 CGI 程序的动态网页.....	93
<b>第 5 章 U-Boot.....</b>	<b>95</b>
5.1 U-Boot 简介.....	95
5.1.1 Bootloader 简介.....	95
5.1.2 U-Boot 介绍.....	95
5.1.3 U-Boot 工作原理.....	96
5.2 U-Boot 基本命令.....	97
5.2.1 查看命令列表.....	97
5.2.2 环境变量.....	98
5.2.3 网络命令.....	100
5.2.4 Nand Flash 命令.....	101

5.2.5 启动命令.....	102
5.2.6 组合命令.....	102
5.2.7 重启命令.....	102
5.3 U-Boot 工具.....	103
<b>第 6 章 Linux 内核.....</b>	<b>104</b>
6.1 内核简介.....	104
6.1.1 概述.....	104
6.1.2 Linux 内核源码.....	104
6.1.3 Linux 内核配置系统.....	105
6.2 编译内核.....	109
<b>第 7 章 文件系统.....</b>	<b>111</b>
7.1 文件系统介绍.....	111
7.1.1 根文件系统目录结构.....	111
7.1.2 根文件系统类型.....	111
7.2 设置开机自启动程序.....	114
7.3 动态加载模块.....	114
<b>第 8 章 系统恢复与更新.....</b>	<b>115</b>
8.1 Nand Flash 分区.....	115
8.2 烧写系统映像.....	115
8.2.1 硬件连接.....	116
8.2.2 使用 MtgTools 软件进行烧写.....	117
8.3 烧写 U-Boot 映像.....	118
8.4 烧写 DTB.....	118
8.5 烧写 Linux 操作系统.....	118
8.6 烧写用户文件系统.....	118



8.7 编译 Linux 操作系统.....	119
第 9 章 免责声明.....	120

# 第 1 章 产品介绍

## 1.1 产品简介

TWEvb-IMX6UL 为 TWCORE-IMX6UL 系列核心板的评估板，以方便用户评估核心板及 CPU 的性能。TWCORE-IMX6UL 系列核心板基于工业级核心板基于 NXP(Freescale) i.MX6UL 系列 Cortex-A7 高性能处理器设计，支持摄像头接口、集成工业级 Wi-Fi、双路 CAN-bus 现场总线接口、双路以太网接口、8 路串口等，适用于快速开发一系列最具创新性的应用，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。

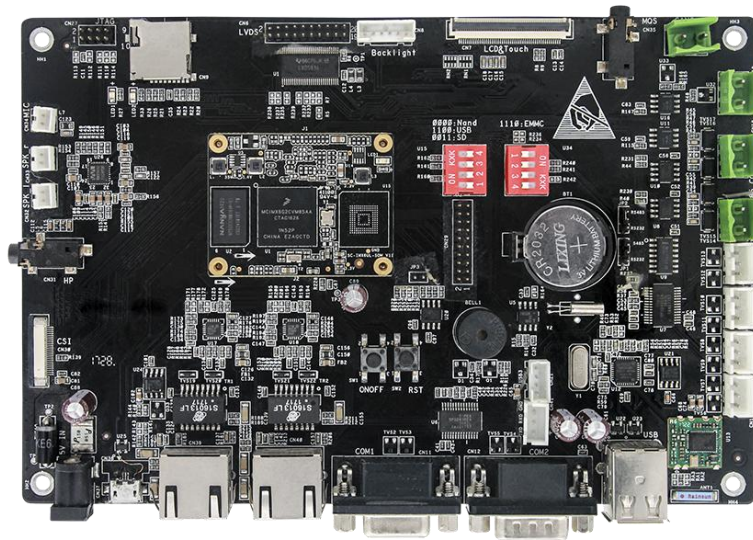


图 1.1 TWEvb-IMX6UL 外观

## 1.2 硬件参数

TWEvb-IMX6UL 板载的外设功能：

- 集成 2 路 10M/100M 自适应以太网接口
- 集成 5 路应用 RS-232 接口，1 路 232 调试串口
- 集成 2 路 RS-485 接口(与 RS-232 复用)
- 集成 2 路 CAN-bus 接口
- 集成 3 路 USB Host，1 路 USB Device
- 集成 USB Wi-Fi
- 集成 1 路 CSI 摄像头接口
- 支持扩展音频输入、输出



- 支持 CPU 独立音频输出
- 支持 1 路 TF 卡接口
- 支持 1 路 LVDS 接口 & 背光控制
- 支持液晶显示接口 (RGB 信号)
- 支持 4 线电阻触摸屏
- 支持实时时钟与后备电池
- 支持蜂鸣器与板载 LED
- 支持 GPIO 接口
- 预留 JTAG 接口
- 直流+5V 电源供电

TWCore-IMX6UL 核心板硬件资源参数:

表 1.1 硬件资源参数

产品名称	TWCore-IMX6UL 核心板
操作系统	Linux
处理器	i.MX6UL Cortex-A7
主频	528MHz
内存	128MB /256MB /更高
电子硬盘	128MB /256MB /更高
摄像头	1 路, CSI, 可扩展模拟摄像头
LCD 最高分辨率	1366 * 768
VGA	可提供方案支持
LVDS	可提供方案支持
触摸屏	支持 4 线电阻式与电容触摸屏
音频接口	1 路输出, 无需声卡, 支持外扩声卡
USB	2 路 USB2.0
串口	最高 8 路 (复用)
CAN-Bus	2 路
以太网	2 路
ADC	2 通道
SD 卡接口	最高 2 路 (复用)
I2C	1 路
PWM	2 路 (复用)
SPI	1 路
GPIO	30 路 (复用)
机械尺寸	35mm * 45mm

注: 受限于评估底板的尺寸与接口布局, 核心板部分资源以插针方式引出。

## 1.3 软件参数

TWCore-IMX6UL 核心板软件资源:

- 操作系统 Linux
- NANDFLASH 驱动
- 显示驱动
- 触摸屏驱动
- Wi-Fi 驱动
- 摄像头驱动
- 以太网驱动
- RS-232&RS-485 驱动
- CAN-bus 驱动
- USB Host & USB Slave 驱动
- SPI 驱动
- IIC 驱动
- PWM 驱动
- IO 驱动
- ADC 驱动
- 音频输出驱动，支持外扩输入、输出驱动
- TF/SD 卡驱动
- 蜂鸣器驱动
- LED 驱动
- RTC 驱动
- 看门狗驱动

# 1.4 产品接口布局

TWEvb-IMX6UL 功能接口布局示意图如图 1 所示。

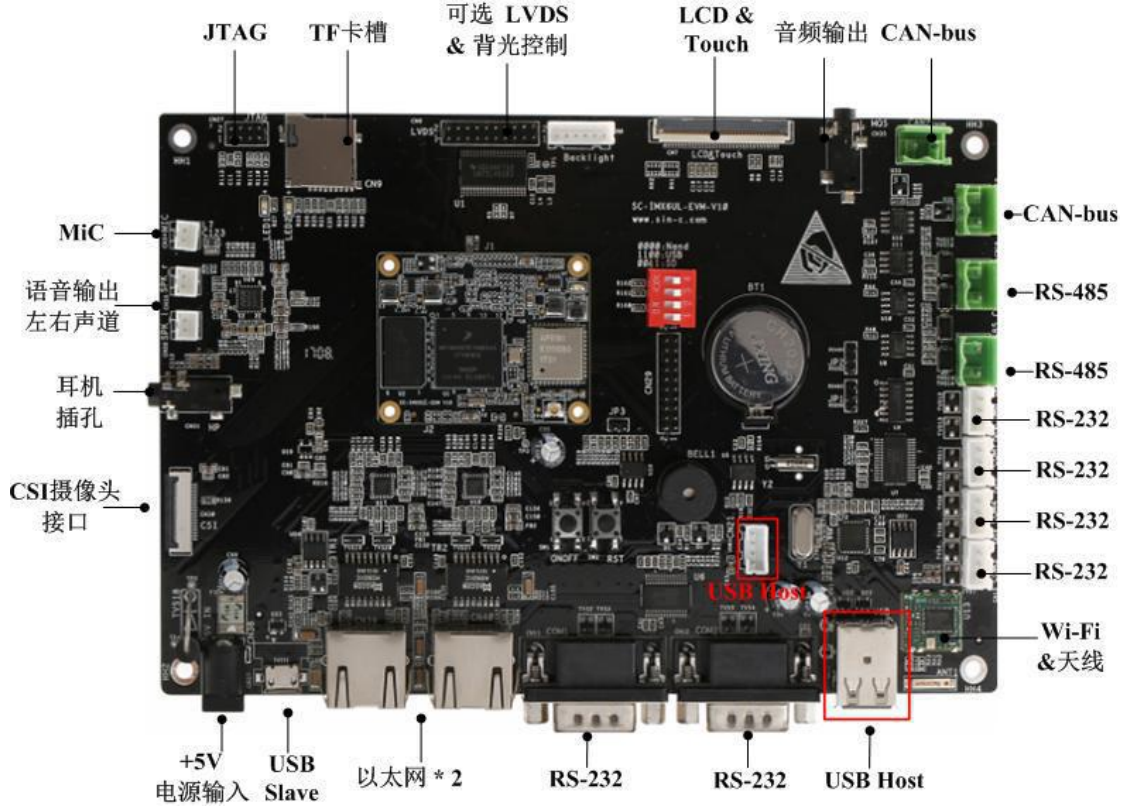


图 1.2 TWEvb-IMX6UL 接口布局

注：图片仅供参考，以实际销售产品为准

## 1.5 跳线器与指示灯说明

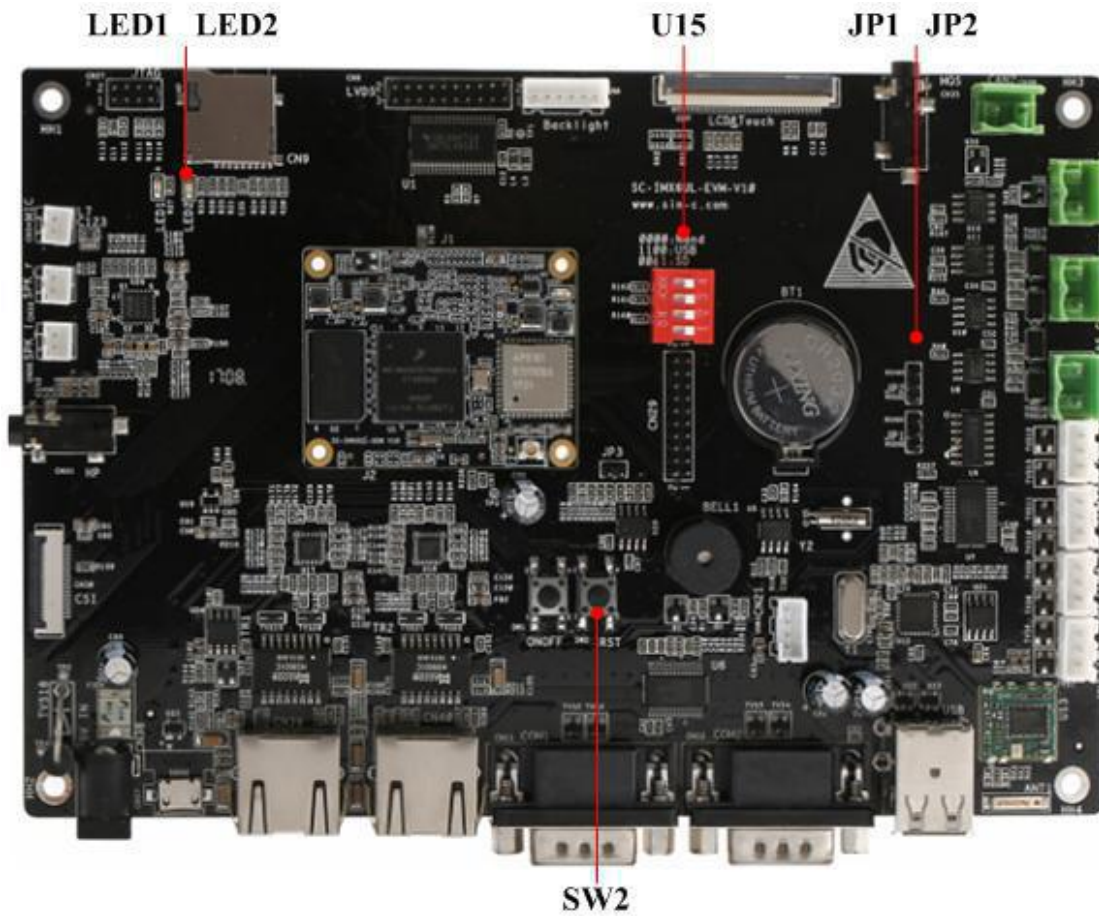


表 1.2 评估底板跳线与指示灯说明

序号	评估底板标号	描述
1	JP1	COM3 端口, 用于选择使用 RS-485 功能或 RS-232 功能
2	JP2	COM9 端口, 用于选择使用 RS-485 功能或 RS-232 功能
3	LED1	运行指示灯, GPIO 控制
4	LED2	电源指示灯
5	SW2	主板复位按键

表 1.3 U15 拨码开关说明

序号	拨码顺序	描述
1	0000	NANDFlash 启动
2	1100	USB 启动
3	0011	SD/TF 启动

## 1.6 软件特性设备管理

TWEvb-IMX6UL 开发板提供完善的 Linux BSP，包括 Linux 内核源码、和开发工具等，具体软件资源如下表所示：

表 1.4 软件资源

软件资源	说明	
Linux 内核	Linux 4.1.15	
文件系统	根文件系统采用 rootfs，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（内核）	arm-linux-gnueabi-gcc4.9.2	
交叉编译器（应用程序）	arm-none-linux-gnueabi-gcc 4.5.1	
外设驱动	NADN Flash	驱动源码：/drivers/mtd
	SD/MMC	驱动源码：/drivers/mmc
	LCD	驱动源码：/drivers/video
	触摸屏	驱动源码：/drivers/input/touchscreen
	I2C	驱动源码：/drivers/i2c
	UART	驱动源码：/drivers/tty/serial
	USB	驱动源码：/drivers/usb
	以太网	驱动源码：/drivers/net/ethernet
	CAN	驱动源码：/drivers/net/can
	WIFI	驱动源码：/drivers/net/wireless
	PWM	驱动源码：/drivers/pwm
	GPIO	驱动源码：/drivers/gpio
RTC	驱动源码：/drivers/rtc	
示例程序	提供串口、LED、网络、Web、数据库等开发例程	
工具软件	如系统镜像烧写工具、串口调试工具、网络调试工具、tftp 服务器软件等	

## 第 2 章 产品快速使用说明

### 2.1 启动选择

请参考跳线器与指示灯说明 1.5 节跳线器与指示灯说明，选择 NandFlash 启动或 TF 卡启动。

### 2.2 串口登陆

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通讯参数设置如下：

表 2.1 登录串口通讯参数设置

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	None

#### 2.2.1 串口硬件连接

在登录前必须确保开发板和主机之间的硬件连接正常。TWEvb-IMX6UL 开发板使用 COM1 作为登录调试串口，因此通过串口登录时必须将主机连接到开发板的 COM1 口上。

开发板的 COM1 口为标准的 RS232 DB9 公头接口。如果主机自带 RS232 串口，则可以使用串口延长线连接主机和开发板的 COM1 口；如果主机不带 RS232 串口，则需要使用 USB 转 RS232 转换器，将转换器的 USB 端接到主机的 USB 接口上，将转换器的 RS232 端接到开发板的 COM1 口上。在 windows 下使用 USB 转 RS232 转换器连接主机和开发板时，需要安装厂商提供的转换器驱动程序（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 RS232 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：

右键单击桌面上的“我的电脑”或者“此电脑”图标，选择“属性”菜单，在弹出的“系统”对话框中选择“设备管理器”（注：不同版本的操作系统进入设备管理器的方法可能有所不同），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转换出来的串口。如下图所示：

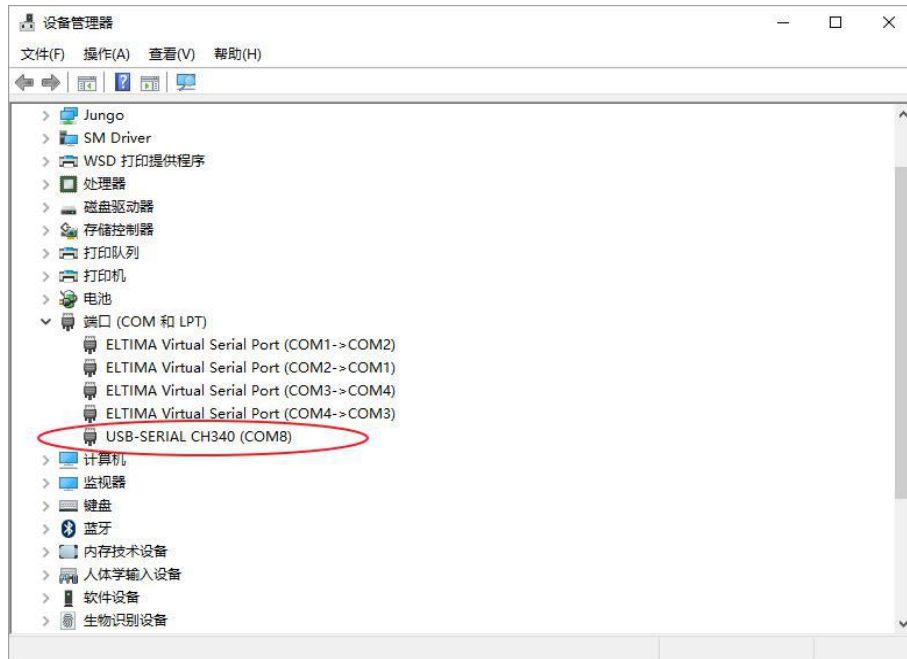


图 2.1 查看 USB 转 RS232 转换器信息

从图中可以看出，系统不仅识别了转换器转换出来的串口，而且还可以看到该串口使用的串口号，后面在使用串口终端登录时需要使用该串口号。

## 2.2.2 使用串口终端登录

Windows 下的串口终端软件比较多，此处以 SecureCRT 软件为例介绍串口登录的方法。

### 1. 安装 SecureCRT

用户可以从 <https://www.vandyke.com/products/securecrt/index.html> 下载 SecureCRT 安装程序。双击该文件开始安装，由于其安装过程比较简单，此处不再详细叙述。

### 2. 使用 SecureCRT 登录

运行 SecureCRT 软件，弹出“连接”对话框，如下图所示：

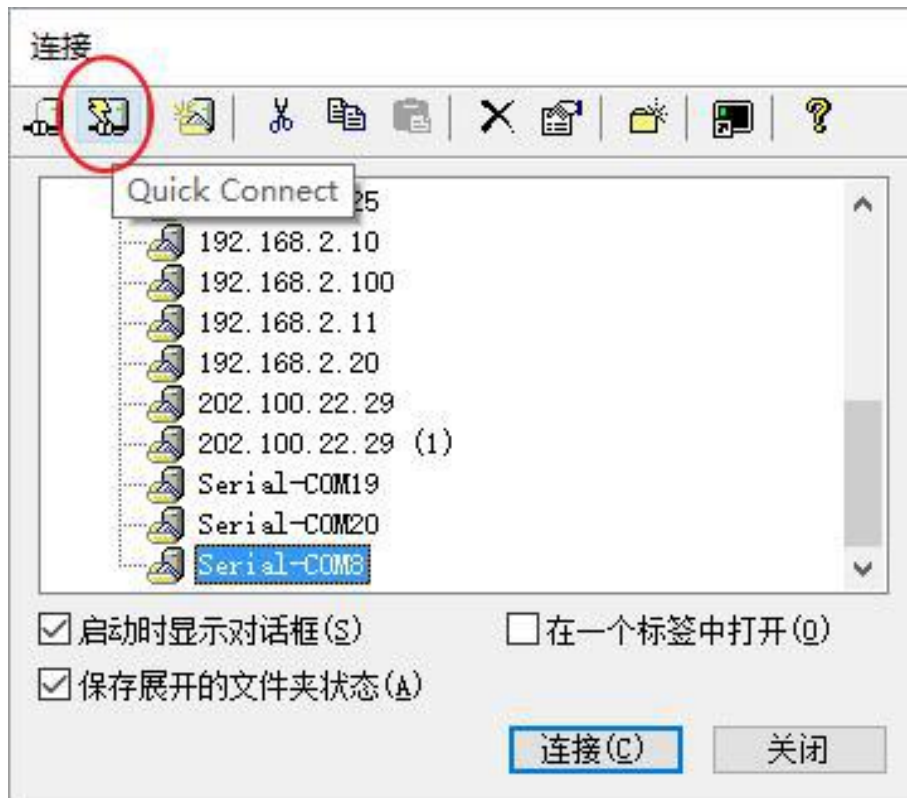


图 2.2 SecureCRT“连接”对话框

点击“Quick Connect”按钮，弹出“快速连接”对话框，如下图所示：

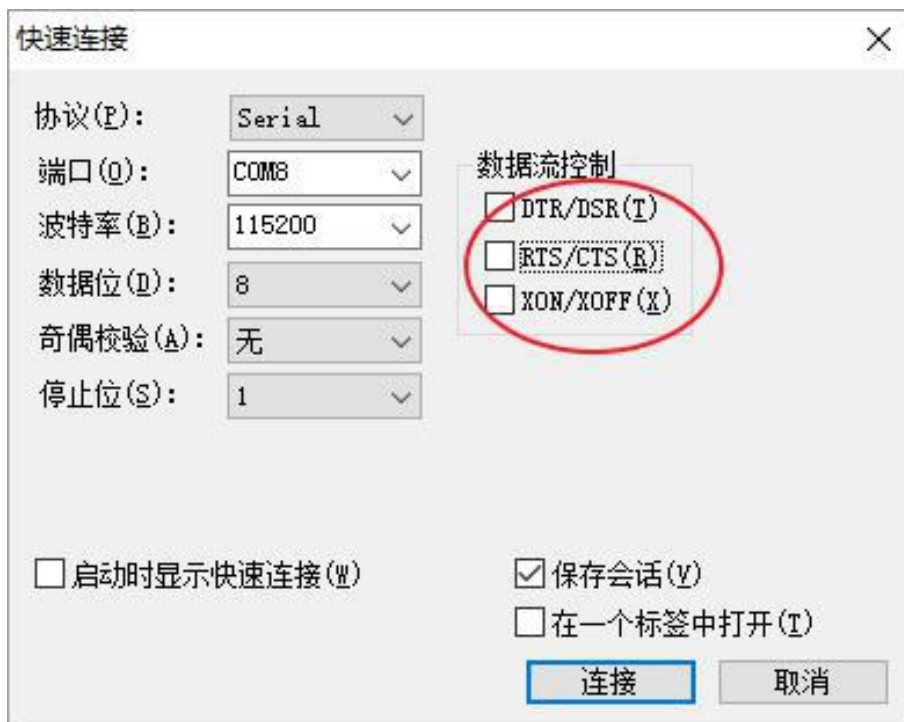


图 2.3 SecureCRT“快速连接”对话框

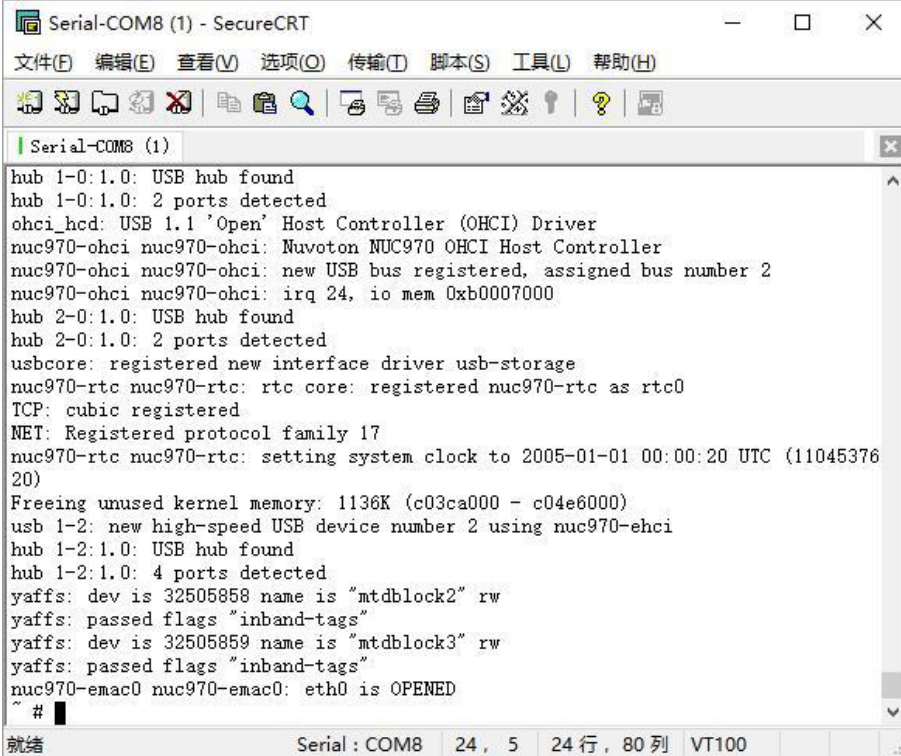
在该对话框中，选择“协议”为“Serial”，“端口”需根据主机实际使用的串口号进行选择（可从操作系统的“设备管理器”中获得该信息），并对串口参数进行相应的设置，具体通讯参数值可参考“表 2-1 登录串



口通讯参数设置”。值得注意的是在设置串口参数时需关闭所有的“数据流控制”选项。在设置完成后，点击“连接”按钮，会出现一个用于登录的 SecureCRT 终端窗口。

此时如果开发板已经上电运行，则在 SecureCRT 终端窗口上直接按回车键，就可在终端窗口上看到命令行提示符；如果开发板还没有上电运行，则先给开发板上电，此时可以看到在 SecureCRT 终端上会

打印系统的启动信息，在系统启动完成后，会自动出现命令行提示符。如下图所示：



```
Serial-COM8 (1) - SecureCRT
文件(F) 编辑(E) 查看(V) 选项(O) 传输(T) 脚本(S) 工具(L) 帮助(H)
Serial-COM8 (1)
hub 1-0:1.0: USB hub found
hub 1-0:1.0: 2 ports detected
ohci_hcd: USB 1.1 'Open' Host Controller (OHCI) Driver
nuc970-ohci nuc970-ohci: Nuvoton NUC970 OHCI Host Controller
nuc970-ohci nuc970-ohci: new USB bus registered, assigned bus number 2
nuc970-ohci nuc970-ohci: irq 24, io mem 0xb0007000
hub 2-0:1.0: USB hub found
hub 2-0:1.0: 2 ports detected
usbcore: registered new interface driver usb-storage
nuc970-rtc nuc970-rtc: rtc core: registered nuc970-rtc as rtc0
TCP: cubic registered
NET: Registered protocol family 17
nuc970-rtc nuc970-rtc: setting system clock to 2005-01-01 00:00:20 UTC (1104537620)
Freeing unused kernel memory: 1136K (c03ca000 - c04e6000)
usb 1-2: new high-speed USB device number 2 using nuc970-ehci
hub 1-2:1.0: USB hub found
hub 1-2:1.0: 4 ports detected
yaffs: dev is 32505858 name is "mtdblock2" rw
yaffs: passed flags "inband-tags"
yaffs: dev is 32505859 name is "mtdblock3" rw
yaffs: passed flags "inband-tags"
nuc970-eth0 nuc970-eth0: eth0 is OPENED
~ #
```

图 2.4 串口终端登录

在命令行提示符下，用户可以输入 linux 命令和系统进行交互。

## 2.3 关机和重启

当需要关机时，如果有数据存储操作，为了确保数据完全写入，可执行 sync 命令：

```
~ # sync
```

完成数据同步后再关闭电源关机。

也可以执行 reboot 命令重启开发板：

```
~ # reboot
```

该命令会自动完成数据同步后再重启系统。

## 2.4 查看系统信息

### 2.4.1 查看内核版本

使用 `uname -a` 命令可以查看内核版本信息：

```
~ # uname -a
Linux (none) 4.1.15 #18 SMP PREEMPT Mon Apr 17 13:33:09 CST 2017 armv7l GNU/Linux
```

也可以通过查看 `/proc/version` 文件，获得系统内核版本信息：

```
~ # cat /proc/version
Linux version 4.1.15 (sinc@sinc-virtual-machine) (gcc version 4.9.2 20140904 (prerelease) (crosstool-NG
linaro-1.13.1-4.9-2014.09 - Linaro GCC 4.9-2014.09) ) #18 SMP PREEMPT Mon Apr 17 13:33:09 CST 2017
```

### 2.4.2 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
              total          used          free          shared          buffers
Mem:          60620           5580          55040             0             0
-/+ buffers:              5580          55040
Swap:           0             0             0
```

### 2.4.3 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
Filesystem      1M-blocks    Used    Available    Use%    Mounted on
devtmpfs         29         0         29         0%     /dev
none             30         0         30         0%     /tmp
/dev/mtdblock2   32         6         26         19%    /root
/dev/mtdblock3   73         2         72         2%     /home
```

### 2.4.4 查看磁盘分区信息

通过查看 `/proc/mtd` 文件，可以获得 NandFlash 的分区信息：

```
~ # cat /proc/mtd
dev: size erasesize name
mtd0: 00200000 00020000 "u-boot"
```

```
mtd1: 01400000 00020000 "Kernel"  
mtd2: 02000000 00020000 "root"  
mtd3: 04a00000 00020000 "home"
```

通过查看/proc/partitions 文件，可以获得系统所有的分区信息：

```
~ # cat /proc/partitions  
major minor #blocks name  
 31      0          2048 mtdblock0  
 31      1         20480 mtdblock1  
 31      2        32768 mtdblock2  
 31      3        75776 mtdblock3
```

## 2.4.5 查看 CPU 信息

通过查看/proc/cpuinfo 文件，可以获得 CPU 等信息：

```
~ # cat /proc/cpuinfo  
processor       : 0  
model name     : ARM926EJ-S rev 5 (v5l)  
BogoMIPS      : 148.88  
Features       : swp half thumb fastmult edsp java  
CPU implementer : 0x41  
CPU architecture: 5TEJ  
CPU variant    : 0x0  
CPU part       : 0x926  
CPU revision   : 5  
  
Hardware       : NUC970  
Revision       : 0000  
Serial         : 0000000000000000
```

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，单位百万。

## 2.5 网络设置

### 2.5.1 获取网络信息

TWEvb-IMX6UL 开发板有两个以太网卡，运行 `ifconfig` 命令可以查看网卡的配置信息：

```
#ifconfig
eth0    Link encap:EthernetHWaddr 00:35:9A:0C:EE:2F
        Bcast:192.168.1.2
        inet addr:192.168.1.10  55          Mask:255.255.255.0
        UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
        RX packets:2237 errors:0 dropped:6 overruns:0 frame:0
        TX packets:2183 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:1000
        RX bytes:1240764 (1.1 MiB) TX bytes:100445 (98.0 KiB)

eth1    Link encap:EthernetHWaddr 08:00:27:00:01:93
        Bcast:192.168.0.2
        inet addr:192.168.0.10  55          Mask:255.255.255.0
        UP BROADCAST MULTICAST  MTU:1500  Metric:1
        RX packets:0 errors:0 dropped:0 overruns:0 frame:0
        TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:1000
        RX bytes:0 (0.0 B) TX bytes:0 (0.0 B)

lo      Link encap:Local Loopback
        inet addr:127.0.0.1  Mask:255.0.0.0
        UP LOOPBACK RUNNING  MTU:65536  Metric:1

        RX packets:0 errors:0 dropped:0 overruns:0 frame:0
        TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0
        txqueuelen:0
        RX bytes:0 (0.0 B) TX bytes:0 (0.0 B)
```

其中接口 `eth0` 表示第一个以太网卡，接口 `eth1` 表示第二个以太网卡，`lo` 表示本地回环接口。

### 2.5.2 设置 IP 与子网掩码

使用 `ifconfig` 命令可以设置网卡的 IP 地址和子网掩码，其命令格式如下：

```
ifconfig 网络接口名 IP 地址 netmask 子网掩码
```

对于以太网卡 1，“网络接口名”为 `eth0`；对于以太网卡 2，“网络接口名”为 `eth1`。例如要将网卡 1 的

ip 地址设置为 192.168.1.10，执行如下命令：

```
#ifconfig eth0 192.168.1.10 netmask 255.255.255.0
```

设置完毕后，可以使用 `ifconfig eth0` 命令查看网卡 1 的配置：

```
~ # ifconfig eth0
eth0 Link encap:Ethernet HWaddr 08:00:27:00:01:92
inet addr:192.168.1.10 Bcast:192.168.1.255 Mask:255.255.255.0
UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
RX packets:217 errors:0 dropped:71 overruns:0 frame:0 TX packets:0 errors:0 dropped:0 overruns:0 carrier:0 collisions:0
txqueuelen:1000
RX bytes:31251 (30.5 KiB) TX bytes:0 (0.0 B)
```

## 2.5.3 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
```

对于以太网卡 1，“网络接口名”为 `eth0`；对于以太网卡 2，“网络接口名”为 `eth1`。例如要将网卡 1 的默认网关设置为 192.168.1.111，执行如下命令：

```
~ # route add default gw 192.168.1.111 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
```

对于以太网卡 1，“网络接口名”为 `eth0`；对于以太网卡 2，“网络接口名”为 `eth1`。例如要将网卡 1 上配置的 IP 为 192.168.1.111 的默认网关删除，执行如下命令：

```
~ # route del default gw 192.168.1.111 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：

```
/etc # route -n
Kernel IP routing table
Destination Gateway Genmask Flags Metric Ref Use Iface
192.168.1.0 0.0.0.0 255.255.255.0 U00 0 eth0
```

## 2.5.4 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
```

对于以太网口 1，“网络接口名”为 `eth0`；对于以太网口 2，“网络接口名”为 `eth1`。例如要关闭网卡 1，执行如下命令：

```
#ifconfig eth0 down
```

要启动网卡 1，执行如下命令：

```
#ifconfig eth0 up
```

## 2.5.5 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
```

## 2.5.6 开机自动设置网络参数

使用 `ifconfig`、`route` 命令直接在命令提示符下设置网络参数或者直接编辑 `/etc/resolv.conf` 文件配置 DNS 服务器，在开发板断电或者复位后，上述配置信息就会丢失。因此为了在每次开机运行时，都能够自动使用用户修改过的参数配置，需要将上述命令添加到系统的启动脚本中。

对于 TWEvb-IMX6UL 开发板，可以通过修改 `/home/etc/sysconf` 配置文件来实现对 IP 地址和网关的设置。`sysconf` 配置文件中关于 IP 地址和网关的配置项如下：

```
IPADDR=192.168.1.10
NETMASK=255.255.255.0
ROUTE=192.168.1.1
ETH1IPADDR=192.168.0.10
ETH1NETMASK=255.255.255.0
ETH1ROUTE=192.168.0.1
```

其中 `IPADDR`、`NETMASK`、`ROUTE` 分别对应第一个网卡的 IP 地址、子网掩码和默认网关；`ETH1IPADDR`、`ETH1NETMASK`、`ETH1ROUTE` 分别对应第二个网卡的 IP 地址、子网掩码和默认网关。开机时启动脚本会读取该配置文件，并根据配置文件的内容调用 `ifconfig` 和 `route` 命令完成对 IP 地址和

网关的设置。

在完成对 `sysconf` 配置文件的修改后，可以在命令行下直接执行 `/home/etc/netcfg` 脚本，使网络设置立即生效，如下所示：

```
#!/home/etc/netcfg
```

也可以直接重启开发板，使网络设置生效。

对于 DNS 的配置，则有所不同。直接修改 `/etc/resolv.conf` 文件，在开发板重启后，配置信息之所以会丢失是因为 `/etc` 目录下的所有文件其实都是存在于内存中，为了验证这一点，可以在命令提示符下运行 `mount` 命令：

```
/root # mount
rootfs on / type rootfs (rw)
none on /proc type proc (rw,relatime)
sysfs on /sys type sysfs (rw,relatime)
devtmpfs on /dev type devtmpfs (rw,relatime,size=107860k,nr_inodes=26965,mode=755)
none on /tmp type tmpfs (rw,relatime)
devpts on /dev/pts type devpts (rw,relatime,mode=600)
/dev/mtdblock4 on /root type yaffs2 (rw,relatime)
/dev/mtdblock5 on /home type yaffs2 (rw,relatime)
none on /debugfs type debugfs (rw,relatime)
none on /sys/kernel/debug type debugfs (rw,relatime)
```

从上述结果可以看出，NandFlash 对应的分区 `/dev/mtdblock4` 和 `/dev/mtdblock5` 分别被挂载在 `/root` 和 `/home` 目录下，也就是说除了 `/root` 和 `/home` 这两个目录及其子目录对应于 NandFlash 上的存储空间，其他的目录其实都在内存中。

而为了使 DNS 配置信息在系统每次重启后都能生效，可以采用如下方案：

- 1.在 `/root` 或者 `/home` 目录（也可以是这两个目录的任意子目录）下新建一个 `resolv.conf` 配置文件，并在其中增加相应的 DNS 服务器配置；

- 2.在 `/root/etc/rc.ini` 文件中增加如下内容：（假定用户新增的 `resolv.conf` 位于 `/home/etc` 目录下）：

```
cp /home/etc/resolv.conf /etc/resolv.conf
```

即在每次系统启动后，将用户创建的 `resolv.conf` 配置文件自动拷贝到 `/etc` 目录下。

## 2.5.7 注意事项

开发板和主机通讯时，它们的 IP 地址必须在同一网段内。例如：假设开发板的 IP 地址为 192.168.1.10，子网掩码为 255.255.255.0，则主机 IP 地址也必须在 192.168.1 这个网段内，如设置为 192.168.1.20。另外在配置好主机 IP 和开发板的 IP 后，可以在主机上运行一下 `ping` 命令，`ping` 一下开发板，看网络是否畅通。

## 2.6 网络登录

当设置好开发板 IP 地址（具体设置方法可参考“设置 IP 与子网掩码”章节），且主机和开发板之间网络通讯正常后，即可以采用网路登录到开发板上。和串口登录类似，在 Windows 上也有很多软件可以实现网络登录，此处仍以 SecureCRT 软件为例介绍网络登录的方法。

### 1. 安装 SecureCRT

用户可以从 <https://www.vandyke.com/products/securecrt/index.html> 下载 SecureCRT 安装程序。双击该文件开始安装，由于其安装过程比较简单，此处不再详细叙述。

### 2. 使用 SecureCRT 登录

运行 SecureCRT 软件，弹出“连接”对话框，如下图所示：

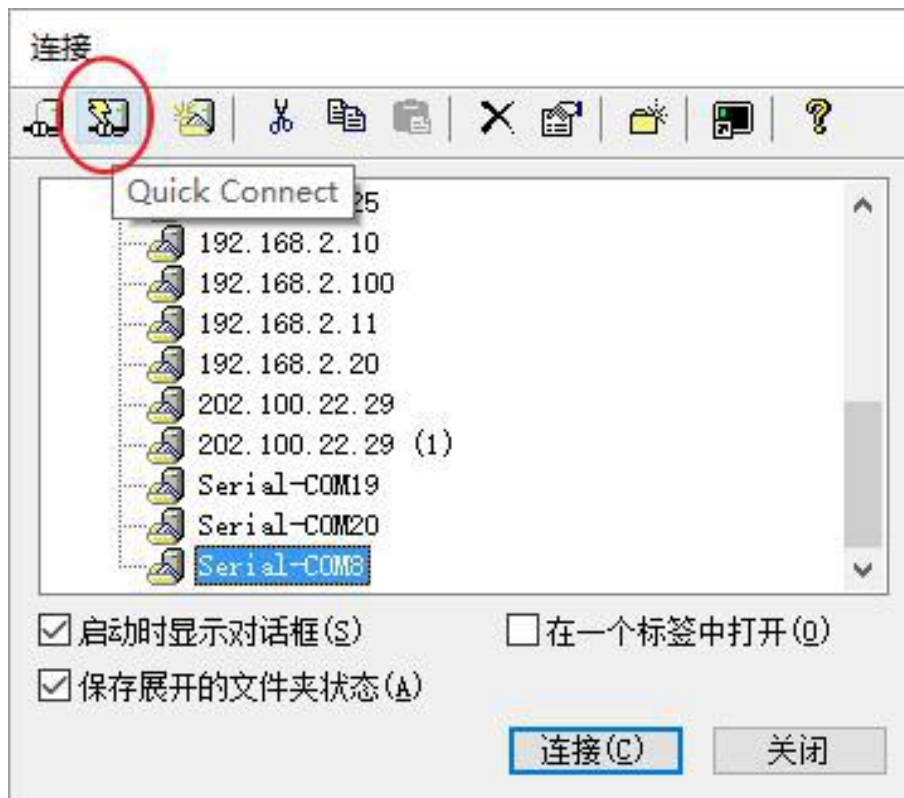


图 2.5 SecureCRT“连接”对话框

点击“Quick Connect”按钮，弹出“快速连接”对话框，如下图所示





图 2.6 SecureCRT“快速连接”对话框

在该对话框中，选择“协议”为“Telnet”，“主机名”为开发板的 IP 地址，端口号使用默认值 23。在设置完成后，点击“连接”按钮，会出现一个用于登录的 SecureCRT 终端窗口。在该终端窗口中，会提示用户登录，登录的用户名和密码均为 root，然后直接按回车键，即可完成登录操作。如下图所示：



图 2.7 网络终端登录

## 2.7 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：

```
~# usb 1-2.2: new low-speed USB device number 10 using nuc970-ehci input: Logitech Lenovo USB Optical Mouse as
```

```
/devices/platform/nuc970-ehci/usb1/1-2/1-2.2/1-2.2:1.0/input/input6
hid-generic 0003:17EF:6019.0006: input: USB HID v1.11 Mouse [Logitech Lenovo USB Optical Mouse]
on usb-nuc970-ehci-2.2/input0
```

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：

```
~# usb 1-2.2: new low-speed USB device number 5 using nuc970-ehci
input: USB USB Keyboard as /devices/platform/nuc970-ehci/usb1/1-2/1-2.2/1-2.2:1.0/input/input3
hid-generic 0003:1A2C:0002.0003: input: USB HID v1.10 Keyboard [USB USB Keyboard]
on usb-nuc970-ehci-2.2/input0
input: USB USB Keyboard as /devices/platform/nuc970-ehci/usb1/1-2/1-2.2/1-2.2:1.1/input/input4
hid-generic 0003:1A2C:0002.0004: input: USB HID v1.10 Device [USB USB Keyboard]
on usb-nuc970-ehci-2.2/input1
```

## 2.8 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：

```
mmc0: new high speed SDHC card at address e624
mmcblk0: mmc0:e624 SL08G 7.40 GiB
mmcblk0: p1 p2 p3 p4
```

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，且 TF 卡有 4 个分区，分别为 p1、p2、p3、p4。这个 4 个分区对应的设备名分别为 mmcblk0p1、mmcblk0p2、mmcblk0p3、mmcblk0p4。

系统会为 TF 卡的每个分区都在/mnt 下生成一个目录，目录的名字为 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。

假定 TF 卡被挂载到/mnt/mmcblk0p1 目录上，为了将 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 /mnt/mmcblk0p1/test /home
```

为了将/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 /mnt/mmcblk0p1
```

TF 卡使用完毕后，在拔出 TF 卡前，需执行 umount 命令卸载 TF 卡所有的分区：

```
~ #umount /mnt/mmcblk0p1
```

此处假定 TF 卡被挂载在/mnt/mmcblk0p1 目录下。umount 会确保所有缓存的数据都被正确的写入 TF 卡。在 umount 成功后，即可拔出 TF 卡。在调用 umount 前，必须确保 TF 卡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 TF 卡的挂载目录下，否则调用 umount 会提示失败，如下所示：

```
umount: can't umount /mnt: Device or resource busy
```

## 2.9 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：

```
~ # usb 1-2.2: new high-speed USB device number 3 using nuc970-ehci
usb 1-2.2: New USB device found, idVendor=0930, idProduct=6544
usb 1-2.2: New USB device strings: Mfr=1, Product=2, SerialNumber=3
usb 1-2.2: Product: DataTraveler 2.0
usb 1-2.2: Manufacturer: Kingston
usb 1-2.2: SerialNumber: C860008863E1CE61DA12AAF3
usb-storage 1-2.2:1.0: USB Mass Storage device detected
scsi0 : usb-storage 1-2.2:1.0
scsi 0:0:0:0: Direct-Access    Kingston DataTraveler 2.0 1.00 PQ: 0 ANSI: 4
sd 0:0:0:0: [sda] 15148608 512-byte logical blocks: (7.75 GB/7.22 GiB)
sd 0:0:0:0: [sda] Write Protect is off
sd 0:0:0:0: [sda] Write cache: disabled, read cache: enabled, doesn't support DPO or FUA sda: sda1
sd 0:0:0:0: [sda] Attached SCSI removable disk
```

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 7.75GB 的 U 盘，其对应的设备名为 sda1。

系统会在/mnt 目录下为 U 盘的每个分区都生成一个目录，目录的名字为 sdxn(x 用于区分不同的 U 盘，n 用于区分不同的分区，x=a、b、c..... n=1、2、3.....)。U 盘的每个分区就挂载在这些目录下。U 盘挂载成功后，即可对 U 盘进行文件查看、文件拷贝等操作。以下以文件拷贝操作为例，介绍 U 盘的使用。

假定 U 盘被挂载到/mnt/sda1 目录下，为了将 U 盘根目录上的 test 文件拷贝到/home 目录下，可执行如下命令：

```
~ #cp /mnt/sda1/test /home
```

为了将/home 目录下的 test1 文件拷贝到 U 盘根目录上，可执行如下命令：

```
~ #cp /home/test1 /mnt/sda1
```

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：

```
~ #umount /mnt/sda1
```

此处假定 U 盘被挂载在/mnt/sda1 目录下。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，如下所示：

```
umount: can't umount /mnt: Device or resource busy
```

## 2.10 与 PC 互传文件

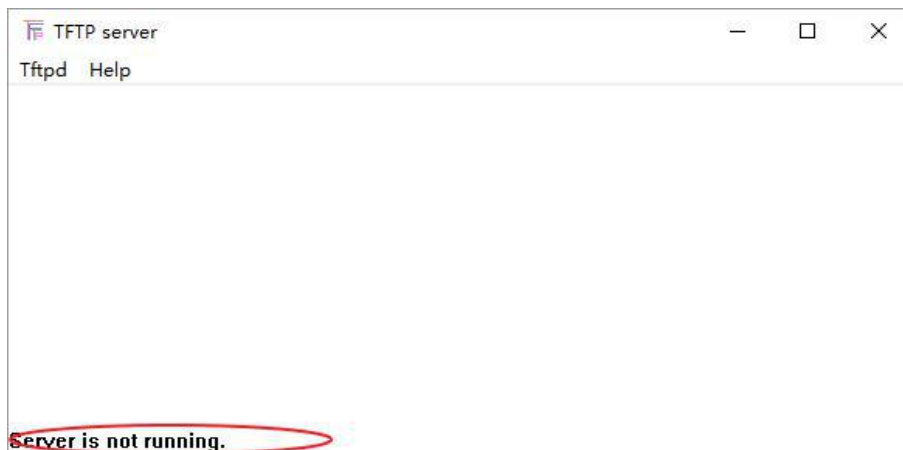
开发板和 PC 机有两种方式传输文件：

- 使用 U 盘。其具体操作方法可参见“U 盘使用”一节。
- 使用 TFTP 协议传输文件。

本节将主要介绍如何通过 TFTP 协议在 PC 机和开发板之间传输文件。

首先需要在 PC 机上运行 TFTP 服务器，在“产品光盘资料/3、软件开发参考资料/2、开发工具软件”中，提供了一个简单的 TFTP 服务器软件 `tftpd.exe`。在主机端配置 TFTP 的步骤如下：

1. 运行 `tftpd.exe` 程序，弹出如下对话框：



从图中可以看出，软件刚运行时，tftp 服务器还有没有启动；

2. 配置 tftp 文件传输目录。点击“Tftpd”菜单下的“Configure”子菜单，弹出“Tftpd Settings”对话框，如下图所示：

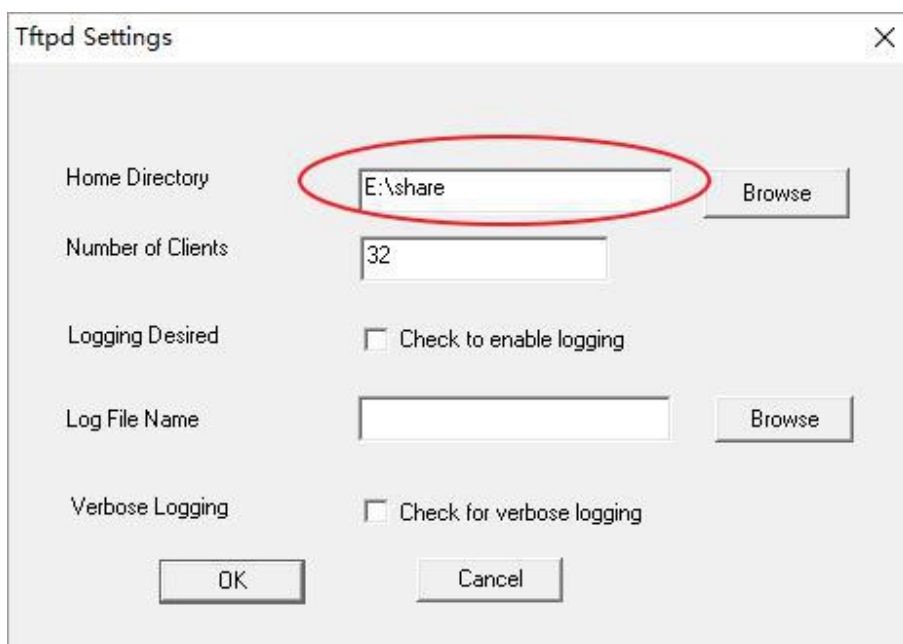


图 2.10 Tftpd Settings 对话框

在该对话框中，需要设置“Home Directory”，该配置用于指定 PC 机和开发板在进行文件传输时所使用的目录。当 PC 机需要把文件下载到开发板时，需要将待下载的文件拷贝到该目录内；同样当开发板需要把文件上传到 PC 机上时，上传的文件也会放在该目录内。配置完成后点击“OK”按钮；

3. 启动 tftpd 服务。点击 Tftpd”菜单下的“Start”子菜单即可。正常启动后的 tftpd.exe 界面如下图所示：



图 2.11 启动 tftp 服务

在 PC 机端配置完 TFTP 服务器后，就可以通过 TFTP 协议在开发板和 PC 机之间传输文件了。在传输文件前，必须首先设置好开发板的 IP 地址，并保证开发板和 PC 机之间网络通讯正常。关于 IP 的设置方法，可参见“[设置 IP 与子网掩码](#)”一节。

如果需要将 PC 机中的文件下载到开发板上，执行如下步骤：

- 将待下载文件拷贝到 TFTP 服务器设置的文件传输目录中，此处假定待下载文件为 rootfs.tar.gz；
- 通过串口或网络登录到开发板上，并使用 tftp 命令下载文件：

```
#cd /home
```

```
# tftp -r rootfs.tar.gz -g 192.168.1.111
```

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 192.168.1.111 的 PC 机上下载 rootfs.tar.gz 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：

通过串口或网络登录到开发板上，并进入文件所在目录，执行如下命令：

```
#cd /home
```

此处假定待上传文件位于/home 目录中；

使用 tftp 命令上传文件：

```
tftp -r rootfs.tar.gz -p 192.168.1.111
```

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。上述

命令行的含义是将开发板中 /home/rootfs.tar.gz 文件上传到 IP 地址为 192.168.1.111 的 PC 机上。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。

如果按照上述步骤执行后，仍然无法通过 TFTP 传输文件，可以试着关闭 PC 机上的 Windows 防火墙，然后再重新进行文件传输。

## 2.11 LED 测试

在开发板/home/demo 目录下，运行 ledtest 程序可以对 LED 进行测试。该程序可控制 LED 闪烁，每 2 秒钟闪烁一次，连续闪烁 10 次。为执行该测试，可在命令行下运行如下命令：

```
~ # cd /home/demo
/home/demo # ./ledtest
```

## 2.12 蜂鸣器测试

在开发板/home/demo 目录下，运行 belltest 程序可以对蜂鸣器进行测试。该程序可控制蜂鸣器连续响 5 次，每次持续时间为 1 秒。为执行该测试，可在命令行下运行如下命令：

```
~ # cd /home/demo
/home/demo # ./belltest
```

## 2.13 串口测试

TWEvb-IMX6UL 开发板提供了多个串口供用户使用：其中 COM1 为调试串口，COM2、COM3、COM8 为 RS232 通讯口，COM4、COM7 为 RS232/RS485 复用通讯口，用户可以通过底板上的跳线进行切换。关于串口的详细说明可以参考《TWEvb-IMX6UL 数据手册》。在开发板/home/demo 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；也可以是设备名，如/dev/ttymx1、/dev/ttymx2、/dev/ttymx3。例如需要通过 COM2 口进行数据收发，在命令行下执行如下命令：

```
~ # cd /home/demo
/home/demo # ./serialtest COM2
```

该程序运行流程如下：

- 打开串口（串口通讯参数为：波特率 9600，8 位数据位，1 位停止位，无数据校验位）；
- 通过串口发送一个 20 字节的数据；
- 从串口接收数据；

- 重复步骤 2~3，实现数据的循环发送和接收。

使用 serialtest 程序进行串口数据收发测试的步骤如下：

- 首先用串口线连接开发板和主机，关于开发板串口引脚的定义可参见文档《TWEvb-IMX6UL 数据手册》。
- 在主机上运行串口调试工具，该工具可接收开发板串口发送的数据，并可发送数据到开发板。Windows 下的串口调试软件比较多，此处以 `sscom32.exe` 为例介绍串口调试工具的使用，该工具可在“眺望产品光盘资料/3、软件开发参考资料/2、开发工具软件\SSCOM32”中直接获取。

运行 `sscom32.exe` 软件，弹出如下对话框，如下图所示：

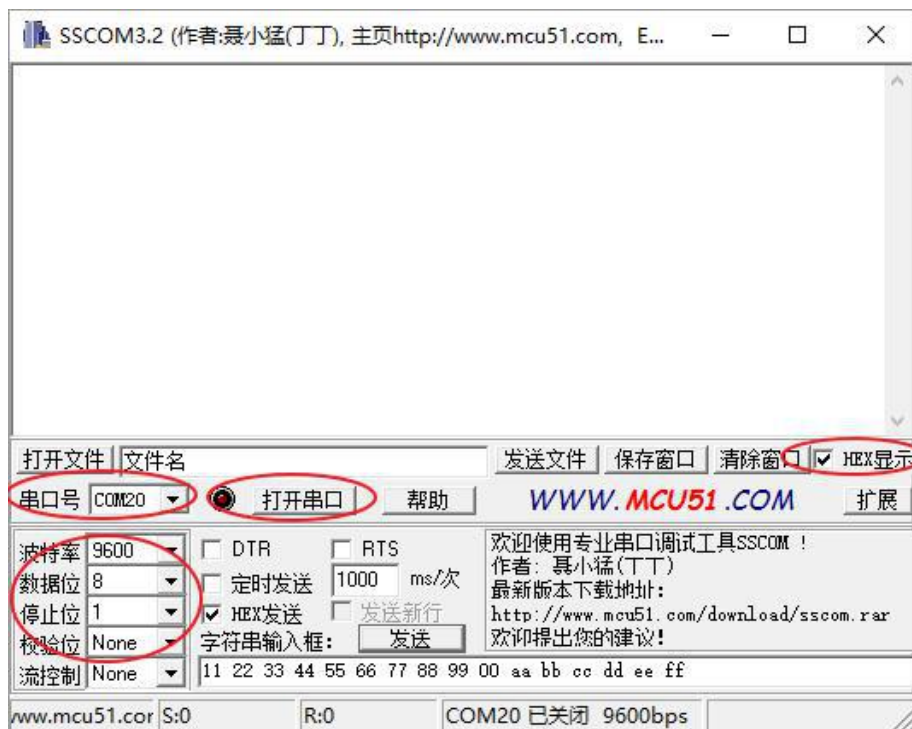


图 2.12 sscom32 主界面

在该对话框中设置主机侧的串口号，并设置串口通讯参数为：波特率 9600，8 位数据位，1 位停止位，无数据校验位。勾选“HEX 显示”选项。设置完成后，点击“打开串口”按钮。如果串口打开成功，`sscom32` 界面如下图所示：

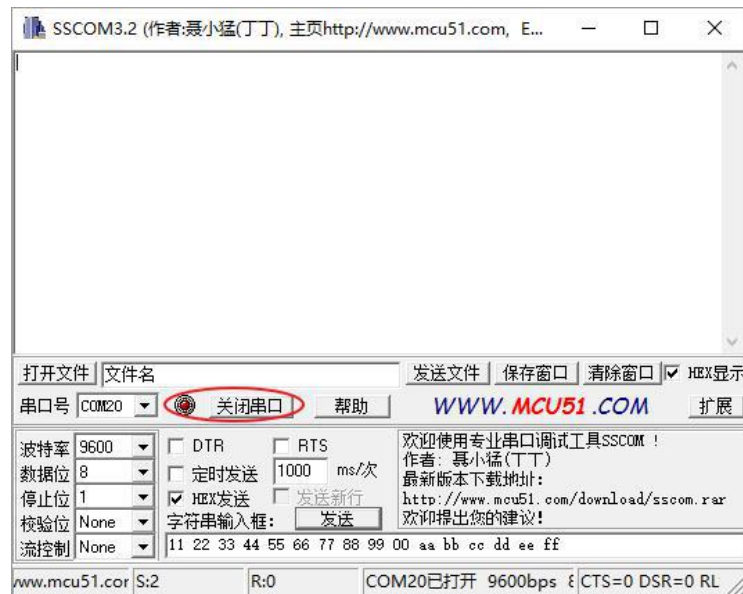


图 2.13 sscom32 串口打开成功

如上图所示，红色标记部分表明串口打开正常。

1. 测试开发板发送数据是否正常。方法如下：

在开发板上运行 serialtest 测试程序。在命令行下执行如下命令：

```
~ # cd /home/demo
/home/demo # ./serialtest COM2
```

此处假定测试开发板 COM2 口。运行 serialtest 后，serialtest 会通过串口发送数据，此时可以在 sscom32 看到 serialtest 发送的数据，如下图所示：



图 2.14 sscom32 接收数据

2.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：



图 2.15 sscom32 发送数据

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：

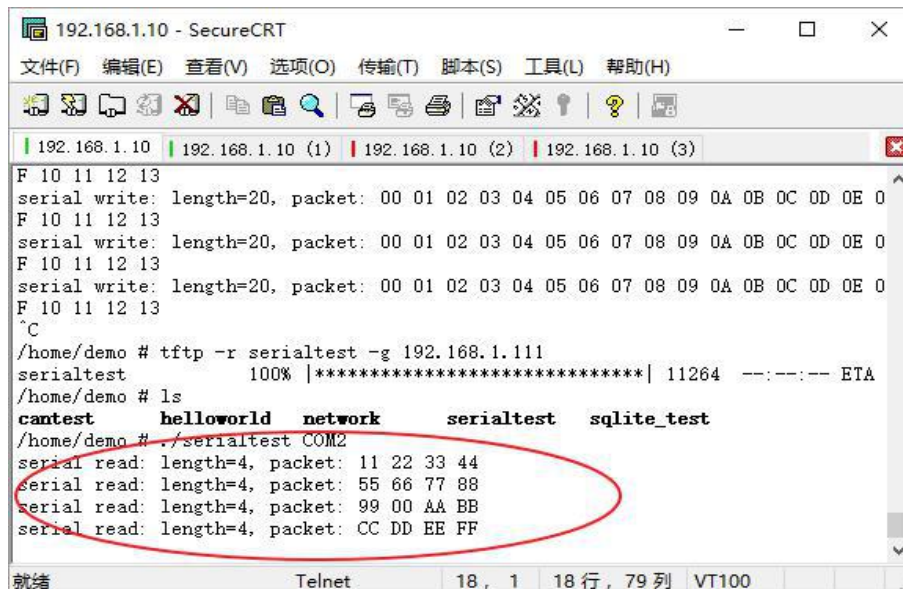


图 2.16 开发板串口接收数据

## 2.14 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 0~7，共 8 个背光级别。当设置为 7 时背光最亮。例如要把背光值设置为 5，可在命令行下执行如下命令：

```
echo 5>/sys/class/backlight/backlight/brightness
```

## 2.15 音频播放测试

TWEvb-IMX6UL 开发板提供 2 种类型的音频输出接口：

- 1、基于 I2S 的音频输出接口（CN31）：提供较高质量的音频输出；
- 2、基于 MQS 的音频输出接口（CN35）：提供中等质量的音频输出。

在开发板/home/audio 目录下，运行 aplay 程序可以对音频播放进行测试。在测试前，需要将耳机插入开发板的耳机插孔中。

为测试基于 I2S 的音频输出接口，在命令行下执行如下命令：

```
~ # cd /home/audio  
/home/demo # ./aplay 1.wav
```

其中 1.wav 为需要播放的音频文件，且此处假定 1.wav 位于/home/audio 目录下。

为测试基于 MQS 的音频输出接口，在命令行下执行如下命令：

```
~ # cd /home/audio  
/home/demo # ./aplay -D mqs 1.wav
```

其中 1.wav 为需要播放的音频文件，且此处假定 1.wav 位于/home/audio 目录下。

在进行音频播放时，对于基于 I2S 的音频输出接口，可以使用 amixer 程序对音量进行调节。该程序位于/home/audio 目录下。使用 amixer 程序对音量进行调节的命令行格式为：

```
amixer cset numid=10 音量值
```

其中音量值取值范围在 0~255 之间，255 表明最大音量。例如要将音量设置为 200，可在命令行下执行如下命令：

```
~ # cd /home/audio  
/home/demo # ./amixer cset numid=10 200
```

## 2.16 WIFI 测试

### 2.16.1 配置 WIFI 网络

与 WIFI 相关的程序和配置文件位于开发板/home/wifi 目录下。其中/home/wifi/wpa\_sup.conf 文件保存了 WIFI 网络的配置信息，如 WIFI 网络的 SSID 和密码等，在连接 WIFI 网络前，必须根据 WIFI 网络的实际情况对该文件进行修改。

在 wpa\_sup.conf 文件中，关于 WIFI 网络 SSID 和密码的配置项如下（通常位于该文件的末尾）：

```
network={
ssid="sincmain"
psk= 2755c6234fd131d6b41e96ffd99c107d6cb019d749c6e19ca15b90b5222bb80d
}
```

其中 ssid 为 WIFI 网络的 SSID，此例为 sincmain；psk 为经过加密后的网络密码。

在开发板/home/wifi 目录下，可以使用 wpa\_passphrase 程序生成上述配置项（包含 SSID 和加密后的密码）。wpa\_passphrase 程序的命令格式如下：

```
wpa_passphrase 网络 SSID 密码
```

其中密码长度为 8~63 个字符。例如要生成 SSID 为 sincmain，密码为 12345678 的配置项，执行如下命令：

```
#cd /home/wifi
# /home/wifi # ./wpa_passphrase sincmain 12345678
```

命令执行完毕后，会在控制台上输出包含 ssid 和 psk 网络信息的配置项，如下所示：

```
network={
ssid="sincmain"
#psk="12345678"
psk=2755c6234fd131d6b41e96ffd99c107d6cb019d749c6e19ca15b90b5222bb80d
}
```

将 wpa\_passphrase 程序生成的配置项拷贝到 wpa\_sup.conf 文件，并删除该文件中原有的关于 ssid 和 psk 的配置项，即可完成对 WIFI 网络的配置。

### 2.16.2 使 WIFI 网络配置生效

在完成对 wpa\_sup.conf 文件的修改后，可以在命令行下直接执行/home/wifi/wifi.sh 脚本，使网络设置立即生效，如下所示：

```
#!/home/wifi/wifi.sh
```

## 2.16.3 测试 WIFI 网络配置

当 WIFI 网络连接成功后，使用 `ifconfig` 命令可以看到 `wlan0` 网络接口已经自动获取了 IP 地址，如下图所示：

```
wlan0 Link encap:Ethernet HWaddr 00:95:69:9B:5C:4A
      inet addr:192.168.1.103 Bcast:255.255.255.255 Mask:255.255.255.0
      UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
      RX packets:22 errors:0 dropped:3 overruns:0 frame:0
      TX packets:18 errors:0 dropped:1 overruns:0 carrier:0
      collisions:0 txqueuelen:1000
      RX bytes:5022 (4.9 KiB) TX bytes:3155 (3.0 KiB)
```

如果 WIFI 网络可以连接互联网，也可以直接在命令行下使用 `ping` 命令，`ping` 某个公网网址，测试网络是否连通。例如 `ping` 百度公司的网址，可在命令行下执行如下命令：

```
~#ping www.baidu.com
```

如果可以 `ping` 通表明 WIFI 网络连接成功。

由于网络信号等问题，开发板有可能在启动过程中，无法使用 `dhcp` 从路由器自动获取 IP，此时可以在命令行下执行 `udhcpc` 命令尝试从路由器再次获取 IP 地址：

```
~#udhcpc -i wlan0
```

# 第 3 章 Linux 应用程序开发

## 3.1 安装 Linux 操作系统

进行嵌入式 Linux 的开发，首先必须要有一个运行 Linux 操作系统的主机环境。搭建主机环境既可以选择在真实的电脑上直接安装 Linux 操作系统，也可以选择虚拟机上安装。所谓虚拟机安装方式，就是指在 Windows 下安装一个虚拟机软件，然后通过虚拟机软件创建一台虚拟电脑，最后在虚拟电脑中安装 Linux 操作系统。在 Windows 下使用虚拟机，除了可以继续使用 Windows 下的工具之外，还有如下优势：

- 一台电脑可以同时存放多台虚拟机，这样就可以存在多个不同版本的 Linux 系统；
- 在硬件允许的情况下，甚至可以同时运行多台虚拟机；
- 安装好的虚拟机可以任意复制和拷贝，方便在不同电脑之间迁移；
- 用户在虚拟机中的任何操作，都不会对主机操作系统带来影响。例如万一用户在执行某个操作或者测试某个功能时，导致虚拟机中的操作系统无法正常启动，那只需重新安装虚拟机中的操作系统就可以了，虚拟机中操作系统出现的任何异常都不会影响到主机上的操作系统。

因此大多数情况下，用户都会选择在虚拟机中安装 Linux 操作系统。目前常用的虚拟机软件有 VMware、Virtual Box 和 Virtual PC 等，不同的虚拟机软件在使用方法上略有不同。本章节将以 VMware 为例进行介绍。

对于主机 Linux 操作系统而言，也存在众多的发行版本，常用的有 RedHat、Fedora、Debian、Ubuntu、SuSe 等。不同的发行版在安装和使用上都存在着一些差异。在进行嵌入 Linux 开发时最常使用的是 Ubuntu 发行版。相比较其他发行版，Ubuntu 有如下优势：

- Ubuntu 流行度广，使用的用户多，这样在遇到问题寻求技术支持时会更为方便；
- Ubuntu 简单易用，尤其是在下载和更新软件时，Ubuntu 可以自动搜索软件包之间的依赖关系，自动完成软件包的安装；
- 现在很多处理器半导体厂商以及开发平台厂商都使用 Ubuntu 发行版，因此如果搭建主机环境时采用 Ubuntu，那么就可以直接使用半导体或者开发平台原厂提供的各种工具，减少开发过程中的障碍。

本章将以 Ubuntu 为例介绍主机环境的搭建过程。

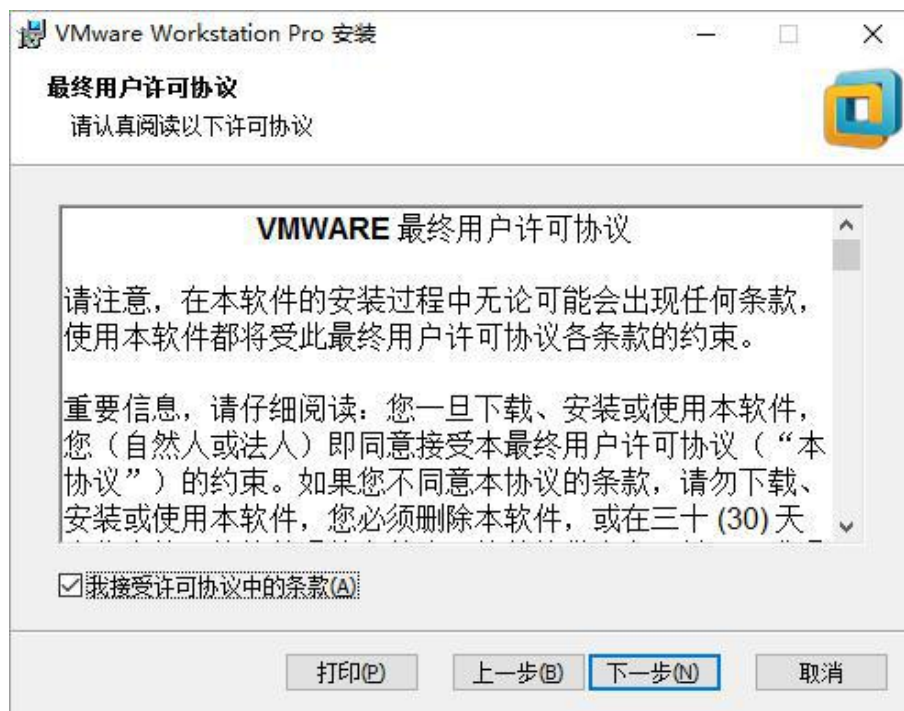
### 3.1.1 VMware 软件

此处以 VMware12.1 版本为例介绍 VMware 软件的安装和使用。

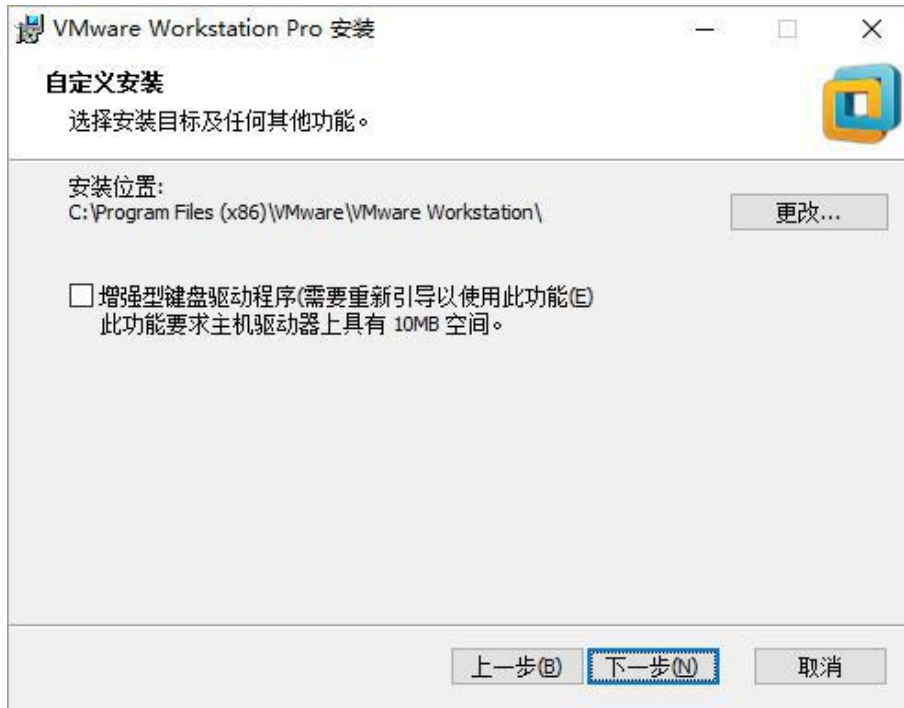
用户可以从 [www.vmware.com](http://www.vmware.com) 下载 VMware 安装程序。双击安装程序，在弹出的对话框中点击“下一步”按钮，如下图所示：



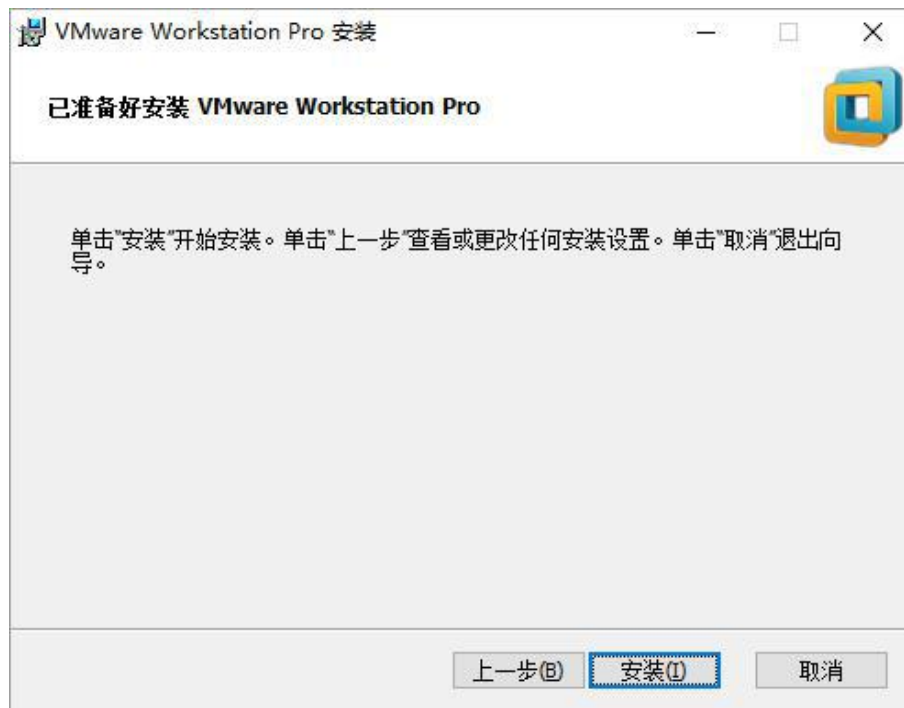
在弹出的“最终用户许可协议”对话框中，勾选“我接受许可协议中的条款”，并点击“下一步”按钮，如下图所示：



在弹出的“自定义安装”对话框中，用户可以更改程序的安装路径，也可以使用软件默认的安装路径，并点击“下一步”按钮，如下图所示：



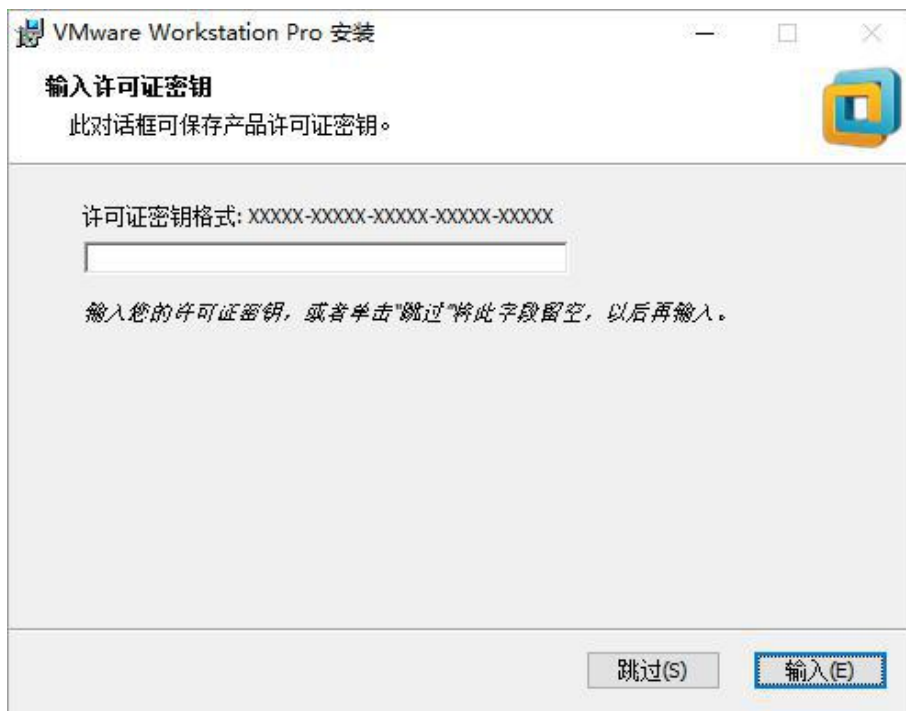
之后采用安装软件的默认设置，一直点击“下一步”按钮，直到出现如下界面：



在该界面上点击“安装”按钮，开始安装。安装完成后，会提示如下界面



在该画面上，用户可以点击“许可证”按钮，输入软件的授权码；也可以先点击“完成”按钮，结束安装，在运行软件时，再输入授权码。如果点击“许可证”按钮，则弹出如下对话框：



在该对话框中，用户输入软件的授权码，然后点击“输入”按钮，弹出如下画面：



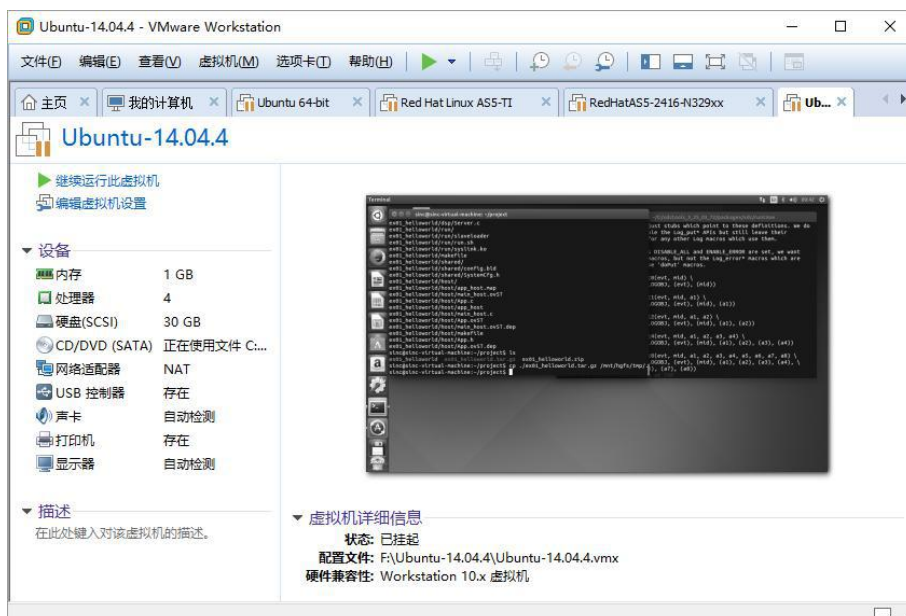


在该画面中点击“完成”按钮即可完成安装。

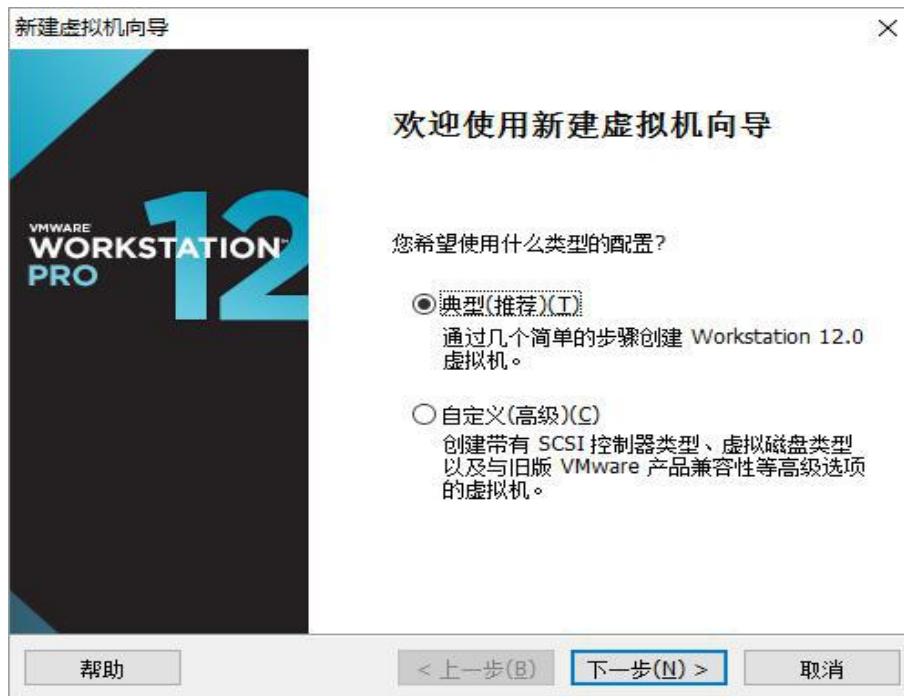
### 3.1.2 创建和配置虚拟机

#### 1. 创建虚拟机

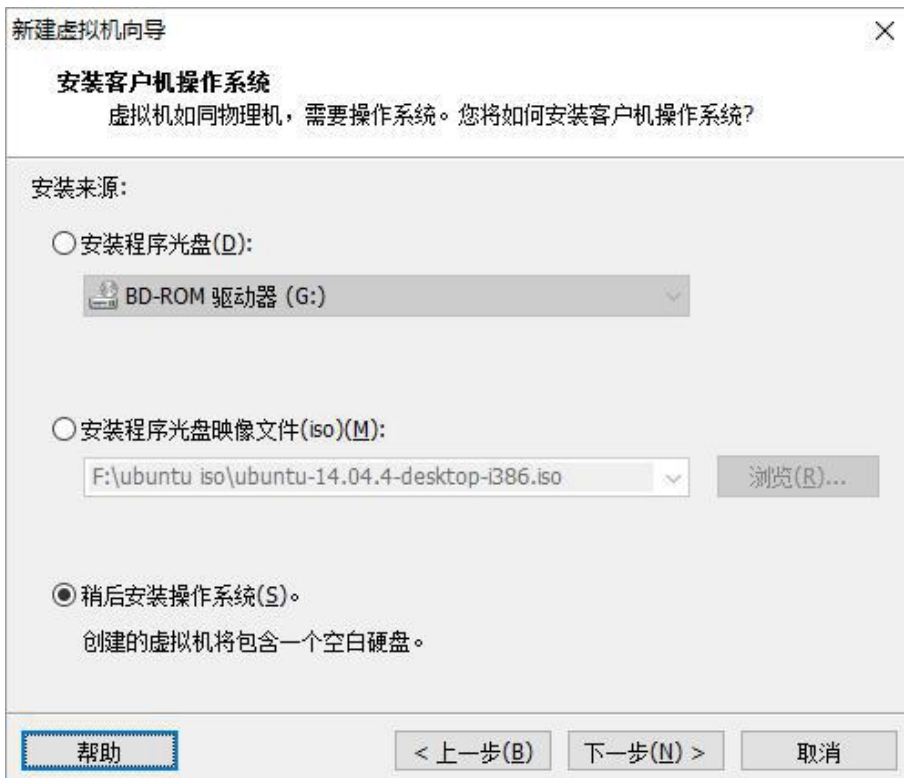
运行 VMware 虚拟机软件，如下图所示：



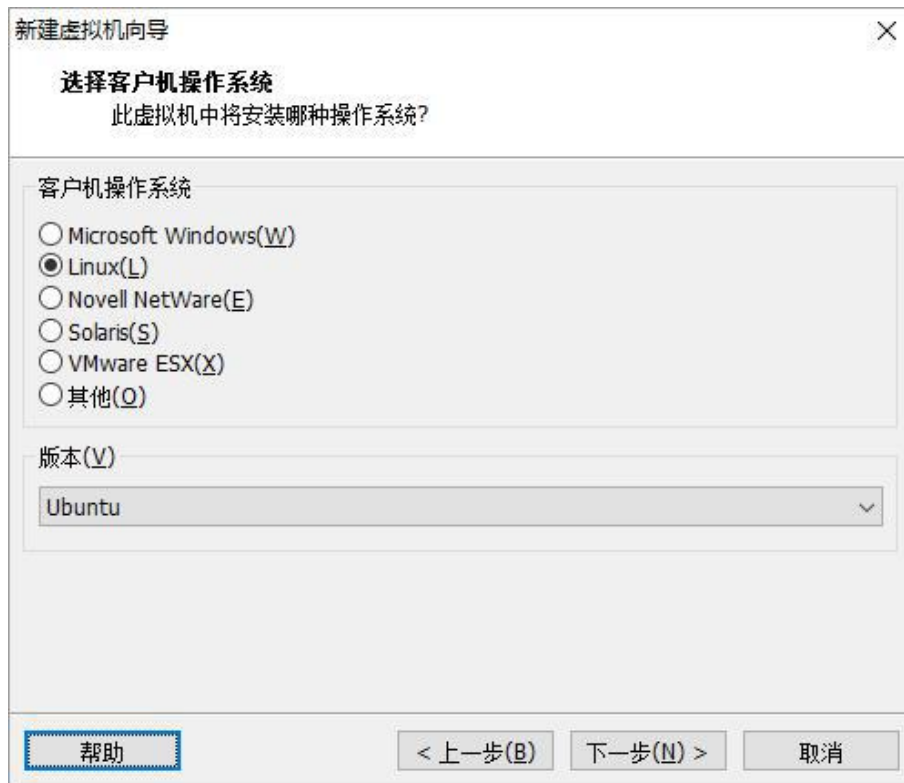
点击“文件”菜单下的“新建虚拟机”子菜单，弹出“新建虚拟机向导”对话框，如下图所示：



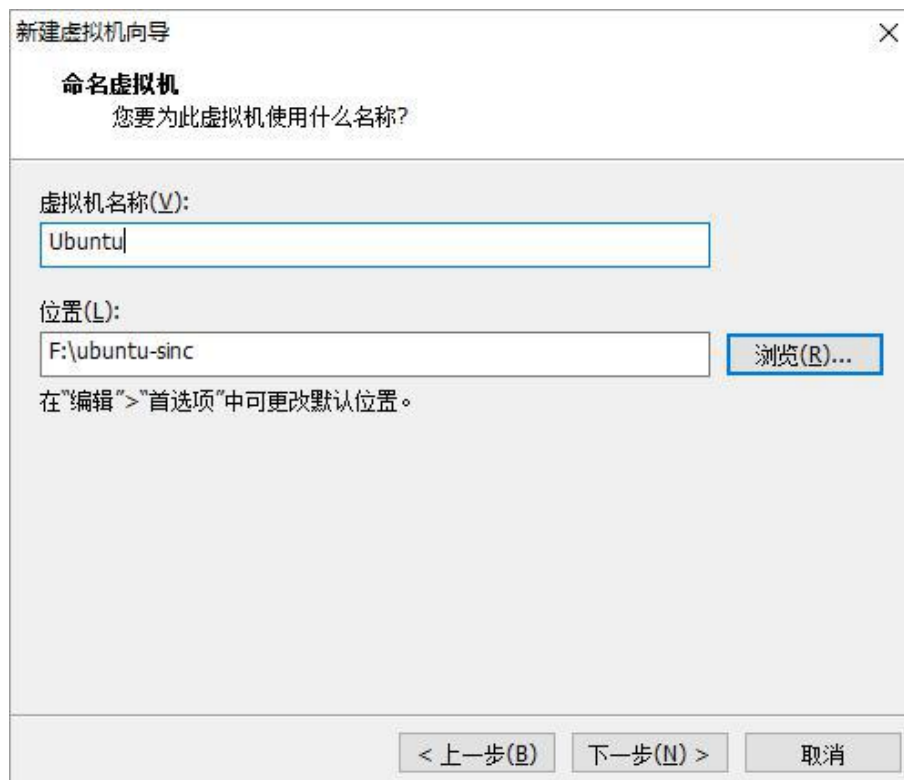
选择“典型（推荐）”，并点击“下一步”按钮，弹出“安装客户机操作系统”对话框，如下图所示：



选择“稍后安装操作系统”选项，然后点击“下一步”按钮。弹出“选择操作系统”对话框，如下图所示：



其中“客户机操作系统”选择 Linux，“版本”则根据用户所下载的 Ubuntu 安装文件的实际版本进行选择，如果要安装的 Ubuntu 是 32 位的版本，则选择 Ubuntu；如果要安装的 Ubuntu 是 64 位的版本，则选择 Ubuntu 64，此处选择安装 32 位版本的 Ubuntu。设置完成后，点击“下一步”按钮，弹出“命名虚拟机”对话框，如下图所示：



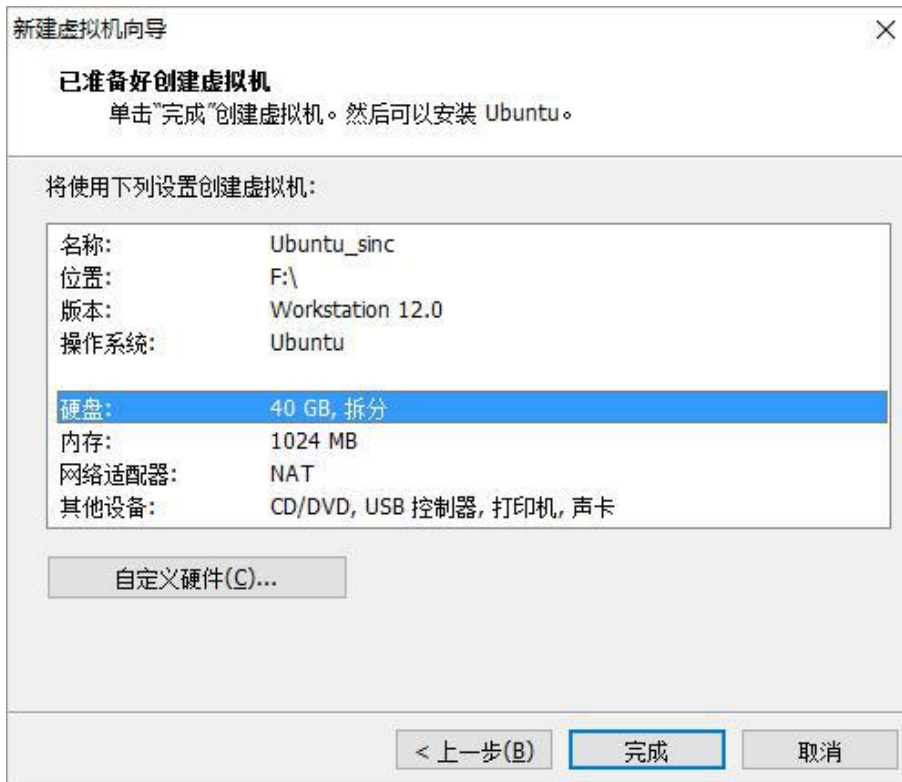
其中“虚拟机名称”可以为虚拟机指定一个任意的名字；“位置”则为新创建的虚拟机指定一个存放路径，虚拟机可以存放在任意目录中，但必须保证该目录所在的磁盘有足够大的剩余空间。设置完成后，点击“下一步”按钮，弹出“指定磁盘容量”对话框，如下图所示：



关于“最大磁盘大小”这个参数说明如下：

- “最大磁盘大小”建议设置到 40G 以上。因为除了安装 Ubuntu 操作系统本身外，还会安装嵌入式 Linux 开发的各种工具，以及对应的源码等，都需要较大的磁盘空间；
- 虽然此处配置了 40G 的磁盘空间，但 VMware 并不会立即占用 40G 实际硬盘空间。虚拟磁盘文件会在使用过程中逐步增大，直到最大容量 40G。尽管不会立即占用 40GB 硬盘空间，但是为了将来方便使用，必须保证放置虚拟机所在的磁盘有超过 40G 的空闲空间。

设置完成后，点击“下一步”按钮，弹出“已准备好创建虚拟机”对话框，如下图所示：



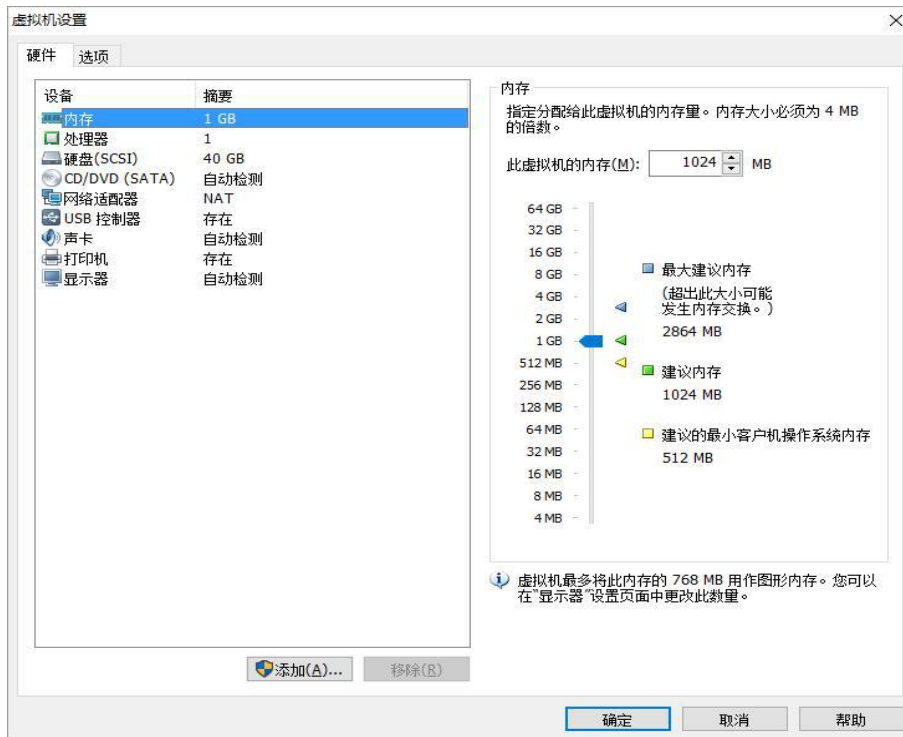
点击“完成”按钮，完成虚拟机的创建工作。

## 2. 配置虚拟机

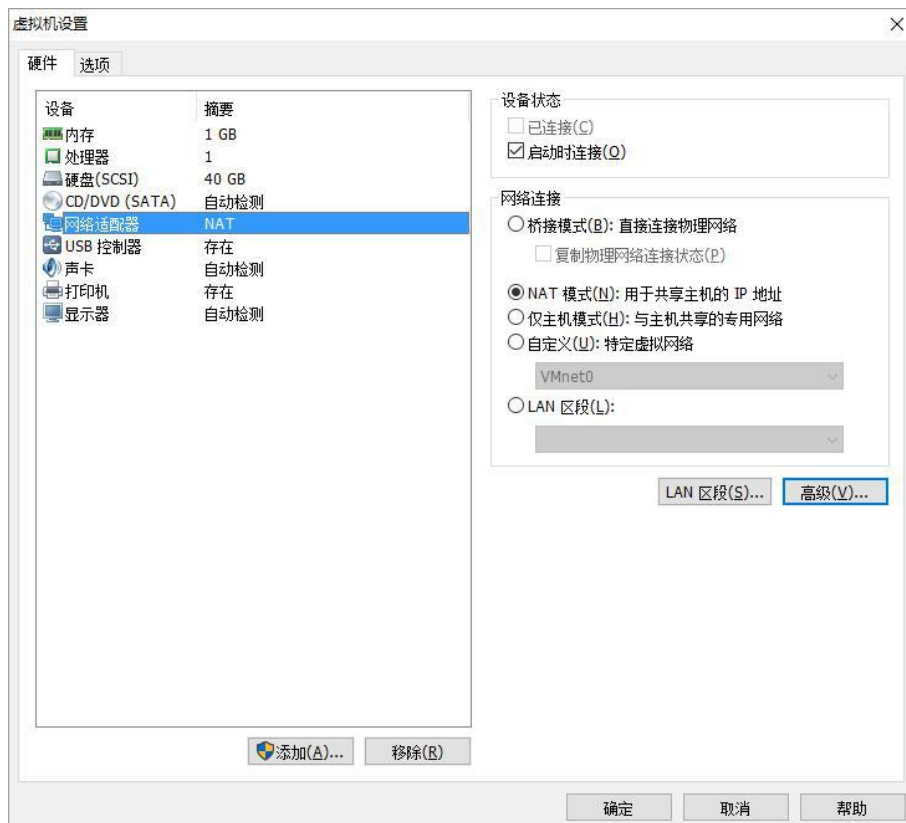
根据需要用户可以修改已创建虚拟机的参数配置，具体方法如下：在虚拟机主界面上，点击需要修改的虚拟机标签页，如下图所示：



点击左侧的“编辑虚拟机设置”按钮，弹出“虚拟机设置”对话框，如下图所示：



在该对话框中，可以对虚拟机的内存、网络等参数进行配置和修改。以下重点介绍网络部分的设置，网络设置界面如下图所示：



如图所示，虚拟机网络可以工作在如下 3 种模式：

#### 1. 桥接模式：

在这种模式下，VMware 虚拟出来的操作系统就像是局域网中的一台独立的主机，它可以访问网内任何一台机器。在桥接模式下，虚拟系统和宿主机器的关系，就像连接在同一个 Hub 上的两台电脑。用户需要手工为虚拟系统配置 IP 地址、子网掩码，而且还要和宿主机器处于同一网段，这样虚拟系统才能和宿主机器进行通信。同时，由于这个虚拟系统是局域网内的一个独立的主机系统，那么就可以手工配置它的 TCP/IP 配置信息，以实现通过局域网的网关或路由器访问互联网。

在进行嵌入式 Linux 开发时，如果需要目标板通过 NFS 文件系统挂载虚拟机的 NFS 共享目录的话，必须将虚拟网卡配置为桥接模式。

#### 2. NAT 模式

使用 NAT 模式，就是让虚拟系统借助 NAT（网络地址转换）功能，通过宿主机器所在的网络来访问公网，也就是说，使用 NAT 模式可以实现在虚拟系统里访问互联网。NAT 模式下虚拟系统的 TCP/IP 配置信息是由 VMnet8(NAT)虚拟网络的 DHCP 服务器提供的，虚拟机无法正常对主机所连网络中的其它主机提供普通的网络服务，如 TFTP、NFS 和 FTP 等。

采用 NAT 模式最大的优势是虚拟系统接入互联网非常简单，用户不需要进行任何其它的配置，只需要宿主机器能访问互联网即可。

#### 3. 仅主机模式

在某些特殊的网络调试环境中，要求将真实环境和虚拟环境隔离开，这时用户就可采用仅主机（Host-Only）模式。在 Host-Only 模式中，所有的虚拟系统是可以相互通信的，但虚拟系统和真实的网络是被隔离开的。

### 3.1.3 安装 Ubuntu

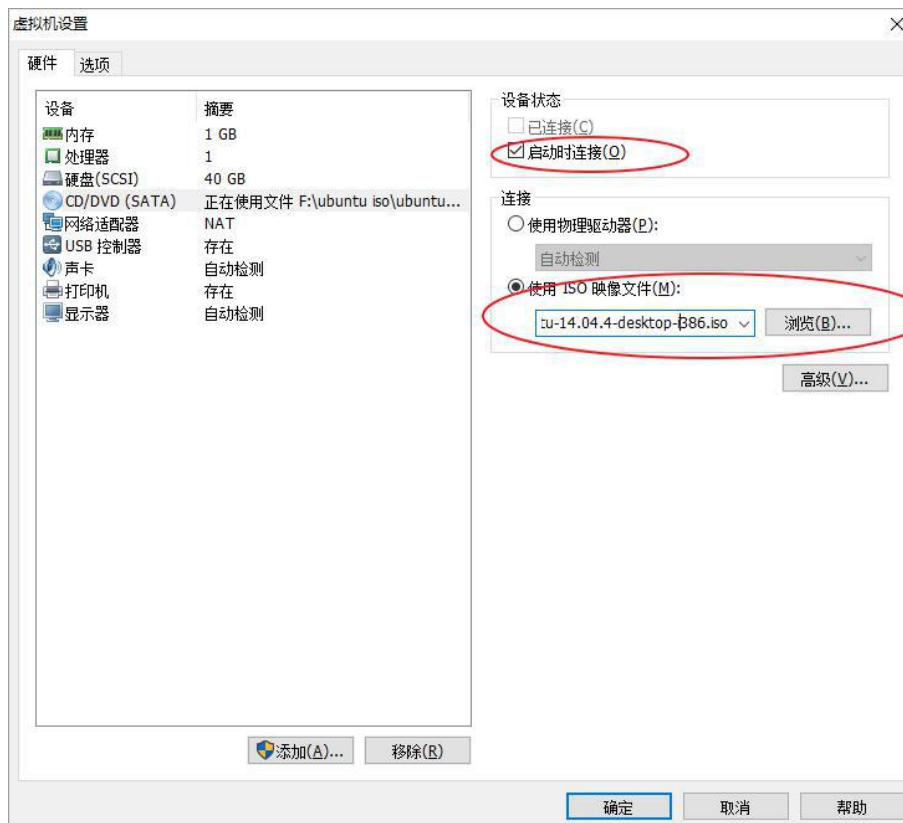
此处以 Ubuntu 14.04.4 版本为例介绍 Ubuntu 的安装和使用。

用户可以从 <http://www.ubuntu.com/download/alternative-downloads> 下载 Ubuntu ISO 光盘映像文件。其中 ubuntu-14.04.4-desktop-i386.iso 为 Ubuntu14.04.4 32 位桌面版本，本节将以该映像文件为例介绍 Ubuntu 的安装过程。

在安装 Ubuntu 前，先按照“创建和配置虚拟机”章节的步骤创建一个 VMware 虚拟机。打开 VMware 软件，在主界面上选中需要安装 Ubuntu 的虚拟机，如下图所示：



点击左侧的“编辑虚拟机设置”按钮，弹出“虚拟机设置”对话框，选择“硬件”标签页下的“CD/DVD (SATA)”标签，进入光驱的配置界面，如下图所示：





选择“使用 ISO 映像文件”，并点击“浏览”按钮，添加 Ubuntu ISO 光盘映像文件，并确认已经勾选了“启动时连接”选项。配置完成后，点击“确定”按钮，返回 VMware 主界面。

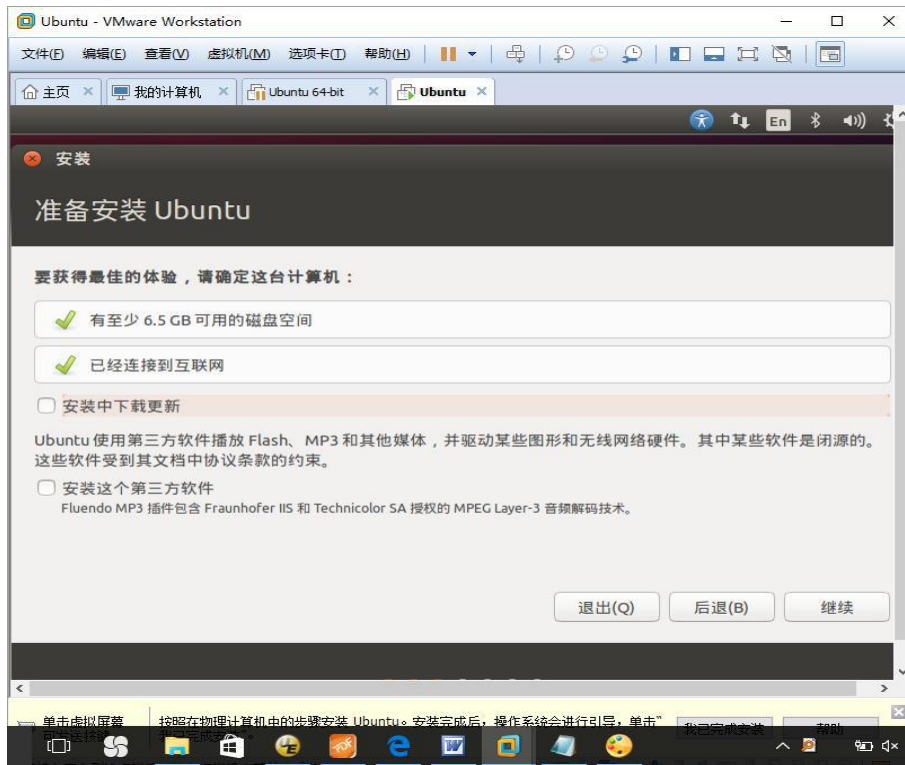
在主界面下，选中刚刚配置过的虚拟机，并点击左侧的“启动此虚拟机”按钮，则可以启动该虚拟机，开始进行 Ubuntu 系统的安装。如下图所示：



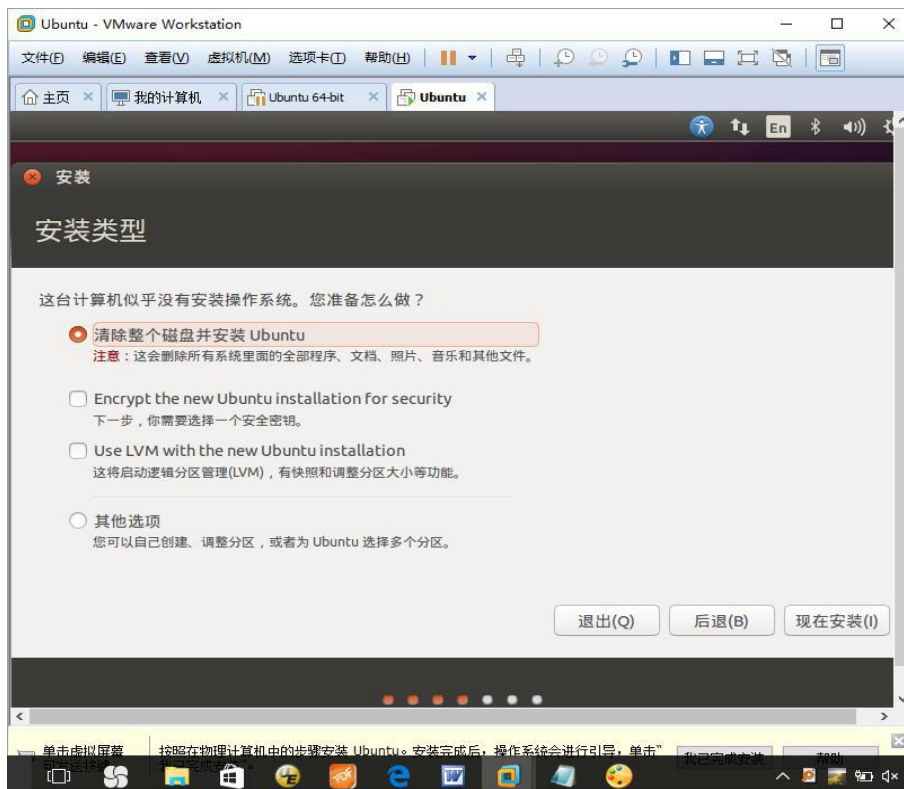
Ubuntu 安装镜像正常启动后，会出现如下图所示的欢迎界面：



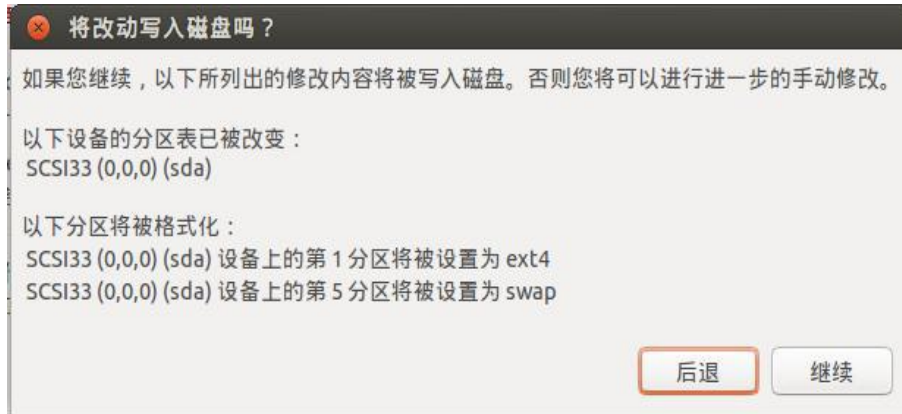
在左侧列表中选择“中文（简体）”，然后单击“安装 Ubuntu”按钮，会出现“准备安装 Ubuntu”界面，如下图所示：



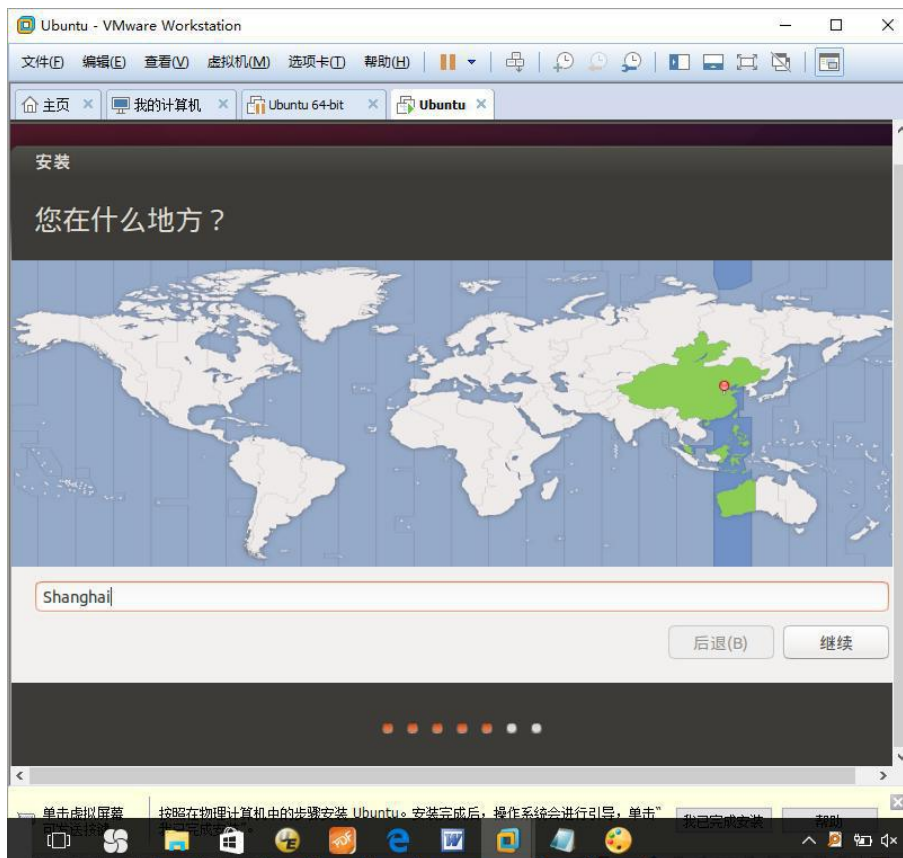
该界面会给出当前安装环境的检测结果，包括系统空闲磁盘以及是否联网等信息。点击“继续”按钮，会出现“安装类型”界面，如下图所示：



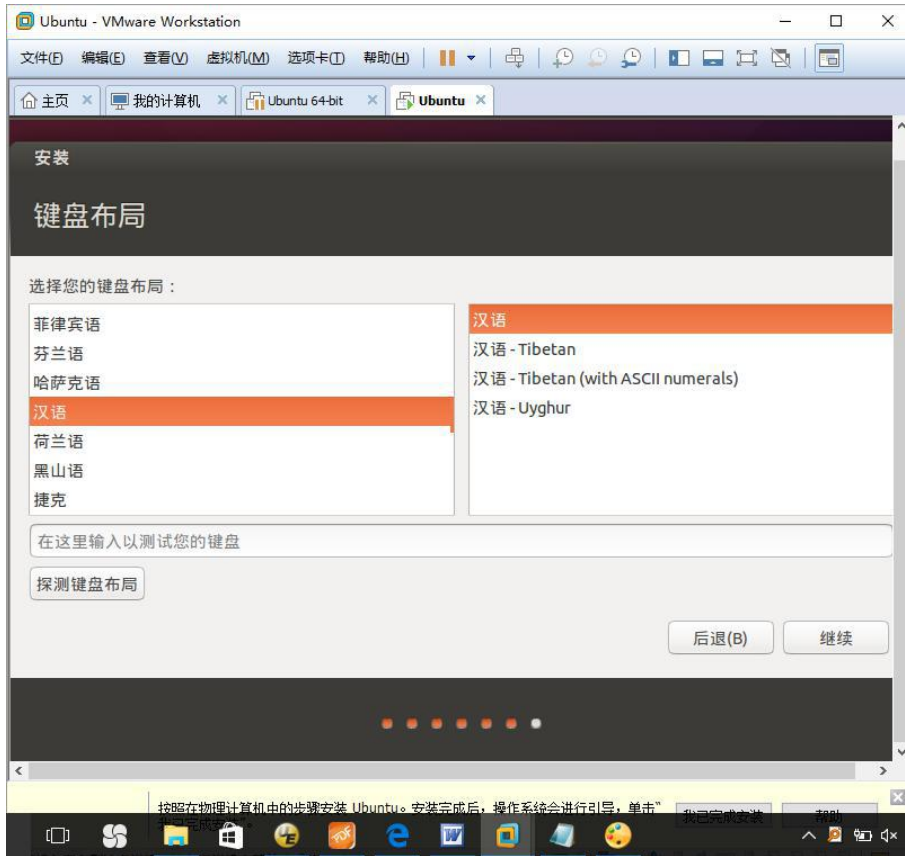
直接使用默认配置：“清除整个磁盘并安装 Ubuntu”，点击“现在安装”按钮，会弹出一个确认“将改动写入磁盘吗？”的提示界面，如下图所示：



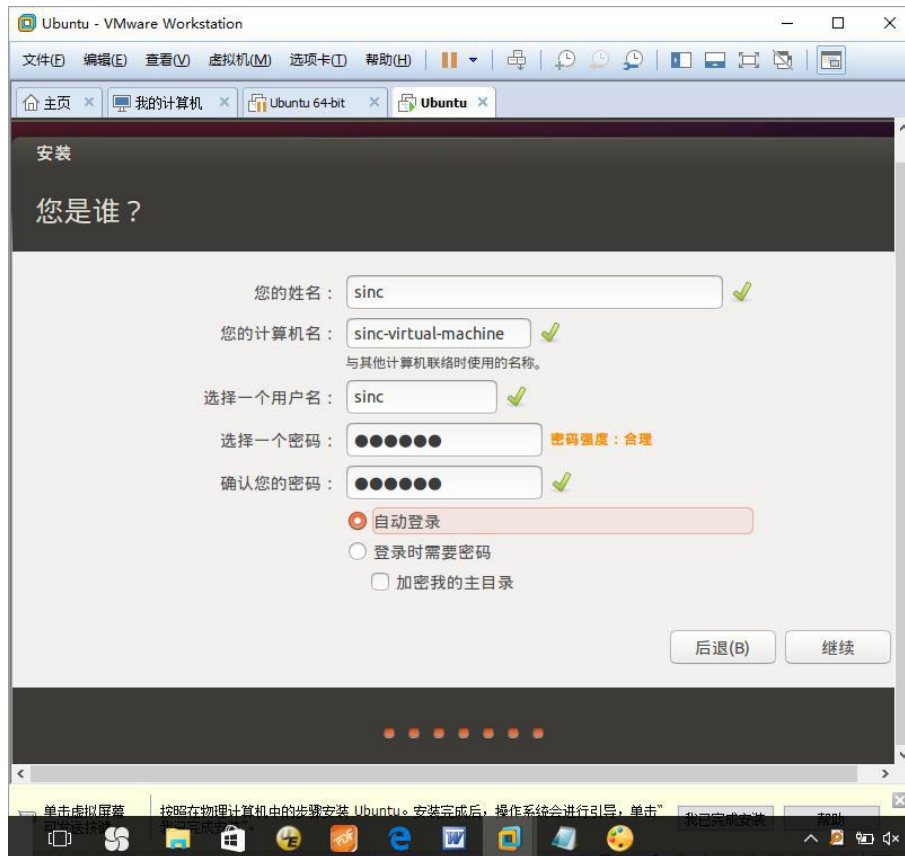
点击“继续”按钮，会出现“地理位置选择”界面，如下图所示：



使用默认值“shanghai”，点击“继续”按钮，会出现“键盘布局”界面，如下图所示：



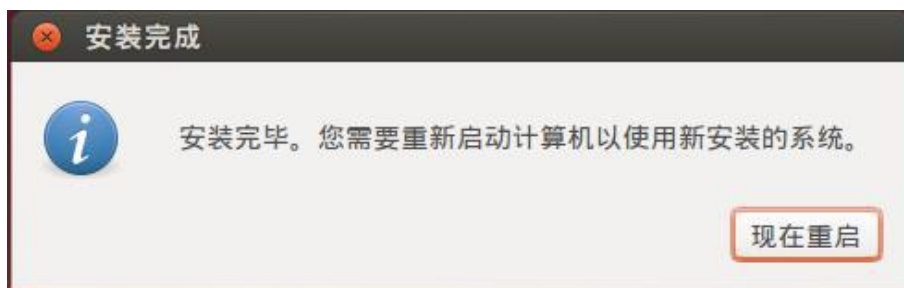
使用默认值“汉语”，点击“继续”按钮，会进入“用户信息设置”界面，如下图所示



在该界面中配置登录 Ubuntu 时的用户名和密码，为了使用更为方便，可以选择“自动登录”选项，这样每次启动系统时，就不会出现系统登录界面，而是直接进入 Ubuntu 桌面。设置完成后，点击“继续”按钮，会出现“正在安装”界面，如下图所示：



耐心等待系统安装完成。系统安装完成后，会出现“安装完成”对话框，如下图所示：



点击“现在重启”按钮，重启后即可进入了 Ubuntu 桌面环境。

主要包含以下要素：常用故障列表、异常操作警告等。

### 3.1.4 安装 VMware Tool

进入 Ubuntu 系统后，屏幕不能全屏，也不能通过共享目录以及通过拖曳方式在 Windows 和虚拟机 Ubuntu 系统之间传输文件，这是因为还需要有安装 VMware Tool 工具。下面介绍 VMware Tools 的安装方法：

点击“虚拟机”菜单下的“安装 VMware Tools”子菜单，系统会自动挂载 VMware Tools

安装包, 安装前需要将该安装包解压然后再进行安装。可以使用“Ctrl+Alt+T”组合按键打开控制台终端, 然后在命令行下执行如下操作(此处假定将安装包解压到/home/sinc 目录下):

```
cd /media/sinc/Vmware\ Tools/  
tar zxvf VMWareTools-8.1.3-203739.tar.gz -C /home/sinc
```

将 VMWareTools-8.1.3-203739.tar.gz 解压到 /home/sinc 目录下。解压成功后, 进入 /home/sinc 目录, 可以看到其中有一个名为 vmware-tools-distrib 的目录。

需要说明的是第一个命令是进入 VMware Tools 安装包所在的路径, 该路径与用户安装 Ubuntu 时所输入的用户名有关, 用户可以采用如下方法确定该目录:

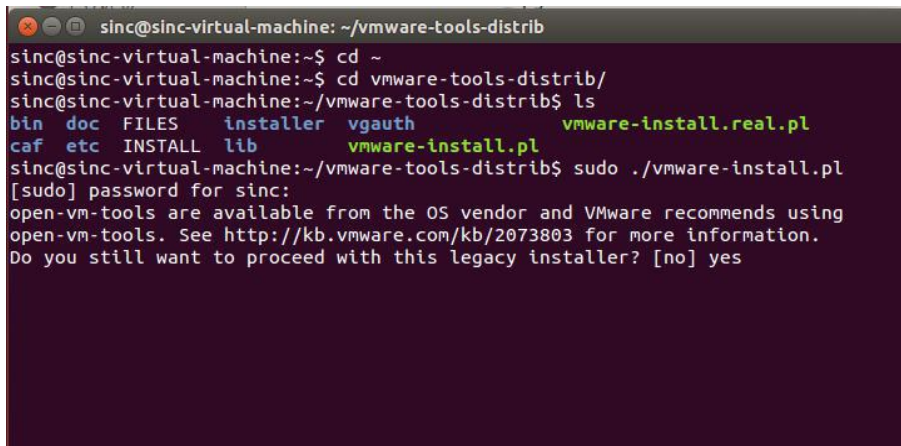
首先在命令行上输入 cd /media/, 然后使用键盘左边的 Tab 键补全剩余的输入, 对于此例当用户第一次按 Tab 键时, 命令行上会出现 cd /media/sinc/, 然后继续再按一次 Tab 键, 命令行上会出现 cd /media/sinc/Vmware\ Tools/, 即可完成输入。

解压完成后, 在命令行下执行如下操作:

```
cd /home/sinc/vmware-tools-distrib  
sudo ./vmware-install.pl
```

执行 sudo 时, 会提示输入 root 用户的密码, 使用 Ubuntu 安装时输入的密码即可。

在安装开始后, VMware Tools 会询问一些安装配置, 第一次询问的内容如下图所示:



```
sinc@sinc-virtual-machine: ~/vmware-tools-distrib  
sinc@sinc-virtual-machine:~$ cd ~  
sinc@sinc-virtual-machine:~$ cd vmware-tools-distrib/  
sinc@sinc-virtual-machine:~/vmware-tools-distrib$ ls  
bin doc FILES installer vgauth vmware-install.real.pl  
caf etc INSTALL lib vmware-install.pl  
sinc@sinc-virtual-machine:~/vmware-tools-distrib$ sudo ./vmware-install.pl  
[sudo] password for sinc:  
open-vm-tools are available from the OS vendor and VMware recommends using  
open-vm-tools. See http://kb.vmware.com/kb/2073803 for more information.  
Do you still want to proceed with this legacy installer? [no] yes
```

除了第一次需要输入 yes, 并按回车外, 后续的询问全部按回车键使用默认配置即可。

### 3.1.5 虚拟机和主机之间传输文件

安装完 VMWare Tool 后, 可以使用如下两种方法在虚拟机和主机之间传输文件:

#### 1. 直接拖曳的方式:

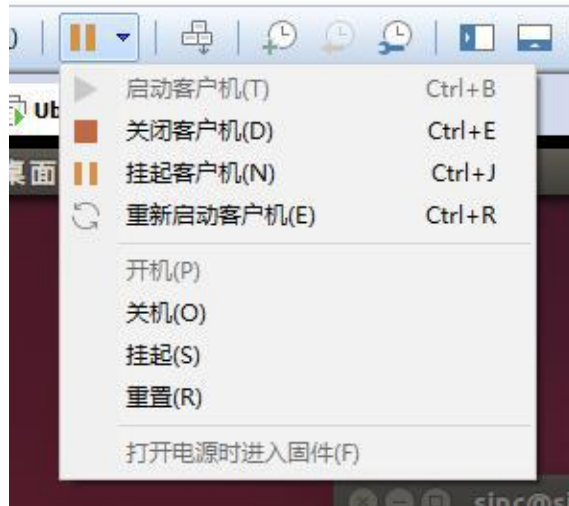
可以直接用鼠标拖曳的方式在主机和虚拟机之间传输文件。如果要将主机文件拷贝到虚拟机中, 只需

用鼠标选中主机文件并将其拖曳到虚拟机相应的目录下即可；而如果要将虚拟机文件拷贝到主机中，也只需鼠标选中虚拟机文件并将其拖曳到主机相应的目录。

## 2. 使用共享目录：

VMwareTools 安装成功后，就可以设置主机和虚拟机之间的共享目录。通过共享目录，主机和虚拟机之间可以方便的传输文件。设置共享目录的方法如下：

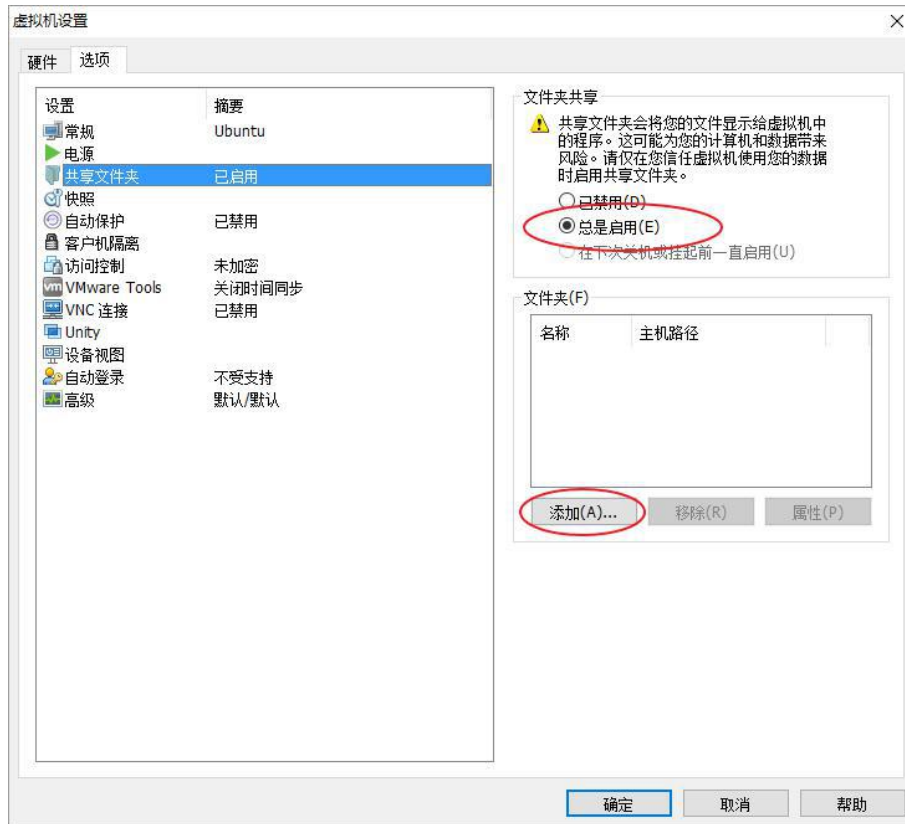
首先点击“关闭客户机”按钮，关闭虚拟机，如下图所示：



虚拟机关闭后，在主界面上选中需要设置共享目录的虚拟机，如下图所示：



点击左侧的“编辑虚拟机设置”按钮，弹出“虚拟机设置”对话框，选择“选项”标签页下的“共享文件夹”标签，如下图所示：



选择“总是启用”，并点击“添加”按钮，弹出“添加共享文件夹向导”对话框，如下图所示：





点击“下一步”按钮，弹出“命名共享文件夹”对话框，如下图所示：



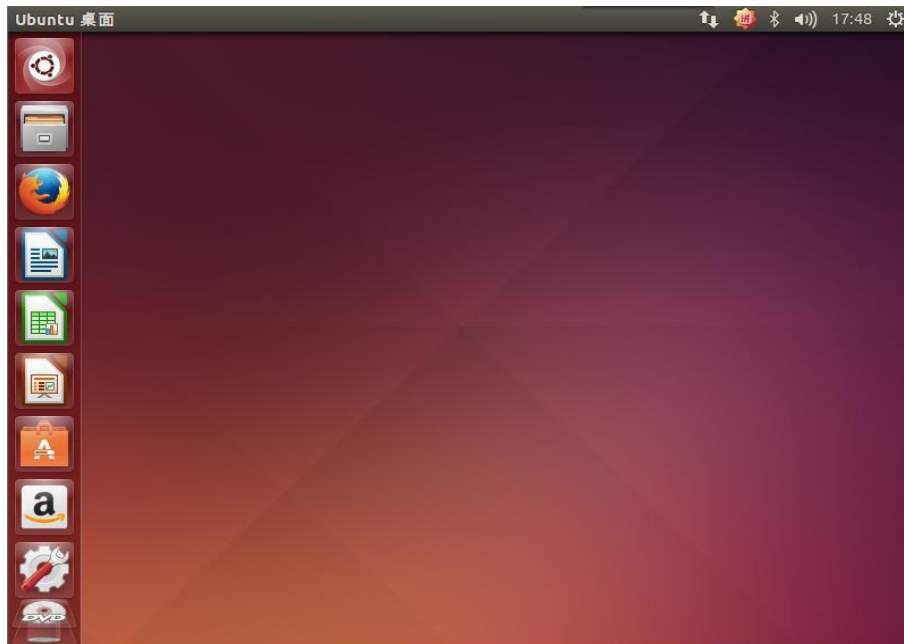
“主机路径”可以是主机上的任意目录，虚拟机和主机将通过这个目录传输文件。“主机路径”可以通过点击“浏览”按钮进行设置。“名称”是在虚拟机中可以看到的目录名。设置完成后，点击“下一步”即可。

重新运行虚拟机，进入 Ubuntu 系统后，可以在/mnt/hgfs 目录下看到前面设置的共享文件夹，通过这个文件夹就可以在主机和虚拟机之间传输文件了。

### 3.1.6 Ubuntu 操作简介

#### 1. Ubuntu 桌面

Ubuntu 启动后，进入桌面系统，桌面环境如下图所示：



在桌面右上角显示的是网络连接、输入法、蓝牙、音量、时间等信息。

桌面的左侧是任务栏。在任务栏上，可以看到 Ubuntu 为用户准备了一些常用的软件：



文件浏览器，用于浏览计算机上的文件；



火狐浏览器；



文档处理软件，类似微软的 Word 软件；



表格处理软件，类似微软的 Excel 软件；



演示文稿软件，类似微软的 PowerPoint；




软件中心，管理已安装的软件，并提供大量软件的下载、安装；



系统设置。


## 2. 输入法

在桌面的右上角点击  图标打开输入法菜单，如下图所示：



从该菜单上可以看到系统已经默认安装了一种中文输入法。按 Shift 键可以用来切换中英文输入。点击“Perference”可以对输入法进行设置。

## 3. 系统设置

在任务栏上点击  图标，即可打开系统设置窗口，如下图所示：




在该窗口中用户可以对系统时间、网络、显示等参数进行设置，该窗口类似于 Windows 下的“控制面板”。

如果用户进入 Ubuntu 系统后，桌面没有全屏显示，此时就可以在“系统设置”里点击“显示”图标，进入“显示”配置界面，对显示的分辨率进行配置。

#### 4. 搜索软件和文件



在 Ubuntu 桌面环境，用户可以用 Dash 工具查找软件、文件和目录。在任务栏点击  图标，即可打开 Dash 的界面，如下图所示：



在 Dash 界面显示了用户最近打开的程序和文件的图标。Dash 界面还有一个“搜索”输入框，用于搜索已安装的程序或者文件。

#### 5. 打开终端

Dash 的搜索输入框输入“terminal”，即可搜索到终端程序。按“Ctrl+Alt+T”组合键也可以打开终端窗口。按“Ctrl+Shift+T”键可以在终端窗口打开一个新的终端标签页，按“Alt+1”、“Alt+2”等可以在各终端标签页之间进行切换。

#### 6. 安装软件

Ubuntu 中一般使用 apt-get 命令安装软件。但前提是虚拟机必须已经连接到互联网。apt-get 命令在执行时会在网上下载指定的软件包，然后完成安装。

例如需要安装 vim 软件，可以执行如下命令：

```
$ sudo apt-get install vim
```

如果要卸载安装好的 vim，可以执行如下命令：

```
$ sudo apt-get remove vim
```

除了使用 `apt-get` 命令外，Ubuntu 也提供了很好的图形界面让用户查找、安装自己所需的软件。在



Ubuntu 桌面的左侧任务栏，点击

图标，即可打开软件中心，如下图所示：



在软件中心，用户可以很方便的查找、下载和安装软件，或者卸载已经安装好的软件。

## 3.2 嵌入式 Linux 开发简介

由于嵌入式系统资源匮乏，一般不能像 PC 机一样安装本地编译器和调试器，不能在本地编写、编译和调试自身运行的程序，而需借助其它系统如 PC 机来完成这些工作，这样的系统通常被称为宿主机。

宿主机通常是 Linux 系统，并安装交叉编译器、调试器等工具；宿主机也可以是 Windows 系统，安装嵌入式 Linux 集成开发环境。在宿主机上编写和编译代码，通过串口、网口或者硬件调试器将程序下载到目标系统里运行。

所谓的交叉编译，就是在宿主机平台上使用某种特定的交叉编译器，为某种与宿主机不同平台的目标系统编译程序，得到的程序在目标系统上运行而非在宿主机本地运行。这里的平台包含两层含义：一是核心处理器的架构，二是所运行的系统。

交叉编译器是在宿主机上运行的编译器，但是编译后得到的二进制程序却不能在宿主机上运行，而只能在目标机上运行。交叉编译器命名方式一般遵循“处理器-系统-gcc”这样的规则，一般通过名称便可以知道交叉编译器的功能。例如下列交叉编译器：

- `arm-uclinuxeabi-gcc`，表示目标处理器是 ARM，运行 uClinux 操作系统；
- `arm-none-linux-gnueabi-gcc`，表示目标处理器是 ARM，运行 Linux 操作系统；
- `mips-linux-gnu-gcc`，表示目标处理器是 MIPS，运行 Linux 操作系统。

进行 ARM Linux 开发，通常选择 `arm-linux-gcc` 交叉编译器。ARM-Linux 交叉编译器可以自行从源代码编译，也可以从第三方获取。在能从第三方获取交叉编译器的情况下，请尽量采用第三方编译器而不要自行编译，一是编译过程繁琐，不能保证成功，二是就算编译成功，也不能保证交叉编译器的稳定性，编译器的不稳定性会对后续的开发带来无限隐患。而第三方提供的交叉编译器通常都经过比较完善的测试，确认是稳定可靠的。

## 3.3 安装交叉编译器

TWEvb-IMX6UL 开发板编译内核使用 `arm-linux-gnueabi-gcc 4.9.2` 交叉编译器，编译用户应用程序使用 `arm-none-linux-gnueabi-gcc 4.5.1` 版本交叉编译器。以下分别说明这两种编译器的安装过程：

### 3.3.1 安装编译内核的交叉编译器

编译用户应用程序的编译器为 `arm-linux-gnueabi-gcc 4.9.2` 版本交叉编译器，其安装包为 `gcc-linaro-arm-linux-gnueabi-gcc-4.9-2014.09_linux.tar.bz2`，可从“产品光盘资料/3、软件开发参考资料/2、开发工具软件/`gcc-linaro-arm-linux-gnueabi-gcc-4.9-2014.09_linux.tar.bz2`”中直接获取。具体安装步骤如下：

1、将 `gcc-linaro-arm-linux-gnueabi-gcc-4.9-2014.09_linux.tar.bz2` 拷贝到 Ubuntu 虚拟机中（方法可参考“虚拟机和主机之间传输文件”节的描述），此处假定拷贝到 `/home/sinc/tools` 目录下；

2、使用“`Ctrl+Alt+T`”组合按键打开控制台终端，然后在命令行下执行如下操作(此处假定将安装包解压到 `/home/sinc/tools` 目录下)：

```
cd /home/sinc/tools/  
tar jxvf gcc-linaro-arm-linux-gnueabi-gcc-4.9-2014.09_linux.tar.bz2
```

将 `gcc-linaro-arm-linux-gnueabi-gcc-4.9-2014.09_linux.tar.bz2` 解压到 `/home/sinc/tools` 目录。解压成功后，进入 `/home/sinc/tools` 目录，可以看到其中有一个名为 `gcc-linaro-arm-linux-gnueabi-gcc-4.9-2014.09_linux` 的目录。

3、设置 `PATH` 环境变量

参考“设置 `PATH` 环境变量”章节的方法，将内核交叉编译器的编译路径设置到 `PATH` 环境变量中。

4、测试编译器是否可以正常运行

在编译器安装完成，并设置好 `PATH` 环境变量后，可以简单的测试一下交叉编译器是否可以正常运行。方法如下：

在命令行下执行：

```
$arm-linux-gnueabi-gcc -v
```

如果能显示编译器的版本信息，则表明编译器已经可以正常运行了。

### 3.3.2 安装编译应用程序的交叉编译器

编译用户应用程序的编译器为 arm-none-linux-gnueabi-gcc 4.5.1 版本交叉编译器，其安装包为 arm-2010.09-50-arm-none-linux-gnueabi.bin，可从“眺望产品光盘资料/3、软件开发参考资料/2、开发工具软件/ arm-2010.09-50-arm-none-linux-gnueabi.bin”中直接获取。具体安装步骤如下：

1、将 arm-2010.09-50-arm-none-linux-gnueabi.bin 拷贝到 Ubuntu 虚拟机中(方法可参考“虚拟机和主机之间传输文件”节的描述)，此处假定拷贝到/home/sinc/tools 目录下；

2、由于 Ubuntu 默认的 SHELL 脚本解析器是 dash,但执行 arm-2010.09-50-arm-none-linux-gnueabi.bin 需要 bash 解析器，因此在安装前，需要将 Ubuntu 的脚本解析器更改为 bash。方法如下：

打开终端窗口，执行如下命令：

```
sudo dpkg-reconfigure dash
```

此时会弹出“正在设定 dash”的提示框，如下图所示：

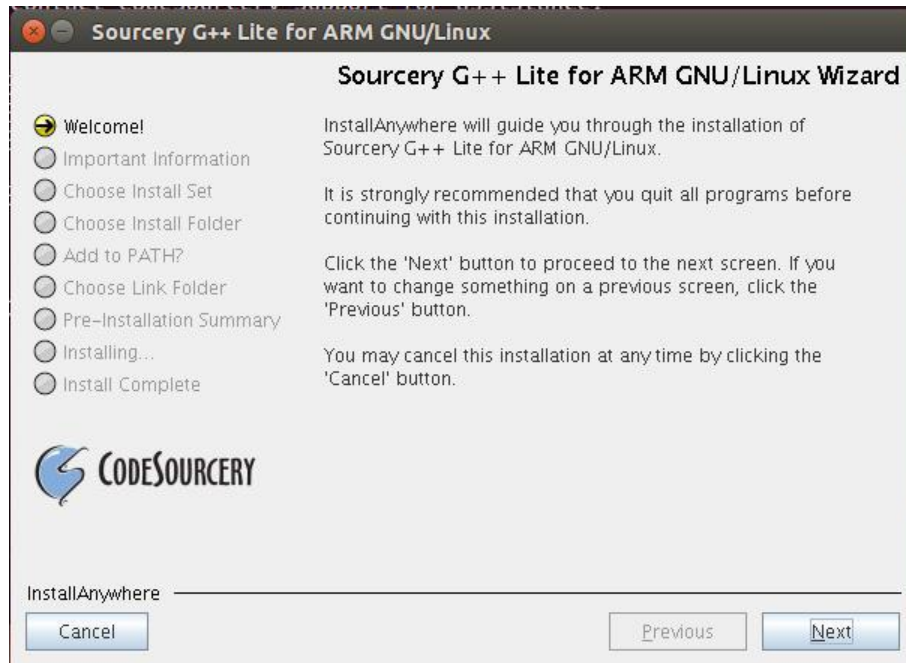


按“Tab”键，选择“否”，然后按确认键完成设置。

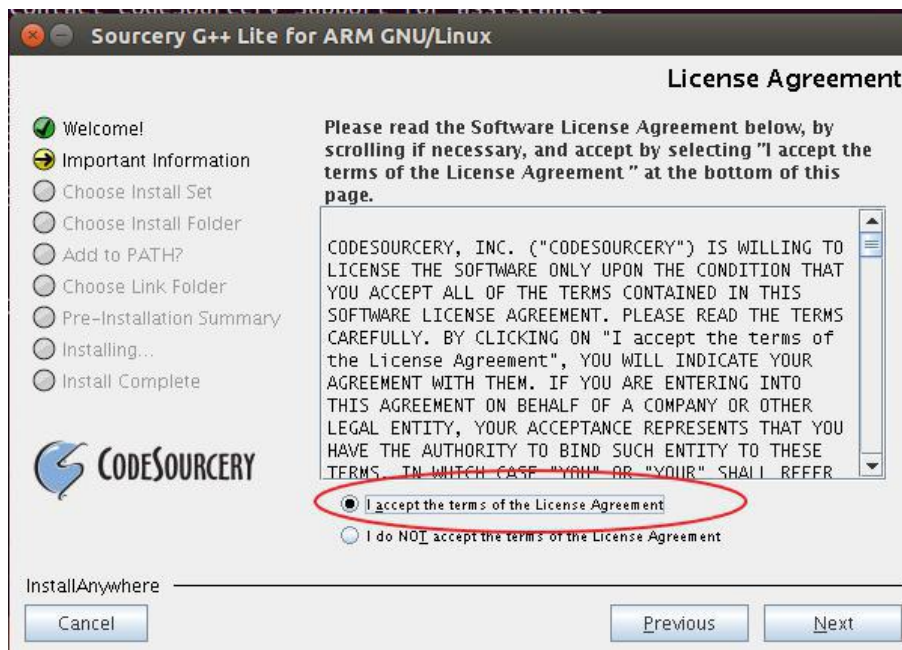
3、在命令行下运行安装程序：

```
$cd /home/sinc/tools $. /arm-2010.09-50-arm-none-linux-gnueabi.bin
```

弹出“Wizard”对话框，如下图所示：

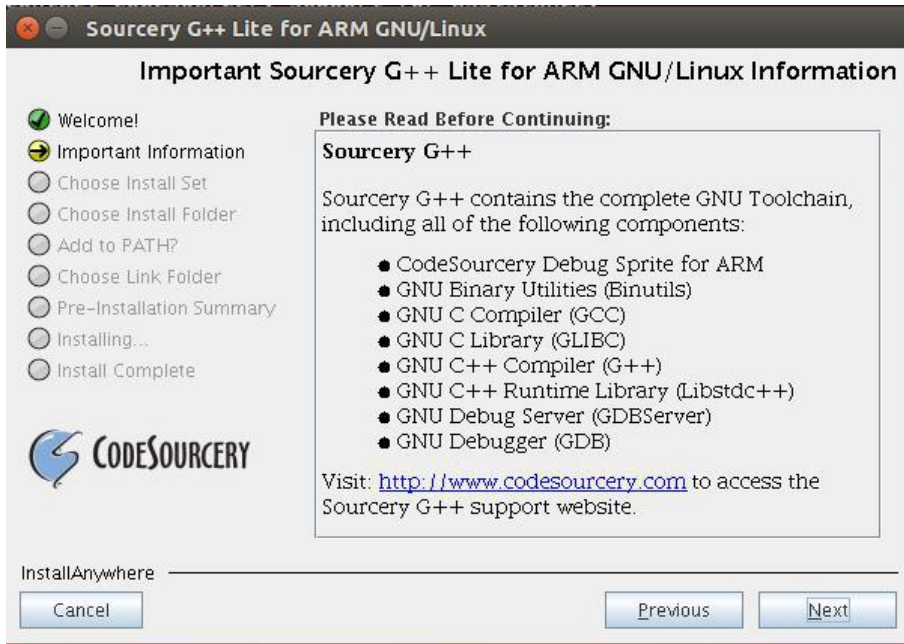


点击“Next”按钮，弹出“License Agreement”对话框，如下图所示：

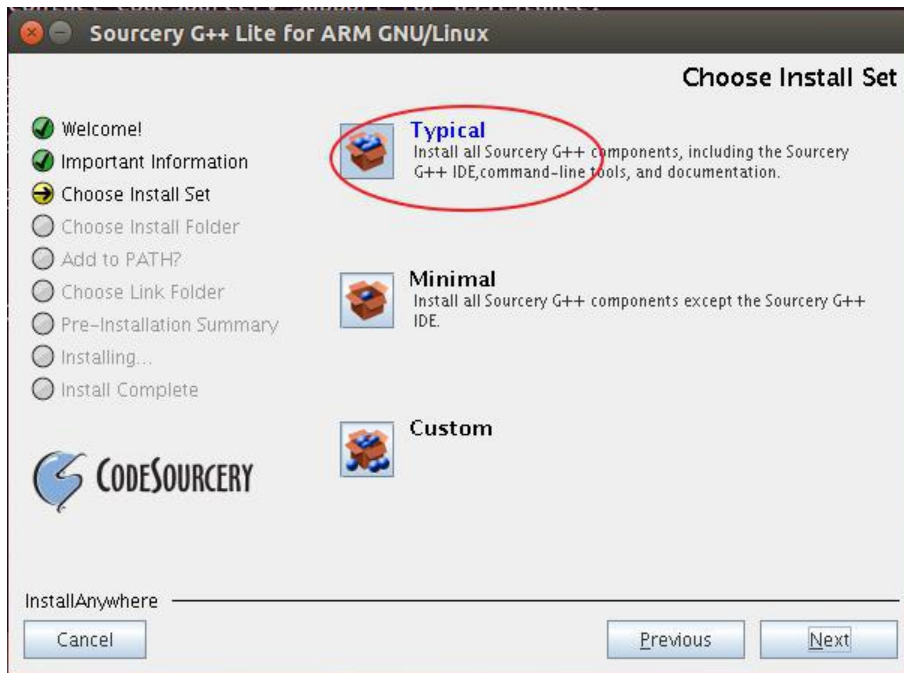


选择“I accept the terms of the License Agreement”，并点击“Next”按钮，弹出“Import information”弹出框，如下图所示：

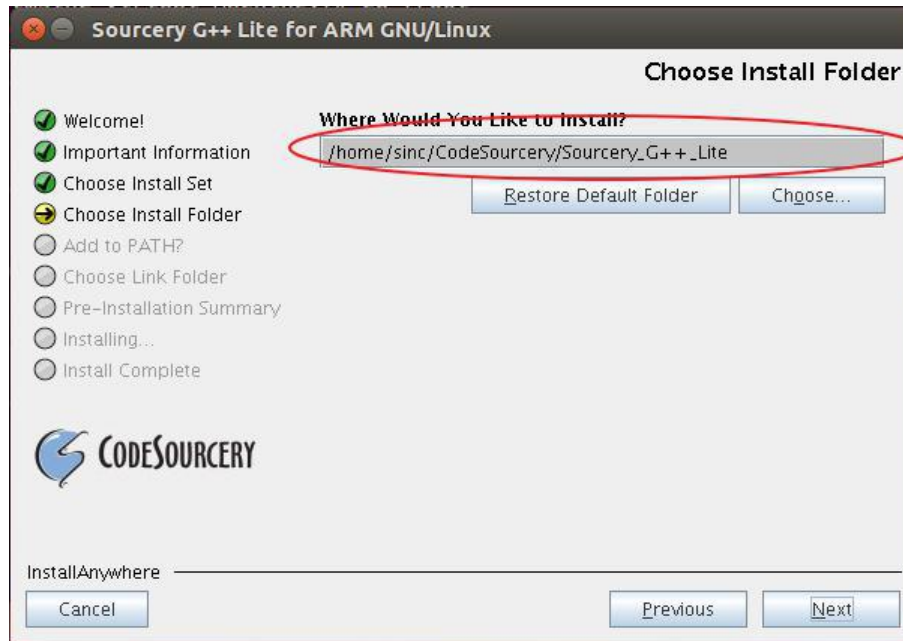




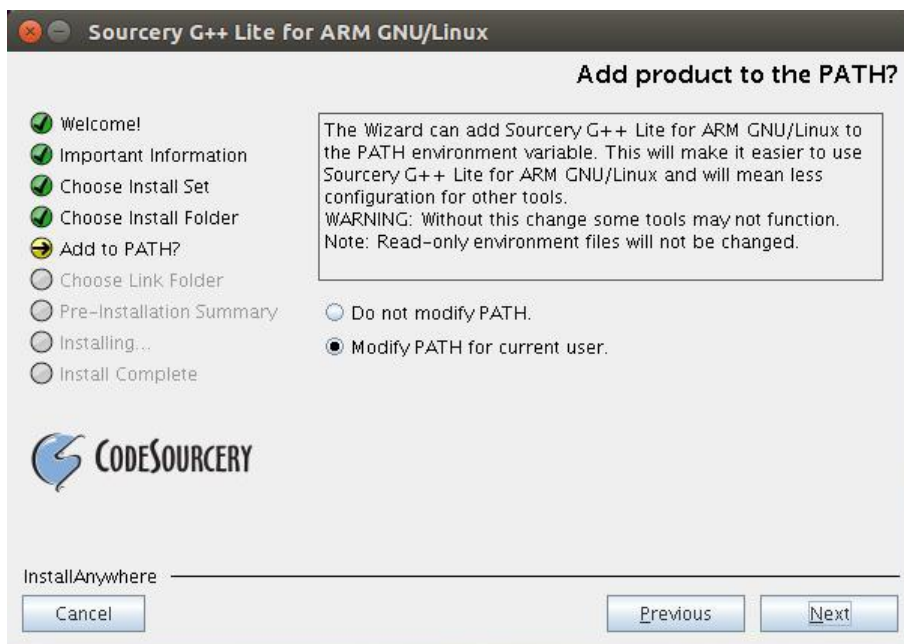
点击“Next”按钮，弹出“Choose Install Set”对话框，如下图所示：



选择“Typical”，并点击“Next”按钮，弹出“Choose Install Folder”对话框，如下图所示：

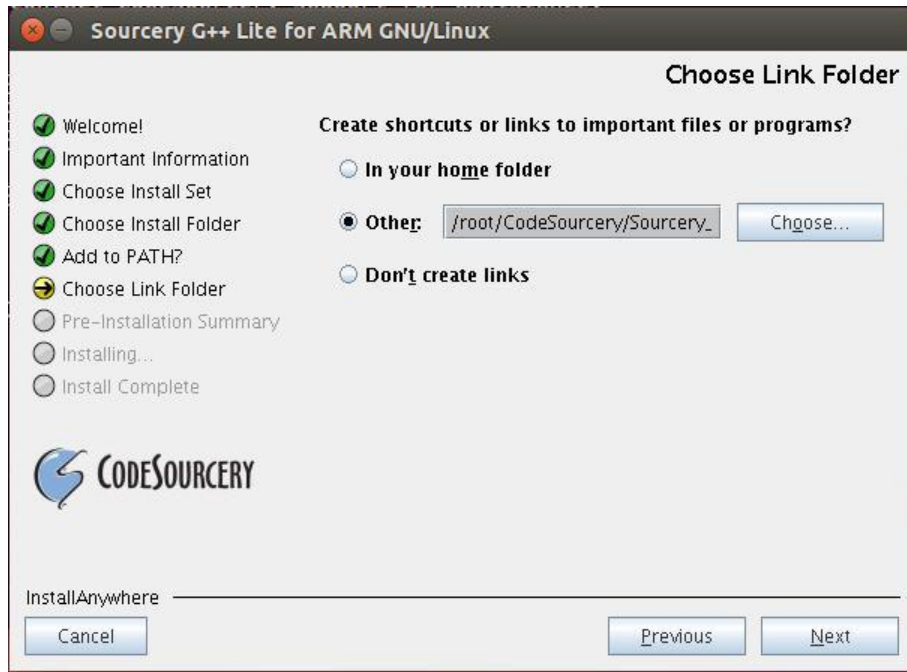


在该对话框中可以设置编译器的安装路径，建议保持默认配置。点击“Next”按钮，弹出“Add product to the PATH”对话框，如下图所示：

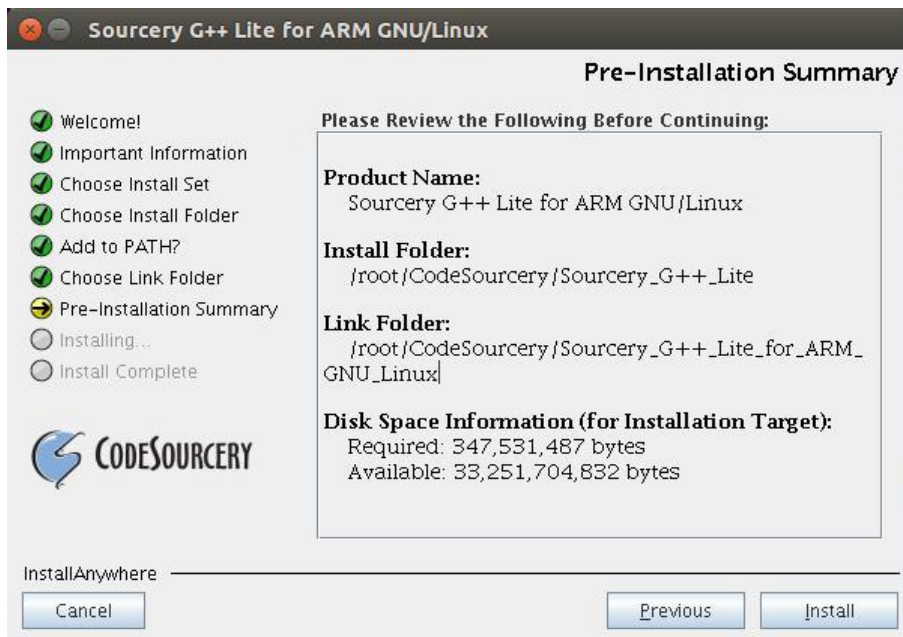


在该对话框中，用户可以选择是否要将编译器路径设置到 PATH 环境变量中。但经过实测，发现即使用户选择“Modify PATH for current user”，在安装完成后，安装程序也没有将编译器路径设置到 PATH 环境变量中，因此安装结束后，必须自己设置 PATH 环境变量。

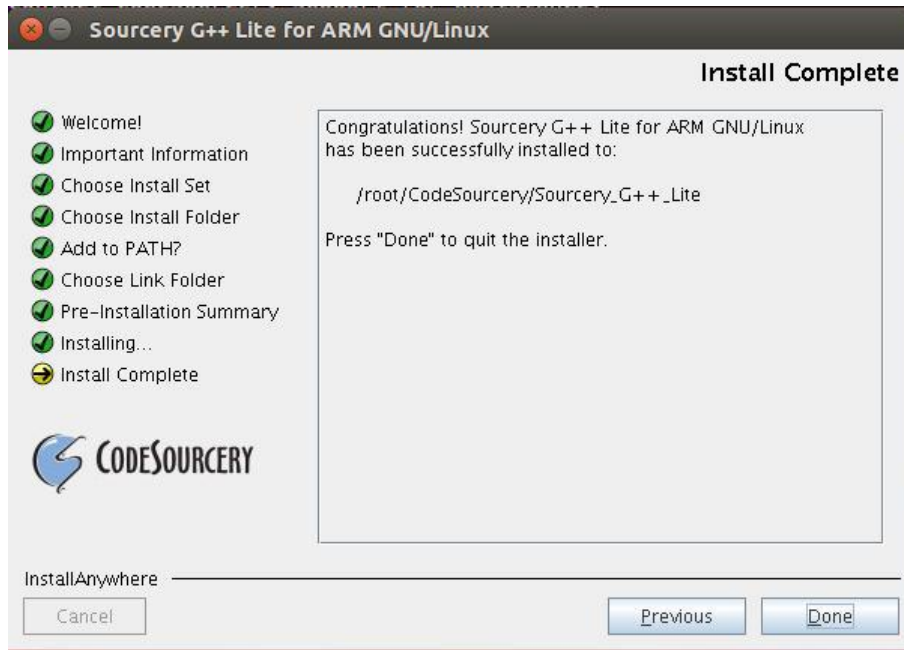
点击“Next”按钮，弹出“Choose Link Folder”对话框，如下图所示：



使用默认值，点击“Next”按钮，弹出“Pre-Installation Summary”对话框，如下图所示：



点击“Install”按钮，开始安装，安装完成后，弹出“Install Complete”对话框，如下图所示：



点击“Done”按钮，完成安装。

#### 4、设置 PATH 环境变量

参考“设置 PATH 环境变量”章节的方法，将应用程序交叉编译器的编译路径设置到 PATH 环境变量中。

#### 5、测试编译器是否可以正常运行

在编译器安装完成，并设置好 PATH 环境变量后，可以简单的测试一下交叉编译器是否可以正常运行。方法如下：

在命令行下执行：

```
$arm-none-linux-gnueabi-gcc -v
```

如果能显示编译器的版本信息，则表明编译器已经可以正常运行了。

### 3.3.3 设置 PATH 环境变量

安装完成后，需要将内核交叉编译器路径和应用程序交叉编译器路径分别设置到 PATH 环境变量中。可采用如下两种方法进行设置：（下文假定内核交叉编译的安装路径为 /home/sinc/tools/gcc-linaro-arm-linux-gnueabi-hf-4.9-2014.09\_linux，应用程序交叉编译器的安装路径为/home/sinc/）

#### A)临时设置：

临时设置系统环境变量，是通过 export 命令将交叉编译器的路径添加到系统 PATH 环境变量中。可在命令行下执行如下命令：（假定交叉编译器安装在/home/sinc 目录下）

```
$ export PATH=$PATH:/home/sinc/  
tools/gcc-linaro-arm-linux-gnueabi-hf-4.9-2014.09_linux/bin:/home/sinc/CodeSourcery/Sourcery_G++_Lite/bin
```

这种方法设置环境变量，只对当前终端有效，将终端关闭并再次打开终端时，该设置就会失效，需要重新设置。

B)修改全局配置文件：

修改/etc/profile 系统配置文件，在该文件中设置交叉编译器的路径。采用这种方法，能够让登录本机的全部用户都可以使用这个编译器，而且在机器重启后依然有效。具体方法为：

打开终端，在命令行下输入：

```
$sudo vi /etc/profile
```

打开/etc/profile 文件，在文件末尾添加：

```
export PATH=$PATH:/home/sinc/  
tools/gcc-linaro-arm-linux-gnueabi-hf-4.9-2014.09_linux/bin:/home/sinc/CodeSourcery/Sourcery_G++_Lite/bin
```

文件修改完毕后，保存并退出。然后，在命令行下执行如下命令：

```
$source /etc/profile
```

执行 profile 文件，使刚才的改动生效。

## 3.4 Hello, World!

本节以一个“HelloWorld”示例来介绍如何在 Linux 宿主机上进行 C 语言程序的编写和编译以及如何将编译出的程序下载到开发板上并在开发板上运行该程序。该例程功能十分简单，就是在控制台上打印“Hello World”字符串。

该示例的源码可以从“产品光盘资料/3、软件开发参考资料/ 3、应用软件示例/HelloWorld”中获取。

### 3.4.1 编写 HelloWorld 源程序

在 Linux 宿主机任意目录下创建一个 HelloWorld 文件夹，用于存放 helloworld.c 源文件。此处假定在/home/sinc/demo 下创建 HelloWorld 文件夹。执行的命令如下：

```
$mkdir /home/sinc/demo
```

```
$mkdir /home/sinc/demo/HelloWorld
```

```
$cd /home/sinc/demo/HelloWorld
```

在该目录下新建一个 helloworld.c 的源文件，其内容如下：

```
#include <stdio.h>
```

```
int main(int argc, char *argv[])
```

```
{
```

```
    printf("Hello World\n");
```

```
return 0;
```

保存文件并退出。

### 3.4.2 编译 helloworld 程序

进入源码所在目录，在命令行下执行如下命令编译程序：

```
$arm-none-linux-gnueabi-gcc-o helloworld helloworld.c
```

其中"helloworld.c"为需要编译的源文件，"-o helloworld"表明交叉编译后可执行文件名为"helloworld"。编译成功后，执行 ls 命令，可以看到在 HelloWorld 目录下会产生一个 helloworld 的可执行文件。

### 3.4.3 下载程序

假设 windows 主机和虚拟机之间的共享目录为/mnt/hgfs/share，则使用如下命令将 helloworld 程序拷贝到 windows 主机上：

```
$cp /home/sinc/HelloWorld/helloworld /mnt/hgfs/share
```

拷贝完成后，在 windows 主机设置的共享目录中，可以找到 helloworld 可执行文件。按照“与 PC 互传文件”一节所示的方法把文件下载到开发板中。

### 3.4.4 运行程序

登录到开发板上，进入 helloworld 所在的目录，为 helloworld 文件增加可执行权限，并执行程序。具体的命令行如下：

```
#chmod a+x helloworld
# ./helloworld
Hello Wold
```

从执行结果可以看出，在终端上已经打印出了“Hello World”字符串。

## 3.5 QT 编程

### 3.5.1 QT 介绍

Qt 是一个跨平台的 C++ 图形用户界面库，由挪威 TrollTech 公司出品(2008 年被诺基亚收购，2012 年被 Digia 收购)，Qt 支持包括 Linux 在内所有的 Unix 系统，同时也支持 Windows 平台。Qt 目前是 Linux 下主流的图形界面支撑环境，Linux 桌面系统 KDE 也是基 Qt 开发的。Qt 具备如下优点：

1. 优良的跨平台特性

Qt 支持下列操作系统: Microsoft Windows 95/98, Microsoft Windows NT, Linux, Solaris, SunOS,

HP-UX, Digital UNIX (OSF/1, Tru64), Irix, FreeBSD, BSD/OS, SCO, AIX, OS390, QNX 等等。

## 2. 面向对象

Qt 的良好封装机制使得 Qt 的模块化程度非常高, 可重用性较好, 对于用户开发来说是非常方便的。Qt 提供了一种称为 signals/slots 的安全类型来替代 callback, 这使得各个元件之间的协同工作变得十分简单。

## 3. 丰富的 API

Qt 包括多达 250 个以上的 C++ 类, 还替供基于模板的 collections, serialization, file, I/O device, directory management, date/time 类。甚至还包括正则表达式的处理功能。

## 4. 支持 2D/3D 图形渲染, 支持 OpenGL

## 5. 大量的开发文档

## 6. XML 支持

### 3.5.2 编译 QT

此处以 QT4.7.4 为例介绍 QT 的编译流程, 编译 QT 时使用的交叉编译器为 arm-none-linux-gnueabi-gcc 4.5.1。

#### 1. 编译环境搭建

在开始编译 QT 前, 需做如下准备工作:

- 在 Ubuntu 虚拟机中安装交叉编译器, 交叉编译器的安装步骤可参考“安装交叉编译器”一节;
- 编译 Qt 需要安装 g++ 编译器。在 Ubuntu 命令行下直接输入 g++, 系统会提示用户是否安装。如果没有安装 g++, 可在命令行提示符下执行如下命令进行安装:

```
$ sudo apt-get install g++
```

- 编译 Qt 需要 tslib 头文件和库文件。在“产品光盘资料/3、软件开发参考资料/5、源代码 /tslib\_arm.tar.gz”中有已经编译好的 tslib 库, 可以直接使用。

#### 2. 编译步骤

环境搭建完毕后, 可按如下步骤编译 QT:

- 将 QT4.7.4 源码“产品光盘资料/3、软件开发参考资料/5、源代码 /qt-everywhere-opensource-src-4.7.4.tar.gz”拷贝到 Ubuntu 虚拟机任意目录下, 此处假定拷贝到 /home/sinc/qt-4.7.4 目录下。

- 对 Qt 源码进行解压。在命令行下执行如下命令：

```
$ cd /home/sinc/qt-4.7.4
```

```
$tar zxvf qt-everywhere-opensource-src-4.7.4.tar.gz
```

- 将“眺望产品光盘资料/3、软件开发参考资料/5、源代码/tslib\_arm.tar.gz”拷贝到 Ubuntu 虚拟机任意目录下，此处假定拷贝到/home/sinc 目录下。并对其进行解压。在命令行下执行如下命令：

```
$ cd /home/sinc
```

```
$tar zxvf tslib_arm.tar.gz
```

- 进入 Qt 源码目录，并将 Qt 交叉编译工具链修改为 arm-none-linux-gnueabi-，在命令行下执行如下命令：

```
$ cd qt-everywhere-opensource-src-4.7.4/
```

```
$ cp -a mkspecs/qws/linux-arm-g++/ mkspecs/qws/linux-sinc-g++
```

```
$ sed -i "s/arm-linux/arm-none-linux-gnueabi/g" mkspecs/qws/linux-sinc-g++/qmake.conf
```

- 配置 Qt，如指定 Qt 安装目录，编译平台，工具链等。在命令行下执行如下命令：

```
$echo -e "o\nyes" | ./configure \ --
```

```
prefix=/home/sinc/qt-arm-4.7.4 \ -
```

```
embedded arm \
```

```
-xplatform qws/linux-sinc-g++ \
```

```
-release -opensource -shared -fast -confirm-license \
```

```
-no-largefile \
```

```
-no-exceptions \ -
```

```
no-accessibility \
```

```
-stl \ -qt-sql-
```

```
sqlite \
```

```
-no-qt3support \ -no-
```

```
xmlpatterns \ -no-
```

```
multimedia \ -no-
```

```
audio-backend \ -no-
```

```
phonon \ -no-phonon-
```

```
backend \ -svg \
```

```
-no-webkit \ -no-
```

```
javascript-jit \ -
```

```
script \ -no-
```

```
scripttools \ -
```

```
declarative \
```

```
-no-mmx -no-3dnow -no-sse -no-sse2 -no-sse3 -no-ssse3 -no-sse4.1 \
```

```
-no-sse4.2 -no-avx -no-neon \
```

```
-qt-zlib \ -qt-
```

```
gif \ -qt-libtiff
```

```
\ -qt-libpng \ -
```

```
qt-libmng \ -
```



```
qt-libjpeg \ -
no-openssl \ -
nomake docs \

-nomake examples
 \ -nomake demos \ -
optimized-qmake \
-no-nis \
-no-iconv \
-no-cups \
-no-pch \ -qt-
freetype \ -
depths all \ -
no-opengl \
-qt-gfx-linuxfb -qt-gfx-transformed \
-qt-kbd-linuxinput -qt-kbd-tty -no-kbd-qvfb-no-kbd-qnx \
-qt-mouse-pc -qt-mouse-linuxtp -qt-mouse-linuxinput \
-qt-mouse-qvfb -no-mouse-qnx \
-qt-mouse-tslib -I/home/sinc/tslib_arm/include -L/home/sinc/tslib_arm/lib \
-qtlibinfix E \
-little-endian -v "$@"
```

其中 `-prefix` 选项用于指定 Qt 的安装目录，“`-qt-mouse-tslib -I/home/sinc/tslib_arm/include-L/home/sinc/tslib_arm/lib`”选项用于指定 `tslib` 库的头文件和库文件路径，用户可根据需要对其进行修改。

- 配置完成后，在命令行下执行如下命令编译并安装 Qt:

```
$ make -j4 && make install
```

编译完成后，可在 `-prefix` 选项所指定的目录中找到编译生成的 Qt 库文件和其他相关文件，此例为 `/home/sinc/qt-arm-4.7.4/`。

### 3.5.3 安装和配置 Qt Creator

Qt Creator 是一个跨平台的集成开发环境，用户可以在 Qt Creator 提供的开发环境中完成 Qt 应用程序开发，包括 Qt 工程的创建、编辑、编译、调试等。

#### 1. 安装 Qt Creator

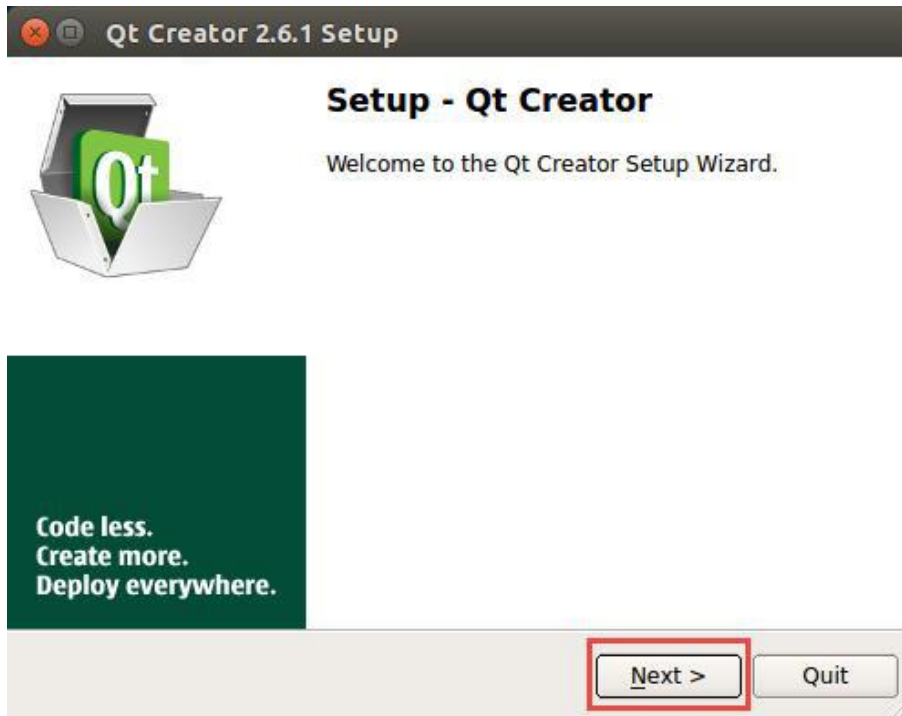
此处以 Qt Creator2.6.1 为例介绍 Qt Creator 的安装过程。Qt Creator2.6.1 的安装程序可以从“产品光盘资料 /3、软件开发参考资料 /2、开发工具软件 /qt-creator-linux-x86-opensource-2.6.1.bin”直接获取。

- 将 Qt Creator2.6.1 安装程序“眺望产品光盘资料/3、软件开发参考资料/2、开发工具软件 /qt-creator-linux-x86-opensource-2.6.1.bin”拷贝到 Ubuntu 虚拟机任意目录下，此处假定拷贝到

/home/sinc/qt-4.7.4 目录下。在命令行下执行如下命令安装 Qt Creator:

```
$ ./qt-creator-linux-x86-opensource-2.6.1.bin
```

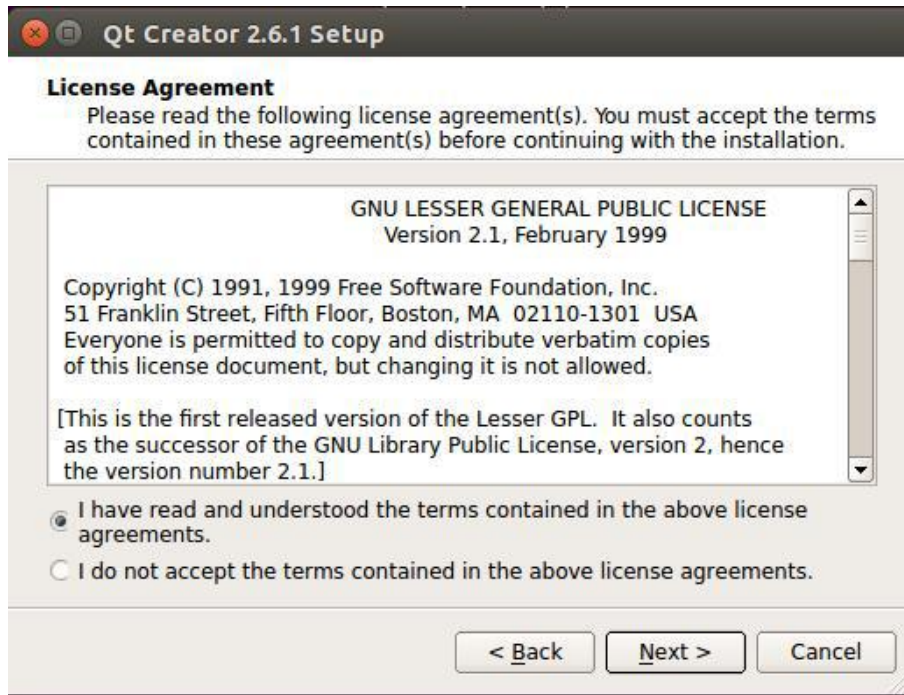
- 在弹出的对话框中，选择“Next”。



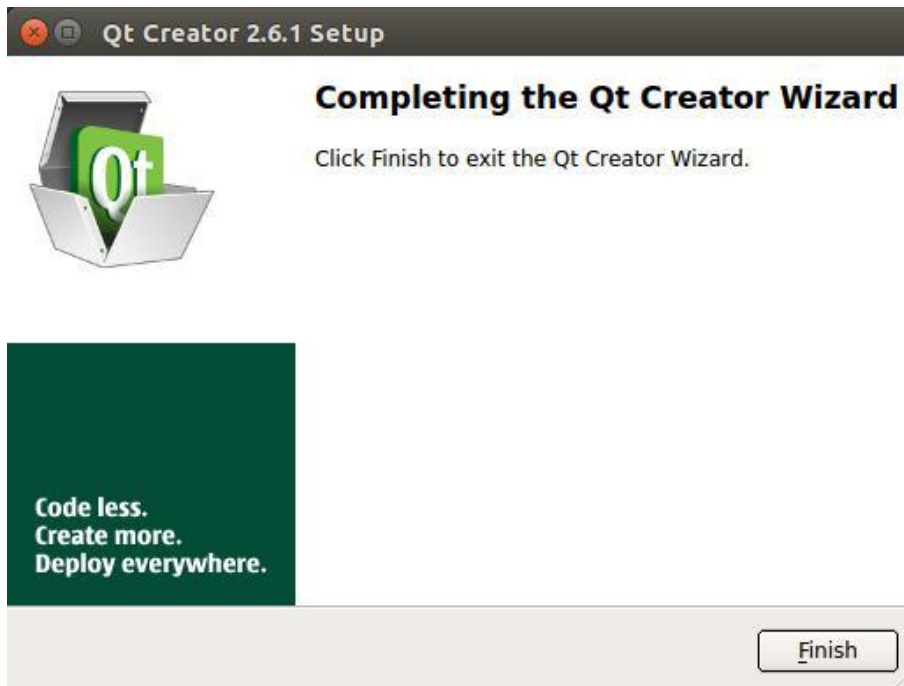
- 在弹出的对话框内，选择 Qt Creator 的安装目录：



- 点击“Next”后，在对话框内，选择同意安装协议：



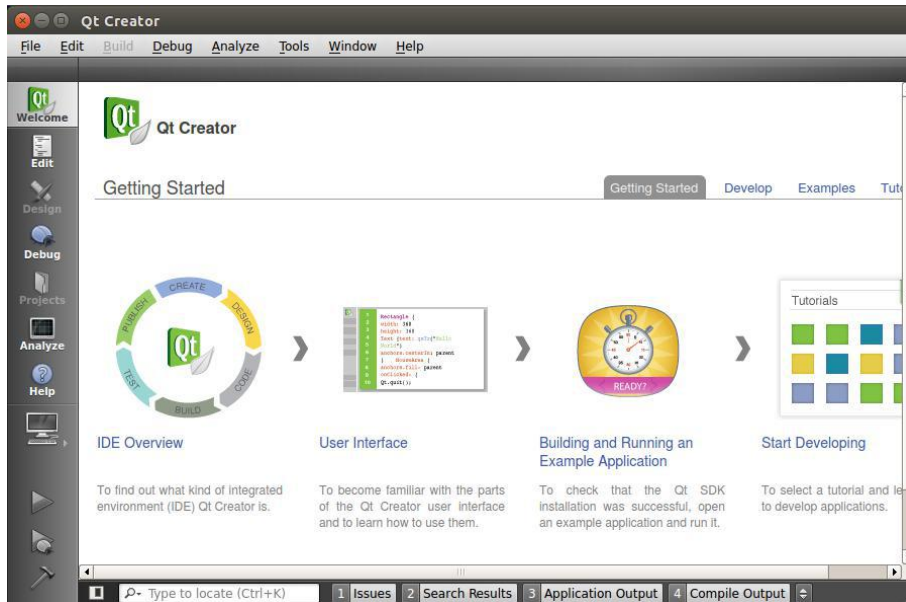
- 点击“Next”后，在对话框里选择“Install”，等待一段时间后，安装完成。



- 运行 Qt Creator。进入 Qt Creator 安装路径的 bin 子目录，并在命令行下执行如下命令：

```
$cd /home/sinc/qtcreator-2.6.1  
$ ./qtcreator
```

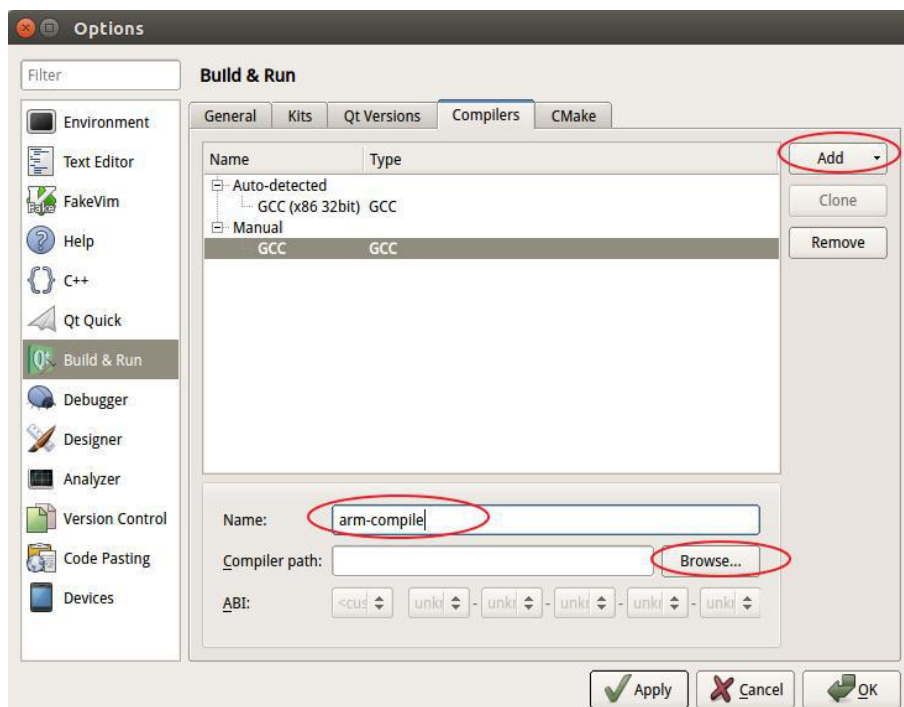
Qt Creator 主界面如下图所示：



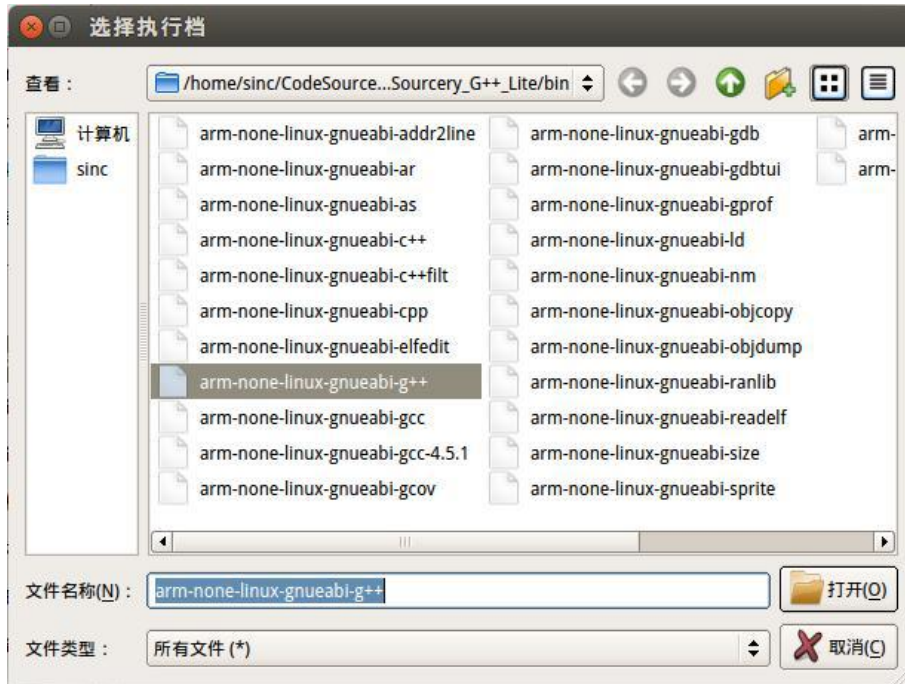
## 配置 Qt Creator

在开始使用 Qt Creator 前，需要配置 Qt Creator 使用的交叉编译工具链以及 Qt 版本信息。

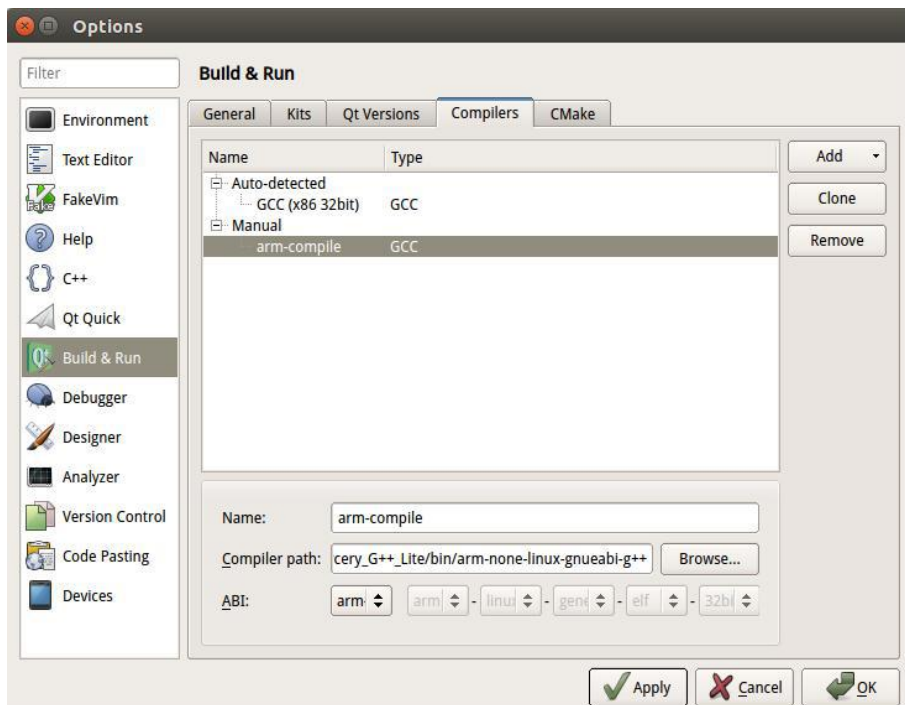
- 配置交叉编译器。点击菜单栏 "Tool->Options->Build & Run->Compilers->Add->GCC"，弹出如下对话框：



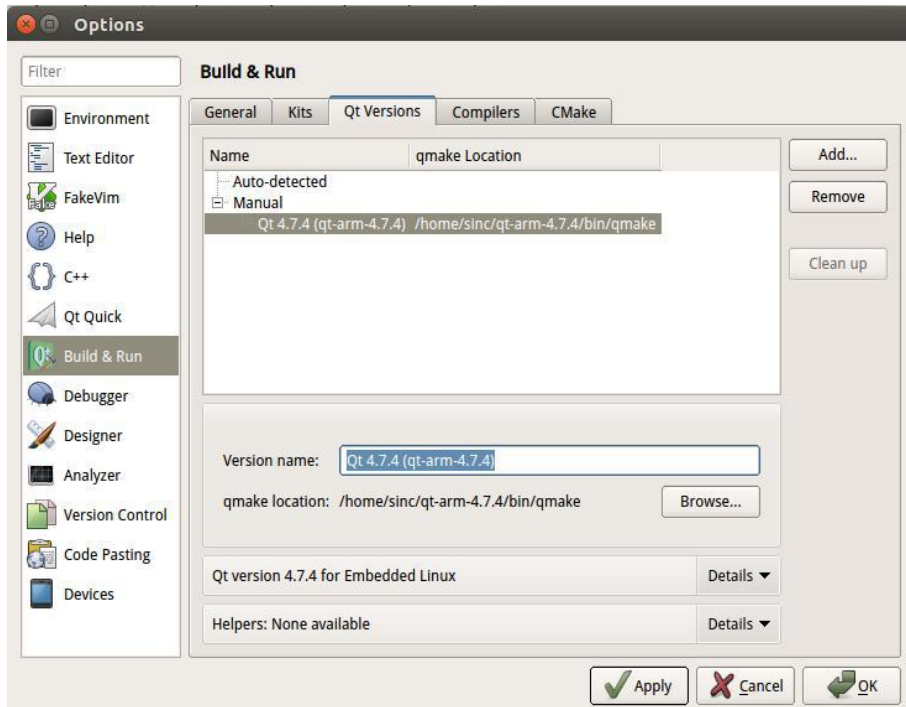
在该对话框中，用户可以修改编译器的名称，此处将其改为“arm-compile”，并点击“Browse”按钮，选择交叉编译工具链安装目录下的" bin/arm-none-linux-gnueabi-g++"，如下图所示：



选择完成后，点击“Apply”即可，配置完成后的界面如下图所示：

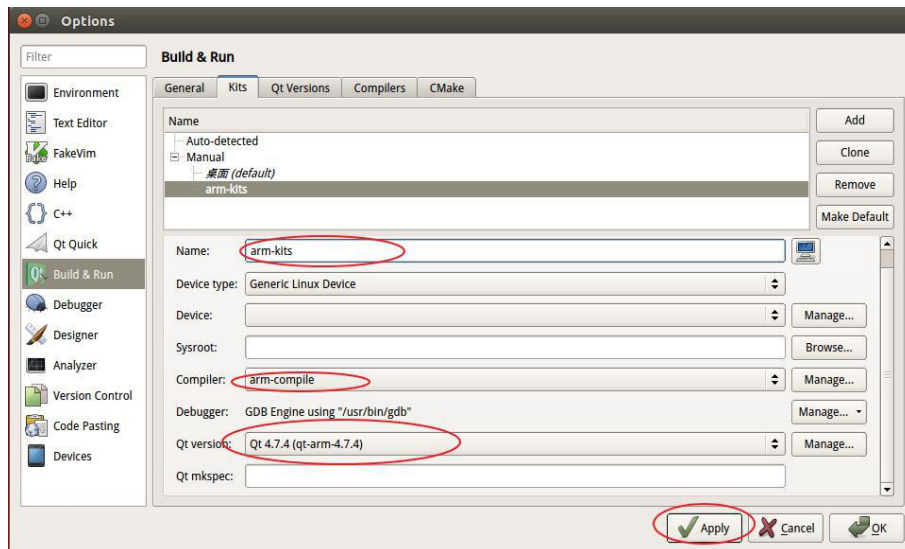


- 配置 Qt 版本。点击" Tool->Options->Build & Run->Qt Versions->Add", 选择编译 Qt 时所设置的 Qt 安装目录下的 "bin/qmake" 文件，此例为 /home/sinc/qt-arm-4.7.4/bin/qmake，如下图所示：



配置完毕后，点击“Apply”按钮。

- 配置开发套件。点击菜单栏"Tool->Options->Build & Run->Kits", 如下图所示：



在该界面中，需进行如下设置：

- 1) 修改开发套件的名称，此处将其改为“arm-kits”；
- 2) 选择开发套件所使用的编译器，该编译器必须首先在“Compilers”标签页下进行配置，此处为“arm-compile”；
- 3) 选择开发套件所使用的 Qt 版本，该版本必须首先在“Qt Versions”标签页下进行配置，此处为“Qt4.7.4(qt-arm-4.7.4)”。

配置完毕后，点击“Apply”按钮，然后点击“OK”按钮完成配置。

### 3.5.4 Tslib—触摸屏校准

Tslib 是一个开源的程序，通常作为触摸屏驱动的适配层，为触摸屏驱动提供获得的采样提供，诸如滤波、去抖、校准等功能。开发板在移植好 tslib 后，需要使用 tslib 上的触摸屏校正程序 ts\_calibrate 对触摸屏进行一次校准，以存储触摸屏的校正信息。使用 tslib 进行触摸屏校正的步骤如下：

1. 配置 tslib 运行环境。在命令行下执行如下命令：

```
~#cd /home/qtapp/tslib
~#source tslib_env.sh
```

2. 运行触摸屏校准程序 ts\_calibrate:

```
~#cd /home/qtapp/tslib/bin
~#./ts_calibrate
```

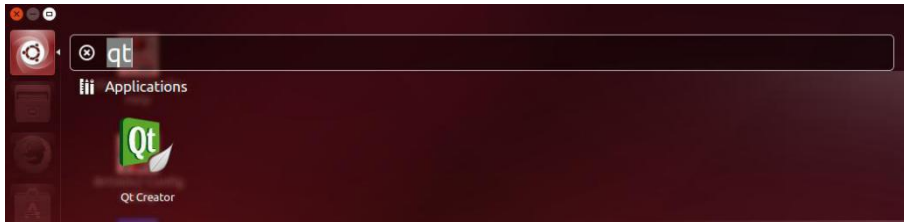
此时会依次在屏幕的左上角、右上角、左下角、右下角和正中间出现“十”字标记，用户用手依次点击“十”字标记的中心点（注意：此时不能使用 USB 鼠标点击，必须用手按压屏幕进行点击），即可完成对触摸屏的校准工作。

### 3.5.5 Hello, World!

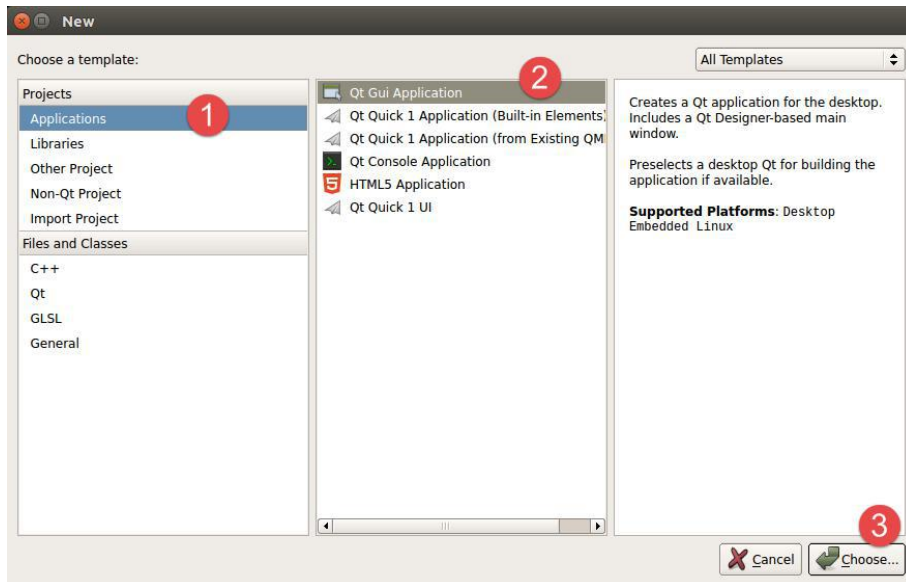
本节以一个“HelloWorld”示例来介绍如何在 Linux 宿主机上进行 QT 程序的编写和编译以及如何将编译出的程序下载到开发板上并在开发板上运行该程序。该例程功能十分简单，就是在屏幕上显示一个“Hello World”窗体。

该示例的源码可以从“眺望产品光盘资料/3、软件开发参考资料/ 3、应用软件示例/qt\_helloworld”中获取。

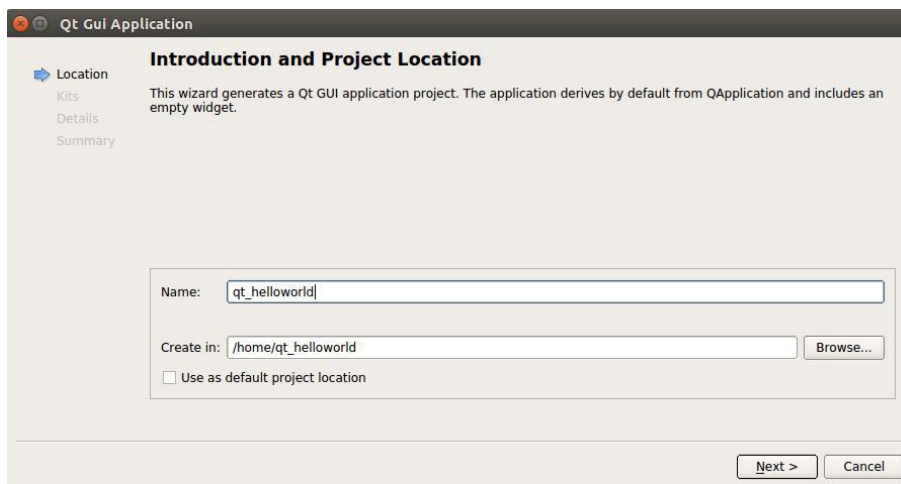
- 1、在 Ubuntu 中打开 QT Creator，单击 Ubuntu 右边 search 图标，找到 QT Creator 并打开。



2、新建工程，在菜单栏点击"File->New File or Project"，在弹出的对话框中选择"Applications>Qt Gui Application"。

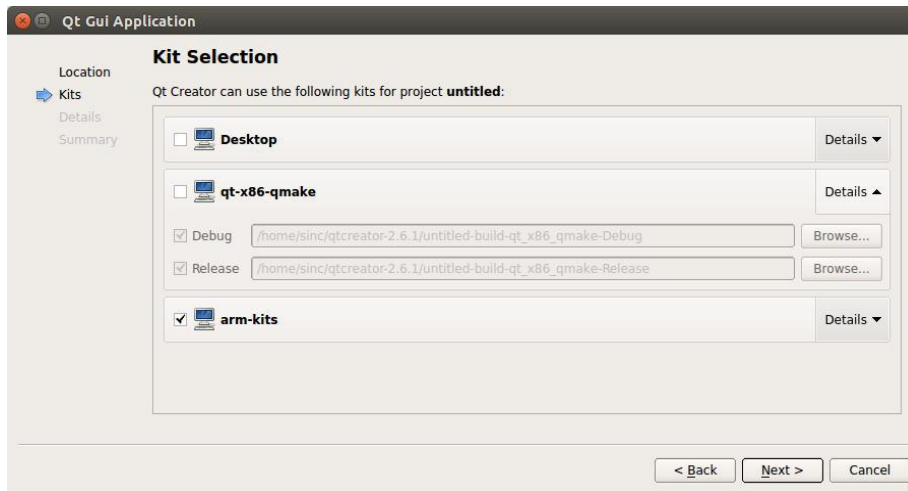


点击 Choose，在弹出的对话框的 Name 栏输入工程名称：helloworld，在"Create in" 栏输入或者通过 Browse 选择工程存放目录，例如："/home/qt\_helloworld"。

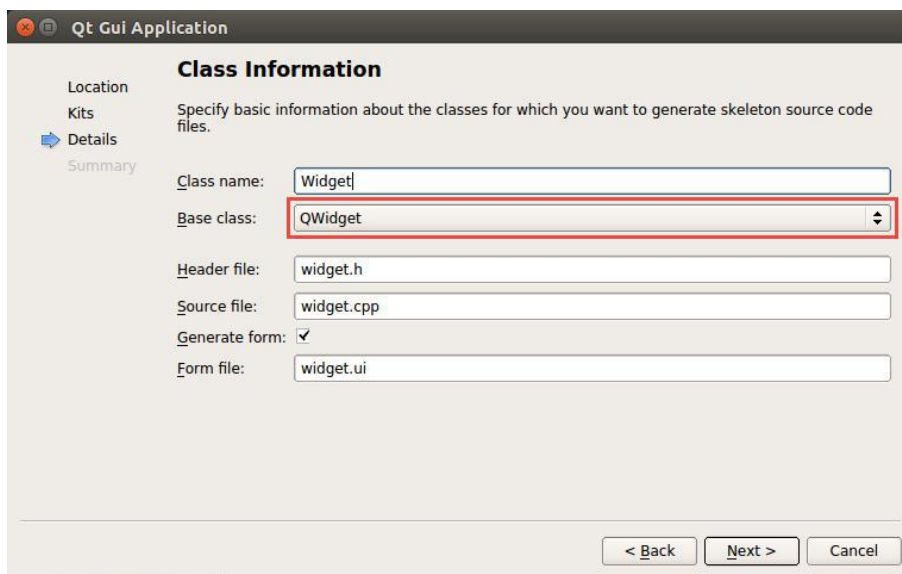


点击—Next|后，选择 arm 端编译链。

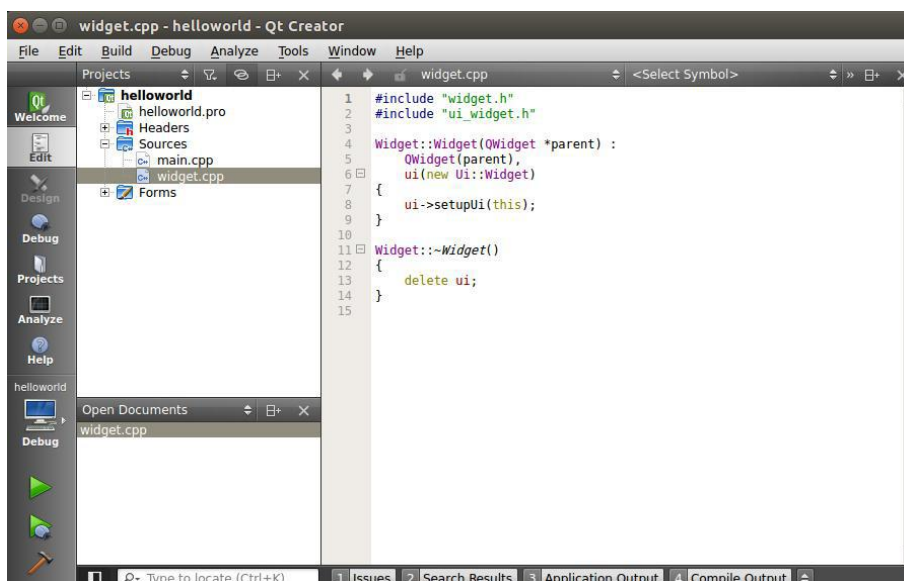




点击“Next”后，弹出的对话框保持默认，继续点击—Next。在新弹出的界面中，点击“Base class”选项下拉选择“QWidget”：

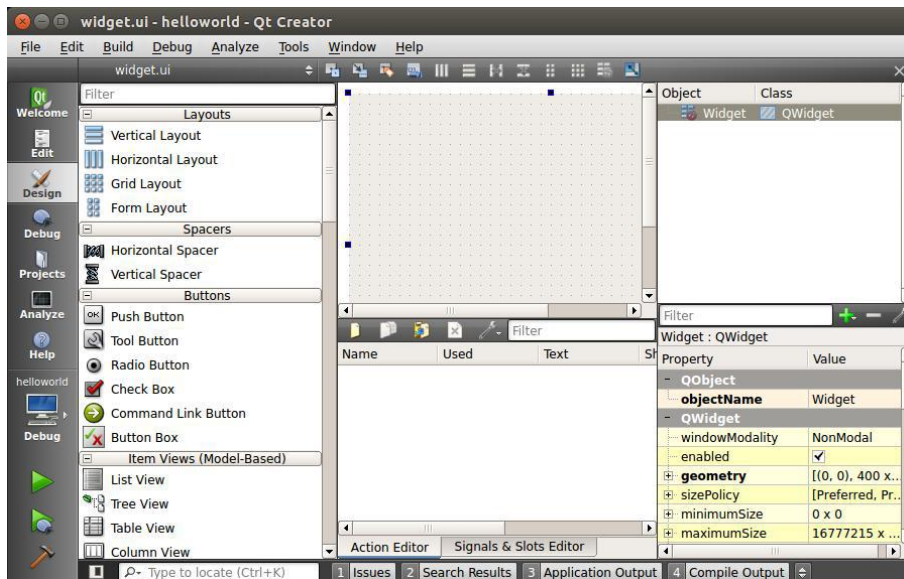


点击“Next”，在弹出的新界面上点击—Finish后，弹出基本的工程界面，如下图：



### 3、设置界面

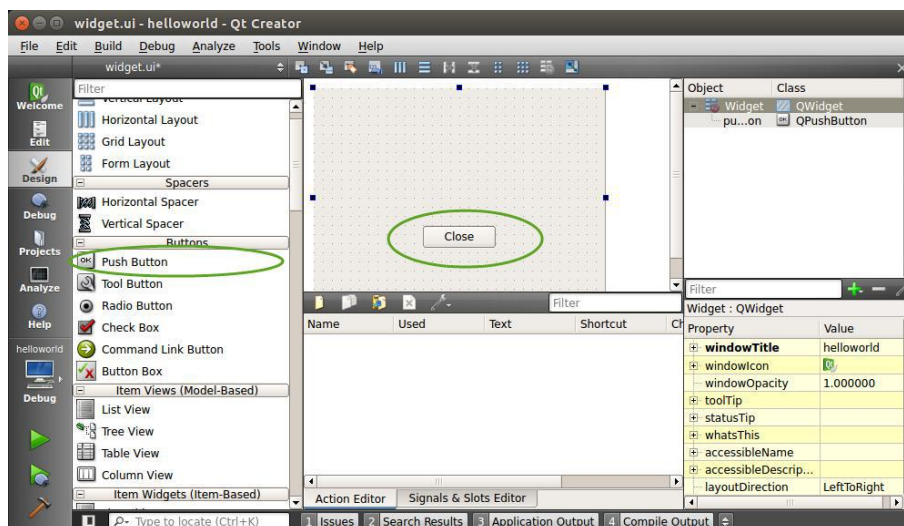
双击“Forms->widget.ui”，弹出如下界面。



在窗口放置以下控件：一个 Push Button（按钮）控件；一个 Label（文本标签）；

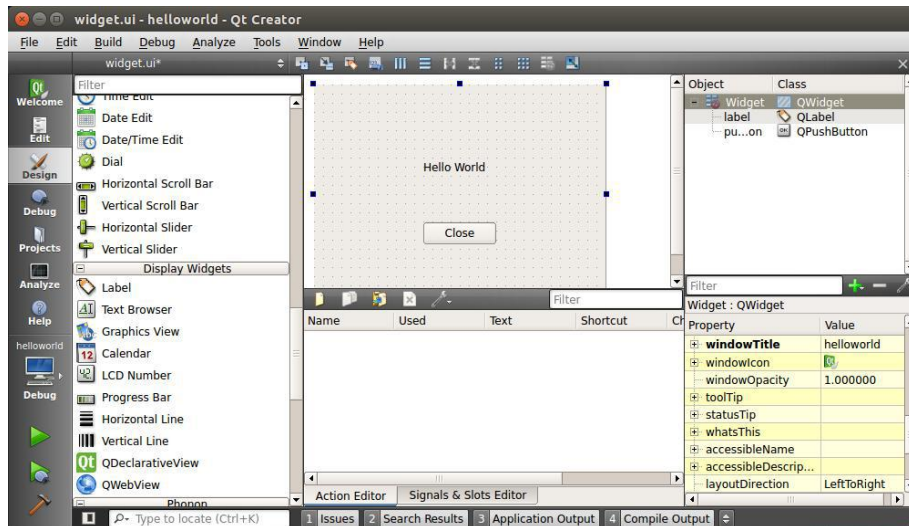
放置 Push Button（按钮）控件：鼠标左击 Buttons 下的 Push Button 控件，鼠标不松开移动到画布中，松开鼠标就放置了一个 Push Button 控件，用于点击退出程序。其他的控件放置方法一样。

放置完成，双击图标，输入字符串“Close”。



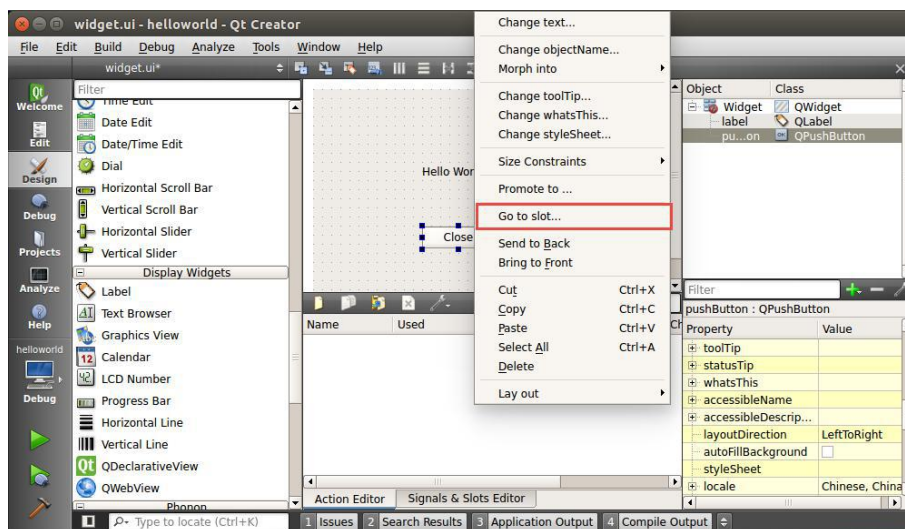
Label（文本标签）：鼠标左击 Display Widgets 下的 Label 控件，鼠标不松开移动到画布中，松开鼠标就放置了一个 Label 控件，用于显示“Hello World”。

放置完成，双击图标，输入字符串“Hello World”。




#### 4、编写代码

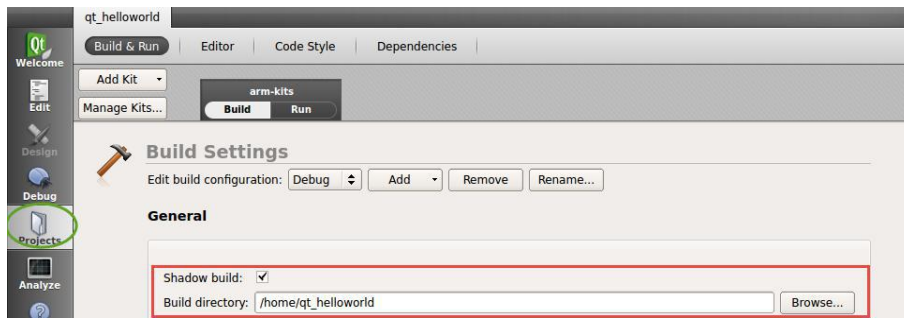
文档中选择了最简单的示例代码—Close 作为演示。右击界面上的 "Close"按钮，点击"Go to slot..."，如下图：




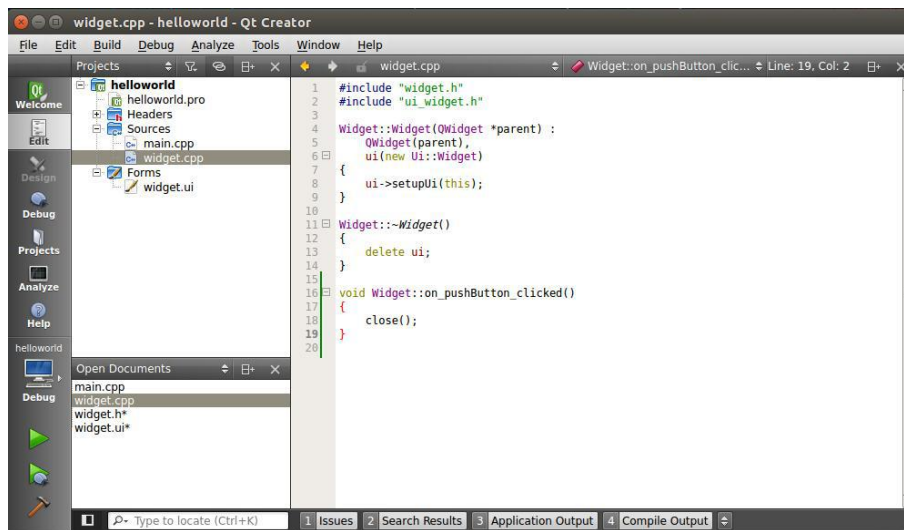
选择"clicked()", 点击 OK, 弹出代码编辑框, 在"void Widget::on\_pushButton\_clicked()"函数内填上"close();"。"ctrl + S"保存代码。

#### 5、编译代码

点击 Qt Creator 界面左侧的工程按钮 ，在弹出界面中的"Build directory"选项处修改或点击 Browse 选择交叉编译镜像的生成路径，例如："/home/qt\_helloworld"，如下图所示：



然后点击左下角的编译按钮，即可在前面设置的目录中产生 Qt 程序镜像 helloworld。



## 6、在开发板上运行 QT 程序

该示例程序源码位于“产品光盘资料/3、软件开发参考资料/ 3、应用软件示例/qt\_helloworld”目录下。将交叉编译产生的 Qt 程序的镜像拷贝到开发板根文件系统/home/qtapp/testappl目录下。

完成所有准备工作后，可以进行以下操作。

- 1、修改—testappl目录下脚本文件 qtapp.sh。

```
EXEC_NAME=—QT 可执行程序镜像名
```

- 2、运行 Qt 程序。

```
~# ./qtapp.sh
```

执行命令后，屏幕将会出现 Qt 界面。

## 3.6 示例程序介绍

在“产品光盘资料/3、软件开发参考资料/ 3、应用软件示例”中，包含了全部的示例程序。下面将对这些示例程序分别加以介绍。

### 3.6.1 Hello World

该示例程序源码位于“产品光盘资料/3、软件开发参考资料/ 3、应用软件示例/HelloWorld”目录下。用户可以直接运行开发板上的/home/demo/helloworld 程序来查看该示例程序的运行结果。

该例程功能十分简单，就是在控制台上打印“Hello World”字符串。关于该示例程序的具体说明可参考“Hello, World!”一节的内容。

### 3.6.2 LED 示例

该示例程序源码位于“产品光盘资料/3、软件开发参考资料/ 3、应用软件示例/ led\_test”目录下。用户可以直接运行开发板上的/home/demo/ledtest 程序来查看该示例程序的运行结果。关于该示例程序的具体说明可参考“LED 测试”一节的内容。

### 3.6.3 蜂鸣器示例

该示例程序源码位于“产品光盘资料/3、软件开发参考资料/ 3、应用软件示例/ bell\_test”目录下。用户可以直接运行开发板上的/home/demo/belltest 程序来查看该示例程序的运行结果。关于该示例程序的具体说明可参考“蜂鸣器测试”一节的内容。

### 3.6.4 串口编程示例

该示例程序源码位于“眺望产品光盘资料/3、软件开发参考资料/ 3、应用软件示例/serial\_test”目录下。用户可以直接运行开发板上的/home/demo/serialtest 程序来查看该示例程序的运行结果。该例程可以通过指定的串口收发数据。关于该示例程序的具体说明可参考“串口测试”一节的内容。

### 3.6.5 CAN 编程示例

该示例程序源码位于“眺望产品光盘资料/3、软件开发参考资料/ 3、应用软件示例/ can\_test”目录下。用户可以直接运行开发板上的/home/demo/cantest 程序来查看该示例程序的运行结果。

TWEvb-IMX6UL 开发板有 2 个 CAN 口。运行 cantest 程序可以通过 CAN 口收发数据。该程序在运行时，需要提供一个命令行参数，即需要打开的 CAN 口名，这个 CAN 名参数可以为“can1”、“can2”。例如需要通过 CAN1 口进行数据收发，在命令行下执行如下命令：

```
~ # cd /home/demo  
/home/demo # ./cantestcan1
```

该程序运行流程如下：

- 打开 CAN1 口，其中 CAN1 口的通讯速率为 125000；
- 通过 CAN1 口发送一个 20 字节的数据；

- 从 CAN1 口接收数据；
- 重复步骤 2~3，实现数据的循环发送和接收。

### 3.6.6 网络编程示例

该示例程序源码位于“眺望产品光盘资料/3、软件开发参考资料/3、应用软件示例/network”目录下。网络编程共包含 4 个例程，分别为 tcp 客户端、tcp 服务器、udp 客户端、udp 服务器，具体说明如下：

表 3.1 网络编程示例

示例程序	源码路径	开发板程序路径
tcp 客户端	tcp_client_test	/home/demo/network/tcp_client_test
tcp 服务端	tcp_server_test	/home/demo/network/tcp_server_test
udp 客户端	udp_client_test	/home/demo/network/udp_client_test
udp 服务端	udp_server_test	/home/demo/network/udp_server_test

其中 tcp\_client\_test 和 tcp\_server\_test 为 tcp 通讯例程，可组合在一起进行测试；udp\_client\_test 和 udp\_server\_test 为 udp 通讯例程，可组合在一起进行测试。

#### 1. tcp 通讯例程

tcp\_client\_test 在运行时，需要提供两个命令行参数，一个为服务器的 IP 地址，一个为服务器的端口号。例如需要和 ip 地址为 127.0.0.1，端口号为 9001 的本机服务器通讯，在命令行下执行如下命令：

```
~ # cd /home/demo/network
/home/demo # ./tcp_client_test 127.0.0.1 9001
```

该程序运行流程如下：

- 与服务器建立连接；
- 向服务器发送字符串数据；
- 从服务器接收字符串数据；
- 重复步骤 2~3，实现数据的循环发送和接收。

tcp\_server\_test 在运行时，需要提供一个命令行参数，即服务器侦听端口号。例如服务器需要在 9001 端口号上进行侦听，在命令行下执行如下命令：

```
~ # cd /home/demo/network
/home/demo # ./tcp_server_test 9001
```

该程序运行流程如下：

- 接受来自客户端的连接请求；
- 从客户端接收字符串数据；

- 向客户端发送字符串数据；
- 重复步骤 2~3，实现数据的循环接收和发送。

在实际测试时，可以在主机上运行两个终端，一个终端运行 `tcp_client_test`，命令如下：

```
# cd /home/demo/network
/home/demo # ./tcp_client_test 127.0.0.1 9001
```

另一个终端运行 `tcp_server_test`，命令如下：

```
/home/demo # ./tcp_server_test 9001
```

当 `tcp_client_test` 和 `tcp_server_test` 都正常运行后，就可以在各自终端上看到报文收发的打印信息。

## 2. udp 通讯例程

`udp_client_test` 在运行时，需要提供两个命令行参数，一个为服务器的 IP 地址，一个为服务器的端口号。例如需要和 ip 地址为 127.0.0.1，端口号为 9001 的本机服务器通讯，在命令行下执行如下命令：

```
~ # cd /home/demo/network
/home/demo # ./udp_client_test 127.0.0.1 9001
```

该程序运行流程如下：

- 与服务器建立连接；
- 向服务器发送字符串数据；
- 从服务器接收字符串数据；
- 重复步骤 2~3，实现数据的循环发送和接收。

`udp_server_test` 在运行时，需要提供一个命令行参数，即服务器侦听端口号。例如服务器需要在 9001 端口号上进行侦听，在命令行下执行如下命令：

```
~ # cd /home/demo/network
/home/demo # ./udp_server_test 9001
```

该程序运行流程如下：

- 从客户端接收字符串数据；
- 向客户端发送字符串数据；
- 重复步骤 1~2，实现数据的循环接收和发送。

在实际测试时，可以在主机上运行两个终端，一个终端运行 `udp_client_test`，命令如下：

```
# cd /home/demo/network
/home/demo # ./udp_client_test 127.0.0.1 9001
```

另一个终端运行 `udp_server_test`，命令如下：

```
/home/demo # ./udp_server_test 9001
```

当 `udp_client_test` 和 `udp_server_test` 都正常运行后，就可以在各自终端上看到报文收发的打印信息。

### 3.6.7 数据库编程示例

该例程使用 `sqlite` 数据库。`sqlite` 数据库的源码可以从“[www.sqlite.org/download.html](http://www.sqlite.org/download.html)”下载或者从“产品光盘资料 /3、软件开发参考资料 /5、源代码/sqlite-autoconf-3140100.tar.gz”中直接获取。

该示例程序源码位于“产品光盘资料/3、软件开发参考资料/ 3、应用软件示例/sqlite\_test”目录下。用户可以直接运行开发板上的 `/home/demo/ sqlite_test` 程序来查看该示例程序的运行结果。

`sqlite_test` 程序运行流程如下：

- 创建 `test.db` 数据库；
- 在数据库中创建名为 `film` 的数据库表；
- 向 `film` 数据表中插入 4 条记录；
- 查询 `film` 数据表中的全部记录；
- 关闭数据库。

## 3.7 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。系统时钟是由 Linux 内核所维护的时钟，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主板硬件时钟。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. 查看系统时钟：

使用 `date` 命令可以查看系统时钟：

```
~ # date
Sat Jan  1 02:03:07 UTC 2005
```

2. 查看 RTC 时钟：

使用 `hwclock` 命令可以查看 RTC 时钟：

```
~ # hwclock
Sat Jan  1 02:04:22 2005    0.000000 seconds
```

3. 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 2016-08-03 22:30:10，可以使用如下命令：

```
~ # date -s "2016-08-03 22:30:10"  
Wed Aug 3 22:30:10 UTC 2016
```

#### 4. 设置 RTC 时钟：

使用 `hwclock -w`，可以将系统时钟写入 RTC 时钟：

```
~ # hwclock -w
```

#### 5. 同步系统时钟：

使用 `hwclock -s`，可以将 RTC 时钟写入系统时钟：

```
~ # hwclock -s
```

通过上面的叙述可以看出，如果想要改变当前的系统时间，且希望系统重启后改变依然生效，需要执行如下两步操作：

- 使用 `date -s` 命令修改当前的系统时钟；
- 使用 `hwclock -w` 命令将修改后的系统时钟写入 RTC 时钟。

例如需要将当前时钟设置为 2016-08-03 22:30:10，并希望该改变在系统重启后依然有效，应执行如下命令：

```
~ # date -s "2016-08-03 22:30:10"  
Wed Aug 3 22:30:10 UTC 2016  
~ # hwclock -w
```

## 3.8 停止示例工程运行

开发板上电后会自动运行一个示例工程。为停止该示例工程的运行，可在命令下执行如下命令：

```
~ # killall lcwatchdog  
~ # killall lcrun
```

为重新运行示例工程，可在命令行下执行如下命令：

```
~ # /home/etc/lcwatchdog &
```

如果用户不希望每次开机自动运行该示例工程，可以修改 `/home/etc/rc.ini` 文件，将其中的

```
/home/app/lcrun.sh &  
/home/etc/lcwatchdog &
```

两行注释掉（在行的最前面加#），即修改为：

```
~/home/app/lcrun.sh &  
~/home/etc/lcwatchdog &
```

## 第 4 章 Web 控制系统

### 4.1 嵌入式 Web 开发简介

随着 Internet 技术的兴起，越来越多的项目使用 Web 浏览器对嵌入式设备进行管理和监控。使用 Web 的好处在于，用户无需安装任何软件，只需要运行 Web 浏览器即可完成对设备的操作，因此使用起来非常方便。

Web 系统采用客户端/服务器架构，Web 客户端通常是 PC 机上的 Web 浏览器，如 IE 浏览器、Chrome 浏览器等。而 Web 服务器则运行在嵌入式设备上，因此在进行嵌入式 Web 开发前首先需要移植一个适合于嵌入式系统运行的 Web 服务器。下面将分别对嵌入式 Web 服务器和在嵌入式 Web 服务器中最常用到的 CGI 标准进行简要介绍。

#### 4.1.1 嵌入式 Web 服务器

PC 机上最常用的 Web 服务器有开源软件 Apache 和 Microsoft 的 IIS 等，这些 Web 服务器功能非常强大，但通常布署比较复杂，对资源的占用也比较大。而在嵌入式设备中资源一般都比较有限，并且也不需要同时处理很多用户的连接请求，因此在嵌入式设备中需要使用一些专门为嵌入式系统设计的 Web 服务器，这些 Web 服务器在存贮空间和运行时所占有的内存空间上都会非常适合于嵌入式应用场合。典型的嵌入式 Web 服务器有 Boa 和 thttpd 等，它们和 Apache、IIS 等高性能的 Web 服务器主要的区别在于它们一般是单进程服务器，只有在完成一个用户请求后才能响应另一个用户的请求，而无法并发响应。

#### 4.1.2 CGI

CGI (Common Gateway Interface) 是外部应用程序 (CGI 程序) 与 Web 服务器之间的接口标准，是在 CGI 程序和 Web 服务器之间传递信息的过程。CGI 规范允许 Web 服务器执行外部程序，并将它们的输出发送给 Web 浏览器。通过 CGI 可以提供许多静态的 HTML 网页无法实现的功能，比如搜索引擎、基于 Web 的数据库访问等。

CGI 工作流程如下：

- 浏览器通过 HTML 表单或超链接请求指向一个 CGI 应用程序的 URL；
- 服务器收到浏览器提交的请求；
- 服务器执行指定的 CGI 应用程序；
- CGI 应用程序根据用户在浏览器上的输入执行所需要的操作；
- CGI 应用程序把结果格式化为 Web 服务器和浏览器能够理解的文档（通常是 HTML 网页）；

- Web 服务器把结果返回到浏览器中。

## 4.2 Boa 服务器

### 4.2.1 Boa 服务器简介

Boa 是嵌入式系统中最常使用的一种支持 CGI 的 Web 服务器，它是一款开源软件，其最新的代码可以从 <http://www.boa.org/> 中下载。Boa 作为一款单任务的 Web 服务器，与其它传统的 Web 服务器不同的是：当有连接请求到来时，它并不为每个连接请求单独创建进程，而是通过建立 HTTP 请求列表来处理多路 HTTP 连接请求，同时它只为 CGI 程序创建新的进程，这样就在最大程度上节省了系统资源，因此 Boa 具有很高的 HTTP 请求处理速度，在嵌入式系统中获得了广泛的应用。

嵌入式 Web 服务器 Boa 和普通 Web 服务器一样，能够完成接收客户端请求、分析请求、向客户端返回请求结果等任务，它的工作流程主要包括：

- 完成 Web 服务器的初始化工作，如创建环境变量、创建 TCP 套接字、绑定端口、开始侦听以及等待接收浏览器的连接请求等；
- 当有客户端连接请求时，Boa 负责接收客户端请求，并对请求进行分析处理；
- Boa 完成相应的处理后，向浏览器发送响应信息，并关闭和浏览器的 TCP 连接。

### 4.2.2 编译 Boa 服务器

此处以 Boa 0.94.13 版本为例，介绍 Boa 服务器的移植步骤。

1. boa-0.94.13.tar.gz 源码可以从“产品光盘资料/3、软件开发参考资料/5、源代码”中直接获取。将 boa-0.94.13.tar.gz 拷贝到 Ubuntu 虚拟机中，此处假定将该文件拷贝到/home/sinc/目录中，并对其进行解压：

```
$cd /home/sinc
$tar zxvf boa-0.94.13.tar.gz
```

进入 src 子目录对源码进行编译。从眺望产品光盘资料中获取的源码压缩包已经设置好了交叉编译器，因此可以直接在命令行下执行 make 命令进行编译：

```
$cd ./src $make
```

编译完成后，在 src 子目录下将生成 boa 可执行程序。

### 4.2.3 Boa 服务器配置

Boa 运行时，需要读取/etc/boa 目录(如果没有该目录则需要自己创建)下的 boa.conf 配置文件。在 Boa 源码目录下有一个 boa.conf 示例文件，用户可以在该文件的基础上对其进行修改，生成自己所需要的配置文件。

boa.conf 一个配置项对应文件的一行，其中以#号开头的行表示注释，因此如果要关闭某个配置项可以在该配置项对应的行前加上#。下面对 boa.conf 文件的主要配置项进行介绍：

表 4.1 boa.conf 配置文件

配置项	说明
Port	监听的端作为哪个用户运行，即它拥有该用户的权限，一般都是 nobody，需要/etc/passwd 中有口号，缺省 80。例如：Port 80
User	boa 以哪个用户身份运行，即它拥有该用户的权限，默认值为 nobody。需要在/etc/passwd 中需要配置该用户。为简单起见，也可以用 root。例如：User nobody
Group	boa 以哪个用户组运行，即它拥有该用户组的权限，默认值为 nogroup。需要在/etc/passwd 中需要配置该用户。为简单起见，也可以用 root。例如：Group root
DocumentRoot	HTML 文档的主目录。如果没有以/开始，则表示从服务器的根路径开始。例如：DocumentRoot /var/www
DirectoryIndex	默认 HTML 文档名。例如：DirectoryIndex index.html
MimeTypes	指明 mime.types 文件位置。如果没有以/开始，则表示从服务器的根路径开始。例如：MimeTypes /etc/mime.types
DefaultType	文件扩展名没有或未知的的话，使用的缺省 MIME 类型。例如：DefaultType text/plain
CGIPath	指定 CGI 程序执行时所使用的 PATH 环境变量。例如：CGIPath /bin:/usr/bin:/usr/local/bin
ScriptAlias	该配置有两个参数，第一个参数指明 cgi 脚本的虚拟路径，第二个参数指明 cgi 脚本，在开发板上的实际存储路径。当用户在浏览器上需要访问 cgi 脚本时，输入的 URL 为：主机 IP/虚拟路径/cgi 脚本文件名。例如：ScriptAlias /cgi-bin/ /var/www/cgi-bin/

## 4.3 Web 开发环境搭建

在开发板上搭建一个基于 Boa 的嵌入式 web 服务器，需要准备如下文件和目录：

文件或目录	说明
boa	Web 服务器可执行程序，可放在开发板任意目录下。在 TWEvb-IMX6UL 开发板，将其放在 /home/webserver 目录下。
boa.conf	boa 配置文件，必须放在/etc/boa 目录下。
mime.types	MIME 类型文件。可放在开发板任意目录下，但必须与 boa.conf 配置文件的“MimeTypes”配置项保持一致。在 TWEvb-IMX6UL 开发板，将其放在/home/webserver 目录下，并在 boa.conf 中配置如下：MimeTypes /home/webserver/mime.types
用户网站目录	需要在开发板上创建一个目录，用于存放用户自己开发的全部网页文件（如：html 文件、css 文件、js 文件等）。该目录的位置可以任意指定，但必须与 boa.conf 配置文件的“DocumentRoot”配置项保持一致。在 TWEvb-IMX6UL 开发板，将其放在/home/www 目录下，并在 boa.conf 中配置如下：DocumentRoot /home/www
CGI 文件目录	需要在开发板上创建一个目录，用于存放用户自己开发的全部 CGI 文件。该目录的位置可以任意指定，但必须与 boa.conf 配置文件的“ScriptAlias”配置项保持一致。在 TWEvb-IMX6UL 开发板，将其放在/home/www/cgi-bin 目录下，并在 boa.conf 中配置如下：ScriptAlias /cgi-bin/ /home/www/cgi-bin/

搭建环境所需要的文件可以从“产品光盘资料/3、软件开发参考资料/3、应用软件示例/web/env”中直接获取。

## 4.4 Web 开发实例

在运行本章示例前，需要先按照“Web 开发环境搭建”一节在开发板上准备好相关文件和目录。并在开发板上运行 boa 服务程序：

```
~ # cd /home/webserver/  
/home/webserver # ./boa
```

### 4.4.1 静态网页

本节以在浏览器上显示一个欢迎界面来演示静态网页的创建过程。该示例的源码可以从“眺望产品光盘资料/3、软件开发参考资料/3、应用软件示例/web/html\_test”中获取。

#### 1、编辑 html 文件：

用户可以使用任意文本编辑器编写一个 html 文件，此处假定编写的 html 文件名为 index.html，文件内容如下：

```
<html>  
<meta http-equiv="Content-Type" content="text/html;charset=UTF-8">  
<h1>  
    欢迎!  
</h1>  
  
<body>  
    欢迎使用 TWEvb-IMX6UL 开发板!<br/>  
</body>  
</html>
```

如果文件中包含中文，应保证中文的编码方式与 html 文件中声明的编码方式一致。此例中 html 声明的字符集为 UTF-8（注：由 meta 标签的 charset 属性指定），因此文件的编码也必须是 UTF-8。

#### 2、将文件下载到开发板中，此例将其下载到/home/www 目录下。执行如下命令：

```
# cd /home/www  
/home/www # tftp -r index.html -g 192.168.1.111
```

此处假定主机的 ip 地址为 192.168.1.111。关于如何使用 TFTP 协议传输文件可参见“与 PC 互传文件”一节。

#### 3、在浏览器的地址栏中输入（此处假定开发板的 ip 地址为 192.168.1.10）：

```
http://192.168.1.10/index.html
```

即可在浏览器上看到 html 文件的内容，如下图所示：



## 4.4.2 使用 CGI 程序的动态网页

本节将介绍如何在浏览器中调用用户编写的 CGI 程序。该例程功能十分简单，就是在浏览器上打印“Hello World”字符串。

该示例的源码可以从“眺望产品光盘资料/3、软件开发参考资料/3、应用软件示例/web/cgi\_test”中获取。

### 1. 编写 CGI 源程序：

在 Linux 主机上编写 CGI 源文件，此处假定文件名为 `cgi_test.c`，文件内容如下：

```
#include <stdio.h>
int main(int argc, char *argv[])
{
    printf("Content-type: text/html\n\n");
    printf("<html>\n");
    printf("<h1><title>CGI Output</title></h1>\n");
    printf("<body>\n");
    printf("Hello World\n"); printf("</body>\n");
    printf("</html>\n"); return 0;
}
```

### 2. 编译 CGI 程序：

进入源码所在目录，在命令行下执行如下命令编译程序：

```
$arm-none-linux-gnueabi-gcc-o cgi_test cgi_test.c
```

编译成功后，可以看到在源码目录下会产生一个 `cgi_test` 的可执行文件。

### 3. 下载 CGI 程序：

将 `cgi_test` 程序下载到开发板 `/home/www/cgi-bin` 目录下，并增加该文件的可执行权限。执行如下命令：

```
/home/www # cd /home/www/cgi-bin/ /home/www/cgi-  
bin # tftp -r cgi_test -g 192.168.1.111 /home/www/cgi-  
bin # chmod a+x cgi_test
```

4. 在浏览器的地址栏中输入（此处假定开发板的 ip 地址为 192.168.1.10）：

```
http://192.168.1.10/cgi-bin/cgi_test
```

即可在浏览器上看到 html 文件的内容，如下图所示：



# 第 5 章 U-Boot

## 5.1 U-Boot 简介

### 5.1.1 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。通用性强且使用方便的莫过于 U-Boot 了。

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。

### 5.1.2 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，从 FADSR0M、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，还能引导 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 的源码很多都是 Linux 内核源码的简化，特别是驱动部分的源码。

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、XScale 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。

### 5.1.3 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（即初始化 SDRAM）；
- 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。

## 5.2 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 2013.04-rc2 (Mar 14 2016 - 15:09:29)
```

```
CPU: NUC972
```

```
DRAM: 64 MiB
```

```
NAND: 128 MiB
```

```
MMC: mmc: 0, mmc: 1
```

```
In: serial
```

```
Out: serial
```

```
Err: serial
```

```
Net: emac
```

```
Hit any key to stop autoboot: 0 U-Boot>
```

下面将针对 U-Boot 中的一些常用命令进行介绍。

### 5.2.1 查看命令列表

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：

U-Boot 命令中，有一些命令归类在某一个子类中，如 I2C、NAND FLASH、UBI 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 nand 相关的子命令，可在命令行下执行：

```
U-Boot> nand
```

```
nand - NAND sub-system
```

```
Usage:
```

```
nand info - show available NAND devices
```

```
nand device [dev] - show or set current
```

```
device nand read - addr off|partition size
```

```
nand write - addr off|partition size
```

```
    read/write 'size' bytes starting at offset 'off'  
    to/from memory address 'addr', skipping bad blocks.
```

```
nand read.raw - addr off|partition [count]
```

```
nand write.raw - addr off|partition [count]
```

```
    Use read.raw/write.raw to avoid ECC and access the flash as-is.
```

```
nand erase[.spread] [clean] off size - erase 'size' bytes from offset 'off'
```

```
    With '.spread', erase enough for given file size, otherwise,  
    'size' includes skipped bad blocks.
```

```
nand erase.part [clean] partition - erase entire mtd partition'  
nand erase.chip [clean] - erase entire chip'  
nand bad - show bad blocks  
nand dump[.oob] off - dump page  
nand scrub [-y] off size | scrub.part partition | scrub.chip  
    really clean NAND erasing bad blocks (UNSAFE)  
nand markbad off [...] - mark bad block(s) at offset (UNSAFE)  
nand biterr off - make a bit error at offset (UNSAFE)
```

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 `nand` 设备，可在命令行下执行：

```
U-Boot> nand info  
Device 0: nand0, sector size 128 KiB  
Page size      2048 b  
OOB size      64 b  
Erase size    131072 b
```

## 5.2.2 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：

```
U-Boot> printenv  
baudrate=115200  
bootcmd=nand read 0x7fc0 0x280000 0x1200000;bootm  
0x7fc0 bootdelay=1  
ethact=emac  
ethaddr=00:00:00:11:66:88  
ipaddr=192.168.1.10  
nand_erasize=20000  
nand_oobsize=40  
nand_writesize=800  
serverip=192.168.1.111  
stderr=serial stdin=serial  
stdout=serial
```

```
Environment size: 290/65532 bytes
```

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：

```
U-Boot> printenv serverip  
serverip=192.168.1.111
```

## 2. 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 192.168.1.111 的 IP 时，可以使用如下命令：

```
U-Boot> ping 192.168.1.111
Using emac device
host 192.168.1.111 is alive
```

但如果在 U-Boot 中已经将 `serverip` 环境变量的值设置为 192.168.1.111，那么上述 `ping` 命令可改成：

```
U-Boot> ping #{serverip}
Using emac device
host 192.168.1.111 is alive
```

## 3. 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，但用户没有指定环境变量的值，则该命令删除环境变量。

例如要设置 `serverip` 环境变量的值为 192.168.1.111，可执行如下命令：

```
U-Boot> setenv serverip 192.168.1.111
```

要删除 `serverip` 环境变量，可执行如下命令：

```
U-Boot> 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：

```
U-Boot> saveenv
Saving Environment to NAND...
Erasing Nand...
Erasing at 0xe0000 -- 100% complete.
Writing to Nand... 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存储器中。

## 4. 特殊环境变量

V-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。

## 5.2.3 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 192.168.1.10 和 192.168.1.111，可使用如下命令：

```
U-Boot> setenv ipaddr 192.168.1.10
U-Boot> setenv serverip 192.168.1.111
```

### 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：

```
U-Boot> ping 192.168.1.111
Using emac device
host 192.168.1.111 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 192.168.1.111 的 tftp 服务器下载 vmlinux.ub 文件到本地内存的 0x7fc0 地址时，应执行如下命令：

```
U-Boot> tftp 0x7fc0 vmlinux.ub Using emac device
TFTP from server 192.168.1.111; our IP address is
192.168.1.10 Filename 'vmlinux.ub'.
Load address: 0x7fc0
Loading: T T T #####
#####
#####
#####
```

```
#####  
#####  
283.2 KiB/s  
done  
Bytes transferred = 5328032 (514ca0 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。

## 5.2.4 Nand Flash 命令

### 1. 读命令

nand read 命令可以把 Nand Flash 指定位置的数据读取到内存的指定位置中。该命令格式为：

```
nand read loadaddr offset_addr size
```

在该命令中，第一个参数 loadaddr 为内存的起始地址；第二个参数 offset\_addr 为要读取 Nand Flash 的起始地址；第三个参数 size 为读取数据的长度。

例如某固件文件的长度为 18M (0x1200000)，保存在 Nand Flash 起始地址为 0x280000 的位置，那么如果要将该固件复制到 0x7fc0 的内存地址上，可执行如下命令：

```
U-Boot> nand read 0x7fc0 0x280000 0x1200000
```

```
NAND read: device 0 offset 0x280000, size 0x1200000  
18874368 bytes read: OK
```

### 2. 擦除命令

根据 Nand Flash 的特点，在对 Nand Flash 执行写入操作之前，必须先对其执行擦除操作。nand erase 命令可以擦除 Nand Flash 指定位置的数据。该命令格式为：

```
nand erase offset size
```

其中“offset”参数为 Nand Flash 要擦除的起始地址；“size”为要擦除的长度。

例如要擦除 Nand Flash 上起始地址为 0x280000，大小为 0x1200000 一段地址区域，可执行如下命令：

```
U-Boot> nand erase 0x280000 0x1200000
```

```
NAND erase: device 0 offset 0x280000, size 0x1200000  
Erasing at 0x1460000 -- 100% complete.
```

### 3. 写命令

nand write 命令可以把内存中指定地址的数据写入 NAND Flash 的指定地址中。该命令格式为：

```
nand write loadaddr offset_addr size
```

在该命令中，第一个参数“loadaddr”为内存的起始地址；第二个参数“offset\_addr”为要写入 NAND Flash

的起始地址；第三个参数“size”为要写入数据的长度。

假如某固件文件的长度为 18M(0x1200000)，保存在 0x7fc0 的内存地址上，要把该固件写入到 Nand Flash 偏移为 0x280000 地址上，可以使用如下命令：

```
U-Boot> nand write 0x7fc0 0x280000 0x1200000
```

```
NAND write: device 0 offset 0x280000, size 0x1200000
18874368 bytes written: OK
```

## 5.2.5 启动命令

bootm 命令可以执行内存中的执行的二进制代码。而这些二进制代码是有特定的格式，通常为 mkimage 命令处理过的文件。bootm 命令的格式为：

```
bootm [loadaddr]
```

其中 loadaddr 参数为二进制代码在内存的起始地址。例如要让 U-Boot 运行内存地址为 0x7fc0 的代码，可以使用如下命令：

```
bootm 0x7fc0
```

## 5.2.6 组合命令

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 bootcmd，假定其被赋值为两个命令的组合：

```
bootcmd=nand read 0x7fc0 0x280000 0x1200000;bootm 0x7fc0
```

其中第一个命令是 nand read 命令，该命令从 nand flash 地址为 0x280000 的地方读取 0x1200000 长度的数据，并将这些数据写入到内存 0x7fc0 中；第二条命令则是让 U-Boot 执行 0x7fc0 处的代码。假如 nand flash 地址为 0x280000 的地方存放的是内核镜像，该组合命令其实就是加载内核到内存并运行内核。当 U-Boot 以启动模式运行时，其会自动执行 bootcmd 中的命令加载内核；但当 U-Boot 运行在程序下载模式时，用户也可以在命令行下输入如下命令来加载并运行内核：

```
U-Boot> run bootcmd
```

执行完该命令，会发现系统已经自动进入了 Linux 内核。

## 5.2.7 重启命令

在 U-Boot 命令行下，执行 reset 命令可以重启系统：

```
U-Boot> reset
```

## 5.3 U-Boot 工具

U-Boot 提供了一些有用的小工具，这些工具的源码位于 U-Boot 源码顶层目录的 `tools` 子目录下。在编译 U-Boot 时，会自动编译这些工具，假设编译 U-Boot 时，设置编译结果的存储路径为 U-Boot 源码顶层目录的 `build` 子目录，那么可以在 U-Boot 源代码顶层目录的 `build/tools` 子目录找到这些工具。这些工具都是在主机上使用的，其中 `mkimage` 工具最为常用。

`mkimage` 是在编译内核的时候需要用到的工具，用于将内核编译时生成的内核映像文件转换成 U-Boot 所需要的映像文件。



# 第 6 章 Linux 内核

## 6.1 内核简介

### 6.1.1 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 系统。Linux 是一个一体化内核（monolithic kernel）系统。—内核指的是一个提供硬件抽象层、磁盘及文件系统控制、多任务等功能的系统软件。一个内核不是一套完整的操作系统，一套基于 Linux 内核的完整操作系统叫作 Linux 操作系统，或是 GNU/Linux。

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”，是相对于“微内核”而言的。几乎所有的嵌入式和实时系统都采用微内核，如 VxWorks、uC/OS-II、PSOS 等；
- 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。

### 6.1.2 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。TWEvb-IMX6UL 开发板使用 Linux 4.1.5 内核版本，本节以该版本为例介绍 Linux 源码的目录结构：

表 6.1 Linux 内核源码目录

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 内核源码中占了很大比例，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

### 6.1.3 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（对配置脚本中使用的配置命令进行解释）和配置用户界面（提供

基于字符界面、基于 Ncurses 图形界面以及基于 Xwindows 图形界面的用户配置界面，各自对应于 Make config、Make menuconfig 和 make xconfig）。

下面将对这三部分分别加以介绍：

## 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。

## 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。

## 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；
- 3、make oldconfig：使用已有的配置文件（.config），但是会询问新增的配置选项；
- 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-get 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：

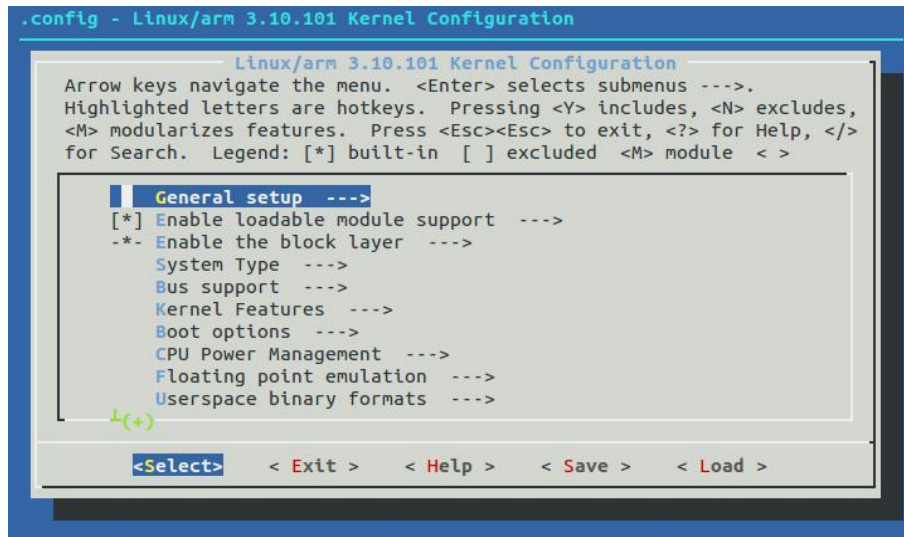


图 6.1 Linux 内核配置主界面

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项菜单项；
- 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，并按回车键，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），此时如果用户对内核配置进行了修改，则将弹出确认修改的提示画面，如下图所示：

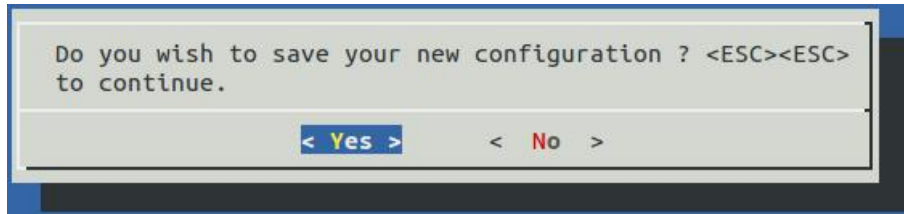


图 6.2 内核配置保存提示画面

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：

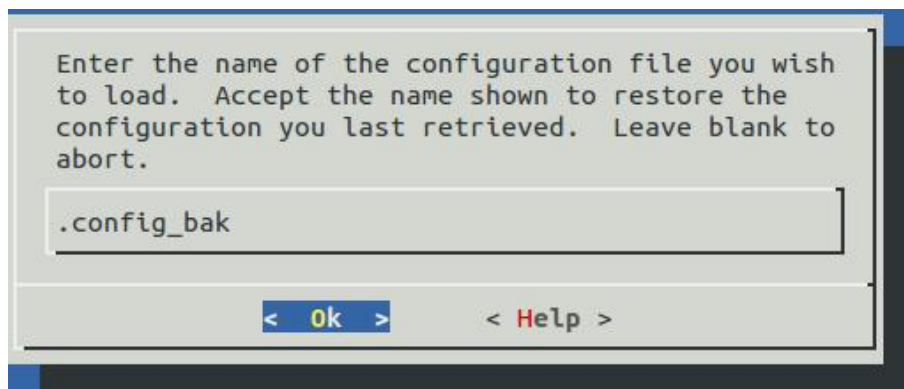


图 6.3 内核配置加载画面

在该画面中输入需要加载的备份文件的名称，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。例如将名为 `sc_imx6ul_som_defconfig` 的内核配置文件拷贝到<arch/arm/configs/>目录下,以后只需执行下列命令即可使用该配置:

```
$ make sc_imx6ul_som_defconfig
```

## 6.2 编译内核

本节介绍如何编译 Linux 操作系统映像文件 `zImage`。如前所述,该映像文件除了包含 Linux 内核外,还内置了一个小型的根文件系统。因此在编译时除了需要准备好 Linux 内核源码外,还需要准备好根文件系统。

编译 `zImage` 使用的是 `arm-linux-gnueabi-gcc 4.9.2` 版本的交叉编译器。关于编译器的安装可参见“安装编译内核的交叉编译器”一节。

在“产品光盘资料/3、软件开发参考资料/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-src.tar.gz` 和根文件系统 `root.tar.gz`。编译 Linux 操作系统的步骤如下:

1. 将 Linux 内核源码文件 `linux-src.tar.gz` 和根文件系统 `root.tar.gz` 拷贝到 Ubuntu 虚拟机中,此处假定将这两个文件拷贝到 `/home/sinc/imx6ul/bsp` 中,并对其进行解压:

```
$cd /home/sinc/imx6ul/bsp/  
$tar zxvf linux-src.tar.gz  
$tar zxvf root.tar.gz
```

2. 清除前一次的编译结果。执行如下命令:

```
$cd /home/sinc/imx6ul/bsp/linux-src  
$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi-gcc distclean
```

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

```
$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi-gcc sc_imx6ul_som_defconfig
```

注: 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 ARCH=arm CROSS_COMPILE=arm-linux-gnueabi-gcc menuconfig
```

注: 如果不需要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 ARCH=arm CROSS_COMPILE=arm-linux-gnueabi-gcc all
```

编译完成后,会在内核源码顶层目录(即 `linux-src` 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
built-all-sc.sh	清除以前的编译结果，全部重新编译
built-dtb-sc.sh	编译设备树
built-modules-sc.sh	编译内核模块
built-zImage-sc.sh	编译内核映像
built-menuconfig.sh	使用 menuconfig 配置内核

# 第 7 章 文件系统

## 7.1 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。

### 7.1.1 根文件系统目录结构

嵌入式 Linux 根文件系统布局，建议还是按照 FHS（Filesystem Hierarchy Standard）标准来安排。事实上大多数嵌入式 Linux 都是这样做的。但是嵌入式系统可能并不需要桌面/服务器那样庞大系统的全部目录，可以酌情对系统目录进行精简，以简化 Linux 的使用。下面以 TWEvb-IMX6UL 开发板为例，介绍根文件系统的目录结构。

TWEvb-IMX6UL 开发板根文件系统各个目录介绍如下：

表 7.1 根文件系统目录结构

目录	说明
bin	用户必须的命令和工具
dev	设备文件和其他特殊文件
etc	系统初始化脚本和配置文件
lib	系统运行所需要的库文件，如 C 库等
proc	proc 文件系统。用来提供内核与进程信息的虚拟文件系统
sbin	系统管理员必须的命令和工具
sys	sysfs 文件系统。用来对系统的设备进行管理的虚拟文件系统
tmp	tmpfs 文件系统。为一种基于内存的文件系统，并可动态调整大小。
var	存放系统运行时产生的不可数据
mnt	挂载点，可用于临时挂载文件系统。例如挂载 U 盘、SD 卡等
usr	包含用户常用的命令和工具
root	用户文件系统，挂载在/root 下。可实现掉电保存
home	用户文件系统，挂载在/home 下。可实现掉电保存

### 7.1.2 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 NOR Flash 还是 NAND Flash，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、 cramfs、 jffs2、 yaffs/yaffs2 和 ubifs 等，各个文件系统的特点如下表所示：

表 7.2 常用文件系统

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
cramfs	-	是	否	-	否
jffs2	NOR Flash	是	是	是	否
yaffs/yaffs2	NAND Flash	否	是	是	否
ubifs	NAND Flash	是	是	是	否

尽管文件系统以某一种文件系统的镜像发布，但是整个文件系统实际上还并存着多种逻辑文件系统。例如一个系统根文件系统以 ubifs 挂载，但是/dev 目录却是以 tmpfs 挂载、/sys 目录挂载的是 sysfs 文件系统。可以在命令行下运行 mount 命令查看当前系统已经挂载的全部文件系统：

```
~ # mount
rootfs on / type rootfs (rw)
none on /proc type proc (rw,relatime)
sysfs on /sys type sysfs (rw,relatime)
devtmpfs on /dev type devtmpfs (rw,relatime,size=29276k,nr_inodes=7319,mode=755)
none on /tmp type tmpfs (rw,relatime)
devpts on /dev/pts type devpts (rw,relatime,mode=600)
/dev/mtdblock5 on /root type yaffs2 (rw,relatime)
/dev/mtdblock6 on /home type yaffs2 (rw,relatime) none
on /debugfs type debugfs (rw,relatime)
```

以下将对一些常用的文件系统分别加以介绍。

## 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。

## 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，sequenceNumber 用 29 bits 表示。

## 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 Flash 的更新频率。

## 7.2 设置开机自启动程序

对于 TWEvb-IMX6UL 开发板，启动脚本为/root/etc/rc.ini 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/root/etc/rc.ini 即可。例如需要开机自动运行/home/demo/ledtest 程序，可以在/root/etc/rc.ini 文件的最后添加如下内容：

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

其中最后的“&”表示程序在后台运行。

## 7.3 动态加载模块

可以使用 insmod 命令动态加载模块。例如需要加载/root/etc/tsdev.ko 模块，在命令行下执行如下命令：

```
~ # insmod /root/etc/tsdev.ko
```

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

```
~ # rmmod tsdev.ko
```

可以使用 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
```

# 第 8 章 系统恢复与更新

## 8.1 Nand Flash 分区

TWEvb-IMX6UL 开发板 Nand Flash 的容量为 256M，其具体分区情况如下：

表 8.1NandFlash 分区

分区	地址范围	大小	用途
bootloader	0x00000000-0x00400000	4M	U-Boot 引导映像
dtb	0x00400000-0x00500000	1M	设备树
kernel	0x00500000-0x01500000	16M	Linux 操作系统映像
logo	0x01500000-0x01800000	3M	3M
root	0x01800000-0x03800000	32M	用户文件系统，挂载在/root 目录下
home	0x03800000-0x10000000	200M	用户文件系统，挂载在/home 目录下

其中，Linux 操作系统映像文件 zImage 除包含 Linux 内核本身外，还内置了一个小型的根文件系统。这样在烧写完 uboot、设备树和 zImage 后，操作系统就可以正常启动，并可在 linux 命令行下执行 Linux 操作系统下的一些常用命令（如 ls 等），这些基础命令就是由操作系统映像文件 zImage 中内置的小型根文件系统提供的。为方便用户增加新的应用程序，并实现断电保存功能，在这个根文件系统里实现了两个可读写的 yaffs2 文件系统分区，并将这两个分区加载到/root 和/home 目录下，在这两个目录下的文件可实现断电不丢失。

## 8.2 烧写系统映像

可以使用 MfgTools 烧写工具通过开发板的 USB 从口烧写系统映像。MfgTools 烧写工具位于“眺望产品光盘资料/3、软件开发参考资料/6、出厂镜像文件/mtgtools”目录下。烧写时，相关的系统映像文件位于 mfgtools 文件夹的 Profiles\Linux\OS Firmware\TWCORE-IMX6UL 子文件夹中，用户可以根据实际需要替换为自己的映像文件。

TWCORE-IMX6UL 目录中各个文件的说明如下：

表 8.2 映像文件说明

文件名	描述
u-boot.imx	U-Boot 映像文件
TWCore-IMX6UL-256m.dtb	设备树文件
zImage	内核文件
root_imx6ul.tar.gz	用户文件系统，挂载在/root 目录下
home_imx6ul.tar.gz	用户文件系统，挂载在/home 目录下
logo	开机画面

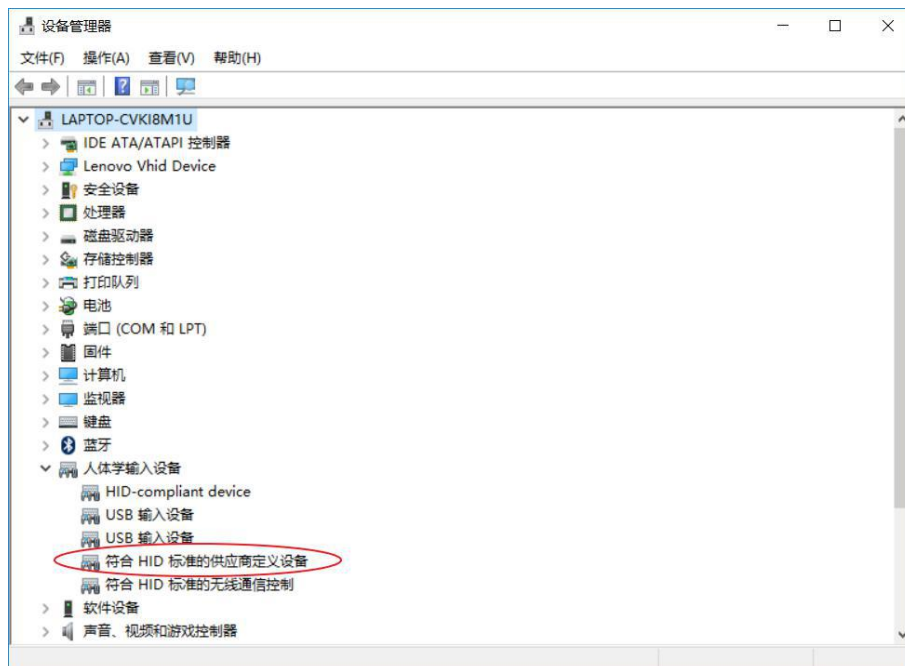
具体烧写步骤如下：

## 8.2.1 硬件连接

硬件连接方法如下：

- (1) 断开 TWEvb-IMX6UL 开发板的供电电源；
- (2) 把 TWEvb-IMX6UL 开发板设置为 USB 启动方式（拨码开关拨到 1100）；
- (3) 使用 MicroUSB 线缆将 TWEvb-IMX6UL 开发板的 USB 从口与计算机的 USB 端口相连；
- (4) 重新给 TWEvb-IMX6UL 开发板上电。

注：若上述 4 个步骤操作正确，则 TWEvb-IMX6UL 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：

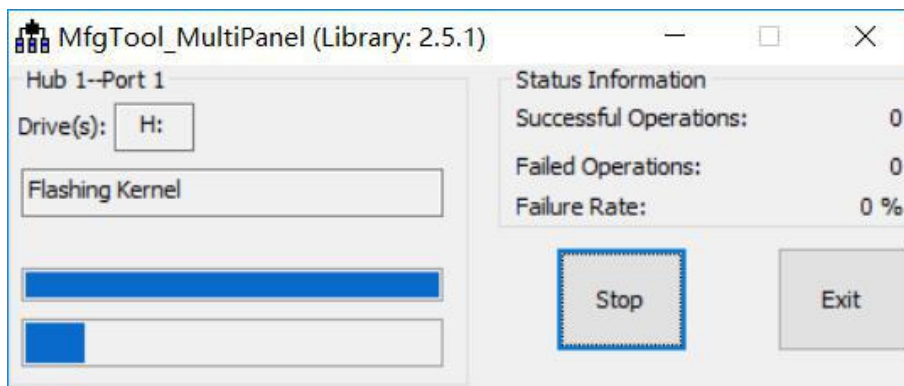


## 8.2.2 使用 MtgTools 软件进行烧写

双击运行 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：

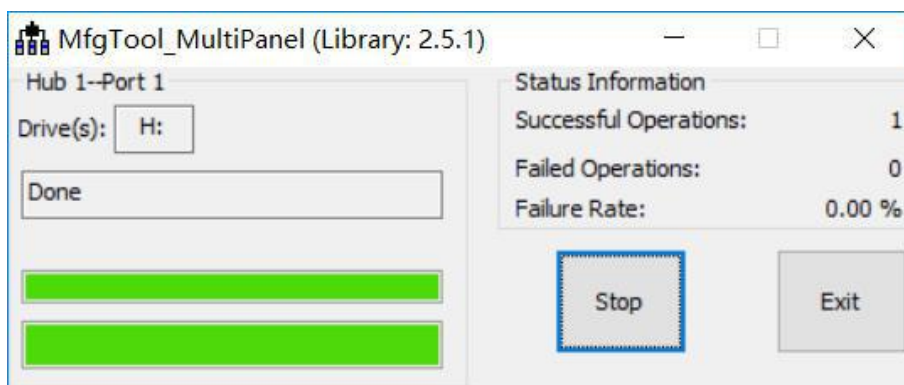


点击 Start 按钮开始进行固件的烧写工作，烧写过程如下图所示：



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 TWEvb-IMX6UL 开发板的供电电源并将拨码开关拨到 0000 位置（即从 NANDFLASH 启动），完成固件烧写工作。

## 8.3 烧写 U-Boot 映像

本节介绍在操作系统命令行下，单独烧写 U-Boot 映像文件（u-boot.imx）的步骤：

- 1、 将 U-Boot 映像文件 u-boot.imx 拷贝到开发板任意目录下：
- 2、 擦除 U-Boot 映像所在的 NAND 分区，执行如下命令：

```
~ # flash_eraseall /dev/mtd0
```

3、 使用 kobs-ng 命令将 U-Boot 映像烧写到 Nand Flash 中，执行如下命令：（此处假定 u-boot.imx 位于开发板/tmp 目录下）

```
~ # /root/bin/kobs-ng init -x -v --chip_0_device_path=/dev/mtd0 /tmp/u-boot.imx
```

## 8.4 烧写 DTB

本节介绍在操作系统命令行下，单独烧写设备树文件 DTB 的步骤：

- 1、 将设备树文件拷贝到开发板任意目录下：
- 2、 擦除设备树所在的 NAND 分区，执行如下命令：

```
~ # flash_eraseall /dev/mtd1
```

3、 将设备树文件烧写到 Nand Flash 中，执行如下命令：（此处假定设备树文件为 TWCORE-IMX6UL-256m.dtb 且位于开发板/tmp 目录下）

```
~ #nandwrite -p /dev/mtd1 /tmp/TWCORE-IMX6UL-256m.dtb
```

## 8.5 烧写 Linux 操作系统

本节介绍在操作系统命令行下，单独烧写操作系统映像文件（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：
- 2、 擦除操作系统所在的 NAND 分区，执行如下命令：

```
~ # flash_eraseall /dev/mtd2
```

3、 将操作系统映像烧写到 Nand Flash 中，执行如下命令：（此处假定操作系统映像文件位于开发板/tmp 目录下）

```
~ #nandwrite -p /dev/mtd2 /tmp/zImage
```

## 8.6 烧写用户文件系统

本节介绍在操作系统命令行下，单独烧写用户文件系统（/root 和/home）的步骤：

1、 将用户文件系统压缩包 `root_imx6ul.tar.gz` 复制到开发板/`tmp` 目录下。拷贝前，如果/`tmp` 目录下有其他文件，建议删除这些文件，以减少内存占用：

2、 将该文件系统压缩包解压到/`root` 目录下。执行如下命令：

```
/tmp # cd /  
~ # tar zxvf /tmp/root_imx6ul.tar.gz
```

3、 将用户文件系统压缩包 `home_imx6ul.tar.gz` 复制到开发板/`tmp` 目录下。拷贝前，如果/`tmp` 目录下有其他文件，建议删除这些文件，以减少内存占用：

4、 将该文件系统压缩包解压到/`home` 目录下。执行如下命令：

```
/tmp # cd /  
~ # tar zxvf /tmp/home_imx6ul.tar.gz
```

## 8.7 编译 Linux 操作系统

本节介绍如何编译 Linux 操作系统映像文件 `zImage`。如前所述，该映像文件除了包含 Linux 内核外，还内置了一个小型的根文件系统。因此在编译时除了需要准备好 Linux 内核源码外，还需要准备好根文件系统。

编译 `zImage` 使用的是 `arm-linux-gnueabi-gcc 4.9.2` 版本的交叉编译器。关于编译器的安装可参见“安装编译内核的交叉编译器”一节。

在“产品光盘资料/3、软件开发参考资料/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-src.tar.gz` 和根文件系统 `root.tar.gz`。用户可以直接在该内核源码文件的基础上进行编译，而无需做任何配置。本节将介绍如何基于这个内核源码文件对

Linux 操作系统进行编译，如果用户需要重新对 Linux 内核进行配置，可参考“编译内核”一节。编译 Linux 操作系统的步骤如下：

将 Linux 内核源码文件 `linux-src.tar.gz` 和根文件系统 `root.tar.gz` 拷贝到 Ubuntu 虚拟机中，此处假定将这两个文件拷贝到/`home/sinc/imx6ul/bsp` 中，并对其解压：

```
$cd /home/sinc/imx6ul/bsp $tar zxvf linux-src.tar.gz $tar zxvf root.tar.gz
```

编译内核。执行如下命令：

```
$cd /home/sinc/imx6ul/bsp/linux-src  
$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi-gcc zImage
```

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



## 第 9 章 免责声明

本档提供有关广州眺望电子科技有限公司产品的信息。本档并未授予任何知识产权的许可，并未以明示或暗示，或以禁止发言或其它方式授予任何知识产权许可。

除眺望电子在其产品的销售条款和条件中声明的责任之外，眺望电子概不承担任何其它责任。并且，眺望电子对产品的销售和 / 或使用不作任何明示或暗示的担保，包括对产品的特定用途适用性、适销性或任何专利权、版权或其它知识产权的侵权责任等，均不作担保。

眺望电子产品并非设计用于医疗、救生或维生等用途。眺望电子可能随时对产品规格及产品描述做出修改，恕不另行通知。

在订购产品之前，请您与当地的广州眺望电子科技有限公司销售处或分销商联系，以获取最新的规格说明。

本档中提及的含有订购号的文档以及其它文献可通过访问 <http://www.iot-tw.com/> 获得。

广州眺望电子科技有限公司保留所有权利。

