君正®
T41 BSP 开发参考 V2.1

Date: 2023-07

北京君正集成电路股份有限公司
Ingenic Semiconductor Co., Ltd.

Trademarks and Permissions

Disclaimer

All the deliverables and data in this folder serve only as a reference for customerdevelopment. Please read through this disclaimer carefully before you use the deliverables and data in this folder. You may use the deliverables in this folder or not. However, by using the deliverables and data in this folder, you agree to accept all the content in this disclaimer unconditional and irrevocable. If you do not find the content in this disclaimer reasonable, you shall not use the deliverables and data in this folder.

The deliverables and data in this folder are provided "AS IS" without representations, guarantees or warranties of any kind (either express or implied). To the maximum extent permitted by law, Ingenic Semiconductor Co., Ltd (Ingenic) provides the deliverables and data in this folder without implied representations, guarantees or warranties, including but not limited to implied representations, guarantees and warranties of merchantability, non-infringement, or fitness for a particular purpose. Deviation of the data provided in this folder may exist under different test environments.

Ingenic takes no liability or legal responsibility for any design and development error, incident, negligence, infringement, and loss (including but not limited to any direct, indirect, consequential, or incidental loss) caused by the use of data in this folder. Users shall be responsible for all risks and consequences caused by the use of data in this folder.

北京君正集成电路股份有限公司

地址: 北京市海淀区西北旺东路 10 号院东区 14 号楼君正大厦

电话:**(86-10)56345000**

传真:**(86-10)56345001**

Http: //www.ingenic.com.cn

前言

概述

本文为 Ingenic-T41 开发参考，旨在帮助用户了解 T41 的功能及开发流程 。

产品版本

与本文档相对应的产品版本如下。

产品名称	产品版本
T41 BSP 开发参考	V2.1

读者对象

本文档（本指南）主要适用于以下工程师：

- 技术支持工程师
- 软件开发工程师

修订记录

修订记录累积了每次文档更新的说明。最新版本的文档包含以前所有文档版本的更新内容。

日期	版本	修订章节
2022-07	1.0	第一次正式版本发布
2022-08	1.1	**5 系统启动与烧录** 5.1 系统启动涉及新增 5.2 系统烧录涉及修改 **6 系统资源与使用调试** 6.1 ISP_sensor

		表 6-1 sample 功能涉及修改 6.3 GPIO 小节涉及新增 6.4 UART 小节涉及修改 6.7 PWM 小节涉及修改 6.9 I2C 小节涉及修改
2022-08-08	1.2	**6 系统资源与使用调试** 6.5 Watchdog 涉及修改
2022-08-22	1.3	**6 系统资源与使用调试** 6.15.2.2 Device 设备 USB RNDIS 配置涉及修改
2022-09	1.4	**3 T41 必读** 3.1 Uboot 编译涉及修改 **6 系统资源与使用调试** 6.1.5 表 6-1sample 的使用涉及更新 6.2 Audio 部分涉及修改 6.14 WIFI 支持的模组涉及更新 6.15 USB 部分涉及新增
2023-04	2.0	**6 系统资源与使用调试** 6.11 SPI 部分涉及修改
2023-07	2.1	**2 开发环境搭建** 2.3 交叉工具链涉及更新

目录

1 开发资源介绍

1.1 SDK 介绍

ISVP SDK，即软件开发工具包，包括 API 库、开源源码、文档、Samples 等。开发者可以通过 SDK 快速的开展产品功能开发。以下是 ISVP SDK 的内容概览图：

图 1-1 SDK 组成结构

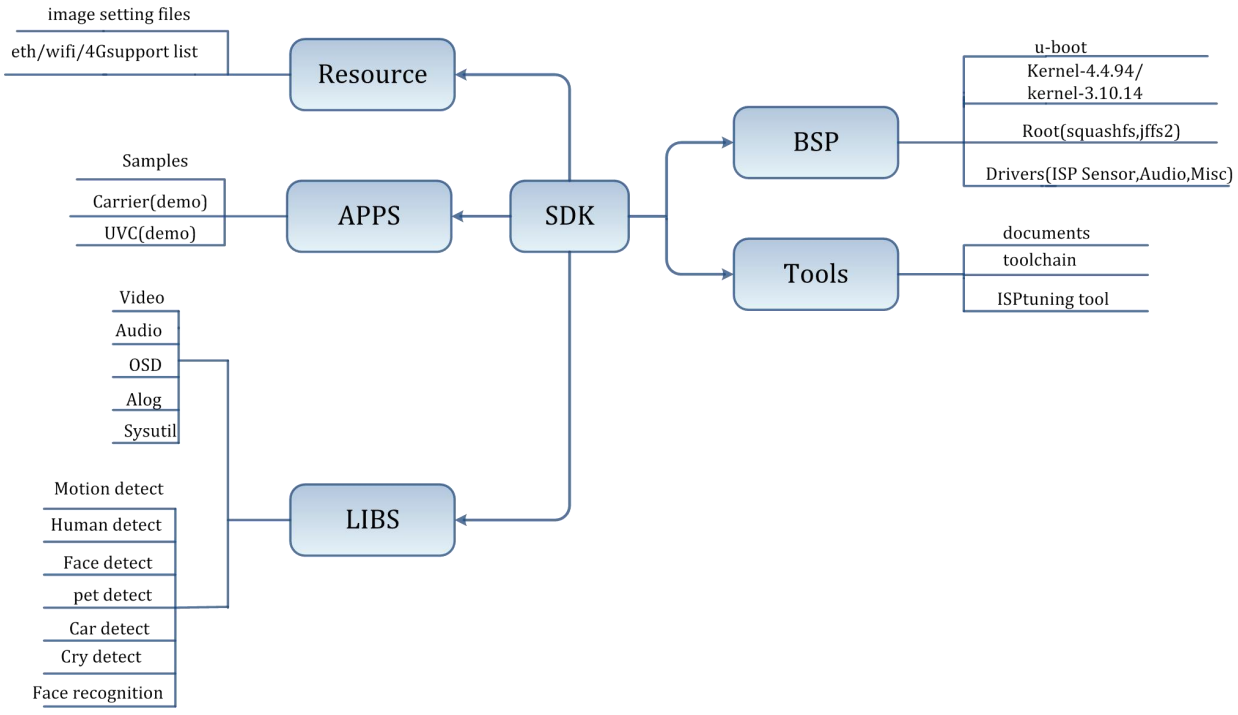
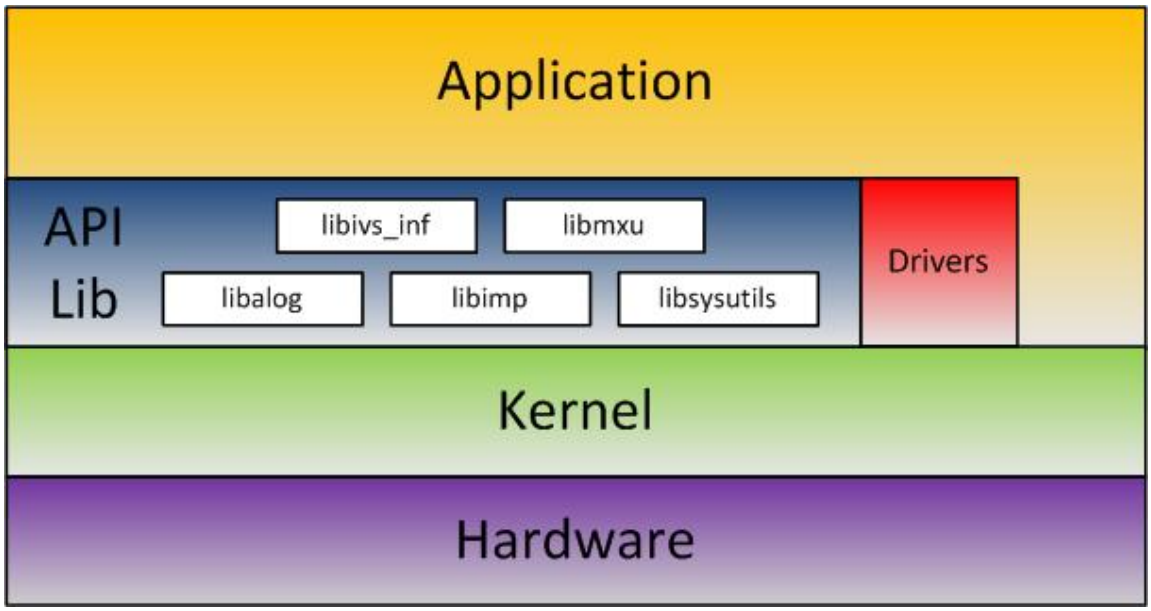


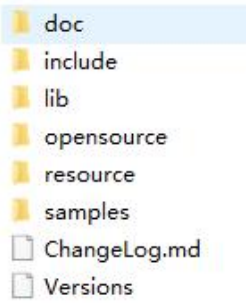
图 1-2 SDK 层次结构



- Hardware: 硬件层，完成 I/O 等具体的硬件功能。
- Linux Kernel: 内核层，完成基础的系统功能，定义硬件资源。
- Drivers: ko 模块驱动，可通过 driver 进行硬件操作。
- API Lib: 接口库，实现硬件功能的抽象，方便于应用层的开发。API 库主要有五部分：
  - libimp: 多媒体功能库，如 H265/H264 编码，JPEG 编码，IVS 和 Audio 等。
  - libsysutils: 系统功能库，如重启，设置系统时间和电池功能等。
  - libalog: ISVP-SDK 的 log 实现库。
  - libivs_inf: IVS 算法库，包括越线检测，周界防范等。
  - libmxu: 512 位 mxu 加速指令算子库。
- Application: 应用层，实现功能逻辑等。

Application 推荐使用 SDK 库提供的 API 并配合 drivers 进行开发。对于一些特殊的功能需求，也可以直接调用内核接口进行开发。

1.2 SDK LIB



doc：指导文档

include：相关头文件

lib：相关库文件

opensource：kernel、uboot、drivers、busybox 等

resource：文件镜像、效果文件等

samples：相关应用代码

2 开发环境搭建

2.1 为什么需要搭建开发环境

由于嵌入式单板的资源有限，不能在单板上运行开发和调试工具，通常需要交叉编译调试的方式进行开发和调试，即“宿主机＋目标机”的形式。宿主机和目标机一般采用串口连接显示交互信息，网口连接传输文件。

宿主机和目标机的处理器一般不相同。宿主机需建立适合于目标机的交叉编译环境。程序在宿主机上经过“编译－链接－定位”得到可执行文件。通过一定的方法将可执行文件烧写到目标机中，然后在目标机上运行。

2.2 安装 Linux 服务器

建议选择常用的 Linux 发行版，便于寻找各类技术资源。例如：

RedHat 较新的发行版如 RedHat Fedora Core 系列和 Redhat Enterprise Linux、RedHat 3.4.4-2、RedHat 较老的发行版如 RedHat 9.0 等。

推荐使用较新版本，以便获取各类资源，如 Fedora Core 系列、Ubuntu12 版本以上。

2.3 交叉工具链

Toolchain 即交叉编译工具链，是 Linux Host 机上用来编译和调试嵌入式设备程序的一系列工具的集合。ISVP 中的 Toolchain 版本信息如下：

1) gcc 版本：7.2.0
2) libc 版本：
   - glibc 版本：2.29
   - uclibc 版本：0.9.33.2

2.3.1 如何安装 Toolchain

安装流程：

第一步：

根据 Host 机 CPU 位宽选择 mips-gcc720-glibc229-r5.1.9.tar.bz2 进行解压。例如：

```
$ tar -jxvf mips-gcc720-glibc229-r5.1.9.tar.bz2
```

第二步：

通过 export PATH=xxxx:$PATH 命令，将 toolchain 下的 bin 目录添加到 PATH 环境变量中或者在~/.bashrc 中加上下面一句永久改变。

```
$ vim ~/.bashrc
$ export PATH=/opt/mips-gcc720-glibc229-r5.1.9/bin:$PATH
```

第三步：

测试 toolchain 可执行：

```
$ mips-linux-gnu-gcc --version
mips-linux-gnu-gcc (Ingenic Linux-Release5.1.9-Default_xburst2_glibc2.2
9 Fix: uclibc popen and pclose 2023.08-15 09:56:16) 7.2.0
Copyright (C) 2017 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is
NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPO
SE.
```

若出现如上信息则可确认 toolchain 安装正确。

2.3.2 如何进行 glibc 和 uclibc 编译

ISVP 的 toolchain 包含了 glibc 和 uclibc，因此基于 glibc 或者 uclibc 的程序均可使用此 toolchan 进行编译。

1) glibc 程序编译方法：

默认 link 的 libc 即为 glibc

2) uclibc 程序编译方法：

C_FLAGS+=-muclibc CXX_FLAGS+=-muclibc，LD_FLAGS+=-muclibc

2.3.3 交叉工具包

交叉工具链是一个软件安装包，是由很多工具的集合。有常用的 gcc 编译工具、o

bjdump 反汇编工具、as 汇编器、ld 链接器、size 查看二进制文件大小等。

```
cmake                       mips-linux-gnu-gcc          mips-linux-gnu-gprof
cpack                       mips-linux-gnu-gcc-7.2.0    mips-linux-gnu-ld
ctest                       mips-linux-gnu-gcc-ar       mips-linux-gnu-ldd
gccgo                       mips-linux-gnu-gccgo        mips-linux-gnu-nm
mips-linux-gnu-addr2line    mips-linux-gnu-gcc-nm       mips-linux-gnu-objcopy
mips-linux-gnu-ar           mips-linux-gnu-gcc-ranlib   mips-linux-gnu-objdump
mips-linux-gnu-as           mips-linux-gnu-gcov         mips-linux-gnu-prelink
mips-linux-gnu-c++          mips-linux-gnu-gcov-dump    mips-linux-gnu-ranlib
mips-linux-gnu-c++filt      mips-linux-gnu-gcov-tool    mips-linux-gnu-readelf
mips-linux-gnu-cpp          mips-linux-gnu-gdb          mips-linux-gnu-size
mips-linux-gnu-elfedit      mips-linux-gnu-gfortran     mips-linux-gnu-strings
mips-linux-gnu-execstack    mips-linux-gnu-go           mips-linux-gnu-strip
mips-linux-gnu-g++          mips-linux-gnu-gofmt
```

3 T41 必读

3.1 Uboot 编译

1. Uboot 编译流程：

u-boot 可单独编译，不依赖其他代码。T41 u-boot 的板机配置文件位于 include/configs/isvp_t41.h。

第一步：$ make distclean 清除旧配置

第二步：$ make isvp_t41x_xxx 根据芯片类型及需求编译 u-boot-with-spl.bin

表 3-1 T41 芯片编译 SD 卡启动的 uboot

型号	编译命令	说明	主频	DDR
T41L	make isvp_t41l_msc0	编译 SD 卡启动的 uboot，针对 T41L 芯片(msc0 接口)	1104	600
T41N	make isvp_t41n_msc0	编译 SD 卡启动的 uboot，针对 T41N 芯片(msc0 接口)	1104	750
T41LQ	make isvp_t41lq_msc0	编译 SD 卡启动的 uboot，针对 T41LQ 芯片(msc0 接口)	1104	600
T41NQ	make isvp_t41nq_msc0	编译 SD 卡启动的 uboot，针对 T41NQ 芯片(msc0 接口)	1104	700
T41XQ	make isvp_t41xq_msc0	编译 SD 卡启动的 uboot，针对 T41XQ 芯片(msc0 接口)	1104	700
T41L	make isvp_t41l_msc0_lp	编译低性能/低功耗（备选方案）的 SD 卡启动的 uboot，针对 T41L 芯片(msc0 接口)，支持 5MP/30fps	1008	550
T41N	make isvp_t41n_msc0_lp	编译低性能/低功耗（备选方案）的 SD 卡启动的 uboot，针对 T41	1008	550

		N 芯片(msc0 接口)，支持 5MP/30 fps		
T41LQ	make isvp_t41lq_msc0_lp	编译低性能/低功耗（备选方案）的 SD 卡启动的 uboot，针对 T41 LQ 芯片(msc0 接口)，支持 5MP/30fps	1008	550
T41NQ	make isvp_t41nq_msc0_lp	编译低性能/低功耗（备选方案）的 SD 卡启动的 uboot，针对 T41 NQ 芯片(msc0 接口)，支持 5MP/30fps	1008	550
T41XQ	make isvp_t41xq_msc0_lp	编译低性能/低功耗（备选方案）的 SD 卡启动的 uboot，针对 T41 XQ 芯片(msc0 接口)，支持 5MP/30fps	1008	550
T41L	make isvp_t41l_msc1	编译 SD 卡启动的 uboot，针对 T41L 芯片(msc1 接口)	1104	600
T41N	make isvp_t41n_msc1	编译 SD 卡启动的 uboot，针对 T41N 芯片(msc1 接口)	1104	700
T41LQ	make isvp_t41lq_msc1	编译 SD 卡启动的 uboot，针对 T41LQ 芯片(msc1 接口)	1104	600
T41NQ	make isvp_t41nq_msc1	编译 SD 卡启动的 uboot，针对 T41NQ 芯片(msc1 接口)	1104	700
T41XQ	make isvp_t41xq_msc1	编译 SD 卡启动的 uboot，针对 T41XQ 芯片(msc1 接口)	1104	700
T41L	make isvp_t41l_msc1_lp	编译低性能/低功耗（备选方案）的 SD 卡启动的 uboot，针对 T41L 芯片(msc1 接口)，支持 5MP/30fps	1008	550
T41N	make isvp_t41n_msc1_lp	编译低性能/低功耗（备选方案）的 SD 卡启动的 uboot，针对 T41N 芯片(msc1 接口)，支持 5MP/30 fps	1008	550
T41LQ	make isvp_t41lq_msc1_lp	编译低性能/低功耗（备选方案）的 SD 卡启动的 uboot，针对 T41	1008	550

		LQ 芯片(msc1 接口)，支持 5MP/30fps		
T41NQ	make isvp_t41nq_msc1_lp	编译低性能/低功耗（备选方案）的 SD 卡启动的 uboot，针对 T41NQ 芯片(msc1 接口)，支持 5MP/30fps	1008	550
T41XQ	make isvp_t41xq_msc1_lp	编译低性能/低功耗（备选方案）的 SD 卡启动的 uboot，针对 T41XQ 芯片(msc1 接口)，支持 5MP/30fps	1008	550

表 3-2 T41 芯片编译 nand flash 启动的 uboot

型号	编译命令	说明	主频	DDR
T41L	make isvp_t41l_sfc0_nand	编译 nand flash 启动的 uboot，针对 T41L 芯片(sfc0 接口)	1104	600
T41N	make isvp_t41n_sfc0_nand	编译 nand flash 启动的 uboot，针对 T41N 芯片(sfc0 接口)	1104	750
T41LQ	make isvp_t41lq_sfc0_nand	编译 nand flash 启动的 uboot，针对 T41LQ 芯片(sfc0 接口)	1104	600
T41NQ	make isvp_t41nq_sfc0_nand	编译 nand flash 启动的 uboot，针对 T41NQ 芯片(sfc0 接口)	1104	700
T41XQ	make isvp_t41xq_sfc0_nand	编译 nand flash 启动的 uboot，针对 T41XQ 芯片(sfc0 接口)	1104	700
T41L	make isvp_t41l_sfc0_nand_lp	编译低性能/低功耗（备选方案）的 nand flash 启动的 uboot，针对 T41L 芯片(sfc0 接口)，支持 5MP/30fps	1008	550
T41N	make isvp_t41n_sfc0_nand_lp	编译低性能/低功耗（备选方案）的 nand flash 启动的 uboot，针对 T41N 芯片(sfc0 接口)，支持 5MP/30fps	1008	550
T41LQ	make isvp_t41lq_sfc0_nand_lp	编译低性能/低功耗（备选方案）的 nand flash 启动的 uboot，针对 T41LQ 芯片(sfc0 接口)，支持 5M	1008	550

		P/30fps		
T41NQ	make isvp_t41nq_sfc0_nand_lp	编译低性能/低功耗（备选方案）的 nand flash 启动的 uboot，针对 T41NQ 芯片(sfc0 接口)，支持 5M P/30fps	1008	550
T41XQ	make isvp_t41xq_sfc0_nand_lp	编译低性能/低功耗（备选方案）的 nand flash 启动的 uboot，针对 T41XQ 芯片(sfc0 接口)，支持 5M P/30fps	1008	550
T41L	make isvp_t41l_sfc1_nand	编译 nand flash 启动的 uboot，针对 T41L 芯片(sfc1 接口)	1104	600
T41N	make isvp_t41n_sfc1_nand	编译 nand flash 启动的 uboot，针对 T41N 芯片(sfc1 接口)	1104	750
T41LQ	make isvp_t41lq_sfc1_nand	编译 nand flash 启动的 uboot，针对 T41LQ 芯片(sfc1 接口)	1104	600
T41NQ	make isvp_t41nq_sfc1_nand	编译 nand flash 启动的 uboot，针对 T41NQ 芯片(sfc1 接口)	1104	700
T41XQ	make isvp_t41xq_sfc1_nand	编译 nand flash 启动的 uboot，针对 T41XQ 芯片(sfc1 接口)	1104	700
T41L	make isvp_t41l_sfc1_nand_lp	编译低性能/低功耗（备选方案）的 nand flash 启动的 uboot，针对 T41L 芯片(sfc1 接口)，支持 5MP/30fps	1008	550
T41N	make isvp_t41n_sfc1_nand_lp	编译低性能/低功耗（备选方案）的 nand flash 启动的 uboot，针对 T41N 芯片(sfc1 接口)，支持 5MP/30fps	1008	550
T41LQ	make isvp_t41lq_sfc1_nand_lp	编译低性能/低功耗（备选方案）的 nand flash 启动的 uboot，针对 T41LQ 芯片(sfc1 接口)，支持 5M P/30fps	1008	550
T41NQ	make isvp_t41nq_sfc1_nand_lp	编译低性能/低功耗（备选方案）的 nand flash 启动的 uboot，针对 T41NQ 芯片(sfc1 接口)，支持 5M	1008	550

		P/30fps		
T41XQ	make isvp_t41xq_sfc1_nand_lp	编译低性能/低功耗（备选方案）的 nand flash 启动的 uboot，针对 T41XQ 芯片(sfc1 接口)，支持 5M P/30fps	1008	550

表 3-3 T41 芯片编译 nor flash 启动的 uboot

型号	编译命令	说明	主频	DDR
T41L	make isvp_t41l_sfc_nor	编译 nor flash 启动的 uboot，针对 T41L 芯片	1104	600
T41N	make isvp_t41n_sfc_nor	编译 nor flash 启动的 uboot，针对 T41N 芯片	1104	750
T41LQ	make isvp_t41lq_sfc_nor	编译 nor flash 启动的 uboot，针对 T41LQ 芯片	1104	600
T41NQ	make isvp_t41nq_sfc_nor	编译 nor flash 启动的 uboot，针对 T41NQ 芯片	1104	700
T41XQ	make isvp_t41xq_sfc_nor	编译 nor flash 启动的 uboot，针对 T41XQ 芯片	1104	700
T41L	make isvp_t41l_sfc_nor_lp	编译低性能/低功耗（备选方案）的 nor flash 启动的 uboot，针对 T41L 芯片，支持 5MP/30fps	1008	550
T41N	make isvp_t41n_sfc_nor_lp	编译低性能/低功耗（备选方案）的 nor flash 启动的 uboot，针对 T41N 芯片，支持 5MP/30fps	1008	550
T41LQ	make isvp_t41lq_sfc_nor_lp	编译低性能/低功耗（备选方案）的 nor flash 启动的 uboot，针对 T41LQ 芯片，支持 5MP/30fps	1008	550
T41NQ	make isvp_t41nq_sfc_nor_lp	编译低性能/低功耗（备选方案）的 nor flash 启动的 uboot，针对 T41NQ 芯片，支持 5MP/30fps	1008	550
T41XQ	make isvp_t41xq_sfc_nor_lp	编译低性能/低功耗（备选方案）的 nor flash 启动的 uboot，针对 T41XQ 芯片，支持 5MP/30fps	1008	550

表 3-4 T41 芯片编译 uart 启动的 uboot

型号	编译命令	说明	主频	DDR
T41L	make isvp_t41l_uart_msc	编译 uart 启动 uboot，且 uboot 命令行支持 msc 相关命令，针对 T41L 芯片	1104	600
T41N	make isvp_t41n_uart_msc	编译 uart 启动 uboot，且 uboot 命令行支持 msc 相关命令，针对 T41N 芯片	1104	750
T41LQ	make isvp_t41lq_uart_msc	编译 uart 启动 uboot，且 uboot 命令行支持 msc 相关命令，针对 T41LQ 芯片	1104	600
T41NQ	make isvp_t41nq_uart_msc	编译 uart 启动 uboot，且 uboot 命令行支持 msc 相关命令，针对 T41NQ 芯片	1104	700
T41XQ	make isvp_t41xq_uart_msc	编译 uart 启动 uboot，且 uboot 命令行支持 msc 相关命令，针对 T41XQ 芯片	1104	700
T41L	make isvp_t41l_uart_msc_lp	编译低性能/低功耗（备选方案）的 uart 启动 uboot，且 uboot 命令行支持 msc 相关命令，针对 T41L 芯片，支持 5MP/30fps	1008	550
T41N	make isvp_t41n_uart_msc_lp	编译低性能/低功耗（备选方案）的 uart 启动 uboot，且 uboot 命令行支持 msc 相关命令，针对 T41N 芯片，支持 5MP/30fps	1008	550
T41LQ	make isvp_t41lq_uart_msc_lp	编译低性能/低功耗（备选方案）的 uart 启动 uboot，且 uboot 命令行支持 msc 相关命令，针对 T41LQ 芯片，支持 5MP/30fps	1008	550
T41NQ	make isvp_t41nq_uart_msc_lp	编译低性能/低功耗（备选方案）的 uart 启动 uboot，且 uboot 命令行支持 msc 相关命令，针对 T41NQ 芯片，支持 5MP/30fps	1008	550
T41XQ	make isvp_t41xq_uart_msc_lp	编译低性能/低功耗（备选方案）的 uart 启动 uboot，且 uboot 命令	1008	550

		行支持 msc 相关命令，针对 T41XQ 芯片，支持 5MP/30fps		
T41L	make isvp_t41l_uart_sfc	编译 uart 启动 uboot，且 uboot 命令行支持 sfc 相关命令，针对 T41L 芯片	1104	600
T41N	make isvp_t41n_uart_sfc	编译 uart 启动 uboot，且 uboot 命令行支持 sfc 相关命令，针对 T41N 芯片	1104	750
T41L	make isvp_t41lq_uart_sfc	编译 uart 启动 uboot，且 uboot 命令行支持 sfc 相关命令，针对 T41LQ 芯片	1104	600
T41NQ	make isvp_t41nq_uart_sfc	编译 uart 启动 uboot，且 uboot 命令行支持 sfc 相关命令，针对 T41NQ 芯片	1104	700
T41XQ	make isvp_t41xq_uart_sfc	编译 uart 启动 uboot，且 uboot 命令行支持 sfc 相关命令，针对 T41XQ 芯片	1104	700
T41L	make isvp_t41l_uart_sfc_lp	编译低性能/低功耗（备选方案）的 uart 启动 uboot，且 uboot 命令行支持 sfc 相关命令，针对 T41L 芯片，支持 5MP/30fps	1008	550
T41N	make isvp_t41n_uart_sfc_lp	编译低性能/低功耗（备选方案）的 uart 启动 uboot，且 uboot 命令行支持 sfc 相关命令，针对 T41N 芯片，支持 5MP/30fps	1008	550
T41LQ	make isvp_t41lq_uart_sfc_lp	编译低性能/低功耗（备选方案）的 uart 启动 uboot，且 uboot 命令行支持 sfc 相关命令，针对 T41LQ 芯片，支持 5MP/30fps	1008	550
T41NQ	make isvp_t41nq_uart_sfc_lp	编译低性能/低功耗（备选方案）的 uart 启动 uboot，且 uboot 命令行支持 sfc 相关命令，针对 T41NQ 芯片，支持 5MP/30fps	1008	550
T41XQ	make isvp_t41xq_uart_	编译低性能/低功耗（备选方案）	1008	550

	sfc_lp	的 uart 启动 uboot，且 uboot 命令行支持 sfc 相关命令，针对 T41XQ 芯片，支持 5MP/30fps		

2. Uboot 配置文件中常见修改点

1) CONFIG_BOOTARGS

主要修改点是内核启动以后的内存配置，分区大小配置。


注意

mem 表示内核启动以后保留内存，rmem 表示预留给 SDK 的内存（包括 ISP 模块的内存），两者相加为芯片真实内存大小，具体大小可参考代码。

2) CONFIG_BOOTCOMMAND

配置 uboot 启动执行的命令。例如：norflash 启动模式下使用 sd 卡启动的命令，"sf probe;sf read 0x80600000 0x40000 0x280000; bootm 0x80600000" 改为 "mmc read 0x80600000 0x1800 0x3000; bootm 0x80600000"。

3) CONFIG_BOOTDELAY

配置 uboot 的等待时间。

4) uboot 中添加密码功能

修改配置文件，修改 isvp_t41.h 中添加如下内容：

#define CONFIG_AUTOBOOT_KEYED // 必配。

#define CONFIG_AUTOBOOT_STOP_STR “123456” //必配，uboot 设置的密码。

#define CONFIG_AUTOBOOT_PROMPT “Press xxx in %d second” // bootdelay,选配，uboot 提示信息。

#define CONFIG_AUTOBOOT_DELAY_STR “linux” //选配，uboot 提示信息代码的具体实现在 common/main.c 中 abortboot_keyed(int bootdelay)；可以根据自己的需要具体改动。

5) SD 卡升级问题

在 isvp_t41.h 中添加 #define CONFIG_AUTO_UPDATE 定义。具体代码实现在 common/cmd_sdupdate.c 中。


注意

- LOAD_ADDR 表示把 SD 卡上的相应内存加载到内存的位置。程序中默认设置为 0x82000000，由于这个地址位于 uboot 的堆上，常见 uboot 的堆大小的设置在 isvp_t41.h 中 CONFIG_SYS_MALLOC_LEN 宏的配置；所以这个地址能够被使用的大小将受到堆大小的限制，还有 uboot 代码中 malloc 空间的限制。
- 需要读取较大文件的时候，可以适当增大 CONFIG_SYS_MALLOC_LEN）

---

6) 编译出的 uboot 大于限制的大小处理方法

uboot 代码中默认限制 spl 部分为 26Kbytes，uboot 部分为 214Kbytes；一共 240Kbytes 大小的限制。如果 uboot 编译生成的 u-boot-with-spl.bin 文件大于 240Kbytes，则无法启动。

解决方案一：

增加 uboot 的限制；修改 isvp_t41.h 中 CONFIG_SYS_MONITOR_LEN 宏的定义；同时修改 CONFIG_BOOTARGS 变量中 boot 分区的大小及以后分区的偏移地址。

解决方案二：

如果生成的 u-boot-with-spl.bin 超出 240Kbytes 较少可以采取压缩 uboot 的方式。在 isvp_t41.h 中 修改#undef CONFIG_SPL_LZOP 为 #define CONFIG_SPL_LZOP；然后重新编译烧录的文件名 u-boot-lzo-with-spl.bin。

7) uboot 网络问题

默认的 isvp 配置是包含以太网部分的代码，如果产品在 uboot 阶段不需要 TFTP 下载或者 NFS 挂载，可以把以太网部分代码裁剪掉，以便减小 uboot。

具体操作：

打开 isvp_t41.h 配置文件，把#define CONFIG_CMD_NET 宏注掉即可。

3.2 Kernel 编译

T41 支持两个版本的内核，分别为 kernel-3.10.14 版本和 kernel-4.4.94 版本。支持两个版本的原因是相同的配置文件两个版本的内核编译出来的内核文件大小不一样。

kernel 可单独编译，不依赖其他代码。以 isvp 板级编译为例，进入 kernel 源码目录。在 arch/mips/configs/ 文件夹下存放了内核的板级配置文件。T41 芯片板级根据 demo 板名称分为：

- Marmot 全功能开发板： isvp_marmot_defconfig
- Tomcat 38 板： isvp_tomcat_defconfig。

下列操作以 38 板为介绍：

第一步：**$make isvp_tomcat_defconfig** 使用相对配置好的板级文件

第二步：**$ make menuconfig** 根据需求选择性编译

第三步：**$ make uImage** 编译内核文件

如果报错，执行 **$ make distclean** ，然后从第一步重新开始。


注意

对于 kernel 之外的 ko 编译，依赖 kernel，且 kernel 必须先编译。每当 kernel 更改之后，可能会出现 ko 无法 insmod 的情况，或者可以 insmod 但会出现未知错误。这是因为 kernel 重新编译后，函数 Symbol 表有变化，需要重新编译 ko driver 以正确 link 函数。

内核默认的 config 留有一定余量，而实际产品往往需要进行内核裁减以释放更多的空间。

以下是几个常用可供裁减的选项及说明：

1) CONFIG_NETWORK_FILESYSTEMS

网络文件系统，一般用来方便开发，但会占用较多空间，也可用 tftp 进行替代。若不支持 NFS，可以 Disable。

2) CONFIG_KALLSYMS &CONFIG_KALLSYMS_ALL

内核函数符号表，会占用较多空间，在 panic 时的函数栈可以显示出函数名。当内核稳定后，可以考虑 Disable 此功能，但建议完全稳定之前使能此功能。

3) 其他文件系统

一般文件系统会占用较多空间，开发者可根据需求对文件系统的选项进行裁减，比如 ext 文件系统等。

4) 和产品定义无关的模块

比如 USB，以太网，TF 卡等等。


注意

内核的深度裁减有一定的技术难度，开发者尽量在深入了解配置的情况下再进行深度裁减。

3.3 T41 需要加载的驱动

驱动名称	驱动介绍
tx-isp-t41.ko	ISP 驱动
sensor_xxxx_t41.ko	Sensor 驱动
avpu.ko	视频编码驱动
sinfo.ko	Sinfo 探测驱动
audio.ko	音频驱动
soc-nna.ko	AI 算法 Nna 驱动
T41-mpsys-4-4-94.ko T41-mpsys-3-10-14.ko	多进程获取 YUV 驱动

4 文件系统

4.1 根文件系统介绍

Linux 的目录结构的最顶层是一个被称为“ / ”的根目录。系统加载 Linux 内核之后，就会挂载一个设备到根目录上。存在于这个设备中的文件系统被称为根文件系统。所有的系统命令、系统配置以及其他文件系统的挂载点都位于这个根文件系统中。

根文件系统通常存放于内存和 Flash 中，或是基于网络的文件系统。根文件系统中存放了嵌入式系统使用的所有应用程序、库以及其他需要用到的服务。下图列出了根文件系统的顶层目录。

```
.                       //根目录
├─── bin                //基本命令的可执行文件
├─── dev                //设备文件
├─── etc                //系统配置文件，包括启动文件
├─── lib                //基本库文件，例如 C 库和内核模块 ko
├─── mnt                //临时文件系统挂载点
├─── opt                //添加的软件包
├─── proc               //内核以及进程信息的虚拟文件系统
├─── root               //root 用户目录
├─── sbin               //系统管理的可执行程序
├─── sys                //系统设备和文件层次结构
├─── system             //可读写系统文件(jffs2),挂载在/dev/mtd3
├─── tmp                //临时文件
├─── usr                //包含一些有用的文件
└─── var                //存放系统日志或一些服务程序的临时文件
```

4.2 文件系统的选择

文件系统有 glibc 与 uclibc 两种选择，glibc 的特点是支持功能全面，但是占用存储稍多；uclibc 的特点是占用存储空间小于 glibc，但是支持功能比 glibc 稍少。开发者应该根据自己实际情况选用适合的 libc 方式。

对于一般的应用场景，推荐客户使用 uclibc 搭建系统。

```
isvp_uclibc_defconfig                --uclibc 默认 config
isvp_uclibc_mini_defconfig           --uclibc 默认裁剪 config
isvp_glibc_defconfig                 --glibc 默认 config
isvp_glibc_mini_defconfig            --glibc 默认裁剪 config
```

对于 NorFlash Based 系统，存储空间一般较小，因此会选用压缩文件系统。推荐以下两种文件系统：

```
A. squashfs：只读文件系统，压缩率高
B. jffs2：可读写文件系统，可选择压缩方式
```

推荐的系统搭建的方案是----系统 rootfs 以及不需要经常修改的系统分区采用 squashfs 文件系统，而配置分区 system 等需要经常读写的分区采用 jffs2 文件系统。

4.3 文件系统的制作

关于文件系统制作的介绍与使用请参见“ T41 文件系统制作指南 ”。

4.4 Demo rootfs 简单说明

ISVP 的 Demo 分区方式为：

```
A. u-boot:256K
B. uImage:2560K
C. rootfs:2048K
D. system:3328K
```

在 ISVP 的文件系统中，有两个指定的文件路径：

```
/etc/sensor/ ——此目录下存放 Sensor 的效果 bin 文件
/etc/webrtc_profile.init ——回音消除参数文件
```

不同的产品以上两处可能会有不同，而且产品升级时也可能升级相关的参数文件，因此建议将以上两处做软链接到可读写或者可更新的路径下。

Demo 的 rootfs 在启动后，会探测 system 分区是否可用，如果不可用，会进行格式化，并创建相关的目录结构。因此，第一次制作 Demo 系统，可以将整片 Flash 擦除，并烧录 u-boot，uImage，rootfs 即可；system 分区会自动创建。

system 分区的结构如下：

```
system
├──── bin    --此目录下放置应用程序
├──── etc
│      └──── sensor    --/etc/sensor 软链接到这里
│              └──── jxf37-t41.bin   --效果文件
├──── init
│      └──── app_init.sh     --开机执行初始化脚本
└──── lib
       ├──── firmware    --/lib/firmware 软链接到这里
       └──── modules     --/lib/modules 软链接到这里
              ├──── sensor_jxf37_t41.ko    --sensor 驱动
              └──── tx-isp-t41.ko     --ISP 驱动
              └──── avpu.ko     --VPU 驱动
              └──── audio.ko     --音频驱动
              └──── sinfo.ko     --sensor 型号探查安装驱动
```

注：Demo rootfs 可作为参考，开发者可以根据产品的实际情况定义 rootfs 方案。

4.5 应用程序编译

4.5.1 编译注意

应用程序编译注意有以下几点：

1. ISVP 的 toolchain 包含了 glibc 和 uclibc，因此基于 glibc 或者 uclibc 的程序均可使用此 toolchan 进行编译。关于 glibc 和 uclibc 编译的区分方法请参考 2.3.2 如何进行 glibc 和 uclibc 编译
   - glibc 程序编译方法：默认 link 的 libc 即为 glibc
   - uclibc 程序编译方法：C_FLAGS+=-muclibc CXX_FLAGS+=-muclibc，LD_FLAGS+=-muclibc
2. 关于 API 库的链接顺序：[IVS 库] [mxu 库] [libimp/libsysutils] [libalog]
3. 由于 libimp 中依赖 C++库，因此需要使用 mips-linux-gnu-g++进行链接，若使用 gcc 链接，需要手动添加 LD_FLAGS+=stdc++。
4. 如何优化 elf 文件的大小：
   - 编译等级选择 O2：C_FLAGS/CXX_FLAGS += -O2
   - DFLAG += -Wl,-gc-sections，不链接不必要的段。
   - elf 文件执行 mips-linux-gnu-strip，链接后删除不必要的段。
5. 若系统中有多个 elf 文件需要链接库文件，可使用动态链接的方式，若只有一

个文件链接库文件，请使用静态链接的方式（注，libimp 相关功能在系统中只能存在一份实例）。在调试时选择动态链接的方式可以方便的进行 debug 及问题反馈。

4.5.2 运行应用程序

要运行编译好的应用程序，首先需要将其添加到目标机中，然后完成以下工作：
将应用程序和需要的库文件（如果有）等添加到目标机的根文件系统相应的目录中。通常将应用程序放到 /bin 目录里，库文件放到 /lib 目录里，配置文件则放到 /etc 目录里。
制作包含新应用程序的根文件系统。


注意
如果执行应用程序，需要读写文件系统。请选择 squashfs、jffs2 文件系统。如果新添加的应用程序需要系统启动后自动运行，请在编写文件系统时，编辑/etc/init.d/rcS 文件，添加需要启动的应用程序路径。

5 系统启动与烧录

5.1 系统启动

一个全新的板子里面什么都没有(准确的说芯片里面有固化的 Bootrom)，只是一个冷冰冰的机器，怎么让这个机器变得有灵魂呢？看下面介绍。

5.1.1 Uboot 启动

5.1.1.1 SD 卡启动

SD 卡启动是一个全新开发板最方便的启动方式。SD 卡插上读卡器就可以在 PC 上操作，使用分区工具把 SD 卡分出一个新分区，把编译好的 uboot 拷贝到这个分区的固定偏移位置上，然后 Bootrom 会从 SD 卡的这个位置进行引导，启动 uboot。（SD 卡启动卡制作查看 7.1 制作启动卡）

5.1.1.2 Nor/Nand 启动

nor/nand 是板子上的存储介质，本质上和 SD 卡里的存储介质类似，但这些存储器是焊接到板子上的，不能直接拿下来操作，而这些闪存是掉电不丢失数据，所以把 uboot 拷贝到 nor/nand 闪存中，重启既不需要再次烧录。拷贝的方法有两种：一是在 uboot 命令行中，把编译好的 uboot.bin 文件先拷贝到内存中，然后通过 flash 烧写命令烧录到 nor/nand flash 中；二是在焊接 flash 前，用烧录器把 uboot.bin 烧录到相应 flash 中。

5.1.1.3 Uart 启动

为了解决板卡贴片空的 flash 或者 EMMC，没有预留卡槽以及 USB 接口，不方便烧录的问题，T41 新增了 UART 启动。T41 UART 通过 YMODEM 协议进行数据传输，支持 UART0/1/2 三个接口通信，通信的波特率为常用的 115200 和 9600。（操作步骤查看 7.2 uart 启动操作方法）

5.2 系统烧录

5.2.1 TFTP 传输与烧录

(1) 目标

将文件通过 tftp 方式，从 PC 端，下载到 Uboot 的内存中，然后写到 flash 中。

(2) 前提

硬件：开发板上网卡；板子通过网线连接到路由器或者交换机上，PC 也连到该路由器或者交换机上，并且处于同一网段上；

软件：PC 端安装并设置好 TFTP 服务，把相应的 u-boot 等文件放到 TFTP 根目录下；uboot 中支持网卡驱动并且支持相应的 TFTP 操作。

(3) 操作

在 uboot 中，执行 $ tftpboot mem_addr file_name ；可以将文件 file_name 传送到 Uboot 的内存地址 mem_addr 中。

(4) 传输实例

通过 tftp 命令 load 文件到内存；

```
$ tftpboot 0x80600000 u-boot-with-spl.bin
$ tftpboot 0x80640000 uImage
$ tftpboot 0x808c0000 root-uclibc-toolchain720.squashfs
```

分别把 u-boot-with-spl.bin、uImage、root-uclibc-toolchain720.squashfs 下载到内存的 0x80600000、0x80640000、0x808c0000 处。


注意

根据实际文件大小来修改偏移位置。

(5) 烧录到 Nor

把下载到内存上的文件，写到开发板上的 nor flash 上，重启后可以从 nor 上直接读取文件，而不必每次手动 load 文件到内存上。烧录命令如下：

```
$ sf probe;sf erase 0x0 0x1000000;sf write 0x80600000 0x0 0x100
0000
```

- sf probe 在使用 sf read、sf write 之前,一定要调用 sf probe ；
- sf erase 擦除指定位置，指定长度的 flash 内容，擦除后内容全 1；
- sf write 写内存数据到 flash 上。

5.2.2 SD 卡传输与烧录

(1) 目标

将某个文件写到内存中，然后写到 flash 上。

(2) 前提

硬件：开发板上有 SD 卡槽；有读卡器把 SD 卡插到 PC 上拷贝数据。

软件：uboot 中支持 SD 卡驱动。

(3) 操作

在 uboot 中，执行 $ fatls mmc 0 查看 SD 卡中的文件，

$ fatload mmc 0 mem_addr file_name 可以将文件 file_name 传送到 Uboot 的内存地址 mem_addr 中了。

(4) 传输实例

通过 fatload 命令 load 文件到内存；

```
$ fatload mmc 0 0x80600000 u-boot-with-spl.bin
$ fatload mmc 0 0x80640000 uImage
$ fatload mmc 0 0x808c0000 root-uclibc-toolchain720.squashfs
```

分别把 u-boot-with-spl.bin、uImage、root-uclibc-toolchain720.squashfs 下载到内存的 0x80600000、0x80640000、0x808c0000 处。


注意

根据实际文件大小来修改偏移位置。

(5) 烧录到 Nor

把下载到内存上的文件，写到开发板上的 nor flash 上，重启后可以从 nor 上直接读取文件，而不必每次手动 load 文件到内存。烧录命令如下：

```
$ sf probe;sf erase 0x0 0x1000000;sf write 0x80600000 0x0 0x1000000
```

- sf probe 在使用 sf read、sf write 之前，一定要调用 sf probe ；
- sf erase 擦除指定位置，指定长度的 flash 内容，擦除后内容全 1；
- sf write 写内存数据到 flash 上。

5.2.3 USB 烧录

1) 目标

将某个文件写到 flash 上。

2) 前提

硬件：开发板上有 USB 烧录接口；数据线。

软件： USB 烧录 PC 工具。

3) 操作

参考“ USBCloner 烧录指南.pdf ”。

5.3 串口连接

开发板串口负责交互信息传输，内核打印会通过串口发送到显示屏上，开发板通过串口接收键盘的输入信息。常用的串口工具有 minicom (linux)、putty、Xshell 等软件。

PC 端串口与开发板串口连接需要配置对应的参数，如串口号，波特率，奇偶校验位，数据位，停止位等。比如配置串口号为 com6，波特率为 115200，8 位数据位，一位停止位，无奇偶校验位。

5.4 Uboot 配置

uboot 最终目的是启动加载内核。内核的启动需要 uboot 的引导并把相应的环境变量传到 kernel，此外还需要挂载一个根文件系统，存放内核配置信息和执行命令操作。

5.4.1 Uboot 常用命令

- reset：重新启动嵌入式系统。
- printenv：打印当前系统环境变量。
- setenv：设置环境变量，格式：setenv name value，表示将 name 变量设置成 value 值；如果没有这个参数，表示删除该变量。
- saveenv：保存环境变量到 flash 中。
- sleep：延迟执行，格式：sleep N，可以延迟 N 秒钟执行。
- run：执行环境变量中的命令，格式：run var，可以跟几个环境变量名。
- cp：复制内存中数据块，格式：cp source target count，第一个参数是源地址，

第二个参数是目的地址，第三个参数是复制数目。

- tftpboot：通过 tftp 协议下载文件到内存，格式 tftpboot mem_addr file_name。
- fatls：显示 SD 卡的文件，格式 fatls mmc 0 。
- bootm：引导启动存储在内存中的程序映像。格式：bootm addr1 addr2，第一个参数是程序映像的地址，第二个参数一般是 ramdisk 地址。

详细介绍可以在 uboot 命令行输入 help 查看。

5.4.2 Uboot 环境变量

板子上电启动，进入 uboot 命令行（读秒时快速多次按下 Enter 键）；通过 uboot 命令 printenv 可以查看 uboot 的环境变量：

```
isvp_t41# printenv
bootargs=console=ttyS1,115200n8 64M@0x0 rmem=64M@0x4000000 init=/linuxr
c rootfstype=squashfs root=/dev/mtdblock2 rw mtdparts=jz_sfc:256k(boot),256
0k(kernel),2048k(root),-(appfs)
bootcmd=sf0 probe;sf0 read 0x80600000 0x40000 0x280000; bootm 0x8060000
0
bootdelay=1
ipaddr=193.169.4.151
serverip=193.169.4.2
```

- bootargs 内核通过 bootargs 找到文件系统。
- console=ttyS1,115200n8 设置串口号为 ttyS1（uart1），波特率为 115200。
- mem=64M@0x0 rmem=64M@0x4000000 分配内存。
- init=/linuxrc 系统首先运行/linuxrc 文件。
- rootfstype=squashfs 使用压缩只读文件系统 squashfs。
- root=/dev/mtdblock2 rw 内核根文件从 mtdblock2 挂载。
- mtdparts=jz_sfc:256k(boot),2560k(kernel),2048k(root),-(appfs) mtd 分区情况。
- bootcmd 启动命令。
- sf0 probe;sf0 read 0x80600000 0x40000 0x280000; bootm 0x80600000 从 nor 启。
- ipaddr 193.169.4.151 开发板 IP 地址。
- serverip=193.169.4.2 服务器 IP 地址。

5.4.3 更改 mtd 分区大小

MTD 分区大小，在 uboot 中以命令行参数的形式传入 kernel，配置文件在 include/configs/isvp_t41.h 中，对于 8M Flash，找到 CONFIG_SFC_NOR，参考平台定义其中：

```
mtdparts=jz_sfc:256k(boot),2560k(kernel),2048k(rootfs),-(appfs)
```

MTD 的分区情况，共 4 块分区，uboot 256k，kernel 2.5M，rootfs(squashfs)2M, -(appfs)具有自适应功能，当 Flash 为 8M 时，appfs 分区会自动分配成 3.25M；当 Flash 为 16M 时会自动分配成 11.25M。如果想修改，请保持总大小不变的情况下，按如上格式修改。一般 uboot、kernel、rootfs 大小固定不变，建议用户不要修改，可以调整 appfs 分区大小。

5.4.4 更改 rmem 内存大小

rmem 大小同样是在 include/configs/isvp_t41.h 中修改，参考平台定义如下

```
#define BOOTARGS_COMMON "console=ttyS1,115200n8 mem=32M@0x0 rmem=32M@0x2000000
```

- mem=32M@0x0 rmem=32M@0x2000000 对 mem 大小的相关设置。
- mem=32M@0x0 从 0 地址开始给系统分配 32M 内存。
- rmem=32M@0x2000000 从 0x2000000（32M） 地址开始给 SDK 分配 32M。

T41 内嵌一块 64M/128M/256M 的内存，如要修改 rmem，请在总大小不变情况下按格式进行修改。示例：

如：以 64M 内存为基准，假设 rmem 减小 4M，mem 增大 4M；如果想减小 rmem，那么 mem 应该增大。所以 mem 大小为 36M，从 0 地址开始。rmem 为 28M，地址从 36M 开始。

```
#define BOOTARGS_COMMON "console=ttyS1,115200n8 mem=36M@0x0 rmem=28M@0x2400000"
```

6 系统资源使用与调试

6.1 ISP_Sensor

T41 芯片内部包含一个 VIC（视频输入控制器）和一个 ISP（图像处理模块）；VIC 支持 DVP 和 CSI 两种接口，DVP 支持 8bit、10bit 和 12bit 并行传输，CSI 通过 CSI_PHY 接入，可以支持一个 2lane 的输入。

6.1.1 ISP 驱动

1) 进入驱动文件夹/opensource/drivers/isp-t41/tx-isp-t41；首先修改 Makefile 中 ISVP_ENV_KERNEL_DIR 宏定义，使之能够索引到正确的 kernel 路径。

2) 然后执行 make clean; make

3) 最后生成的 tx-isp-t41.ko 拷贝到系统中。

4) ISP 驱动注册时提供多个 module_param 参数。

A. isp_clk 参数，设置 ISP 频率

```
$ insmod tx-isp-t41.ko isp_clk=300000000
```

B. direct_mode 参数，设置 IVDC 模式；0 为非直通，1 为直通，2 为半直通

```
$ insmod tx-isp-t41.ko direct_mode=0
```

C. ivdc_mem_line 参数，配置 IVDC 中 DDR 存储的行数

```
$ insmod tx-isp-t41.ko direct_mode=1 ivdc_mem_line=height / 2
```

D. ivdc_threshold_line 参数，配置 IVDC 编码缓存的阈值（单位：行数 width，256 对齐）； 0：配置为 height/2，表示阈值为半帧（默认值）， 大于 0：表示缓存为 width * ivdc_threshold_line

```
$ insmod tx-isp-t41.ko direct_mode=1 ivdc_threshold_line=heigh
t / 2
```

6.1.2 Sensor 驱动

1) sensor 驱动依赖于 kernel 和 ISP 驱动，所以在编译 sensor 驱动之前 kernel 和 I

SP 驱动必须已经编译完成。

2) 进入具体的 sensor 驱动文件夹，首先修改 Makefile 中 ISVP_ENV_KERNEL_DIR 和 ISP_DRIVER_DIR 宏定义，使之能够索引到正确的 kernel 和 ISP 驱动路径。

3) 然后执行 make clean;make

4) 最后将生成的 sensor_xxx_t41.ko 拷贝到系统中。

5) sensor 驱动加载也依赖 ISP 驱动，所以加载 sensor 驱动需先加载 ISP 驱动。

```
$ insmod sensor_xx_t41.ko
```

6.1.3 Avpu 视频编码驱动

1) 进入驱动文件夹/opensource/drivers/avpu/；首先修改 Makefile 中 ISVP_ENV_KERNEL_DIR 宏定义，使之能够索引到正确的 kernel 路径，视频编码驱动只依赖内核，所以在编译 avpu 驱动之前 kernel 必须已经编译完成。

2) 然后执行 make clean;make

3) 最后生成的 avpu.ko 拷贝到系统中。

4) avpu 驱动注册时提供了多个 module_param 可配置参数，例如：

```
$ insmod avpu.ko clk_name='vpll' avpu_clk=400000000
```

其中 clk_name 是 vpu 选择使用的时钟源，avpu_clk 用于设置 avpu 的时钟频率。


注意

如果产品硬件遵循参考设计 insmod 时无需跟参数，使用默认值即可。

6.1.4 图像效果文件

不同的 Sensor 以及镜头可能需要不同的效果参数配置，配置文件位置为：/etc/sensor 目录下，文件名为[sensor]-t41.bin，例如：jxf37-t41.bin。实际产品中效果 bin 文件往往需要随版本迭代更新，因此/etc/sensor 目录需要有读写权限。可供参考的方法之一是将/etc/sensor 目录做成一个软链接，链接到一个 rw 的分区中，这样就可以在版本更新时单独更新 bin 文件。


注意

如果没有这个文件图像颜色等可能不正常。

6.1.5 Sample 的使用

Sample 程序位于/samples/libimp-samples，里面是依赖 SDK 库的应用程序，包括图片的抓取、图片格式的转换、智能检测等应用程序。设置好 Makefile，直接在 sdk 目录下 make 即可以编译。用户可以参考提供的 sample 程序，编写相应的工程代码。

完成编译后，可以在目录下找到对应的可执行程序；拷贝到开发板上测试即可。详细操作查看下表：

表 6-1 sample 功能

应用文件	功能	执行命令	执行结果
sample-Framesource.c	保存对应格式的图片	./sample-Framesource	会在 /tmp 下生成对应格式的图片
sample-Encoder-h265-ivpu-jpeg.c	保存 H265 码流和抓取图片	./sample-Encoder-h265-ivpu-jpeg	会在/tmp 下生成 h265 码流和 Jpeg 图片
sample-Encoder-jpeg.c	把 NV12 图片编码成 jpeg	./sample-Encoder-jpeg	把 NV12 图片编码成 jpeg 图片，并保存在/tmp 下
sample-Encoder-video.c	获取一段对应格式的视频流	./sample-Encoder-video	会在 /tmp 下产生对应编码格式的视频
sample-Encoder-video-direct.c	主码流开启直通功能，并获取直通后的 h265 码流	./sample-Encoder-video-direct	会在 /tmp 下产生 h265 的视频
sample-Encoder-jpeg-ivpu.c	抓取 helix jpeg 图片	./sample-Encoder-jpeg-ivpu	会在/tmp 下生成 jpeg 图片
sample-Snap-YUV.c	抓取 yuv 图片	./sample-Snap-YUV	会在 /tmp 下生成 yuv 文件
sample-Snap-Raw.c	抓取 raw 图片	./sample-Snap-Raw	会在 /tmp 下生成 raw 文件
sample-OSD.c	在视频上叠加图片	./sample-OSD	会在 /tmp 下产生抓取的一张带时间戳的图

			片
sample-ISP-flip.c	图像的镜像翻转	./sample-ISP-flip	会在 /tmp 下生成镜像翻转后的视频
sample-Setfps.c	设置帧率	./sample-Setfps	设置帧率
sample-AutoZoom.c	AF 功能种实现某一部分图片放大	./sample-AutoZoom	会在/tmp 下生成分辨率变化的使用
sample-Change-Resolution.c	改变分辨率流程	./sample-Change-Resolution.	会在/tmp 生成多种分辨率 h265 视频

6.2 Audio

音频分为内部 codec，外部 codec，数字 dmic 三部分。

6.2.1 Audio 驱动

驱动代码位于/opensource/drivers/audio/oss3；在编译前需要确认 Makefile 中的 kernel 路径正确性。驱动参数使用如下。

1) Audio 驱动加载时提供了参数 dmic_gpio 用于配置是否使用 dmic，驱动默认配置的是-1（即不使用 dmic）；如果需要使用 dmic 则配置该参数为 dmic_gpio=1。

```
使用 dmic:
# insmod audio.ko  dmic_gpio=1
```

2) Audio 驱动加载时提供了参数 i2c_bus 用于配置外部 codec 所使用的 I2C 控制器，驱动默认配置的是 1（即使用 I2C1）；如果需要使用其他 I2C 则配置参数 i2c_bus。

```
使用 I2C0:
# insmod audio.ko  i2c_bus=0
```

3) Audio 驱动加载时提供了参数 excodec_name 用于配置外部 codec 标识，驱动默认配置的是 0；如果需要使用外部 codec 则需要配置参数 excodec_name（此标识与外部 codec 的 i2c 驱动中的 driver name 是一致的）。

```
使用外部 codec es8374:
# insmod audio.ko  excodec_name=es8374
```

4) Audio 驱动加载时提供了参数 excodec_addr 用于配置外部 codec 的 I2C 地址，驱动默认配置的是 0xFF（即不使用外部 codec）；如果需要使用外部 codec 则需要配置参数 excodec_addr。

```
配置外部 codec I2C 地址为 0x17：
# insmod audio.ko  excodec_addr=0x17
```

5) Audio 驱动加载时提供了参数 dmic_gpio 用于选择 dmic 的 GPIO pin，驱动默认配置的是 PA14，PA16，PA17；如果要使用 PB28，PB29，PB30 作为 DMIC 的功能引脚，需要使用参数 dmic_gpio 来配置。

```
使用 PB 组 GPIO 作为 DMIC 的功能引脚：
# insmod audio.ko  dmic_gpio=1
```

6) Audio 驱动加载时提供了参数 spk_gpio 用于选择内部 codec 的功放 GPIO 引脚，驱动默认配置的是 PB31；如果要使用其他引脚，需要使用参数 spk_gpio 来配置。

```
使用 PA（10）做为功放口：
GPIO_PA(n)  = (0*32 + n)
GPIO_PB(n)  = (1*32 + n)
GPIO_PC(n)  = (2*32 + n)
GPIO_PD(n)  = (3*32 + n)
# insmod audio.ko  spk_gpio=10
```

7) Audio 驱动加载时提供了参数 spk_level 用于选择内部 codec 功放引脚使能电平，驱动默认配置的是高电平使能；如果要使用低电平使能或者不需要驱动内部控制这个 GPIO，需要使用参数 spk_level 来配置（内部 codec，外部 codec 通用）。

```
低电平使能功放：
# insmod audio.ko  spk_level=0
不使用功放 GPIO：
# insmod audio.ko spk_level=-1
```

8) Audio 驱动加载时提供了参数 LEFT_MIC_GAIN 用于配置内部 codec 左声道的 mic 增益，驱动默认配置的是 20db；如果想要修改 mic 增益值，需要使用参数 LEFT_MIC_GAIN 来配置。

```
Mic gain 共有三级增益：LEFT_MIC_GAIN=0（对应 0db），LEFT_MIC_GAIN=1（对应 6db），LEFT_MIC_GAIN=2（对应 20db），LEFT_MIC_GAIN=3（对应 30db）
配置左声道 mic 增益为 30db：
# insmod audio.ko  LEFT_MIC_GAIN=3
```

9) Audio 驱动加载时提供了参数 gpio_func 用于选择 I2S 的 GPIO pin，驱动默认没有进行配置；如果要使用外部 CODEC，需要使用参数 gpio_func 来配置。

```
使用 PC 组 GPIO：
# insmod audio.ko  gpio_func=2,1,0xFF0000
```

6.2.2 Sample 使用

Sample 程序位于/samples/libimp-samples，里面是依赖 SDK 库的应用程序，包括

录音放音、回应消除等应用程序。设置好 Makefile，直接在 sdk 目录下 make 即可以编译。用户可以参考提供的 sample 程序，编写自己相应工程代码。

完成编译后，可以在目录下找到对应的可执行程序；拷贝到开发板上测试即可。详细操作查看下表：

表 6-2 音频 sample 介绍

应用文件	功能	执行命令	执行结果
sample-Ai.c	模拟 mic 录音	./sample-Ai	生成录音文件 ai_record.pcm
sample-Ao.c	音频文件播放	./sample-Ao	播放采样率 16K 的音频文件 ao_paly.pcm
sample-Ai-AEC.c	音频回声消除	./sample-Ai-AEC	程序运行后，生成了 test_for_play.pcm 和 test_aec_record.pcm 两个录音文件
sample-Ai-Ref.c	验证模拟 mic 音频获取回声消除算法参考帧	./sample-Ai-Ref	程序运行后，生成了 test_for_play.pcm 和 test_aec_record.pcm 两个录音文件
sample-dmic.c	数字麦克阵列录音	./sample-dmic	程序运行后，录音生成 dmic0_record.pcm，dmic1_record.pcm，分别对应麦克阵列中的 MIC0，MIC1 录制的声音数据

6.3 GPIO

GPIO 是“General Purpose Input/Output”的简称，具有输入输出，功能复用的功能。开发者在开发硬件驱动时往往需要操作 GPIO，以控制外设硬件。

ISVP 使用 Linux 标准的 GPIOLIB 接口。GPIOLIB 提供了统一的 GPIO 申请/释放/设置/获取接口，按照 GPIOLIB 的设定，需要在 Kernel space 进行调用。如果是 User space 需要操作 GPIO，有两种方法可以选择：

A. 通过 sysfs GPIO 接口进行操作。

B. 应用程序调用相关的驱动，驱动中实现 GPIO 的设置。

6.3.1 头文件及 API

GPIOLIB 的头文件为：include/linux/gpio.h

在驱动程序中加入头文件引用：

```
#include <linux/gpio.h>
```

API 在头文件 include/asm-generic/gpio.h 中定义，例如：

```
int gpio_request(unsigned gpio, const char *label);
void gpio_free(unsigned gpio);
int gpio_direction_input(unsigned gpio);
int gpio_direction_output(unsigned gpio, int value);
int gpio_get_value(unsigned int gpio);
void gpio_set_value(unsigned int gpio, int value);
int gpio_to_irq(unsigned int gpio);
```

详细的文档说明可参考 kernel/Documentation/gpio.txt; 参考代码 Linux kernel 中标准驱动的 GPIO 操作均使用标准的 GPIOLIB，比如 Software I2C，Fixed regulator 以及中断等。

注意

如果 gpio 已被使用，gpio_request()是无法申请到的，所以在使用 gpio 前，应先通过 gpio_request()函数的返回值判断是否成功申请到 gpio。

6.3.2 sysfs GPIO

sysfs GPIO 是 Linux 标准的用户空间操作 GPIO 的接口。用户可通过命令行或者应用程序直接设置 GPIO 的输入/输出，高低电平等属性。一般情况下，GPIO 调试或者简单的 GPIO 应用（比如 IR-Cut 操作），可通过 sysfs GPIO 接口进行快速开发。

6.3.2.1 内核选项

在内核源码根目录下执行$ make menuconfig 命令进入配置界面，选中以下选项：

```
Device Drivers  --->
     -*- GPIO Support  --->
         [*]   /sys/class/gpio/... (sysfs interface)
```

一般情况下，内核的默认配置已经勾选了此选项。

6.3.2.2 sysfs GPIO 的申请与释放

在操作 sysfs GPIO 之前需要对其进行申请。值得注意的是，由于申请 sysfs GPIO 会在内核 request_gpio，因此在内核中已经申请过的 GPIO 在 sysfs GPIO 再次申请会失败。

申请/释放 GPIO 方法如下：

```
$ cd /sys/class/gpio
$ echo [gpio_num] > export          #申请 GPIO
$ echo [gpio_num] > unexport        #释放 GPIO
```

注：gpio_num 即 GPIO 号。计算公式为：

PA(n) = 0 * 32 + n

PB(n) = 1 * 32 + n

PC(n) = 2 * 32 + n

...

例如：申请 PB(10) = 1 * 32 + 10 = 42

```
$ echo 42 > export
```

申请后在/sys/class/gpio 目录下即会出现 gpio42 目录。

```
$ echo 42 > unexport
```

释放后 gpio42 目录也会消失。释放后的 GPIO 状态并不会恢复，会保持申请时的状态（电平等）。

6.3.2.3 设置输入/输出方式

在申请 GPIO 后，进入 gpioN 目录，例如 gpio42，进行如下操作：

```
$ echo out > direction              #设置 PB10 为输出模式
$ echo in > direction               #设置 PB10 为输入模式
```

6.3.2.4 设置有效电平

gpioN 目录下有 active_low 节点，表示当前 GPIO 的有限电平，默认为 0，其意义为，当输入/输出 value 为 0 时，GPIO 为低电平，当输入/输出 value 为 1 时，GPIO 为高电平。同样的，当 active_low 为 1 时：输入/输出 value 为 0，GPIO 为高电平；输入/输出 value 为 1，GPIO 为低电平。

也就是说，GPIO 的真实电平=value^active_low。

```
$ echo 0 > active_low    #value 是 0,表示低电平；value 是 1,表示高电平
$ echo 1 > active_low    #value 是 1,表示低电平；value 是 0,表示高电平
```

6.3.2.5 输入/输出

gpioN 目录下有 value 节点，表示 gpioN 的电平：当 GPIO 为输入模式时，读取到 value 的值异或 active_low 即为 GPIO 的电平；当 GPIO 为输出模式时，写入到 value 的值异或 active_low 即为 GPIO 的输出电平。

```
$ cat value                #读取电平（输入模式）
$ echo 0 > value           #设置电平（输出模式）
```

6.3.2.6 GPIO 冲突

在不加载任何驱动的情况下，T41 kernel make isvp_marmot_defconfig 默认配置已占用 GPIO 如下表。

表 6-3 默认配置占用 GPIO 情况表

GPIO	功能	取消占用方法
PB25	I2C0_SDA	EEPROM（产品很少用，该功能口可用于其他用处） 参考 6.9 I2C 章节，取消 I2C0 功能。
PB26	I2C0_SCK	
PA23	SFC0_DT	四线 SFC0 外接 SPI-Flash
PA24	SFC0_DR	
PA25	SFC0_GPC	
PA26	SFC0_CE1_	
PA27	SFC0_CLK	
PA28	SFC0_CE0_	
PA20	SFC1_DT	四线 SFC1 外接 SPI-Flash
PA21	SFC1_DR	
PA22	SFC1_CLK	
PA29	SFC1_CE0_	
PA30	SFC1_CE1_	
PA31	SFC1_GPC	
PB00	MSC_D0	参考 6.10 SD/EMMC 章节。
PB01	MSC_D1	
PB02	MSC_D2	
PB03	MSC_D3	
PB04	MSC_CLK	

PB05	MSC_CMD	
PB27	MSC-DETECT	
PB06	GMAC_CLKIN	参考 6.13 GMAC 章节，外置 ETH PHY RMII 总线信号
PB07	GMAC_CLK_OUT	
PB08	GMAC_TXEN	
PB09	GMAC_RXDV	
PB10	GMAC_MDC	
PB11	GMAC_MDIO	
PB13	GMAC_TXD0	
PB14	GMAC_TXD1	
PB15	GMAC_RXD0	
PB16	GMAC_RXD1	
PB23	UART1_TXD	Debug uart1 参考 6.4 uart 章节。
PB24	UART1_RXD	
PB27	SD 卡检测	设备树 marmot.dts 文件中修改，msc0 节点，ingenic,cd-gpios
PB28	外置 PHY 复位信号	在 uboot isvp_t41.h 文件中修改（CONFIG_GPIO_PHY_RESET）；设备树 marmot.dts 文件中修改，mac0 节点，ingenic,rst-gpio

上表格中没有的 gpio 如果发生冲突，可能是加载的驱动，kernel 有修改不是我司原版 kernel 等原因造成的。

6.4 UART

T41 支持 6 个 UART，仅 UART0 和 UART2 支持硬件流控。默认使用 UART1 进行 debug。

6.4.1 UART 配置

- **Kernel-3.10.14**

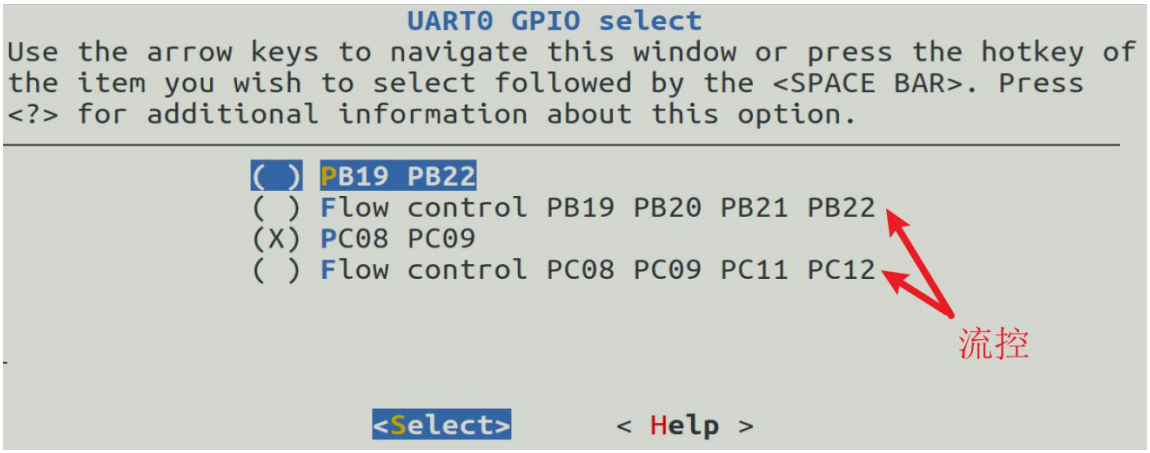
  打开 make menuconfig 配置界面，选中需要使用的 UART。

```
Device Drivers > Character devices > Serial drivers
<*> ingenic T41 serial port support
```

```
      [*]   Console on T41 and compatible serial port
      [*]   enable uart0
              UART0 GPIO select (PC08 PC09)  --->
      [*]   enable uart1
              UART1 GPIO select (PB23 PB24)  --->
      [ ]   enable uart2
      [ ]   enable uart3
      [ ]   enable uart4
      [ ]   enable uart5
```

如下图所示，进入“UART0 GPIO select”选项，可以配置 UART 使用的 GPIO 组。其中“Flow control”是对硬件流控的支持。

图 6-1 UART GPIO 配置图



注意：这里如果选中“Flow control”组 GPIO，仅仅是配置了硬件流控所使用的 GPIO，如果要使用硬件流控功能，还需要在 UART 应用程序中配置硬件流控功能。

- **Kernel-4.4.94**

  打开 marmot.dts 设备树文件，配置需要使用的 UART。

  将需要使用的 UART 节点的 status 属性的值改为"okay" 即可。pinctrl-0 属性用来配置 UART 所使用的 GPIO，可选的 gpio 组可在 t41-pinctrl.dtsi 文件查看。

```
&uart0 {
    status = "okay";
    pinctrl-names = "default";
    pinctrl-0 = <&uart0_pc>;
};
```

6.4.2 修改 Debug uart

uboot 阶段使用的 Debug uart，在 include/configs/isvp_t41.h 文件中通过 CONFIG_SYS_UART_INDEX 宏指定。

kernel 阶段使用的 Debug uart，在 BOOTARGS_COMMON 中“console”字段指定，如下：

```
BOOTARGS_COMMON "console=ttyS1,115200n8 mem=64M@0x0 rmem=64M@0x4000000"
```

“console=ttySn”决定 kernel 阶段 Debug 所用串口，参考 6.4.1 UART 配置 章节配置对应 UART。

6.5 Watchdog

1) 内核配置

在内核源码根目录下执行 make menuconfig 命令进入配置界面，按下面方式配置

```
Device Drivers → Watchdog Timer Support
<*>   Ingenic ingenic SoC hardware watchdog
```

2) 应用例程参考

如果客户使用 Watchdog，请参考 /samples/libsysutils-samples/sample-wdt.c

在关闭看门狗时，需要先调用 wdt_disable()，然后再调用 close()关闭看门狗；不然会导致关不掉从而出现错误！

6.6 ADC

1) 内核配置

在内核源码根目录下执行 make menuconfig 命令进入配置界面，按下面方式配置

```
Device Drivers  --->
    Multifunction device drivers  --->
        <*> Support for the XBurst SADC core
        <*> Support for the XBurst SADC AUX
```

2) 应用例程参考

参考代码在 sample_adc.c

T41 ADC 的基准电压为 1.8V。

ADC 的使能在 kernel-4.4.94/arch/mips/boot/dts/ingenic/marmot.dts 中修改。

注意

参考代码中 ADC_PATH 表示选择哪个 ADC；STD_VAL_VOLATAGE 表示参考电压，单位为 mv！

6.7 PWM

T41 共有 8 路 PWM，打开 make menuconfig 配置界面，选择需要使用的 PWM 通道，并配置对应的 GPIO。

1) 内核配置

在内核源码根目录下执行 make menuconfig 命令进入配置界面

```
Device Drivers → Pulse-Width Modulation (PWM) Support
  --- Pulse-Width Modulation (PWM) Support
  <*>   Ingenic PWM support
  [*]     PWM channel0 Enable
            PWM channel0 GPIO select (PWM channel0 PC15)  --->
  [*]     PWM channel1 Enable
            PWM channel1 GPIO select (PWM channel1 PC16)  --->
  [ ]     PWM channel2 Enable
  [ ]     PWM channel3 Enable
  [ ]     PWM channel4 Enable
  [ ]     PWM channel5 Enable
  [ ]     PWM channel6 Enable
  [ ]     PWM channel7 Enable
```

导出 PWM 通道 N：

```
echo N > /sys/class/pwm/pwmchip0/export
```

取消导出的 PWM 通道 N

```
echo N > /sys/class/pwm/pwmchip0/unexport
```

控制 PWM

导出 PWM 通道 N 后，进入/sys/class/pwm/pwmchip0/pwm*N* 目录，可以看到一些属性文件，通过他们来控制 PWM 输出。

enable：可读可写，写入"0"表示禁止 PWM；写入"1"表示使能 PWM。读取该文

件获取 PWM 当前是禁止还是使能状态。

```
echo 0 > enable  #禁止 PWM 输出
echo 1 > enable  #使能 PWM 输出
```

polarity：用于设置极性，可读可写，可写入的值如下：

```
echo normal > polarity #默认极性
echo inversed > polarity #极性反转
```

period：用于配置 PWM 周期，可读可写。以 ns 为单位。

配置 PWM 周期为 10us：

```
echo 10000 > period
```

duty_cycle：用于配置 PWM 的占空比时间，可读可写。以 ns 为单位，

配置 PWM 占空比为 5us：

```
echo 5000 > duty_cycle
```

2) 驱动参考

参考代码位于/opensource/drivers/misc/sample_pwm，具体使用方法参考驱动目录下的 README。

6.8 LCD

1) 内核配置

```
Device Drivers  --->
    <*>  Multimedia support  --->
         Graphics support  --->
             <*> Support for frame buffer devices  --->
```

2) 驱动编译

驱动代码位于/opensource/drivers/sample_slcd，在编译前需要确认 Makefile 中的 kernel 路径正确性且 lcd_device 目录中的初始化 GPIO 需要和硬件对应。

A. /lcd-device_truly240320.c

```
#define GPIO_LCD_CS          GPIO_PB(20)      //slcd 片选
#define GPIO_LCD_RD          GPIO_PB(19)      //读数据使能(未使用)
#define GPIO_BL_PWR_EN       GPIO_PC(20)      //背光使能
#define GPIO_LCD_RST         GPIO_PB(22)      //slcd 复位
```

B. /lcd-driver_truly240320.c

```
jzgpio_set_func(GPIO_PORT_B, GPIO_FUNC_3, 0x3f<<6 | 0x3<<13 | 0x3<<15 |
0x7<<20);           //设置 slcd 控制器的功能引脚
```

6.9 I2C

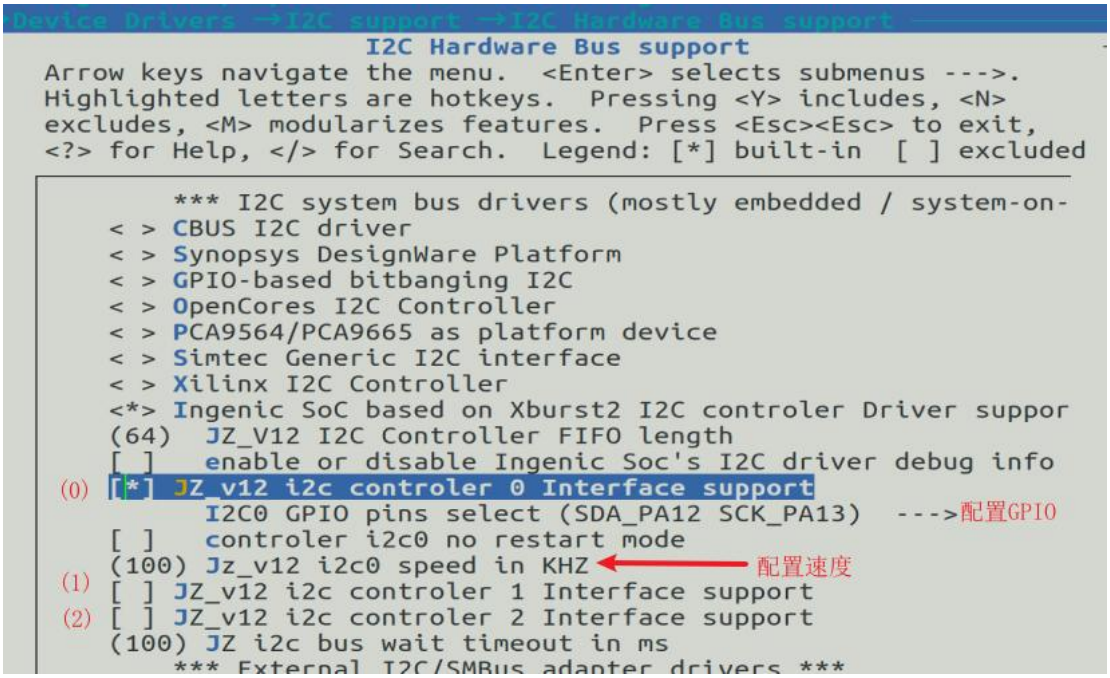
T41 共有三个 I2C。根据需求选择硬件 i2c 或者软件 i2c；硬件 i2c 是确定的引脚，软件 i2c gpio 可以在 arch/mips/boot/dts/ingenic/marmot.dts 中修改。

设备驱动参考：在/opensource/drivers/i2c/sample_eeprom 目录下有使用内核 i2c 驱动的 at24.c 例子，在编译前需要确认 Makefile 中的 kernel 路径正确性。可以实现 i2c 读写内存。

- **Kernel-3.10.14**

  打开 make menuconfig 配置界面，参照下图配置需要使用的 I2C。

图 6-2 I2C 配置图



- **Kernel-4.4.94**

  打开 marmot.dts 设备树文件，配置需要使用的 I2C。

  将需要使用的 I2C 节点的 status 属性的值改为"okay"即可。

  pinctrl-0 属性用来配置 I2C 所使用的 GPIO，可选的 gpio 组可在 t41-pinctrl.dtsi 文件查看。

  clock-frequency 属性用来配置 I2C 的速度.

```
&i2c0 {
    pinctrl-0 = <&i2c0_pa>;
```

```
    clock-frequency = <100000>;
    pinctrl-names = "default";
    status = "okay";
};
```

6.10 SD/EMMC

1) 内核配置

在内核源码根目录下执行 make menuconfig 命令进入配置界面，

按下面方式配置

```
Device Drivers  --->
<*> MMC/SD/SDIO card support  --->
<*>   Ingenic(XBurst2)  MMC/SD Card Controller(MSC) support
```

2) 使用说明

内核默认只支持 fat32 的 SD 卡文件系统，如果使用的卡不是 fat32，可在内核中选上对应的文件系统，或者把卡格式化成 fat32 来使用。

- Ext4 文件系统

在内核按如下配置即可

```
 1、
    [*] Enable the block layer  --->
    [*]   Support for large (2TB+) block devices and files
2、
        File systems  --->
    <*> The Extended 4 (ext4) filesystem
    [*]   Use ext4 for ext2/ext3 file systems (NEW)
```

- Exfat 文件系统

Windows 上超过 32G 的卡不支持格式 fat32，在 Windows 上只能格式化成 exfat。但 Linux 内核默认不支持 exfat 文件系统，如果客户需要支持 exfat 文件系统，编译加载 opensource/drivers/misc/exfat-nofuse 下面的驱动即可，内核使用默认的就可以，不需要额外的内核配置。不提供设备上格式化 exfat 的工具，格式化在 PC 上操作。

3) mdev 自动挂载

Linux 已经支持探测热插拔设备，但需要确保选上相应功能：

- kernel 务必要支持 mdev 和 hotplug 功能。（默认开启）
- busybox 需要选上 mdev 功能。（默认开启）

第一步：创建设备节点

添加热插拔功能，命令内核在增删设备时执行 /sbin/mdev

```
# echo /sbin/mdev > /proc/sys/kernel/hotplug
```

设置 mdev，让它在系统启动时创建所有的设备节点

```
# mdev -s
```

第二步：脚本修改

默认文件系统没有 mdev.conf 文件和 /etc/mdev/ 目录，需要自己创建。如果你的 /etc 目录是只读目录，需要更改为可读写目录。

在 /etc 目录下建立这个文件 mdev.conf，内容如下（其他的类似）：

```
mmcblk0p[0-9]     0:0 666         @/etc/mdev/sd_insert.sh
mmcblk0           0:0 666         $/etc/mdev/sd_remove.sh
```

在 /etc/mdev/ 目录下建立 sd_insert.sh 和 sd_remove.sh 两个脚本，内容分别如下：

sd_insert.sh ：

```
#!/bin/sh
echo "@@@@@@@@ sd card insert! @@@@@@@@" > /dev/console
if [ -e "/dev/$MDEV" ]; then
  mkdir /tmp/sdcard
  mount -rw /dev/$MDEV /tmp/sdcard
fi
```

sd_remove.sh ：

```
#!/bin/sh
echo "@@@@@@@@ sd card remove! @@@@@@@@" > /dev/console
umount -l /tmp/sdcard*
rm -rf /tmp/sdcard*
```

6.11 SPI

SPI, Serial Perripheral Interface, 串行外围设备接口，它允许 MCU 以全双工的同步串行方式，与各种外围设备进行高速数据通信。

6.11.1 内核配置

在内核源码根目录下执行 make menuconfig 命令后进入配置界面，

按下面方式配置：

- **Kernel-3.10.14**

```
Device Drivers  --->
```

```
    [*] SPI support  --->
```

```
--- SPI support
[ ]   Debug support for SPI drivers
      *** SPI Master Controller Drivers ***
< >   Altera SPI Controller
< >   Ingeinc JZ47XX SPI controller test driver
<*>   Ingenic JZ series SPI driver
[ ]     Ingenic SoC SSI controller 0 for SPI Host driver
[*]     Ingenic SoC SSI controller 1 for SPI Host driver
[ ]       Disable DMA (always use PIO) on JZ SSI controller 1
          JZ SSI1 controller function pins select (GPIO A(20,21,22,29))  --->
[*]     Board info associated by spi master
[ ]     Use GPIO CE on JZ SSI controller 0
< >     Ingenic spi test driver
[ ]     Ingenic SoC SSI SLAVE controller for device driver
-*-   Utilities for Bitbanging SPI masters
< >   GPIO-based bitbanging SPI Master
< >   OpenCores tiny SPI
< >   NXP SC18IS602/602B/603 I2C to SPI bridge
< >   Analog Devices AD-FMCOMMS1-EBZ SPI-I2C-bridge driver
< >   Xilinx SPI controller common module
< >   DesignWare SPI controller core support
      *** SPI Protocol Masters ***
<*>   User mode SPI device driver support
< >   Infineon TLE62X0 (for power switching)
```

1 使能ingenic spi控制器驱动
2 使能spi1
3 使能dma模式
4 选择spi1复用引脚PA组
5 注册板级信息
6 使能linux 通用spi设备驱动

- **Kernel-4.4.94**

```
Device Drivers  --->
    [*] SPI support  --->
```

```
--- SPI support
[ ]   Debug support for SPI drivers
      *** SPI Master Controller Drivers ***
<*>   Ingenic SPI Controller
< >   Altera SPI Controller
-*-   Utilities for Bitbanging SPI masters
< >   Cadence SPI controller
< >   GPIO-based bitbanging SPI Master
< >   IMG SPFI controller
< >   Freescale SPI controller and Aeroflex Gaisler GRLIB SPI controller
< >   OpenCores tiny SPI
< >   Rockchip SPI controller driver
< >   NXP SC18IS602/602B/603 I2C to SPI bridge
< >   Analog Devices AD-FMCOMMS1-EBZ SPI-I2C-bridge driver
< >   Xilinx SPI controller common module
< >   Xilinx ZynqMP GQSPI controller
< >   DesignWare SPI controller core support
      *** SPI Protocol Masters ***
<*>   User mode SPI device driver support
< >   Infineon TLE62X0 (for power switching)
```

1 使能ingenic spi控制器驱动
2 使能linux 通用spi设备驱动

设备树添加一个设备驱动节点：

```
spidev: spidev@0 {
        compatible = "ingenic,spidev";
        reg = <0>;
        spi-max-frequency = <50000000>;
};
```

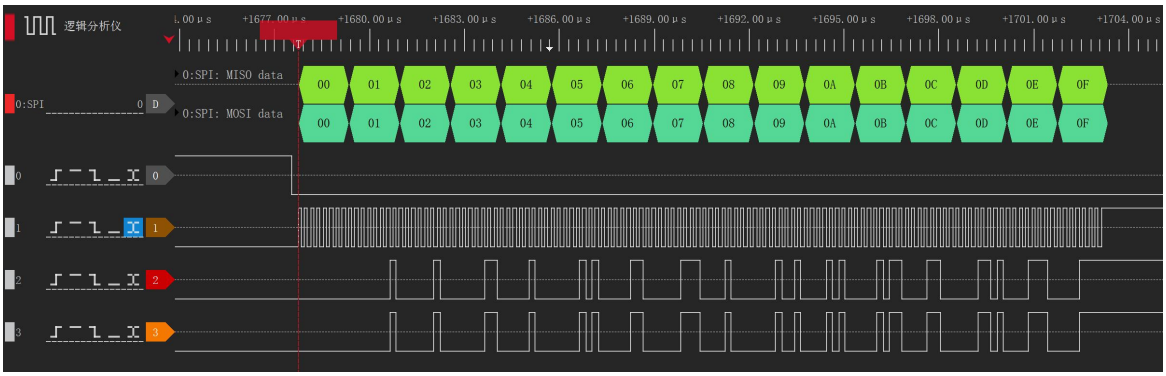
```
&spi1 {
    status = "okay";                    ① 使能spi1
    pinctrl-names = "default";
    pinctrl-0 = <&spi1_pa>;             ② 选择spi1 PA组引脚

    spi-max-frequency = <50000000>;     ③ 设置最大频率
    num-cs = <2>;
    cs-gpios = <0>, <0>;
    ingenic,chnl = <1>;
    ingenic,allow_cs_same = <1>;
    ingenic,bus_num = <1>;
    ingenic,has_dma_support = <1>;
    spidev: spidev@0 {
        compatible = "ingenic,spidev";
        reg = <0>;                      ④ 添加一个设备驱动
        spi-max-frequency = <50000000>;
    };
};
```

内核配置完成以后，以 SPI1 为例，使用 PA 组引脚，配置完成重新编译内核烧录会出现设备节点/dev/spidev1.0。

6.11.2 测试 sample

Sample 路径：samples/libsysutils-samples/sample_spi

根据 SPI 测试 sample，编译运行，发送数据并用逻辑分析仪抓图如下则测验证成功。



6.12 Motor

6.12.1 驱动介绍

电机的 sample 驱动是使用 GPIO 和定时器来控制四相八拍步进电机的转动。驱动

支持两个电机同时使用。参考代码/opensource/drivers/misc/下面；sample_motor 目录为不限位开关的电机驱动；sample_motor2 为带有限位开关的电机驱动；sample_motor3 为带细分器的电机驱动。

电机的控制单位是"步"。在实际产品中，两个电机受到结构的限制有最大转动角度。驱动中定义了两个坐标点(0,0),(vmaxstep,hmaxstep)分别对应两个电机转动的结构限制点。vmaxstep 和 hmaxstep 两个参数可以在加载电机驱动时以参数的形式设置，跟产品的结构转动限制和齿轮转速比有关。

6.12.2 接口介绍

表 6-4 Motor API

函数	功能
MOTOR_RESET	RESET 是使用电机前必须设置的。RESET 操作会控制电机先运动到(0，0)坐标点，然后运动到(vmaxstep/2，hmaxstep/2)坐标中心点。上电前，每个电机所处结构的位置可能都不一样，RESET 操作就是把两个电机都初始化到(vmaxstep/2，hmaxstep/2)坐标中心点。
MOTOR_MOVE	MOVE 操作是控制两个电机转动到一定的步数，参数 x，y 对应两个方向，是相对坐标。
MOTOR_GET_STATUS	GET_STATUS 接口可以得到目前电机的状态和坐标，这个坐标是绝对坐标。
MOTOR_SPEED	电机的转速范围是 100-900。值越大，速度越快。一般设置 400/500。
MOTOR_GOBACK	控制电机转动到(vmaxstep/2，hmaxstep/2)坐标中心点。
MOTOR_STOP	STOP 接口控制电机停止。
MOTOR_CRUIS	CRUISE 接口控制两个电机以最大行程巡航。

6.12.3 驱动适配

根据硬件设计，驱动需要修改 GPIO 配置。在文件 motor.h 中。例如：

```
#define HORIZONTAL_ST1_GPIO     GPIO_PB(22) /**< Phase A */
#define HORIZONTAL_ST2_GPIO     GPIO_PB(21) /**< Phase B */
#define HORIZONTAL_ST3_GPIO     GPIO_PB(20) /**< Phase C */
#define HORIZONTAL_ST4_GPIO     GPIO_PB(19) /**< Phase D */
```

```
#define VERTICAL_ST1_GPIO       GPIO_PC(7)
#define VERTICAL_ST2_GPIO       GPIO_PC(6)
#define VERTICAL_ST3_GPIO       GPIO_PC(5)
#define VERTICAL_ST4_GPIO       GPIO_PC(4)
```

注意

产品使用的 gpio 不一样，这点需要认真对比。驱动使用 TCU 定时器，如果客户使用 PWM 驱动，请确认 PWM 驱动使用的 TCU 通道，以防和电机使用 TCU 产生冲突！

6.13 GMAC

以太网模块提供了 1 个 Ethernet MAC，实现网络接口数据的接收和发送，可以工作在 10Mbit/s 或 100Mbit/s 模式下，支持全双工或者半双工工作模式，提供 RMII 接口。

1) 内核配置

在内核源码根目录下执行 make menuconfig 命令后进入配置界面，按下面方式配置：

```
Device Drivers  --->
     [*] Network device support  --->
          [*] Ethernet driver support  --->
               <*>  ingenic on-chip MAC support
               [*]     Ingenic mac dma interfaces
```

内核默认配置的是 IPv4，如果想使用 IPv6 参考下面配置：

```
[*] Networking support  --->
  Networking options  --->
      <*>   The IPv6 protocol
```

```
--- The IPv6 protocol
[*]   IPv6: Privacy Extensions (RFC 3041) support
[*]   IPv6: Router Preference (RFC 4191) support
[ ]     IPv6: Route Information (RFC 4191) support (NEW)
[ ]   IPv6: Enable RFC 4429 Optimistic DAD (NEW)
<*>   IPv6: AH transformation
<*>   IPv6: ESP transformation
<*>   IPv6: IPComp transformation
< >   IPv6: Mobility (NEW)
<*>   IPv6: IPsec transport mode (NEW)
<*>   IPv6: IPsec tunnel mode (NEW)
<*>   IPv6: IPsec BEET mode (NEW)
< >   IPv6: MIPv6 route optimization mode (NEW)
<*>   IPv6: IPv6-in-IPv4 tunnel (SIT driver) (NEW)
[ ]     IPv6: IPv6 Rapid Deployment (6RD) (NEW)
-*-   IPv6: IP-in-IPv6 tunnel (RFC2473)
<*>   IPv6: GRE tunnel
[ ]   IPv6: Multiple Routing Tables (NEW)
[ ]   IPv6: multicast routing (NEW)
```

- 配置 ip 地址及缺省网关

```
ip -6 addr add <ipv6address>/<ipv6_prefixlen> dev <port>
示例：ip -6 addr add 2001:da8:2::4a58/64 dev eth0
```

- Ping 某个 IPv6 地址

```
ping -6 <ipv6address>
示例：ping -6 2001:da8:2::4a59
```

6.14 Wifi

目前 wifi 模组主要存在两种接口方式：USB 和 SDIO； T 系列芯片两种接口都支持。现在主流 wifi 芯片调试完成的型号如下：

- USB 接口：MT7601、RTL8188FTV、RTL8723BU、ATBM6022、RDA5995、RTL8189FTV、RTL8188GTV、ssv6xxx。
- SDIO 接口：AP6212A、RTL8189FS、RTL8189ES、ESP8089、MARVELL8801、SSV6x5x、Hi3881V100R001C00SPC021、ATBM602x、ATBM603x、ATBM6441。

6.14.1 Wifi 内核配置

不同 wifi 芯片内核的配置存在不同的地方；下面将逐步介绍以上 wifi 的内核配置部分。

6.14.1.1 USB 接口 WIFI

使用内核默认配置即可支持。

6.14.1.2 AP6212A

这款 wifi 内核默认不支持，需要打上相应的 patch，打上 patch 后按如下配置内核。

第一步：先选择 BCM 驱动

```
Device Drivers--->
     Misc devices --->
         <*> BCM module power control core driver
```

第二步：选择 AP6212A 驱动

```
Device Drivers--->
   [*]  Network device support --->
      [*]  Wireless LAN --->
          <*> Broadcom bcm43438 wireless cards support
          [*]    SDIO bus interface support
```

第三步：设备树开启 MSC1

```
    文件：arch/mips/boot/dts/ingenic/shark.dts
      &msc1 {
           status = "okay";
           ...
      }
```

6.14.1.3 除了 AP6212A 以外的 SDIO 接口 WIFI

设备树开启 MSC1

```
    文件：arch/mips/boot/dts/ingenic/shark.dts
      &msc1 {
           status = "okay";
           ...
      }
```

6.14.2 Wifi 驱动编译

1) USB 接口的 wifi 驱动，只需要指定编译工具链和 kernel 路径，然后直接编译

即可。以 RTL8723BU 为例：打开 Makefile，参照添加 CONFIG_PLATFORM_INGENIC = y；然后参照现有板机继续添加：

```
ifeq ($(CONFIG_PLATFORM_INGENIC), y)
EXTRA_CFLAGS += -DCONFIG_LITTLE_ENDIAN -DCONFIG_MINIMAL_MEMORY_USAGE
ARCH ?= mips
CROSS_COMPILE ?= mips-linux-gnu-
KSRC ?= /home_d/ywhan/work/isvp/opensource/kernel

ifeq ($(CONFIG_SDIO_HCI), y)
EXTRA_CFLAGS += -DCONFIG_PLATFORM_OPS
_PLATFORM_FILES += platform/platform_ingenic_sdio.o
endif
endif
```

2) SDIO 接口的 wifi 驱动，首先需要在 platform 文件夹下面添加 power_en 引脚控制代码 platform_ingenic_sdio.c，该文件主要定义了 WIFI 模组的 power_en 引脚的控制，代码默认引脚为 PB(30)。以 RTL8189ES 为例：

A. 首先添加 platform_ingenic_sdio.c，确定硬件对应的 power_en 引脚和软件是否匹配。

```
#define GPIO_WIFI_WAKEUP        GPIO_PC(17)
#define GPIO_WIFI_RST_N         GPIO_PC(16)
#define SDIO_WIFI_POWER         GPIO_PB(30)
#define WLAN_SDIO_INDEX         1
```

B. 然后打开 Makefile，参照添加 CONFIG_PLATFORM_INGENIC = y ，然后参照现有板机继续添加

```
ifeq ($(CONFIG_PLATFORM_INGENIC), y)
EXTRA_CFLAGS += -DCONFIG_LITTLE_ENDIAN -DCONFIG_MINIMAL_MEMORY_USAGE
#EXTRA_CFLAGS += -DCONFIG_IOCTL_CFG80211 -DRTW_USE_CFG80211_STA_EVENT
ARCH ?= mips
CROSS_COMPILE ?= mips-linux-gnu-
KSRC ?= /home_d/ywhan/work/isvp/opensource/kernel

ifeq ($(CONFIG_SDIO_HCI), y)
EXTRA_CFLAGS += -DCONFIG_PLATFORM_OPS
_PLATFORM_FILES += platform/platform_ingenic_sdio.o
endif
endif
```

注：AP6212A 驱动已经集成到内核不需要单独编译。

6.14.3 Wifi 启动操作流程

加载 KO， 成功会生成 wlan0 节点，通过 ifconfig -a 可以看到。

第一步：启动 WIFI（可选操作）

```
$ ifconfig wlan0 up
```

第二步：配置无线网络

```
$ wpa_supplicant -D wext -i wlan0 -c wpa_supplicant.conf -B
```

(wpa_supplicant 应用软件的参考附录 7.3 wpa_supplicant)。

第三步：连接 WIFI：自动配置 IP；获取 IP 地址，网关等信息

```
$ udhcpc -i wlan0            //注意 udhcpc.script 的路径匹配
```

或者手动配置 IP；

```
$ ifconfig wlan0 193.169.4.75
$ route add default gw 193.169.4.1
$ echo "nameserver 193.169.1.57" > /etc/resolv.conf
```

6.15 USB

6.15.1 USB 工作模式配置

USB 设备模式和 USB 主机模式。在 USB 设备模式下，外部 USB 硬件充当 USB 主机，开发板上的 USB 属于从机设备；USB 主机模式正好相反，开发板属于主机，外部 USB 设备属于从机设备。

6.15.1.1 Device Only 模式

```
Device Drivers  --->
    [*] USB support  --->
        <*> DesignWare USB2 DRD Core Support
            Driver Mode (Device Mode Only)  --->
    <*> USB Gadget Support  --->
```

6.15.1.2 Host Only 模式

```
Device Driver --->
     [*] USB support  --->
         <*> Support for Host-side USB
     <*> DesignWare USB2 DRD Core Support
         Driver Mode (Host Mode Only)
```

6.15.2 USB 接口配置

由于 USB 外设种类繁多，内核默认配置不能完全兼顾，用户可以根据需要自行添加。

6.15.2.1 Host 设备

1) U 盘

第一步：进入配置界面 make menuconfig 配置如下：

```
a) SCSI Config
    Device Driver --->
        SCSI device support --->
            <*> SCSI device support
            <*> SCSI disk support
   b) U-disk Config
   Device Driver --->
      [*] USB support  --->
         <*> USB Mass Storage support 要先配置 SCSI Config 才会出现该选项
```

第二步：编译内核 make uImage 并烧录到开发板上；

第三步：插上 U 盘，内核会打印相应提示信息。

2) USB 转串口

第一步：进入配置界面 make menuconfig 配置如下：

```
   Device Driver --->
       [*] USB support  --->
           <*> Support for Host-side USB
           <*> USB Modem (CDC ACM) support
           <*> USB Serial Converter support  --->
                <*> USB Prolific 2303 Single Port Serial Driver //选择串口芯片，以 2303 的芯片配置为例。
```

第二步：编译内核 make uImage 并烧录到开发板上；

第三步：插上串口线，内核会打印相应提示信息。

3) USB 硬盘

第一步：进入配置界面 make menuconfig 配置如下：

```
   a) Device Driver --->
         SCSI device support --->
            <*> SCSI device support
            <*> SCSI disk support
```

```
        [*] SCSI low-level drivers (NEW)  --->
b) Device Driver --->
     [*] USB support  --->
         <*> Support for Host-side USB
         <*> USB Mass Storage support
         <*> DesignWare USB2 DRD Core Support
               Driver Mode (Host Mode Only)
         <*> USB Gadget Support  --->
c)  [*] Enable the block layer  --->
         [*]  Support for large (2TB+) block devices and files
```

第二步：编译内核 make uImage 并烧录到开发板上；

第三步：连接硬盘，启动系统，内核会打印相应提示信息。

4) USB 以太网卡

第一步：进入配置界面 make menuconfig 配置如下：

```
a) Eth Config
     Device Drivers  --->
         [*] Network device support  --->
               USB Network Adapters  --->
               <*> Multi-purpose USB Networking Framework
               <*> ASIX AX88xxx Based USB 2.0 Ethernet Adapters (NEW)
b) USB Config
     Device Drivers  --->
         [*] USB support  --->
             <*> DesignWare USB2 DRD Core Support
                    Driver Mode (Host Mode Only)  --->
         <*> USB Gadget Support  --->
```

第二步：编译内核 make uImage 并烧录到开发板上；

第三步：启动系统，查看相应信息，用 ifconfig –a 查看网络节点（正常是 eth1），然后配置 ip 地址和网关；

```
$ ifconfig eth1 193.169.4.228 up
$ route add default gw 193.169.4.1
```


注意

如果 IP 地址配置成功还是无法 ping 通服务器，检测 GMAC 的 eth0 是否关闭。

5) USB 鼠标

第一步：进入配置界面 make menuconfig 配置如下：

```
a) USB Mode Config
    Device Driver --->
      [*] USB support   --->
        <*> Support for Host-side USB
        <*> DesignWare USB2 DRD Core Support
           Driver Mode (Host Mode Only)
b) Input system Config
    Device Driver --->
       Input device support  --->
         <*>   Mouse interface
            [*]   Mice  --->
c) USB His Config
    Device Drivers  --->
       HID support  --->
       -*- HID bus support
       /dev/hidraw raw HID device support
       USB HID support  --->
          <*> USB HID transport layer
          [*] /dev/hiddev raw HID device support
```

第二步：编译内核 make uImage 并烧录到开发板上；

第三步：启动系统，插入鼠标，查看相应的信息。

```
[   23.101552] usb 1-1: new low-speed USB device number 2 using dwc2
[   23.313232] input: Logitech USB Optical Mouse as /devices/platform/jz-dwc2/dwc2/usb1/1-1/1-1:1.0/input/input1
[   23.329602] hid-generic 0003:046D:C05A.0001: input,hidraw0: USB HID v1.11 Mouse [Logitech USB Optical Mouse] on usb-dwc2-1/input0
```

我们 cat /dev/input/event1 这个节点，然后移动鼠标，在串口上可以看到一连串的打印（打印信息乱码显示，这些信息是鼠标的坐标信息等），停止移动鼠标就没有这些信息。

6.15.2.2 Device 设备

在 Linux-4.4.94 中，推荐在用户空间使用 configfs 文件系统对 USB Gadget 设备进行配置，这种方法配置方便，可在任何阶段使用，将成为未来的主流方式。

对于 USB Configfs 的介绍可以参考 Linux 内核文档：kernel-4.4.94/Documentation/usb/gadget_configfs.txt

下面根据该文档介绍使用 configfs 配置 USB Gadget 的流程：

1. 先在 make menuconfig 中配置你想使用的 USB Function，比如是大容量存储设备、键盘、鼠标或者网卡，具体的配置方式在下面会介绍。
2. 挂载 configfs 文件系统，挂载成功后会在挂载点自动生成 usb_gadget 目录。

```
$ mount -t configfs none /sys/kernel/config
```

3. 在 usb_gadget 目录下创建名为 g1 的 USB Gadget 设备（名称可以任取），创建成功之后会自动生成一系列属性文件，然后根据需要配置这些属性文件。

```
[root@Ingenic-uc1_1:g1]# pwd
/sys/kernel/config/usb_gadget/g1
[root@Ingenic-uc1_1:g1]# ls -lh
-rw-r--r--    1 root     root        4.0K Jan  1 00:05 UDC
-rw-r--r--    1 root     root        4.0K Jan  1 00:05 bDeviceClass
-rw-r--r--    1 root     root        4.0K Jan  1 00:05 bDeviceProtocol
-rw-r--r--    1 root     root        4.0K Jan  1 00:05 bDeviceSubClass
-rw-r--r--    1 root     root        4.0K Jan  1 00:05 bMaxPacketSize0
-rw-r--r--    1 root     root        4.0K Jan  1 00:05 bcdDevice
-rw-r--r--    1 root     root        4.0K Jan  1 00:05 bcdUSB
drwxr-xr-x    2 root     root           0 Jan  1 00:03 configs
drwxr-xr-x    2 root     root           0 Jan  1 00:03 functions
-rw-r--r--    1 root     root        4.0K Jan  1 00:05 idProduct
-rw-r--r--    1 root     root        4.0K Jan  1 00:05 idVendor
drwxr-xr-x    2 root     root           0 Jan  1 00:03 strings
```

4. 配置 PID、VID、class、subclass 和 protocol 等，这些根据需要进行配置。
5. 创建并配置 string 子目录，这个对应该 USB 设备的字符串描述符。
6. 创建 configuration 和字符串。
7. 创建 functions。
8. 将 functions 和 configuration 关联起来。
9. 绑定该 USB Gadget 到 UDC 上。

1) USB 挂载 SD 卡

第一步：进入配置界面 make menuconfig 配置如下：

```
    a) Device Drivers  --->
          [*] USB support  --->
              <*> DesignWare USB2 DRD Core Support
                     Driver Mode (Device Mode Only)  --->
          <*> USB Gadget Support  --->
              <*>   USB Gadget Drivers (USB functions configurable through co
nfigfs)  --->
              [*]   Mass storage
```

第二步：编译内核 make uImage 并烧录到开发板上；

第三步：插入 SD 卡，启动系统，内核会打印相应提示信息；

第四步：进入命令行，根据前文说的 configfs 的配置过程执行下面操作：

```
# mount -t configfs none /sys/kernel/config
# cd /sys/kernel/config/usb_gadget
# mkdir g1
# cd g1
# echo 0x18d1 > idVendor
# echo 0x4ee2 > idProduct
# mkdir strings/0x409
# echo "123456" > strings/0x409/serialnumber
# echo "Ingenic" > strings/0x409/manufacturer
# echo "Mass Storage" > strings/0x409/product
# mkdir configs/c.1
# mkdir configs/c.1/strings/0x409
echo "mass_storage" > configs/c.1/strings/0x409/configuration
# mkdir functions/mass_storage.0
echo "/dev/mmcblk0p1" > functions/mass_storage.0/lun.0/file
# ln -s functions/mass_storage.0 configs/c.1
# ls /sys/class/udc > UDC
```

稍等，PC 即会自动识别出 U 盘（即 sd 卡），进入到 U 盘之中，从 PC 机中拷贝文件到此 U 盘中，之后弹出 U 盘，通过 mount /dev/mmc0blkp1 /dev/mnt 将 sd 卡挂载到 /mnt 目录中，切换到 mnt 中，可以看到 PC 机拷贝的文件。

2) USB 鼠标键盘

工作在 device 模式时，可以将其配置为键盘和鼠标，其实就是仿真(emulation)键盘和鼠标的功能。对该功能的说明可以参考 Linux 内核文档： Documentation/usb/gadget_hid.txt。想了解更详细的关于 USB HID 的信息，可以访问下面这个网站：https://www.usb.org/hid。

使用 configfs 配置鼠标和键盘的功能，使得 T41 同时支持鼠标和键盘功能。

第一步：进入配置界面 make menuconfig 配置如下：

```
a) Device Drivers  --->
      [*] USB support  --->
          <*> DesignWare USB2 DRD Core Support
                 Driver Mode (Device Mode Only)  --->
      <*> USB Gadget Support  --->
          <*>   USB Gadget Drivers (USB functions configurable through co
nfigfs)  --->
          [*]   HID function
```

第二步：编译内核 make uImage 并烧录到开发板上；

第三步：进入命令行，根据前文说的 configfs 的配置过程执行下面操作。

```
    # mount -t configfs none /sys/kernel/config
    # cd /sys/kernel/config/usb_gadget
    # mkdir g1
    # cd g1
    # echo 0x18d1 > idVendor
    # echo 0x4ee2 > idProduct
    # mkdir strings/0x409
    # echo "123456" > strings/0x409/serialnumber
    # echo "Ingenic" > strings/0x409/manufacturer
    # echo "HID Gadget" > strings/0x409/product
    # mkdir configs/c.1
    # mkdir configs/c.1/strings/0x409
    # echo "HID" > configs/c.1/strings/0x409/configuration
    （模拟键盘）
    # mkdir functions/hid.usb0
    # echo 0x1 > functions/hid.usb0/protocol
    # echo 0x1 > functions/hid.usb0/subclass
    # echo 0x8 > functions/hid.usb0/report_length
    (设置键盘的报告描述符)
    # echo -ne \\x05\\x01\\x09\\x06\\xa1\\x01\\x05\\x07\\x19\\xe0\\x29\\xe7\\x1
5\\x00\\x25\\x01\\x75\\x01\\x95\\x08\\x81\\x02\\x95\\x01\\x75\\x08\\x81\\x03\\
x95\\x05\\x75\\x01\\x05\\x08\\x19\\x01\\x29\\x05\\x91\\x02\\x95\\x01\\x75\\x03
\\x91\\x03\\x95\\x06\\x75\\x08\\x15\\x00\\x25\\x65\\x05\\x07\\x19\\x00\\x29\\x
65\\x81\\x00\\xc0 > functions/hid.usb0/report_desc
    # ln -s functions/hid.usb0 configs/c.1
    （模拟鼠标）
    # mkdir functions/hid.usb1
    # echo 0x2 > functions/hid.usb1/protocol
    # echo 0x1 > functions/hid.usb1/subclass
    # echo 0x4 > functions/hid.usb1/report_length
    （设置鼠标的报告描述符）
    # echo -ne \\x05\\x01\\x09\\x02\\xa1\\x01\\x09\\x01\\xa1\\x00\\x05\\x09\\x1
9\\x01\\x29\\x03\\x15\\x00\\x25\\x01\\x95\\x03\\x75\\x01\\x81\\x02\\x95\\x01\\
x75\\x05\\x81\\x03\\x05\\x01\\x09\\x30\\x09\\x31\\x15\\x81\\x25\\x7f\\x75\\x08
\\x95\\x02\\x81\\x06\\xc0\\xc0 > functions/hid.usb1/report_desc
    # ln -s functions/hid_usb1 configs/c.1

    # ls /sys/class/udc > UDC
```

第四步：等待进入系统以后，将 otg 接口接到 ubuntu 主机上，发现有设备枚举，下面是 ubuntu dmesg 中设备的枚举信息：

```
    [48624.817076] usb 1-2: USB disconnect, device number 7
    [48644.342747] usb 1-2: new high-speed USB device number 8 using ehci-pci
    [48644.620913] usb 1-2: New USB device found, idVendor=0525, idProduct=a4ac, bcdDevi
ce= 3.10
    [48644.620917] usb 1-2: New USB device strings: Mfr=1, Product=2, SerialNumber=0
    [48644.620919] usb 1-2: Product: HID Gadget
    [48644.620921] usb 1-2: Manufacturer: Linux 3.10.14__isvp_swan_1.0__ with dwc2-gadge
t
    [48644.683174] input: Linux 3.10.14__isvp_swan_1.0__ with dwc2-gadget HID Gadget as
/devices/pci0000:00/0000:00:11.0/0000:02:02.0/usb1/1-2/1-2:1.0/0003:0525:A4AC.0007/inpu
t/input11
    [48644.747696] hid-generic 0003:0525:A4AC.0007: input,hidraw1: USB HID v1.01 Keyboar
d [Linux 3.10.14__isvp_swan_1.0__ with dwc2-gadget HID Gadget] on usb-0000:02:02.0-2/inp
ut0
    [48644.785440] input: Linux 3.10.14__isvp_swan_1.0__ with dwc2-gadget HID Gadget as
/devices/pci0000:00/0000:00:11.0/0000:02:02.0/usb1/1-2/1-2:1.1/0003:0525:A4AC.0008/inpu
t/input12
    [48644.854093] hid-generic 0003:0525:A4AC.0008: input,hidraw2: USB HID v1.01 Mouse
[Linux 3.10.14__isvp_swan_1.0__ with dwc2-gadget HID Gadget] on usb-0000:02:02.0-2/inpu
t1
```

进入设备的串口查看/dev 目录，可以看到生成了/dev/hidg0 和/dev/hidg1 两个节点，分别对应键盘和鼠标。

第五步：测试键盘鼠标

可参考/Documentation/usb/gadget_hid.txt，其中给了一个测试的 demo。

3) USB RNDIS 配置

USB RNDIS（远程网络驱动接口规范），产品为 USB 从设备。

第一步：进入配置界面 make menuconfig 配置如下

```
    a) Device Drivers  --->
          [*] USB support  --->
              <*> DesignWare USB2 DRD Core Support
                     DWC2 Mode Selection (Gadget only mode) --->
          <*> USB Gadget Support  --->
              <*>   USB Gadget Drivers (USB functions configurable through co
nfigfs)  --->
              [*]   RNDIS
```

第二步：编译内核 make uImage 并烧录到开发板上；

第三步：进入命令行，根据前文说的 configfs 的配置过程执行下面操作。

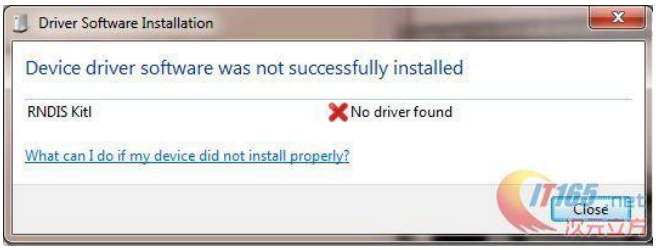
```
    # mount -t configfs none /sys/kernel/config
    # cd /sys/kernel/config/usb_gadget
```

```
# mkdir g1
# cd g1
# echo 0x18d1 > idVendor
# echo 0x4ee2 > idProduct
# mkdir strings/0x409
# echo "123456" > strings/0x409/serialnumber
# echo "Ingenic" > strings/0x409/manufacturer
# echo "RNDIS" > strings/0x409/product
# mkdir g1/configs/c.1
# mkdir configs/c.1/strings/0x409
echo "rndis" > configs/c.1/strings/0x409/configuration
# mkdir functions/rndis.usb0
# ln -s functions/mass_storage.0 configs/c.1
# ls /sys/class/udc > UDC
```

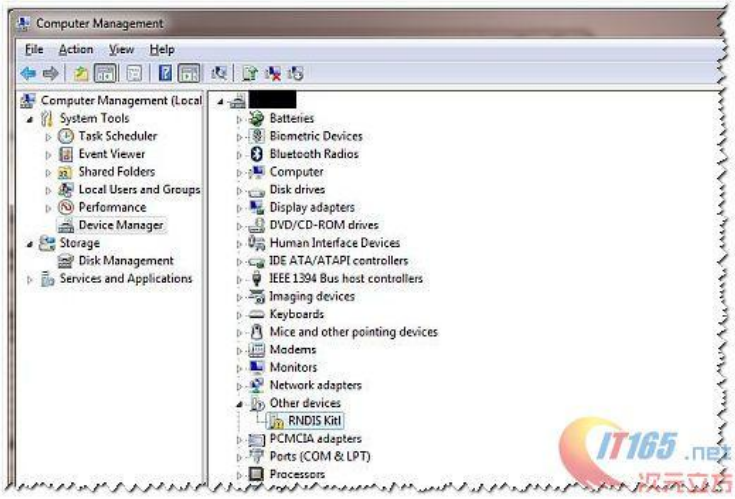
第四步：PC 端配置

（1） RNDIS（Remote Network Driver Interface Specification）也叫远端网络驱动接口协议，设备通过 USB 方式同主机连接，模拟网络连接以便用于下载和调试工作。RNDIS 驱动是 Windows7 的一部分，遗憾的是如果默认安装（插上符合 RNDIS 的设备时）一般均会安装失败（以后可能会修正此问题）。可通过如下方法重新安装 RNDIS 驱动。

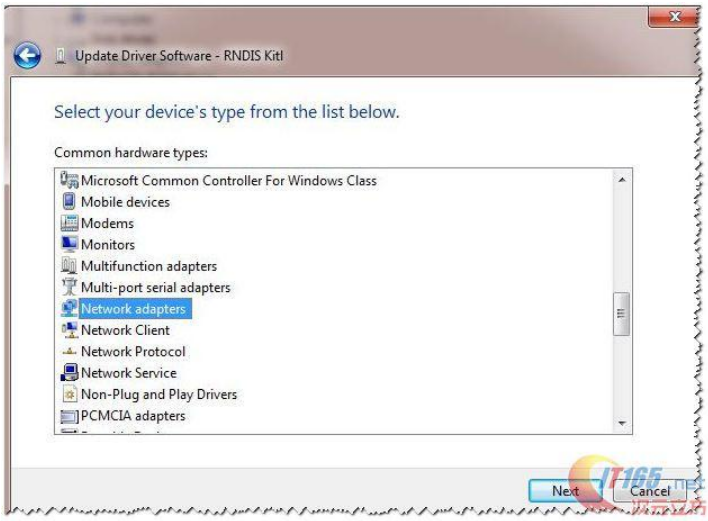
（2）设备同计算机连接时，操作系统会自动搜索并安装 RNDIS 驱动，不过，片刻之后您会发现安装失败。



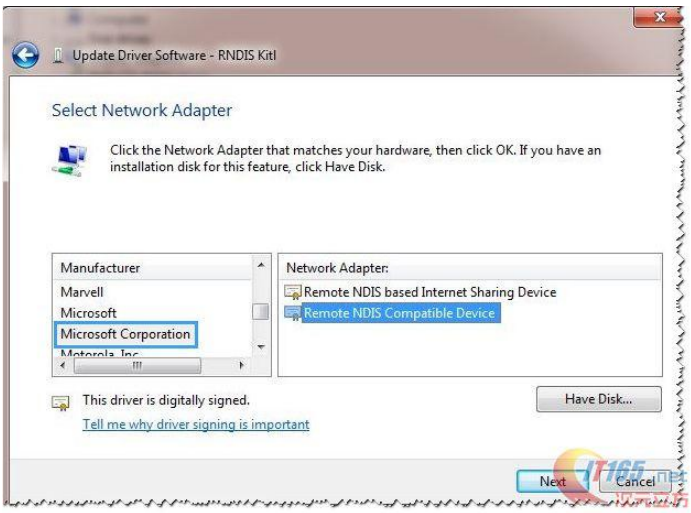
（3）右键点击桌面“计算机”图标，选择“管理”——“设备管理”，可以看到“RNDIS Kitl”设备，并且处于驱动未安装状态。



（4）右键点击该设备，选择更新驱动软件，在如何搜索设备软件提示窗口中，选择“浏览我的计算机”。选择从设备列表中选择“网络适配器”。

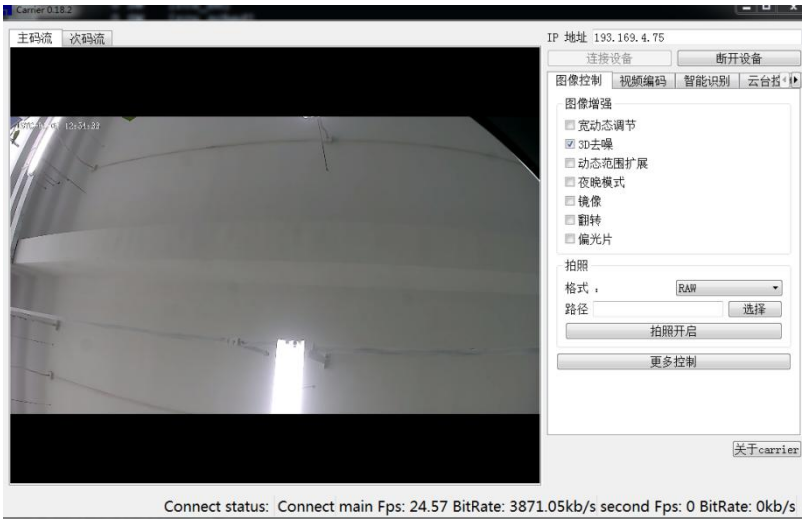


（5）在网络适配器窗口的制造商列表中选择微软公司（Microsoft Corporation），右侧列表中选择远端 NDIS 兼容设备。



（6）点击下一步并等待安装结束，RNDIS Kitl 设备将会安装成功。

（7）然后可以执行 carrier tool 工具查看图像。



（8）Win10 RNDIS

安装一个驱动 kindle_rndis.inf_amd64-v1.0.0.1.zip ，在 SDK 中的 /tools/pc 目录有驱动安装包。

第一步：在需要操作的电脑上，解压安装包；

第二步：找到解压文件中的 5-runasadmin_register-CA-cer.cmd ，以管理员身份安装程序；

第三步： 在我的电脑 管理->设备管理器，在右边目录中有“端口（COM 和 LPT）”，右键单击“串行 USB 设备（COM3）”->更新驱动程序软件...

第四步：在控制面板中“网络和共享中心”>更改适配器设置中看到一个新的网卡 Kindle USB RNDIS 设备（已启用 USBNetwork）。分配一个静态 IP（例

如 193.169.4.xx），可以 ping 设备 ip。

6.16 4G

6.16.1 PPP

第一步：修改内核中需要的代码

主要修改下面几个文件的代码；参考附录 7.4.1 PPP 模式

```
modified:   drivers/net/usb/qmi_wwan.c
modified:   drivers/usb/serial/option.c
modified:   drivers/usb/serial/qcserial.c
modified:   drivers/usb/serial/usb_wwan.c
```

第二步：内核配置（编译烧录）

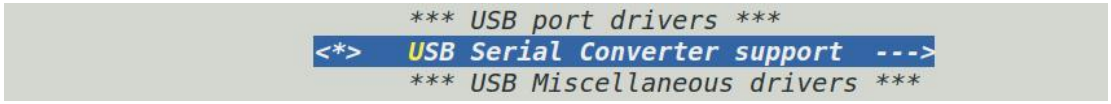
A. network 部分修改

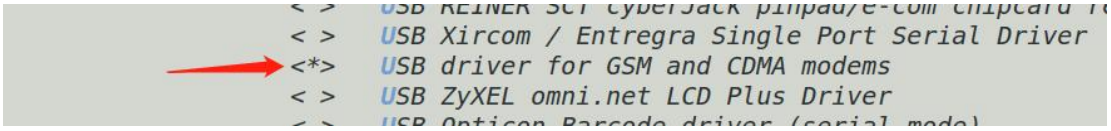
```
<*>   PPP (point-to-point protocol) support
<*>     PPP BSD-Compress compression
<*>     PPP Deflate compression
[*]     PPP filtering
<*>     PPP MPPE compression (encryption)
[*]     PPP multilink support
<*>     PPP over Ethernet
< >     PPP on L2TP Access Concentrator
< >     PPP on PPTP Network Server
<*>     PPP support for async serial ports
<*>     PPP support for sync tty ports
< >   SLIP (serial line) support
```

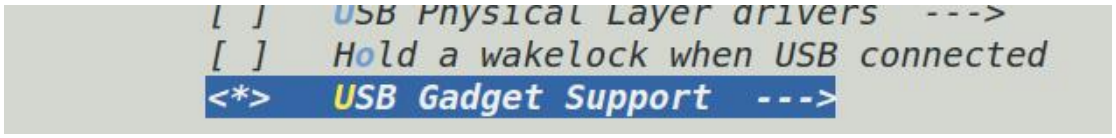
```
<*>     PPP support for sync tty ports
< >   SLIP (serial line) support
      USB Network Adapters  --->
[*]   Wireless LAN  --->
```

```
< > USB RTL8150 based ethernet device support
< > Realtek RTL8152 Based USB 2.0 Ethernet Adapters
<*> Multi-purpose USB Networking Framework
<*>   ASIX AX88xxx Based USB 2.0 Ethernet Adapters
<*>   ASIX AX88179/178A USB 3.0/2.0 to Gigabit Ethernet
```

B. usb 部分修改







第三步：文件系统的修改

A. 请在文件系统的 /etc 目录下通过创建 ppp 软连接；可参考如下操作

```
cd /etc
ln -s ../system/etc/ppp ppp
```

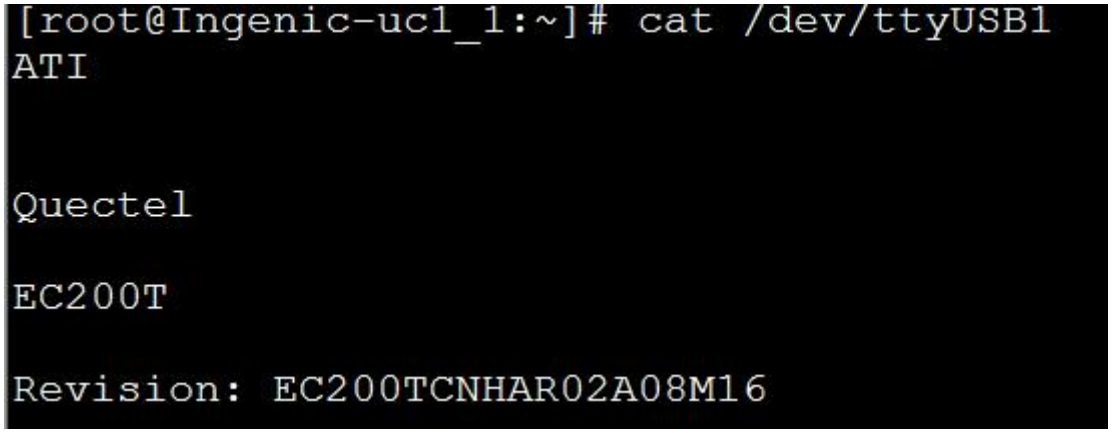
B. 制作 squashfs 文件系统。

linux-ppp-scripts 文件夹里面的 lib 库拷贝到文件系统中， sbin 下的拷贝到系统的 bin 下面；其它脚本参考 readme 使用方法，拷贝到 /etc/ppp/peers 文件夹下面；ip-up 需要拷贝到 /etc/ppp 目录下！

第四步：系统调试

A. 优先查看是否生成了 /dev/ttyUSB*

B. 执行 `echo -e "ATI\r\n" > /dev/ttyUSB1; cat /dev/ttyUSB1`



出现上图所示；表明可以和 4G 模块通信了！

C. 执行命令 pppd call quectel-ppp &

```
Script chat -s -v -f /etc/ppp/peers/quectel-chat-connect finished (pid 299), status = 0x0
Serial connection established.
using channel 1
Using interface ppp0
Connect: ppp0 <--> /dev/ttyUSB2
sent [LCP ConfReq id=0x1 <asyncmap 0x0> <magic 0x97e58dc8> <pcomp> <accomp>]
rcvd [LCP ConfReq id=0x1 <asyncmap 0x0> <auth pap> <magic 0x33bafefe> <pcomp> <accomp>]
sent [LCP ConfAck id=0x1 <asyncmap 0x0> <auth pap> <magic 0x33bafefe> <pcomp> <accomp>]
rcvd [LCP ConfAck id=0x1 <asyncmap 0x0> <magic 0x97e58dc8> <pcomp> <accomp>]
sent [PAP AuthReq id=0x1 user="test" password=<hidden>]
rcvd [PAP AuthAck id=0x1 "" 00]
PAP authentication succeeded
sent [IPCP ConfReq id=0x1 <addr 0.0.0.0> <ms-dns1 0.0.0.0> <ms-dns2 0.0.0.0>]
rcvd [IPCP ConfReq id=0x2]
sent [IPCP ConfNak id=0x2 <addr 0.0.0.0>]
rcvd [IPCP ConfNak id=0x1 <addr 100.185.143.235> <ms-dns1 202.102.213.68> <ms-dns2 61.132.163.68>]
sent [IPCP ConfReq id=0x2 <addr 100.185.143.235> <ms-dns1 202.102.213.68> <ms-dns2 61.132.163.68>]
rcvd [IPCP ConfReq id=0x3]
sent [IPCP ConfAck id=0x3]
rcvd [IPCP ConfAck id=0x2 <addr 100.185.143.235> <ms-dns1 202.102.213.68> <ms-dns2 61.132.163.68>]
Could not determine remote IP address: defaulting to 10.64.64.64
local  IP address 100.185.143.235
remote IP address 10.64.64.64
primary   DNS address 202.102.213.68
secondary DNS address 61.132.163.68

[root@Ingenic-uc1_1:~]# ping www.baidu.com
ping: bad address 'www.baidu.com'
[root@Ingenic-uc1_1:~]# route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         0.0.0.0         0.0.0.0         U     0      0        0 ppp0
10.64.64.64     0.0.0.0         255.255.255.255 UH    0      0        0 ppp0
[root@Ingenic-uc1_1:~]#
```

出现上图打印说明配网成功！

D. 检测 /etc/resolv.conf 是否有配置 DNS，如果有就可以进行下一步

```
[root@Ingenic-uc1_1:~]# cat /etc/resolv.conf
nameserver 61.132.163.68
nameserver 202.102.213.68
```

E. ping www.baidu.com 测试

```
[root@Ingenic-uc1_1:~]# ping www.baidu.com
PING www.baidu.com (14.215.177.38): 56 data bytes
64 bytes from 14.215.177.38: seq=0 ttl=54 time=38.556 ms
64 bytes from 14.215.177.38: seq=1 ttl=54 time=42.371 ms
64 bytes from 14.215.177.38: seq=2 ttl=54 time=37.308 ms
64 bytes from 14.215.177.38: seq=3 ttl=54 time=37.153 ms
```

6.16.2 GobNet

第一步：在配置 gobnet 时需要先在内核中修改以下代码：

```
A. /drivers/net/usb/Makefile (添加下面两行)
+     obj-y += GobiNet.o
+     GobiNet-objs := GobiUSBNet.o QMIDevice.o QMI.o
B. /drivers/net/usb/qmi_wwan.c  (注释下面一行代码)
-     //{QMI_GOBI_DEVICE(0x05c6, 0x9215)},      /* Acer Gobi 2000 Modem
device (VP413) */
C. /drivers/usb/serial/qcserial.c  (注释下面一行代码)
-     //{USB_DEVICE(0x05c6, 0x9215)},   /* Acer Gobi 2000 Modem device
(VP413) */
```

第二步：内核配置（编译烧录）

```
[*] Device Drivers →
   -*- Network device support →
       USB Network Adapters →
           {*} Multi-purpose USB Networking Framework
```

第三步：烧录完内核后执行 quectel-CM &即可。

6.16.3 WWAN

第一步：在配置 gobnet 时需要先在内核中修改以下代码：参考附录 7.4.2 Wwan 模式

第二步：内核配置（编译烧录）

```
[*] Device Drivers →
   -*- Network device support →
       USB Network Adapters →
           {*} Multi-purpose USB Networking Framework
               <*> QMI WWAN driver for Qualcomm MSM based 3G and LTE mode
ms
```

第三步：烧录完内核后执行 quectel-CM &即可。

7 附录

7.1 制作启动卡

对于第一次启动开发板，需要从 SD 卡启动，再将编译好的 uboot 烧录到 flash 中。以后就可直接从 flash 中启动。

7.1.1 制作新分区

格式化 sd 卡，并将卡启动 uboot 烧录到 SD 卡。
此步骤只需执行一次，若不更新卡起 uboot 则不需要再次执行此步骤。经过此步骤处理的 SD 卡可当作正常卡使用。


注意
确定插入你的电脑上 TF 卡对应的设备节点文件是不是 sdb(有可能是 sdc 或者 sdd，可以通过插拔 SD 卡确认)；如果对应错了可能会破坏电脑的硬盘文件系统。

A．卸载 SD 卡

```
umount /media/user/*
```

B．将 SD 卡重新分区

```
# sudo fdisk /dev/sdb
Command (m for help): o                    --创建一个新的分区表
Command (m for help): n                    --添加一个分区 n
--此时会提示：
Partition type:
   p   primary (0 primary, 0 extended, 4 free)
   e   extended
Select (default p):
--回车默认是 p，创建主分区
Partition number (1-4, default 1):
```

```
    --回车默认创建 1 号分区
    First sector (2048-15261695, default 2048):
    --回车使用默认值 2048
    --2048 号扇区在 1MB 的位置，给 uboot 空出了空间
    Last sector, +sectors or +size{K,M,G,T,P} (2048-15261695, default 15261
695):
    --回车使用默认值 15261695
    Command (m for help): w              --最后一步，向 SD 卡中写入分区表
    The partition table has been altered!
    Calling ioctl() to re-read partition table.
    Syncing disks.
```

到这里分区表已创建完成

D．执行 sync，拔卡，重新插卡，PC 将重新识别分区。

E．格式化 sd 卡为通用的 VFat 文件系统：

```
# mkfs.vfat /dev/sdb1        //这里重新分区的是 sdb1
```

F．编译卡起 u-boot，选择对应的 uboot 类型(3.1 uboot 编译)，将生成的 u-boot-with-spl.bin 文件烧录进 sd 卡的 17KB 偏移处(注意是 dev/sdb 分区，不是 dev/sdb1)：

```
# dd if=u-boot-with-spl.bin of=/dev/sdb bs=1024 seek=17
```

到此，可以用来烧录的 SD 卡制作完成。下面可以使用这个启动卡启动开发板。

7.1.2 SD 卡启动

1) bootload 默认最先从 nor 读取启动信息，如果 nor 原本烧写过 uboot，现在想要从卡启，需要做如下操作：

A. 把制作好的启动卡插入开发板。

B. 长按 boot 键，然后断开电源，上电等 3s，松开 boot 键即可进入卡启 uboot。

2) 使用 sd 卡启动开发板，开机前几秒迅速按下回车键，进入 uboot 命令行；

A．首先，将需要烧录的文件读到内存中，以 u-boot 为例：

B. 把内存先全部清为全 f

```
isvp# mw.b 0x80600000 0xff 0x1000000  --第三个参数是大小，这里清了 16MB
```

C．ls SD 卡中的文件

```
isvp# fatls mmc 0
   222944   u-boot-with-spl.bin
   2368879   uimage
5 file(s), 0 dir(s)
```

D．load 文件到内存(这里烧录的是 nor 启 uboot)

```
isvp# fatload mmc 0 0x80600000 u-boot-with-spl.bin
```

E．然后将内存中的数据写入 Nor

```
isvp# sf probe
isvp# sf erase 0x0 0x40000  --擦除 uboot 分区大小 256K
isvp# sf write 0x80600000 0x0 0x40000  --将 uboot 写入 Nor 0x0~0x40000 处
```

F. 此时已经把 uboot 烧录到 flash 中，下次启动可以直接从 flash 中的 uboot 启动。

7.2 UART 启动操作方法

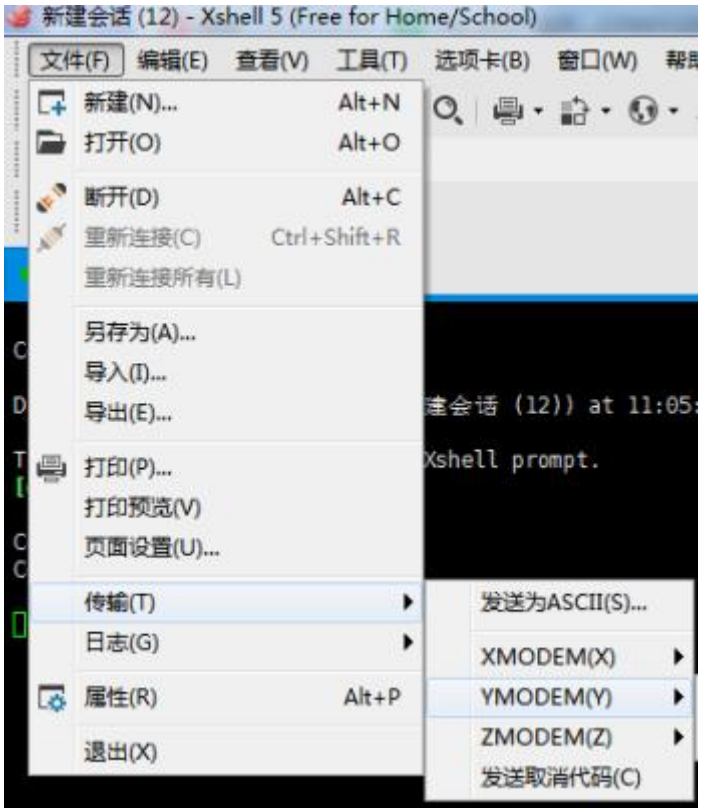
BOOT_SEL1 配置为 1，BOOT_SEL0 配置为 0 进入 UART 启动。

isvp_t41n_uart_msc 配置可以支持启动后烧录 SD/EMMC 等 MSC 接口设备。

isvp_t41n_uart_sfc 配置可以支持启动后烧录 flash 等 SFC 接口设备。

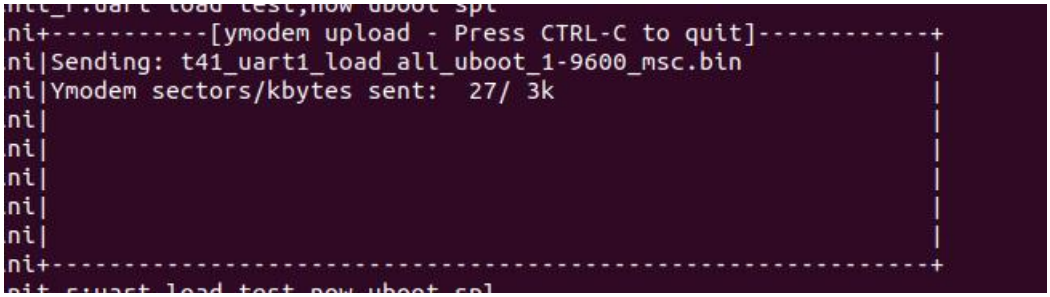
1) Xshell 操作：右键---->传输--->选择 YMODEM-->用 YMODEM 发送--->选择传输文件

图 7-1 Xshell 操作方式



2) Minicom 操作：

图 7-2 Minicom 操作方式



7.3 Wpa_supplicant 使用方法

wpa_supplicant 及 wpa_cli 使用方法。

7.3.1 概述

wpa_supplicant 是一个连接、配置 WIFI 的工具。它主要包含两个程序：wpa_supplicant 与 wpa_cli。二者的关系就是 server 与 client 的关系。通常情况下，可以通过 wpa_cli 来进行 WIFI 的配置与连接，如果有特殊的需要，可以编写应用程序直接调用 wpa_supplicant 的接口直接开发。

本文主要讲述如何通过 wpa_cli 进行 WIFI 的配置与连接。

7.3.2 使用方法

1) 启动 wpa_supplicant 应用

```
$ wpa_supplicant -D nl80211 -i wlan0 -c /etc/wpa_supplicant.conf -B
```

注意：/etc/wpa_supplicant.conf 文件里，添加下面代码。

```
ctrl_interface=/var/run/wpa_supplicant
update_config=1

network={
    ssid="TP-LINK_XXXXX"
    psk="YOUR_PASSWORD"
    key_mgmt=WPA-PSK
}
```

2）启动 wpa_cli 应用

```
$ wpa_cli -i wlan0 scan            搜索附近 wifi 网络
```

```
$ wpa_cli -i wlan0 scan_result      打印搜索 wifi 网络结果
$ wpa_cli -i wlan0 add_network      添加一个网络连接
如果要连接加密方式是：[WPA-PSK-CCMP+TKIP][WPA2-PSK-CCMP+TKIP][ESS] (wpa 加密)
wifi 名称是：wifi_name
wifi 密码是：wifi_psk
$ wpa_cli -i wlan0 set_network 0 ssid '"wifi_name"'
$ wpa_cli -i wlan0 set_network 0 psk '"wifi_psk"'
$ wpa_cli -i wlan0 enable_network 0

如果要连接加密方式是：[WEP][ESS] (wep 加密)
wifi 名称是：wifi_name
wifi 密码是：wifi_psk
$ wpa_cli -i wlan0 set_network 0 ssid '"wpa_name"'
$ wpa_cli -i wlan0 set_network 0 key_mgmt NONE
$ wpa_cli -i wlan0 set_network 0 wep_key0 '"wap_psk"'
$ wpa_cli -i wlan0 enable_network 0

如果要连接加密方式是：[ESS] (无加密)
wifi 名称是：wifi_name
$ wpa_cli -i wlan0 set_network 0 ssid '"wifi_name"'
$ wpa_cli -i wlan0 set_network 0 key_mgmt NONE
$ wpa_cli -i wlan0 enable_network 0
```

3） 分配 ip,netmask,gateway,dns

```
$ udhcpc -i wlan0
```

执行完毕，即可连接网络。

4） 保存连接

```
$ wpa_cli -i wlan0 save_config
```

5）断开连接

```
$ wpa_cli -i wlan0 disable_network 0
```

6）连接已有的连接

```
$ wpa_cli -i wlan0 list_network             列举所有保存的连接
$ wpa_cli -i wlan0 select_network 0         连接第 1 个保存的连接
$ wpa_cli -i wlan0 enable_network 0         使能第 1 个保存的连接
```

7）断开 wifi

```
$ ifconfig wlan0 down
$ killall udhcpc
$ killall wpa_supplicant
```

7.3.3 wpa_wifi_tool 使用方法

wpa_wifi_tool 是基于 wpa_supplicant 及 wpa_cli 的一个用于快速设置 wifi 的工具，方便调试时连接 wifi 使用。使用者无需运行步骤 2 中复杂的命令，即可实现 wifi 设置和连接等功能。

注：此工具仅作为调试工具使用，实际 wifi 功能开发推荐使用 wpa_cli 或者直接调用 wpa_supplicant 实现。

使用方法：

A. 运行 wpa_wifi_tool

B. 输入 help 进行命令查看

C. s 进行 SSID 扫描

D. c[n]进行 wifi 连接，连接时若为新的 SSID 则需输入密码，若为已保存的 SSID 则可以使用保存过的密码或者重新输入密码。

E. e 退出工具

其他使用方法请参考 help

7.4 4G 代码参考

7.4.1 PPP 模式

A drivers/net/usb/qmi_wwan.c

```
diff --git a/drivers/net/usb/qmi_wwan.c b/drivers/net/usb/qmi_wwan.c
index 5645921..f8f0020 100644
--- a/drivers/net/usb/qmi_wwan.c
+++ b/drivers/net/usb/qmi_wwan.c
@@ -614,7 +614,7 @@ static const struct usb_device_id products[] = {
        {QMI_GOBI_DEVICE(0x05c6, 0x9225)},      /* Sony Gobi 2000 Modem d
evice (N0279, VU730) */
        {QMI_GOBI_DEVICE(0x05c6, 0x9245)},      /* Samsung Gobi 2000 Mod
em device (VL176) */
        {QMI_GOBI_DEVICE(0x03f0, 0x251d)},      /* HP Gobi 2000 Modem dev
ice (VP412) */
-       {QMI_GOBI_DEVICE(0x05c6, 0x9215)},      /* Acer Gobi 2000 Modem
device (VP413) */
+//     {QMI_GOBI_DEVICE(0x05c6, 0x9215)},      /* Acer Gobi 2000 Modem
device (VP413) */
```

```
          {QMI_GOBI_DEVICE(0x05c6, 0x9265)},      /* Asus Gobi 2000 Modem d
evice (VR305) */
          {QMI_GOBI_DEVICE(0x05c6, 0x9235)},      /* Top Global Gobi 2000 M
odem device (VR306) */
          {QMI_GOBI_DEVICE(0x05c6, 0x9275)},      /* iRex Technologies Gob
i 2000 Modem device (VR307) */
```

B drivers/usb/serial/option.c

```
   --- a/drivers/usb/serial/option.c
   +++ b/drivers/usb/serial/option.c
   @@ -530,6 +530,22 @@ static const struct option_blacklist_info telit_le
920_blacklist = {
    };

    static const struct usb_device_id option_ids[] = {
   +#if 1 //Added by Quectel
   +      { USB_DEVICE(0x05C6, 0x9090) }, /* Quectel UC15 */
   +      { USB_DEVICE(0x05C6, 0x9003) }, /* Quectel UC20 */
   +      { USB_DEVICE(0x2C7C, 0x0125) }, /* Quectel EC25 */
   +      { USB_DEVICE(0x2C7C, 0x0121) }, /* Quectel EC21 */
   +      { USB_DEVICE(0x05C6, 0x9215) }, /* Quectel EC20 */
   +      { USB_DEVICE(0x2C7C, 0x0191) }, /* Quectel EG91 */
   +      { USB_DEVICE(0x2C7C, 0x0195) }, /* Quectel EG95 */
   +      { USB_DEVICE(0x2C7C, 0x0306) }, /* Quectel EG06/EP06/EM06 */
   +      { USB_DEVICE(0x2C7C, 0x0296) }, /* Quectel BG96 */
   +      { USB_DEVICE(0x2C7C, 0x0435) }, /* Quectel AG35 */
   +      { USB_DEVICE(0x2C7C, 0x0512) }, /* Quectel EG12/EG18 */
   +      { USB_DEVICE(0x2C7C, 0x6026) }, /* Quectel EC200 */
   +      { USB_DEVICE(0x2C7C, 0x6120) }, /* Quectel UC200 */
   +      { USB_DEVICE(0x2C7C, 0x6000) }, /* Quectel EC200/UC200 */
   +#endif
          { USB_DEVICE(OPTION_VENDOR_ID, OPTION_PRODUCT_COLT) },
          { USB_DEVICE(OPTION_VENDOR_ID, OPTION_PRODUCT_RICOLA) },
          { USB_DEVICE(OPTION_VENDOR_ID, OPTION_PRODUCT_RICOLA_LIGHT) },
   @@ -1377,6 +1393,9 @@ static struct usb_serial_driver option_1port_devi
ce = {
    #ifdef CONFIG_PM
          .suspend           = usb_wwan_suspend,
          .resume            = usb_wwan_resume,
   +#if 1 //add by Quectel
   +      .reset_resume = usb_wwan_resume,
```

```
    +#endif
     #endif
     };

    @@ -1421,6 +1440,27 @@ static int option_probe(struct usb_serial *seria
l,
                                &serial->interface->cur_altsetting->desc;
            struct usb_device_descriptor *dev_desc = &serial->dev->descripto
r;

    +#if 1 //Added by Quectel
    +       //Quectel UC20's interface 4 can be used as USB Network device
    +       if (serial->dev->descriptor.idVendor == cpu_to_le16(0x05C6) && s
erial->dev->descriptor.idProduct ==
    +                       cpu_to_le16(0x9003) && serial->interface->cur_alt
setting->desc.bInterfaceNumber >= 4)
    +               return -ENODEV;
    +       //Quectel EC20's interface 4 can be used as USB Network device
    +       if (serial->dev->descriptor.idVendor == cpu_to_le16(0x05C6) && s
erial->dev->descriptor.idProduct ==
    +                       cpu_to_le16(0x9215) && serial->interface->cur_alt
setting->desc.bInterfaceNumber >= 4)
    +               return -ENODEV;
    +       if (serial->dev->descriptor.idVendor == cpu_to_le16(0x2C7C)) {
    +               __u16 idProduct = le16_to_cpu(serial->dev->descriptor.id
Product);
    +
    +               //Quectel EC200&UC200's interface 0 can be used as USB Ne
twork device (ecm, rndis)
    +               if (serial->interface->cur_altsetting->desc.bInterfaceCl
ass != 0xFF)
    +                       return -ENODEV;
    +               //Quectel EC25&EC21&EG91&EG95&EG06&EP06&EM06&BG96&AG35&E
G12&EG18's interface 4 can be used as USB network device (qmi,ecm,mbim)
    +               if ((idProduct != 0x6026 && idProduct != 0x6126) && seria
l->interface->cur_altsetting->desc.bInterfaceNumber >= 4)
    +                       return -ENODEV;
    +       }
    +#endif
    +
            /* Never bind to the CD-Rom emulation interface */
            if (iface_desc->bInterfaceClass == 0x08)
```

```
                return -ENODEV;
```

C drivers/usb/serial/qcserial.c

```
--- a/drivers/usb/serial/qcserial.c
+++ b/drivers/usb/serial/qcserial.c
@@ -53,7 +53,7 @@ static const struct usb_device_id id_table[] = {
        {DEVICE_G1K(0x05c6, 0x9202)},   /* Generic Gobi Modem device */
        {DEVICE_G1K(0x05c6, 0x9203)},   /* Generic Gobi Modem device */
        {DEVICE_G1K(0x05c6, 0x9222)},   /* Generic Gobi Modem device */
-       {DEVICE_G1K(0x05c6, 0x9008)},   /* Generic Gobi QDL device */
+//     {DEVICE_G1K(0x05c6, 0x9008)},   /* Generic Gobi QDL device */
        {DEVICE_G1K(0x05c6, 0x9009)},   /* Generic Gobi Modem device */
        {DEVICE_G1K(0x05c6, 0x9201)},   /* Generic Gobi QDL device */
        {DEVICE_G1K(0x05c6, 0x9221)},   /* Generic Gobi QDL device */
```

D drivers/net/usb/qmi_wwan.c

```
--- a/drivers/usb/serial/usb_wwan.c
+++ b/drivers/usb/serial/usb_wwan.c
@@ -456,8 +456,22 @@ static struct urb *usb_wwan_setup_urb(struct usb_s
erial_port *port,

        /* Fill URB using supplied data. */
        usb_fill_bulk_urb(urb, serial->dev,
-                         usb_sndbulkpipe(serial->dev, endpoint) | dir,
-                         buf, len, callback, ctx);
+                        usb_sndbulkpipe(serial->dev, endpoint) | dir,
+                        buf, len, callback, ctx);
+
+#if 1 //Added by Quectel for Zero Packet
+       if (dir == USB_DIR_OUT) {
+               struct usb_device_descriptor *desc = &serial->dev->descr
iptor;
+               if (desc->idVendor == cpu_to_le16(0x05C6) && desc->idProd
uct == cpu_to_le16(0x9090))
+                       urb->transfer_flags |= URB_ZERO_PACKET;
+               if (desc->idVendor == cpu_to_le16(0x05C6) && desc->idProd
uct == cpu_to_le16(0x9003))
+                       urb->transfer_flags |= URB_ZERO_PACKET;
+               if (desc->idVendor == cpu_to_le16(0x05C6) && desc->idProd
uct == cpu_to_le16(0x9215))
+                       urb->transfer_flags |= URB_ZERO_PACKET;
```

```
+               if (desc->idVendor == cpu_to_le16(0x2C7C))
+                       urb->transfer_flags |= URB_ZERO_PACKET;
+       }
+#endif

        return urb;
 }
```

7.4.2 Wwan 模式

A drivers/net/usb/qmi_wwan.c 添加如下代码

```
--- a/drivers/net/usb/qmi_wwan.c
+++ b/drivers/net/usb/qmi_wwan.c
@@ -20,6 +20,72 @@
 #include <linux/usb/usbnet.h>
 #include <linux/usb/cdc-wdm.h>

+#if 1 //Added by Quectel
+#include <linux/etherdevice.h>
+struct sk_buff *qmi_wwan_tx_fixup(struct usbnet *dev, struct sk_buff *
skb, gfp_t flags)
+{
+       if (dev->udev->descriptor.idVendor != cpu_to_le16(0x2C7C))
+               return skb;
+       //Skip Ethernet header from message
+       if (dev->net->hard_header_len == 0)
+               return skb;
+       else
+               skb_reset_mac_header(skb);
+       if (skb_pull(skb, ETH_HLEN)) {
+               return skb;
+       } else {
+               dev_err(&dev->intf->dev, "Packet Dropped ");
+       }
+       // Filter the packet out, release it
+       dev_kfree_skb_any(skb);
+       return NULL;
+}
+
+#include <linux/version.h>
+#if (LINUX_VERSION_CODE < KERNEL_VERSION( 3,9,1 ))
```

```
+static int qmi_wwan_rx_fixup(struct usbnet *dev, struct sk_buff *skb)
+{
+       __be16 proto;
+       if (dev->udev->descriptor.idVendor != cpu_to_le16(0x2C7C))
+              return 1;
+       /* This check is no longer done by usbnet */
+       if (skb->len < dev->net->hard_header_len)
+              return 0;
+       switch (skb->data[0] & 0xf0) {
+              case 0x40:
+                     proto = htons(ETH_P_IP);
+                     break;
+              case 0x60:
+                     proto = htons(ETH_P_IPV6);
+                     break;
+              case 0x00:
+                     if (is_multicast_ether_addr(skb->data))
+                            return 1;
+                     /* possibly bogus destination - rewrite just in ca
se */
+                     skb_reset_mac_header(skb);
+                     goto fix_dest;
+              default:
+                     /* pass along other packets without modifications
 */
+                     return 1;
+       }
+       if (skb_headroom(skb) < ETH_HLEN)
+              return 0;
+       skb_push(skb, ETH_HLEN);
+       skb_reset_mac_header(skb);
+       eth_hdr(skb)->h_proto = proto;
+       memset(eth_hdr(skb)->h_source, 0, ETH_ALEN);
+fix_dest:
+       memcpy(eth_hdr(skb)->h_dest, dev->net->dev_addr, ETH_ALEN);
+       return 1;
+}
+/* very simplistic detection of IPv4 or IPv6 headers */
+static bool possibly_iphdr(const char *data)
+{
+       return (data[0] & 0xd0) == 0x40;
+}
```

```
    +#endif
    +#endif
    +
     /* This driver supports wwan (3G/LTE/?) devices using a vendor
      * specific management protocol called Qualcomm MSM Interface (QMI) -
      * in addition to the more common AT commands over serial interface
    @@ -332,6 +398,33 @@ next_desc:
                    dev->net->dev_addr[0] &= 0xbf;  /* clear "IP" bit */
            }
            dev->net->netdev_ops = &qmi_wwan_netdev_ops;
    +
    +#if 1 //Added by Quectel
    +       if (dev->udev->descriptor.idVendor == cpu_to_le16(0x2C7C)) {
    +               if (intf->cur_altsetting->desc.bInterfaceClass != 0xff)
{
    +                       dev_dbg(&intf->dev, "Quectel module not qmi_wwan
mode! please check 'at+qcfg=\"usbnet\"'\n");
    +                       return -ENODEV;
    +               }
    +               dev_dbg(&intf->dev, "Quectel EC25&EC21&EG91&EG95&EG06&EP
06&EM06&EG12&EP12&EM12&EG16&EG18&BG96&AG35 work on RawIP mode\n");
    +               dev->net->flags |= IFF_NOARP;
    +#if (LINUX_VERSION_CODE < KERNEL_VERSION( 3,9,1 ))
    +               /* make MAC addr easily distinguishable from an IP header
 */
    +               if (possibly_iphdr(dev->net->dev_addr)) {
    +                       dev->net->dev_addr[0] |= 0x02; /* set local assig
nment bit */
    +                       dev->net->dev_addr[0] &= 0xbf; /* clear "IP" bit
*/
    +               }
    +#endif
    +               usb_control_msg(
    +                               interface_to_usbdev(intf),
    +                               usb_sndctrlpipe(interface_to_usbdev(intf),
 0),
    +                               0x22, //USB_CDC_REQ_SET_CONTROL_LINE_STATE
    +                               0x21, //USB_DIR_OUT | USB_TYPE_CLASS | USB
_RECIP_INTERFACE
    +                               1, //active CDC DTR
    +                               intf->cur_altsetting->desc.bInterfaceNumbe
r,
```

```
+                               NULL, 0, 100);
+       }
+#endif
+
 err:
        return status;
 }
@@ -416,6 +509,10 @@ static const struct driver_info    qmi_wwan_info =
{
        .unbind         = qmi_wwan_unbind,
        .manage_power   = qmi_wwan_manage_power,
        .rx_fixup       = qmi_wwan_rx_fixup,
+#if 1 //Added by Quectel
+       .tx_fixup = qmi_wwan_tx_fixup,
+       .rx_fixup = qmi_wwan_rx_fixup,
+#endif
 };

 #define HUAWEI_VENDOR_ID       0x12D1
@@ -434,6 +531,31 @@ static const struct driver_info    qmi_wwan_info =
{
        QMI_FIXED_INTF(vend, prod, 0)

 static const struct usb_device_id products[] = {
+#if 1 //Added by Quectel
+#ifndef QMI_FIXED_INTF
+       /* map QMI/wwan function by a fixed interface number */
+#define QMI_FIXED_INTF(vend, prod, num) \
+       .match_flags = USB_DEVICE_ID_MATCH_DEVICE |
+       USB_DEVICE_ID_MATCH_INT_INFO, \
+               .idVendor = vend, \
+               .idProduct = prod, \
+               .bInterfaceClass = 0xff, \
+               .bInterfaceSubClass = 0xff, \
+               .bInterfaceProtocol = 0xff, \
+               .driver_info = (unsigned long)&qmi_wwan_force_int##num,
+#endif
+       { QMI_FIXED_INTF(0x05C6, 0x9003, 4) }, /* Quectel UC20 */
+       { QMI_FIXED_INTF(0x2C7C, 0x0125, 4) }, /* Quectel EC25 */
+       { QMI_FIXED_INTF(0x2C7C, 0x0121, 4) }, /* Quectel EC21 */
+       { QMI_FIXED_INTF(0x05C6, 0x9215, 4) }, /* Quectel EC20 */
+       { QMI_FIXED_INTF(0x2C7C, 0x0191, 4) }, /* Quectel EG91 */
```

```
    +       { QMI_FIXED_INTF(0x2C7C, 0x0195, 4) }, /* Quectel EG95 */
    +       { QMI_FIXED_INTF(0x2C7C, 0x0306, 4) }, /* Quectel EG06/EP06/EM06
 */
    +       { QMI_FIXED_INTF(0x2C7C, 0x0512, 4) }, /* Quectel EG12/EP12/EM12
/EG16/EG18 */
    +       { QMI_FIXED_INTF(0x2C7C, 0x0296, 4) }, /* Quectel BG96 */
    +       { QMI_FIXED_INTF(0x2C7C, 0x0435, 4) }, /* Quectel AG35 */
    +#endif
    +
            /* 1. CDC ECM like devices match on the control interface */
            {       /* Huawei E392, E398 and possibly others sharing both dev
ice id and more... */
                    USB_VENDOR_AND_INTERFACE_INFO(HUAWEI_VENDOR_ID, USB_CLAS
S_VENDOR_SPEC, 1, 9),
    @@ -614,7 +736,7 @@ static const struct usb_device_id products[] = {
            {QMI_GOBI_DEVICE(0x05c6, 0x9225)},      /* Sony Gobi 2000 Modem d
evice (N0279, VU730) */
            {QMI_GOBI_DEVICE(0x05c6, 0x9245)},      /* Samsung Gobi 2000 Mod
em device (VL176) */
            {QMI_GOBI_DEVICE(0x03f0, 0x251d)},      /* HP Gobi 2000 Modem dev
ice (VP412) */
    -       {QMI_GOBI_DEVICE(0x05c6, 0x9215)},      /* Acer Gobi 2000 Modem
device (VP413) */
    +//     {QMI_GOBI_DEVICE(0x05c6, 0x9215)},      /* Acer Gobi 2000 Modem
device (VP413) */
            {QMI_GOBI_DEVICE(0x05c6, 0x9265)},      /* Asus Gobi 2000 Modem d
evice (VR305) */
            {QMI_GOBI_DEVICE(0x05c6, 0x9235)},      /* Top Global Gobi 2000 M
odem device (VR306) */
            {QMI_GOBI_DEVICE(0x05c6, 0x9275)},      /* iRex Technologies Gob
i 2000 Modem device (VR307) */
```

B drivers/net/usb/qmi_wwan.c

```
     --- a/drivers/usb/serial/option.c
    +++ b/drivers/usb/serial/option.c
    @@ -530,6 +530,19 @@ static const struct option_blacklist_info telit_le
920_blacklist = {
     };

     static const struct usb_device_id option_ids[] = {
    +#if 1 //Added by Quectel
```

```
    +       { USB_DEVICE(0x05C6, 0x9090) }, /* Quectel UC15 */
    +       { USB_DEVICE(0x05C6, 0x9003) }, /* Quectel UC20 */
    +       { USB_DEVICE(0x2C7C, 0x0125) }, /* Quectel EC25 */
    +       { USB_DEVICE(0x2C7C, 0x0121) }, /* Quectel EC21 */
    +       { USB_DEVICE(0x05C6, 0x9215) }, /* Quectel EC20 */
    +       { USB_DEVICE(0x2C7C, 0x0191) }, /* Quectel EG91 */
    +       { USB_DEVICE(0x2C7C, 0x0195) }, /* Quectel EG95 */
    +       { USB_DEVICE(0x2C7C, 0x0306) }, /* Quectel EG06/EP06/EM06 */
    +       { USB_DEVICE(0x2C7C, 0x0512) }, /* Quectel EG12/EP12/EM12/EG16/E
G18 */
    +       { USB_DEVICE(0x2C7C, 0x0296) }, /* Quectel BG96 */
    +       { USB_DEVICE(0x2C7C, 0x0435) }, /* Quectel AG35 */
    +#endif
            { USB_DEVICE(OPTION_VENDOR_ID, OPTION_PRODUCT_COLT) },
            { USB_DEVICE(OPTION_VENDOR_ID, OPTION_PRODUCT_RICOLA) },
            { USB_DEVICE(OPTION_VENDOR_ID, OPTION_PRODUCT_RICOLA_LIGHT) },
    @@ -1245,7 +1258,7 @@ static const struct usb_device_id option_ids[] = {
            { USB_DEVICE(CINTERION_VENDOR_ID, CINTERION_PRODUCT_AHXX) },
            { USB_DEVICE(CINTERION_VENDOR_ID, CINTERION_PRODUCT_PLXX),
                    .driver_info = (kernel_ulong_t)&net_intf4_blacklist },
    -       { USB_DEVICE(CINTERION_VENDOR_ID, CINTERION_PRODUCT_HC28_MDM) },
    +       { USB_DEVICE(CINTERION_VENDOR_ID, CINTERION_PRODUCT_HC28_MDM) },
            { USB_DEVICE(CINTERION_VENDOR_ID, CINTERION_PRODUCT_HC28_MDMNET)
 },
            { USB_DEVICE(SIEMENS_VENDOR_ID, CINTERION_PRODUCT_HC25_MDM) },
            { USB_DEVICE(SIEMENS_VENDOR_ID, CINTERION_PRODUCT_HC25_MDMNET)
 },
    @@ -1377,6 +1390,9 @@ static struct usb_serial_driver option_1port_devi
ce = {
     #ifdef CONFIG_PM
            .suspend           = usb_wwan_suspend,
            .resume            = usb_wwan_resume,
    +#if 1 //Added by Quectel
    +       .reset_resume = usb_wwan_resume,
    +#endif
     #endif
     };

    @@ -1444,6 +1460,22 @@ static int option_probe(struct usb_serial *seria
l,
                iface_desc->bInterfaceClass != USB_CLASS_CDC_DATA)
                    return -ENODEV;
```

```
    +#if 1 //Added by Quectel
    +       //Quectel UC20's interface 4 can be used as USB network device
    +       if (serial->dev->descriptor.idVendor == cpu_to_le16(0x05C6)
    +                       && serial->dev->descriptor.idProduct == cpu_to_le
16(0x9003)
    +                       && serial->interface->cur_altsetting->desc.bInter
faceNumber >= 4)
    +               return -ENODEV;
    +       //Quectel EC20's interface 4 can be used as USB network device
    +       if (serial->dev->descriptor.idVendor == cpu_to_le16(0x05C6)
    +                       && serial->dev->descriptor.idProduct == cpu_to_le
16(0x9215)
    +                       && serial->interface->cur_altsetting->desc.bInter
faceNumber >= 4)
    +               return -ENODEV;
    +       //Quectel EC25&EC21&EG91&EG95&EG06&EP06&EM06&BG96/AG35's interfa
ce 4 can be used as USB network device
    +       if (serial->dev->descriptor.idVendor == cpu_to_le16(0x2C7C)
    +                       && serial->interface->cur_altsetting->desc.bInter
faceNumber >= 4)
    +               return -ENODEV;
    +#endif
            /* Store device id so we can use it during attach. */
            usb_set_serial_data(serial, (void *)id);
```

C drivers/usb/serial/qcserial.c

```
    --- a/drivers/usb/serial/qcserial.c
    +++ b/drivers/usb/serial/qcserial.c
    @@ -78,7 +78,7 @@ static const struct usb_device_id id_table[] = {
            {USB_DEVICE(0x03f0, 0x241d)},   /* HP Gobi 2000 QDL device (VP41
2) */
            {USB_DEVICE(0x03f0, 0x251d)},   /* HP Gobi 2000 Modem device (VP
412) */
            {USB_DEVICE(0x05c6, 0x9214)},   /* Acer Gobi 2000 QDL device (VP
413) */
    -       {USB_DEVICE(0x05c6, 0x9215)},   /* Acer Gobi 2000 Modem device
(VP413) */
    +//     {USB_DEVICE(0x05c6, 0x9215)},   /* Acer Gobi 2000 Modem device
(VP413) */
            {USB_DEVICE(0x05c6, 0x9264)},   /* Asus Gobi 2000 QDL device (VR
```

```
305) */
           {USB_DEVICE(0x05c6, 0x9265)},   /* Asus Gobi 2000 Modem device
(VR305) */
           {USB_DEVICE(0x05c6, 0x9234)},   /* Top Global Gobi 2000 QDL devi
ce (VR306) */
```

D drivers/usb/serial/usb_wwan.c

```
    --- a/drivers/usb/serial/usb_wwan.c
    +++ b/drivers/usb/serial/usb_wwan.c
    @@ -459,6 +459,20 @@ static struct urb *usb_wwan_setup_urb(struct usb_s
erial_port *port,
                              usb_sndbulkpipe(serial->dev, endpoint) | dir,
                              buf, len, callback, ctx);

    +#if 1 //Added by Quectel for zero packet
    +       if (dir == USB_DIR_OUT) {
    +               struct usb_device_descriptor *desc = &serial->dev->descr
iptor;
    +               if (desc->idVendor == cpu_to_le16(0x05C6) && desc->idProd
uct == cpu_to_le16(0x9090))
    +                       urb->transfer_flags |= URB_ZERO_PACKET;
    +               if (desc->idVendor == cpu_to_le16(0x05C6) && desc->idProd
uct == cpu_to_le16(0x9003))
    +                       urb->transfer_flags |= URB_ZERO_PACKET;
    +               if (desc->idVendor == cpu_to_le16(0x05C6) && desc->idProd
uct == cpu_to_le16(0x9215))
    +                       urb->transfer_flags |= URB_ZERO_PACKET;
    +               if (desc->idVendor == cpu_to_le16(0x2C7C))
    +                       urb->transfer_flags |= URB_ZERO_PACKET;
    +       }
    +#endif
    +
            return urb;
    }
```

7.5 版本说明

ISVP_Devkit 版本升级说明

ISVP 版本升级（主要以软件资源及 SDK 升级为主）是一项持续的工作，新的版本会提供新的功能、修复问题、以及系统优化。

ISVP 软件系统主要包含：uboot，kernel，drivers，lib，sensor bin，rootfs，toolchain 几个部分，对于每次 Devkit 更新，这几个部分间可能存在一定的依赖关系，需要同步升级，否则可能会出现错误。

版本升级有以下注意事项：

A. 开发人员需要认真阅读 ChangeLog，了解有哪些更新。

B. 一般情况下建议 drivers(ISP, Sensor)、sensor bin、与 lib 同时更新。

C. kernel 更新较少，但是如果需要更新 kernel，需要完全重新编译所有 ko，其中 ISP driver 和 Sensor driver 的编译顺序是先编译前者后编译后者。

D. ISP driver 与 Sensor driver 及 Sensor bin 文件三者有匹配关系，不匹配可能造成图像异常。

E. API 更新需要完全替换 lib 与头文件，头文件与 lib 不匹配可能造成程序崩溃。

新的版本也可能会带来新的问题，ISVP 的开发团队会尽力保证新版本的正确性。若开发者发现了 Bug，请及时向君正的技术支持人员反馈，描述现象与复现情况等细节，ISVP 的开发团队会第一时间进行处理。

调试过程中出现问题，需要与技术支持人员进行沟通反馈。正确的提供 Log 以及记录问题，可以有效提高解决问题的效率。主要有以下几点：

A. 正确的描述现象，复现方法，复现概率

B. 提供 Log 以及截图

C. 提供 SDK 版本信息（或提供完整的 logcat）

D. 找到复现规律