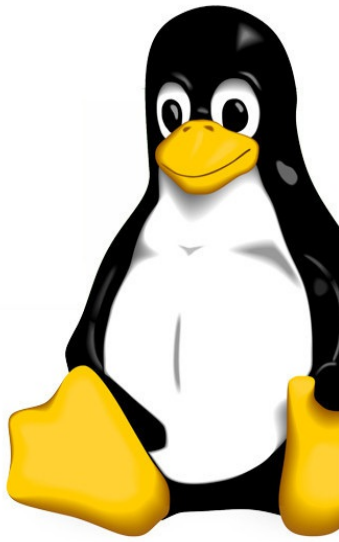

# MYD-AM437X 系列 Linux 4.1.18 开发手册

MYiR™ Make Your Idea Real

## MYD-C437X

## MYD-C437X-PRU

# 目錄

# MYD-AM437X 系列 Linux-4.1.18开发手册

## 前言

本文主要讲述如何在MYD-AM437X系列开发板上安装运行Linux系统以及嵌入式Linux驱动和应用程序的开发流程。其中包括开发环境的搭建、源码编译、Linux应用程序的实例分析、映像的下载等。

本文档适合有一定开发经验的嵌入式linux开发工程师。

版本历史:

| 版本号 | 描述 | 时间 |
| --- | --- | --- |
| V1.0 | 初始版本 | 2017.06.19 |

硬件版本:

本文档仅适用于米尔MYD-C437X和MYD-C437X-PRU开发板。

| MYD-C437X | MYD-C437X-PRU |
| --- | --- |
| am437x通用开发板 | am437x PRU开发板 |

> 注意: 开发板Linux系统默认的root账户密码为空。

# 1. 软件资源介绍

MYD-AM437X系列开发板出厂附带嵌入式Linux系统开发所需要的交叉编译工具链，U-boot源代码，Linux 内核和各驱动模块的源代码，以及各种配套的开发调试工具，应用开发样例等。
本节内容以表格形式描述MYD-C437X和MYD-C437X-PRU的软件资源。
表 1-1 软件资源列表

| 类别 | 名称 | 描述 | 源码 | MYD-C437X | MYD-C437X-PRU |
|---|---|---|---|---|---|
| 引导程序 | U-boot201605 | 一，二级引导程序SPL和U-Boot，负责系统初始化和引导内核 | YES | √ | √ |
| 内核 | Linux 4.1.18 | 专为MYD-AM437X系列开发板的硬件制定的Linux内核 | YES | √ | √ |
| 驱动 | USB Host | USB Host驱动，支持OHCI和EHCI两种传输模式 | YES | √ | √ |
| 驱动 | USB Device | USB Device驱动（Gadget） | YES | √ | √ |
| 驱动 | Ethernet | 以太网驱动 | YES | √ | √ |
| 驱动 | PRU Ethernet | 工业以太网驱动 | YES | × | √ |
| 驱动 | MMC/SD | MMC/SD卡驱动 | YES | √ | √ |
| 驱动 | EMMC | EMMC驱动 | YES | √ | √ |
| 驱动 | I2C | I2C驱动 | YES | √ | √ |
| 驱动 | SPI | SPI驱动 | YES | √ | √ |
| 驱动 | LCD | LCD屏驱动，默认7寸800*480液晶屏 | YES | √ | √ |
| 驱动 | RTC | 内置RTC时钟驱动 | YES | √ | √ |
| 驱动 | RX-8025T | 外置RTC时钟驱动 | YES | x | √ |
| 驱动 | RS485 | RS485驱动，提供源码 | YES | √ | √ |
| 驱动 | ADC | ADC驱动，提供源码 | YES | √ | √ |
| 驱动 | Resistive TouchScreen | 4线电阻触摸屏驱动 | YES | √ | √ |
| 驱动 | Capacitive TouchScreen | FT5X06电容触摸 | YES | √ | √ |
| 驱动 | UART | 串口驱动 | YES | √ | √ |
| 驱动 | CAN | CAN驱动 | YES | √ | √ |
| 驱动 | PMU | 电源管理驱动 | YES | √ | √ |
| 驱动 | LED | LED驱动，包括GPIO LED和PWM LED驱动 | YES | √ | √ |
| 驱动 | Button | GPIO Button 驱动 | YES | √ | √ |
| 驱动 | Camera | 摄像头驱动 | YES | √ | √ |
| 驱动 | Audio | 音频驱动 | YES | √ | × |

| | | | | | |
|---|---|---|---|---|---|
| 文件系统 | RAMDISK | 基于Buildroot定制的文件系统 | 二进制压缩包 | √ | √ |
| 文件系统 | UBIFS | 基于Buildroot定制的Qt文件系统 | 二进制压缩包 | √ | √ |
| 应用程序 | CAN | CAN 测试程序 | YES | √ | √ |
| 应用程序 | EEPROM | EEPROM测试程序 | YES | √ | √ |
| 应用程序 | FrameBuffer | 显示设备测试程序 | YES | √ | √ |
| 应用程序 | Keypad | 按键测试程序 | YES | √ | √ |
| 应用程序 | LED | LED测试程序 | YES | √ | √ |
| 应用程序 | PRU | PRU LED测试程序 | YES | × | √ |
| 应用程序 | RTC | RTC时钟测试实验 | YES | √ | √ |
| 应用程序 | GPIO | GPIO测试程序 | YES | √ | √ |
| 应用程序 | RS232 | RS232测试程序 | YES | √ | √ |
| 应用程序 | RS485 | RS485测试程序 | YES | √ | √ |
| 应用程序 | Camera | 双摄像头测试程序 | YES | √ | √ |
| 应用程序 | Audio | 音频测试程序 | YES | √ | × |
| 应用程序 | Qt | Qt环境以及演示程序 | YES | √ | √ |
| 工具 | Cross Compiler | Linaro GCC 5.3 | BIN | √ | √ |
| 工具 | Win32DiskImager | TF/SD镜像烧写工具 | EXE | √ | √ |
| 工具 | Format Tools | TF/SD格式化工具 | EXE | √ | √ |

# 2. 部署开发环境

本节主要介绍开发过程中所使用的软件开发环境选择，硬件调试环境的搭建和交叉编译工具链的配置以及验证。

软件开发环境：

- Ubuntu12.04/14.04/16.04 64位桌面版
- Windows7/Windows10主机

交叉编译器：

- gcc-linaro-5.3-2016.02-x86_64_arm-linux-gnueabihf.tar.xz

硬件调试环境：

**MYD-C437X**

将RS232电平调试串口J16（图中Debug UART）通过USB转RS232电平调试线连到PC上，并设置PC端串口的波特率设为115200，数据位为8，停止位为1，无奇偶校验。如需要网络调试，请用网线连接开发板J11(图中Ethernet 0)。
具体如下图：

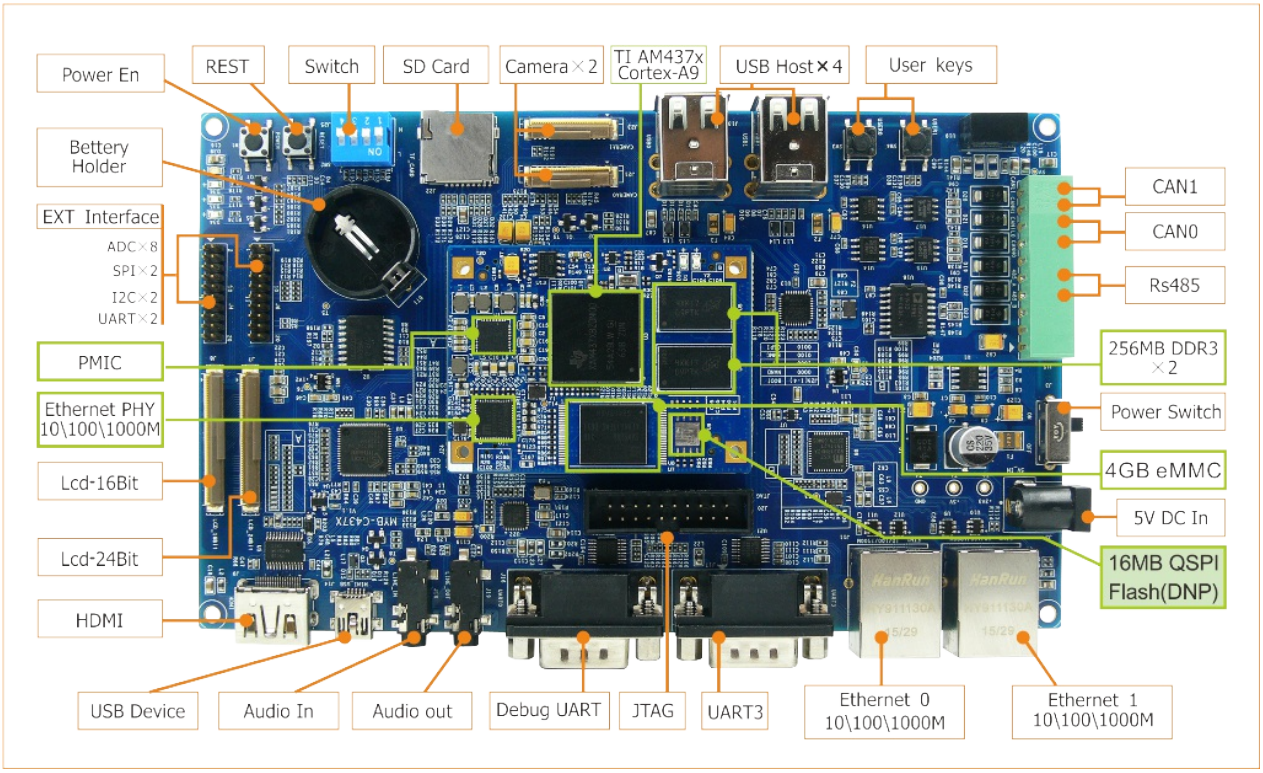

图2-1 MYD-C437X接口定义

**MYD-C437X-PRU**

将TTL电平调试串口J25（图中Debug UART）通过USB转TTL电平调试线连到PC上，并设置PC端串口的波特率设为115200，数据位为8，停止位为1，无奇偶校验。如需要网络调试，请用网线连接开发板J6(图中Giga Ethernet)。

具体如下图：

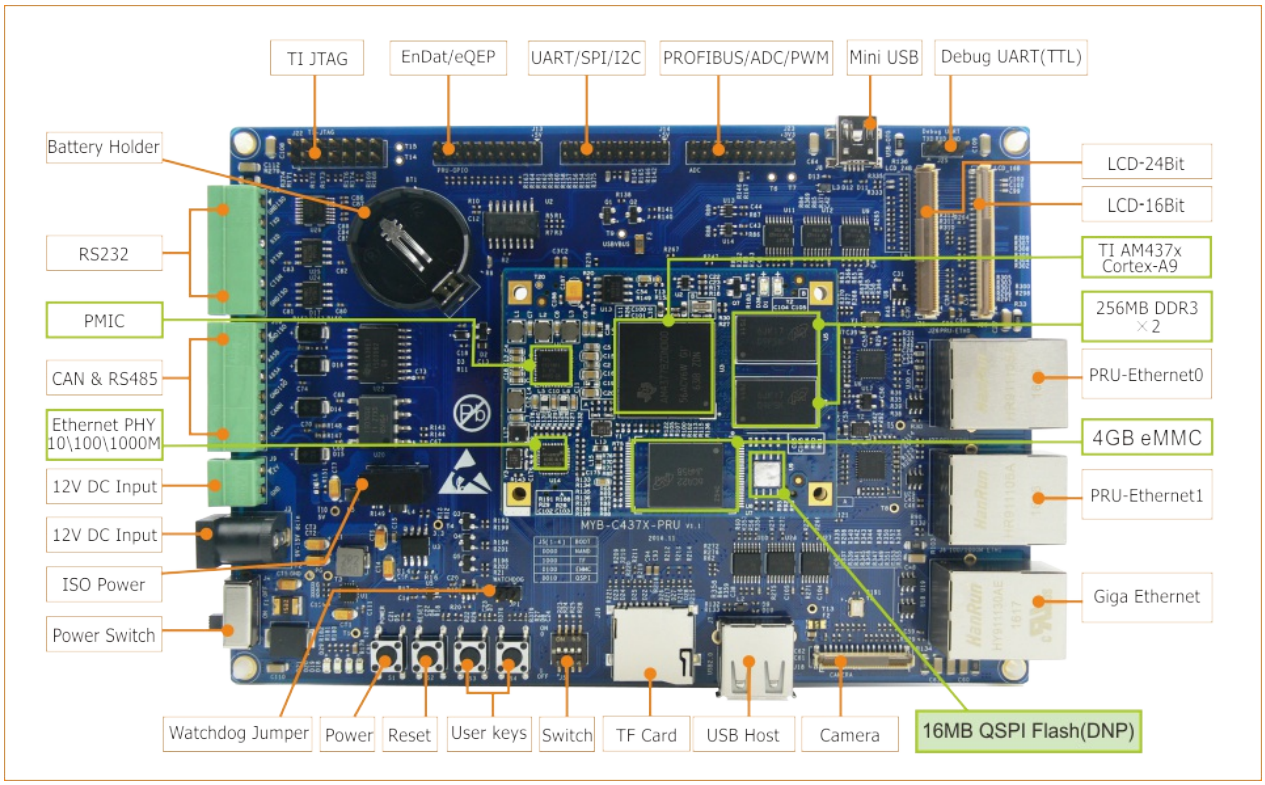


图2-2 MYD-C437X-PRU接口定义

建立工作目录:

创建工作目录, 并拷贝MYD-AM437X系列开发板出厂附带资料04-Linux_Source目录到Linux开发主机中， `<WORKDIR>` 具体目录用户可根据实际情况调整，后文不再赘述，如下所示：

```
$ mkdir -p <WORKDIR>
$ cp /media/cdrom/04-Linux_Source/* <WORKDIR> -rf
$ ls  <WORKDIR>
Bootloader/  Examples/  Filesystem/  Kernel/  ToolChain/  Tools/
```

## 2.1 安装工具

Linux编译过程中需要用到一些常用的开发工具或库，比如build-essential（提供编译程序必须软件包的列表信息）zip unzip xz-utils解压缩工具等，这里先进行安装。后续编译过程中发现缺少某些工具或库文件，也可以用类似的方法安装。

**Ubuntu 12.04**

```
$ sudo apt-get install build-essential git-core libncurses5-dev
$ sudo apt-get install flex bison texinfo zip unzip zlib1g-dev gettext
$ sudo apt-get install gperf libsdl-dev libesd0-dev libwxgtk2.6-dev
$ sudo apt-get install uboot-mkimage
$ sudo apt-get install g++ xz-utils
```

**Ubuntu 14.04**

```
$ sudo apt-get install build-essential git-core libncurses5-dev u-boot-tools
$ sudo apt-get install flex bison texinfo zip unzip zlib1g-dev gettext
$ sudo apt-get install gperf libsdl-dev libesd0-dev
$ sudo apt-get install g++ xz-utils
$ sudo apt-get install subversion
```

**Ubuntu 16.04**

```
$ sudo apt-get install build-essential git-core libncurses5-dev u-boot-tools
$ sudo apt-get install flex bison texinfo zip unzip zlib1g-dev gettext
$ sudo apt-get install gperf libsdl-dev libesd0-dev
$ sudo apt-get install g++ xz-utils
$ sudo apt-get install subversion
```

在64位的Ubuntu系统中，还需要安装一些32位的运行库，如下所示:

```
$sudo apt-get install libc6-i386 lib32stdc++6 lib32z1
```

## 2.2 设置交叉编译工具

设置交叉编译工具主要是设置PATH， ARCH和CROSS_COMPILE三个环境变量，下面介绍具体设置方法。

```
$ cd <WORKDIR>/ToolChain
$ tar Jxvf  gcc-linaro-5.3-2016.02-x86_64_arm-linux-gnueabihf.tar.xz
$ export PATH=$PATH:<WORKDIR>/ToolChain/gcc-linaro-5.3-2016.02-x86_64_arm-linux-gnueabihf/bin
$ export ARCH=arm
$ export CROSS_COMPILE=arm-linux-gnueabihf-
```

执行完“export”命令后，该设置只对当前终端有效，如需永久修改，请修改用户配置文件, Ubuntu系统下，修改如下：

```
vi ~/.profile
```

在行尾添加或修改：

```
export ARCH=arm
export CROSS_COMPILE=arm-linux-gnueabihf-
export PATH=$PATH:<WORKDIR>/ToolChain/gcc-linaro-5.3-2016.02-x86_64_arm-linux-gnueabihf/bin
```

测试环境变量：

```
$source ~/.profile
$echo $ARCH
arm
$echo $CROSS_COMPILE
arm-linux-gnueabihf-
```

测试交叉编译器：

```
$ arm-linux-gnueabihf-gcc -v
Using built-in specs.
COLLECT_GCC=arm-linux-gnueabihf-gcc
......
Thread model: posix
gcc version 5.3.1 20160113 (Linaro GCC 5.3-2016.02)
```

# 3. 构建系统

Linux平台上有许多开源的嵌入式linux系统构建框架，这些框架极大的方便了开发者进行嵌入式系统的定制化构建，目前比较常见的有openWRT, Buildroot, Yocto, Arago等等。其中Buildroot功能强大，使用简单，而且采用了类似于linux kernel的配置和编译框架，所以受到广大嵌入式开发人员的欢迎。

本节重点介绍使用Buildroot构建文件系统和u-boot, kernel镜像的方法，并从这三个部分入手，描述如何使用Buildroot构建一个适合MYD-AM437X系列开发板的嵌入式Linux系统。
在构建文件系统时，还简要介绍了如何通过Buildroot将QT5图形系统集成到文件系统中, 方便用户后续开发QT5的应用程序。

TI 官方提供了使用Arago构建系统的方法和相应的文件系统镜像文件，也可以在这款开发板上运行。

## 3.1 Bootloader

- 进入Bootloader目录，解压U-boot源码：

```
$ cd <WORKDIR>/Bootloader
$ tar -zxvf myir-u-boot.tar.gz
$ cd myir-u-boot
```

- 开始编译U-Boot：

不同的开发板对应不同的配置文件，配置文件位于 `myir-u-boot/configs/` 目录

| 开发板 | 编译选项 | 输出文件 |
| --- | --- | --- |
| MYD-C437X | myd_c437x_evm_defconfig | MLO和u-boot.img |
| MYD-C437X-PRU | myd_c437x_idk_defconfig | MLO和u-boot.img |

下面以MYD-C437X开发板为例，说明u-boot的编译过程：

```
$ make distclean
$ make myd_c437x_evm_defconfig
$ make
```

其中第二个make的配置选项见上表，MYD-C437X-PRU的编译与MYD-C437X类似。

编译完成之后生成MLO和u-boot.img可以用于TF卡启动和EMMC启动，优先选择从TF卡启动，启动过程中，通过按空格键可以进入U-Boot命令控制台执行需要的命令。例如：

```
># help            -- 获取帮助
># echo            -- 查看变量
```

## 3.2 编译Linux内核

- 进入Kernel目录，解压内核源码：

```
$ cd <WORKDIR>/Kernel
$ tar -zxvf myir-kernel.tar.gz
$ cd myir-kernel
```

- 开始编译Kernel：

不同的开发板对应不同的配置文件，配置文件位于*myir-kernel/arch/arm/configs/*目录

| 开发板 | 内核配置 |
| --- | --- |
| MYD-C437X | myd_c437x_evm_defconfig |
| MYD-C437X-PRU | myd_c437x_idk_defconfig |

下面以MYD-C437X开发板为例，说明kernel的编译过程：

```
$ make mrproper
$ make myd_c437x_evm_defconfig
$ make zImage
$ make dtbs
```

其中第二个make的配置选项见上表，MYD-C437X-PRU的编译与MYD-C437X类似。

编译完成后，在arch/arm/boot目录下生成zImage文件, 在arch/arm/boot/dts目录下生成设备树的二进制.dtb文件, 同一块开发板，适当修改DTS文件可以适应于不同的硬件配置。

不同开发板有不同的设备树文件，设备树文件位于*myir-kernel/arch/arm/boot/dts*目录

| 开发板 | 设备树 |
| --- | --- |
| MYD-C437X | myd_c437x_evm.dts、myd_c437x_evm_hdmi.dts |
| MYD-C437X-PRU | myd_c437x_idk.dts、myd_c437x_idk_lcd.dts |

例如AM4372,AM4376和AM4377三种型号的处理器，不具备SGX图形加速的功能，所以需要在设备树中禁用SGX，如下：

```
&sgx {
        status = "disabled";
};
```

反之，如果选择AM4378或AM4379这两款支持SGX图形加速的处理器，则需要使能SGX，如下：

```
&sgx {
        status = "okay";
};
```

MYD-C437X-PRU开发板中LCD和PRU Ethernet复用，也可以修改设备树中GPIO4的配置，通过GPIO4-19和GPIO4-21来选择。 当GPIO4-19和GPIO4-21都为低时，选择PRU Ethernet和CAMERA0可用，LCD不能工作，所以同时还要设置&dss节点status="disabled", &vpfe0节点status="okay", pruss1_eth节点status="okay"，如下所示：

```
/*
 * File: myd_c437x_idk.dts
 */

......

    gpio4_pins: gpio4_pins_default {
        pinctrl-single,pins = <
        0x1fc ( PIN_OUTPUT_PULLUP | MUX_MODE7 ) /* (AE23) cam1_data5.gpio4[19] */
        0x204 ( PIN_OUTPUT_PULLUP | MUX_MODE7 ) /* (AE24) cam1_data7.gpio4[21] */
        >;
    };

......

&gpio4 {
    pinctrl-names = "default";
    pinctrl-0 = <&gpio4_pins>;
    status = "okay";

    p19 {
        gpio-hog;
        gpios = <19 GPIO_ACTIVE_HIGH>;
        output-low;
        line-name = "SelPRUorLCDEN";
    };
    p21 {
        gpio-hog;
        gpios = <21 GPIO_ACTIVE_HIGH>;
        output-low;
        line-name = "SelPRUorLCDSEL";
    };
    /* SelPRUorLCDEN enable selects between PRU_ETH and LCD :
     * P19(OE)    | P21 (SEL)    | FUNCTION
     *----------|-----------|----------
     * LOW          |  LOW        |  PRU_ETH + CAMERA0
     * LOW          |     HIGHT      |     LCD
     * HIGH          |     ANY          |     NONE
     *
     * When changing this line make sure the newly
     * selected device node is enabled and the previously
     * selected device node is disabled.
     */
};

......

&dss {
    status = "disabled";
```

```
......
};

&vpfe0 {
    status = "okay";
......
};

&pruss1 {
    pruss1_mdio: mdio@54432400 {
        pinctrl-0 = <&pruss1_mdio_default>;
        pinctrl-names = "default";
        reset-gpios = <&gpio4 20 GPIO_ACTIVE_LOW>;
        status = "okay";

        pruss1_eth0_phy: ethernet-phy@0 {
            reg = <0>;
        };

        pruss1_eth1_phy: ethernet-phy@1 {
            reg = <1>;
        };
    };

    /* Dual mac ethernet application node on icss1 */
    pruss1_eth {
        compatible = "ti,am4372-prueth";
        pruss = <&pruss1>;
        sram = <&ocmcram_nocache>;
        status = "ok";

        pinctrl-0 = <&pruss1_eth_default>;
        pinctrl-names = "default";

        pruss1_emac0: ethernet-mii0 {
            phy-handle = <&pruss1_eth0_phy>;
            phy-mode = "mii";
            sysevent-rx = <20>;    /* PRU_ARM_EVENT0 */
            /* Filled in by bootloader */
            local-mac-address = [00 00 00 00 00 00];
        };

        pruss1_emac1: ethernet-mii1 {
            phy-handle = <&pruss1_eth1_phy>;
            phy-mode = "mii";
            sysevent-rx = <21>;    /* PRU_ARM_EVENT1 */
            /* Filled in by bootloader */
            local-mac-address = [00 00 00 00 00 00];
        };
    };
};
```

反之，当GPIO4-19为低,GPIO4-21为高时，相应的节点设置也要反过来，具体参照myd_c437x_idk_lcd.dts。

## 3.3 制作文件系统

本节主要介绍使用Buildroot进行文件系统的制作。

> 注意: 当用户修改了U-boot或Kernel的代码之后，Buildroot不会自动更新，必须手动提交到相应的GIT仓库。当用户更新了Kernel代码之后，再重新快速编译Buildroot时，需要手动删除"myir-buildroot/dl/linux-master.tar.gz"文件以及"myir-buildroot/output/build/linux-master" 和"myir-buildroot\output\build\linux-headers-master"这两个目录。U-boot的更新也类似。

### 3.3.1 准备编译Buildroot

在本手册的开头介绍了开发环境的部署，同样适用于Buildroot。不过需要注意的是在64位的系统上，需要安装32位兼容的库。

```
$sudo apt-get install libc6-i386 lib32stdc++6 lib32z1
```

拷贝出厂附带资料中的*04-Linux_Source/Filesystem/myir-buildroot.tar.gz*到本地开发主机，并解压到本地工作目录(注意用本地主机上实际工作目录替换)，如下所示：

```
$ ls <WORKDIR>/Filesystem/myir-buildroot
arch    CHANGES           configs     dl    linux            output   support
board   Config.in         COPYING     docs  Makefile         package  system
boot    Config.in.legacy  DEVELOPERS  fs    Makefile.legacy  README   toolchain
```

关于 `Buildroot` 的目录结构可以参照https://buildroot.org/downloads/manual/manual.html. 其中和MYIR-AM437X系列开发板相关的部分主要位于 `<WORKDIR>/Filesystem/myir-buildroot/board/myir/` 目录。

### 3.3.2 配置说明

MYD-AM437X系列开发板默认的配置文件位于 `<WORKDIR>/Filesystem/myir-buildroot/configs/` ，关于MYD-AM437X系列开发板的配置有如下表所示：

| 开发板 | 配置项 | 含义 |
| --- | --- | --- |
| MYD-C437X | myd_c437x_evm_defconfig | MYD-C437X不带qt5运行环境的的Buildroot配置 |
| MYD-C437X | myd_c437x_evm_qt5_defconfig | MYD-C437X带qt5运行环境的的Buildroot配置 |
| MYD-C437X-PRU | myd_c437x_idk_defconfig | MYD-C437X-PRU不带qt5运行环境的Buildroot配置 |
| MYD-C437X-PRU | myd_c437x_idk_qt5_defconfig | MYD-C437X-PRU带qt5运行环境的Buildroot配置 |

下面以MYD-C437X开发板为例，说明 `Buildroot` 的配置过程：

```
$ make clean
$ make myd_c437x_evm_defconfig
$ make menuconfig
```

其中第二个make的配置选项见上表，MYD-C437X-PRU的配置与MYD-C437X类似。

- 配置交叉编译工具链:

Buildroot可以使用外部交叉编译工具链，也可以自行编译产生内部交叉编译工具链，本手册采用的是内部交叉编译工具链。编译完成之后位于 `<WORKDIR>/Filesystem/myir-buildroot/output/host/usr/bin/` 。

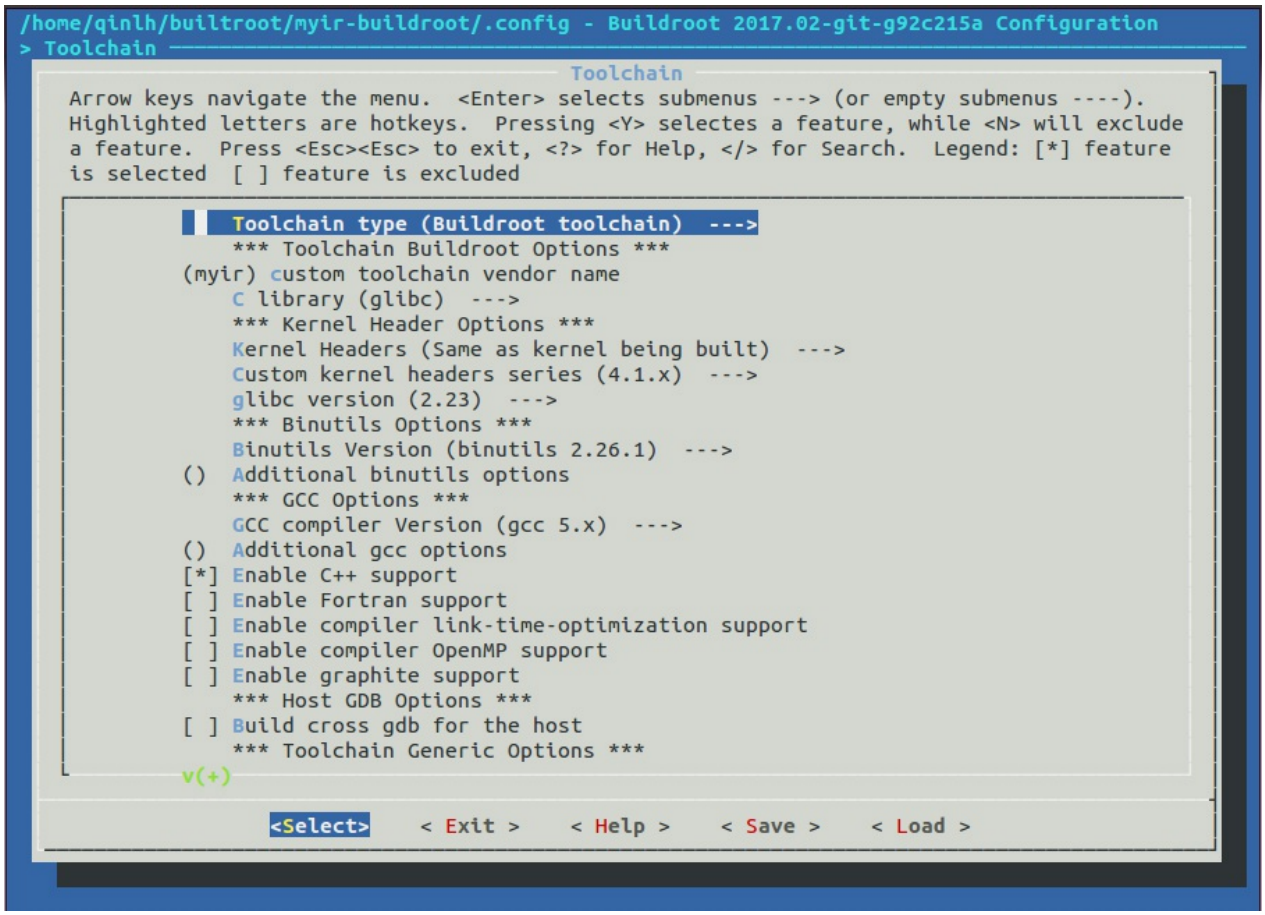

图3-3-1 配置Buildroot交叉编译工具链

- 配置系统参数：

系统配置主要配置目标系统的主机名称，欢迎信息，Init子系统（busybox/systemv/systemd)和对应的设备管理子系统，这里还可以配置root的登陆密码，如下图所示为目标系统配置了root登陆密码 `myirtech` 。 如果不配置的话，密码为空。

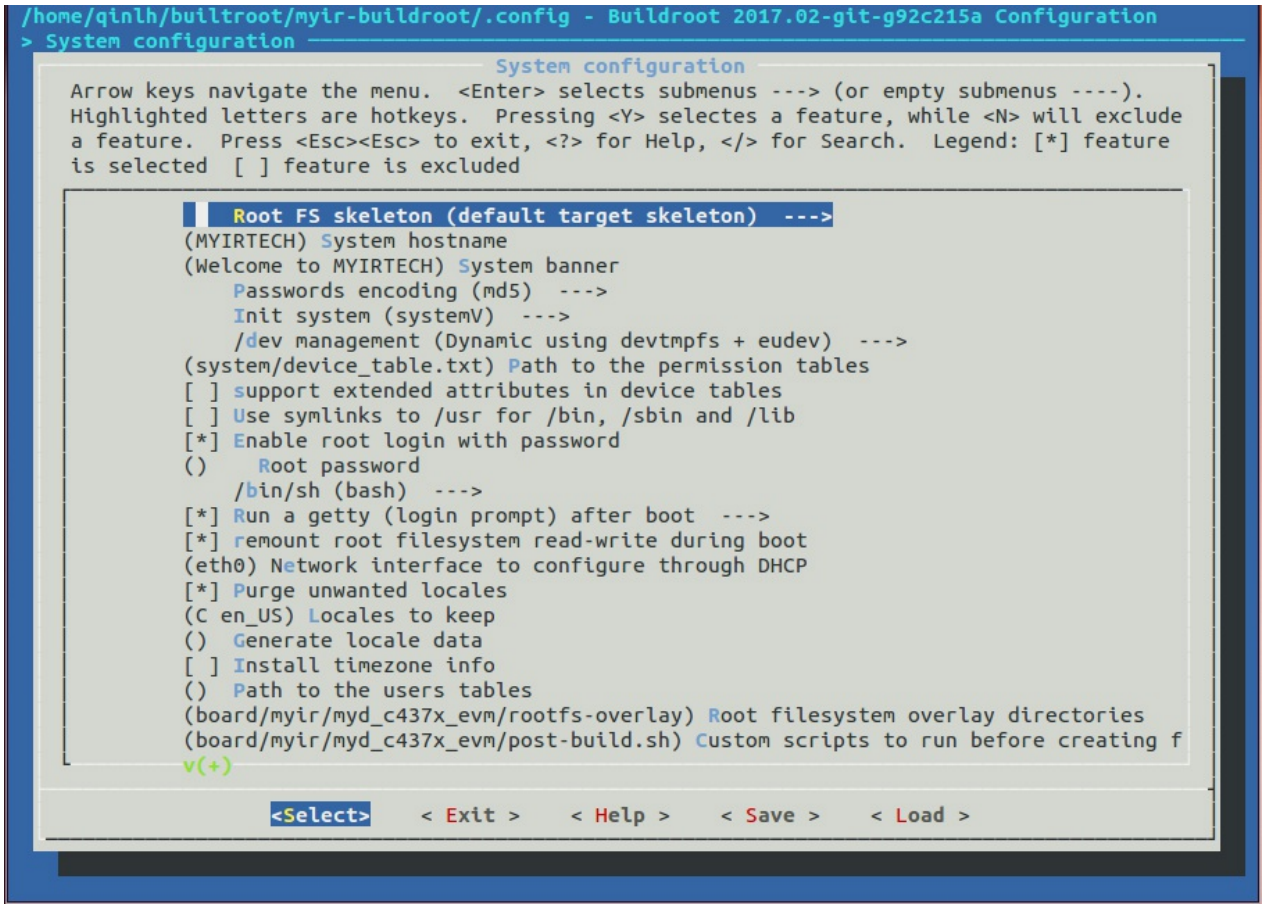

图3-3-2 系统配置

- 配置bootloader：

用户获取到U-boot代码之后，请自行建立Git仓库，替换配置中的BR2_TARGET_UBOOT_CUSTOM_REPO_URL配置操作如下：

建立u-boot代码仓库

```
$ cd ~/
$ tar zxvf myir-u-boot.tar.gz
$ cd myir-u-boot
$ git init
$ git add . -f
$ git commit -m "Initial Version" -a
```

修改位于 `<WORKDIR>/Filesystem/buildroot/configs/myd_c437x_evm_defconfig` 配置文件，将下面两项的内容改为如下所示：

```
BR2_TARGET_UBOOT_CUSTOM_REPO_URL="~/myir-u-boot/.git"
BR2_TARGET_UBOOT_CUSTOM_REPO_VERSION="master"
```

Bootloader配置主要配置bootloader的代码来源，以及代码的配置，编译，安装，如下图所示。这里采用的是git协议从内网的git服务器获取代码。用户也可以根据实际情况配置合适的代码获取方式，具体的配置方式也可以参考Buildroot用户手册。

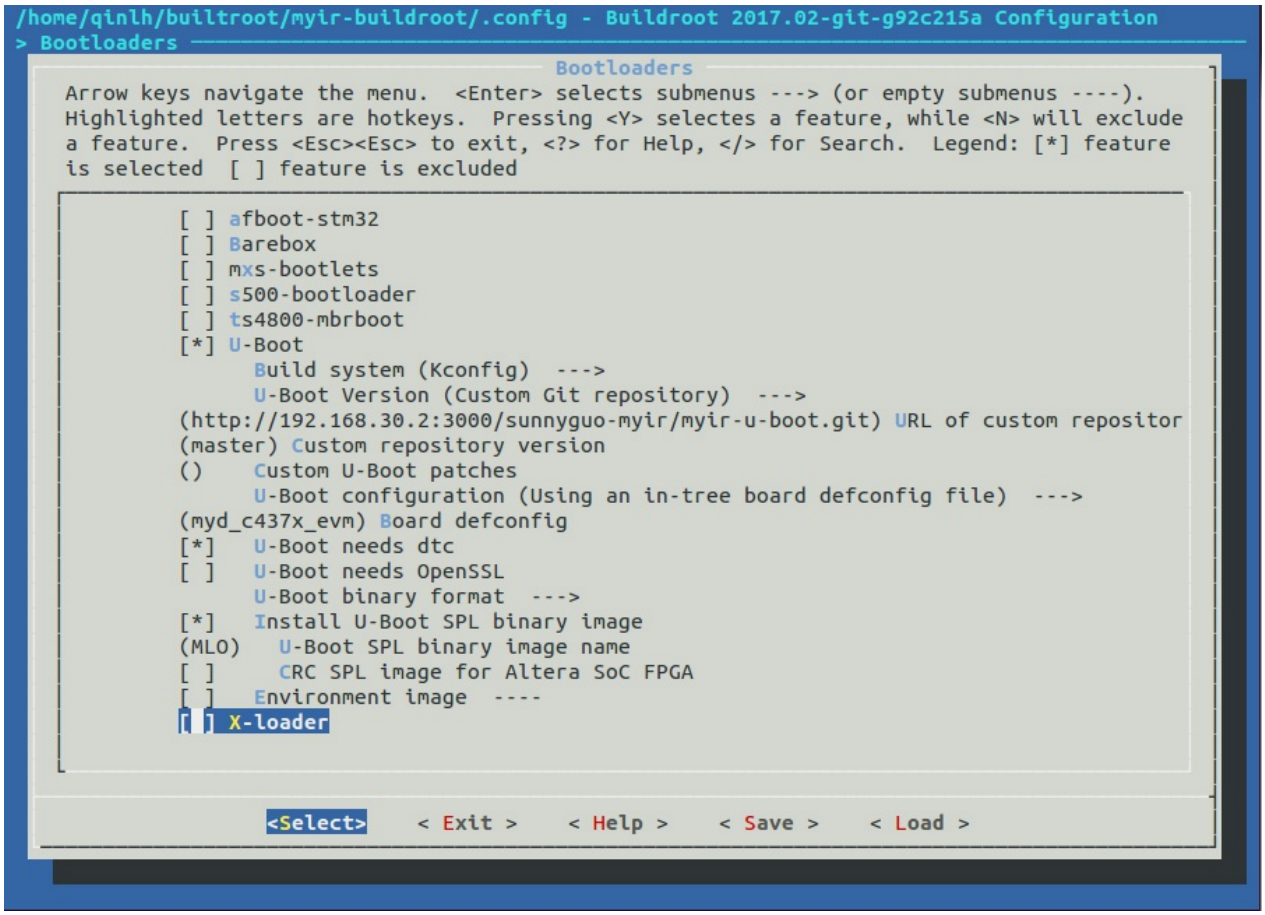

图3-3-3 bootloader配置

- 配置Kernel：
  用户获取到Kernel代码之后，请自行建立Git仓库，替换配置中的
  BR2_LINUX_KERNEL_CUSTOM_REPO_URL配置
  建立kernel代码仓库

```
$ cd ~/
$ tar zxvf myir-kernel.tar.gz
$ cd myir-kernel
$ git init
$ git add . -f
$ git commit -m "Initial Version" -a
```

修改位于 `<WORKDIR>/Filesystem/buildroot/configs/myd_c437x_evm_defconfig` 配置文件，将下面两项的内容改为如下所示：

```
BR2_LINUX_KERNEL_CUSTOM_REPO_URL="~/myir-kernel/.git"
BR2_LINUX_KERNEL_CUSTOM_REPO_VERSION="master"
```

Kernel的配置和bootloader类似，主要配置内核代码的获取方式，内核的配置文件和输出文件等。

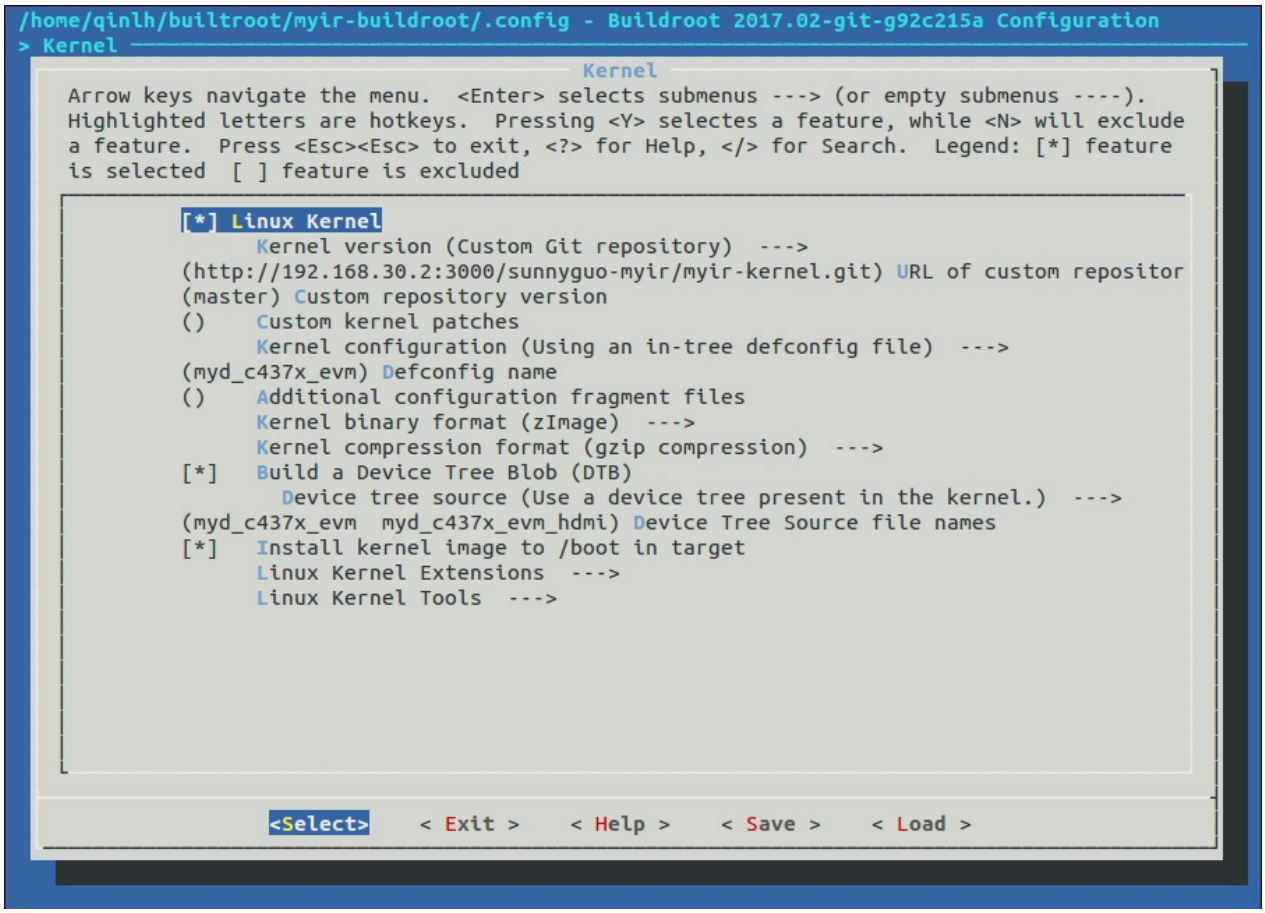

图3-3-4 Kernel配置

- 配置文件系统：

文件系统的配置最终决定了 `<WORKDIR>/Filesystem/myir-buildroot/output/images` 目录下生成哪些格式的文件系统镜像，如下图3-3-5所示，我们配置了ramdisk, EXT2/4以及UBIFS这几种文件系统镜像和rootfs.tar.gz根文件系统压缩包。用户拿这个压缩包用于nfsroot文件系统加载，也可以生成其它格式的文件系统镜像。
除此之外，编译完成之后还在 `<WORKDIR>/Filesystem/myir-buildroot/output/host/usr/sbin` 目录下生成一些本地主机上的文件系统工具，如制作ubi文件系统的工具mkfs.ubifs, ubinize等。我们可以用这些工具重新制作基于rootfs.tar.gz 的UBIFS格式文件系统(注意用本地主机上实际工作目录替换 `<WORKDIR>` )。

创建一个ubinize.cfg文件,内容如下：

```
[ubifs]
mode=ubi
vol_id=0
vol_type=dynamic
vol_name=rootfs
vol_alignment=1
vol_flags=autoresize
image=rootfs.ubifs
```

准备需要打包的根文件系统目录rootfs并打包，步骤如下:

```
$ export PATH=$PATH:<WORKDIR>/Filesystem/myir-buildroot/output/host/usr/sbin
$ mkdir rootfs
$ tar zxvf rootfs.tar.gz -C ./rootfs
$ mkfs.ubifs -d rootfs -e 0x1f000 -c 2048 -m 0x800 -x lzo -F -o  rootfs.ubifs
$ ubinize -o rootfs.ubi -m 0x800 -p 0x20000 -s 512 -m 2048 -O 2048 ubinize.cfg
```

> 注意：系统中可能默认已安装mkfs.ubifs, 请使用which mkfs.ubifs确认这里执行的命令位于 `<WORKDIR>/Filesystem/myir-buildroot/output/host/usr/sbin` 之下。

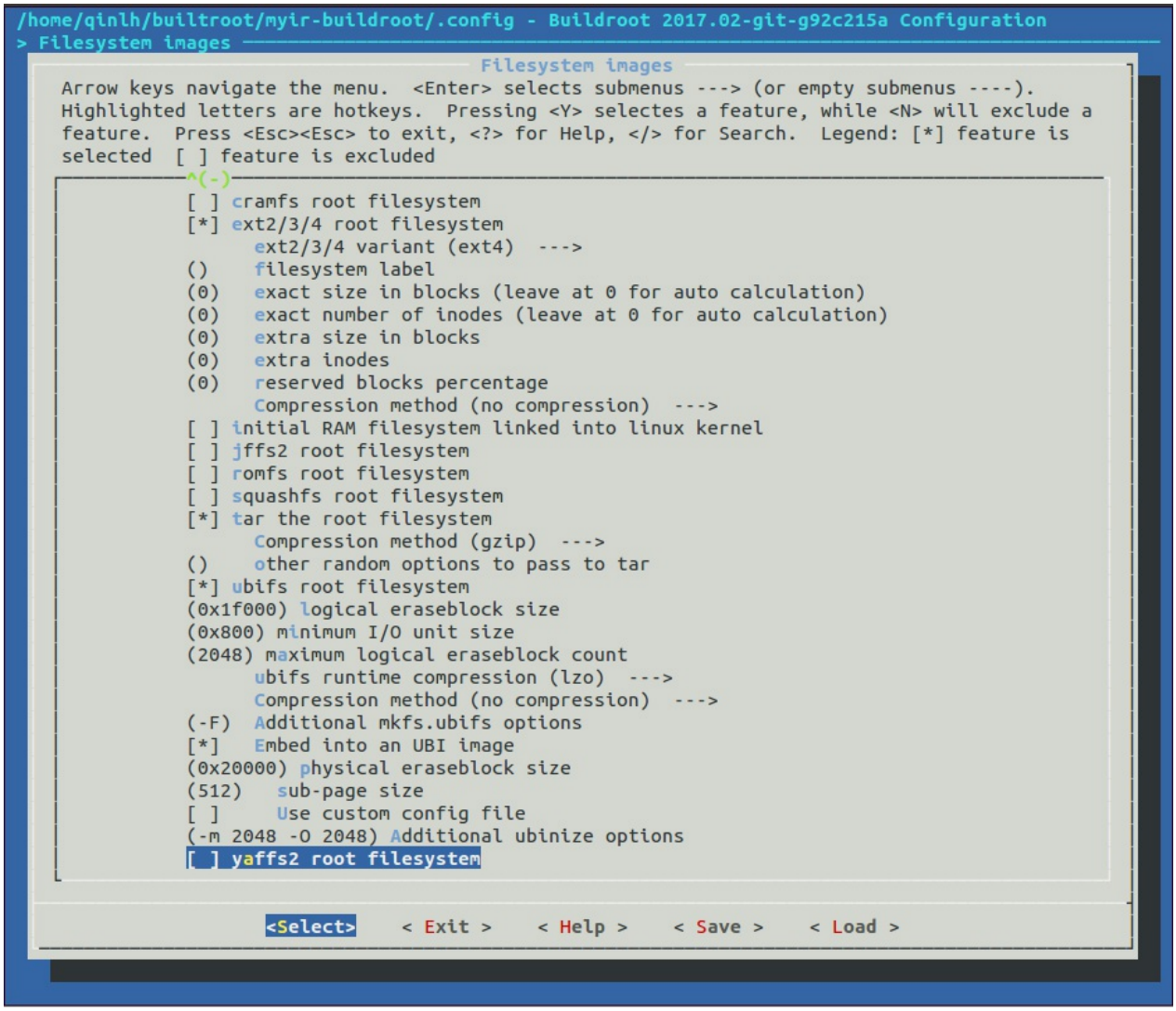


图3-3-5 文件系统配置

- 配置工具包：

工具包的配置相对比较简单，但也是用户经常会改动的配置项。linux下一些常见的工具基本上都能够在这里找到。比如硬件测试相关的I2C-tools, spi-tools, can-utils等，网络相关的DHCP, TFTP, SSH等, 用户可以根据需要自行配制，也可以添加自己编写的其他工具包。关于如何添加自己的工具包，在Buildroot中也有详细的介绍，参见 https://buildroot.org/downloads/manual/manual.html#adding-packages,这里不再赘述。

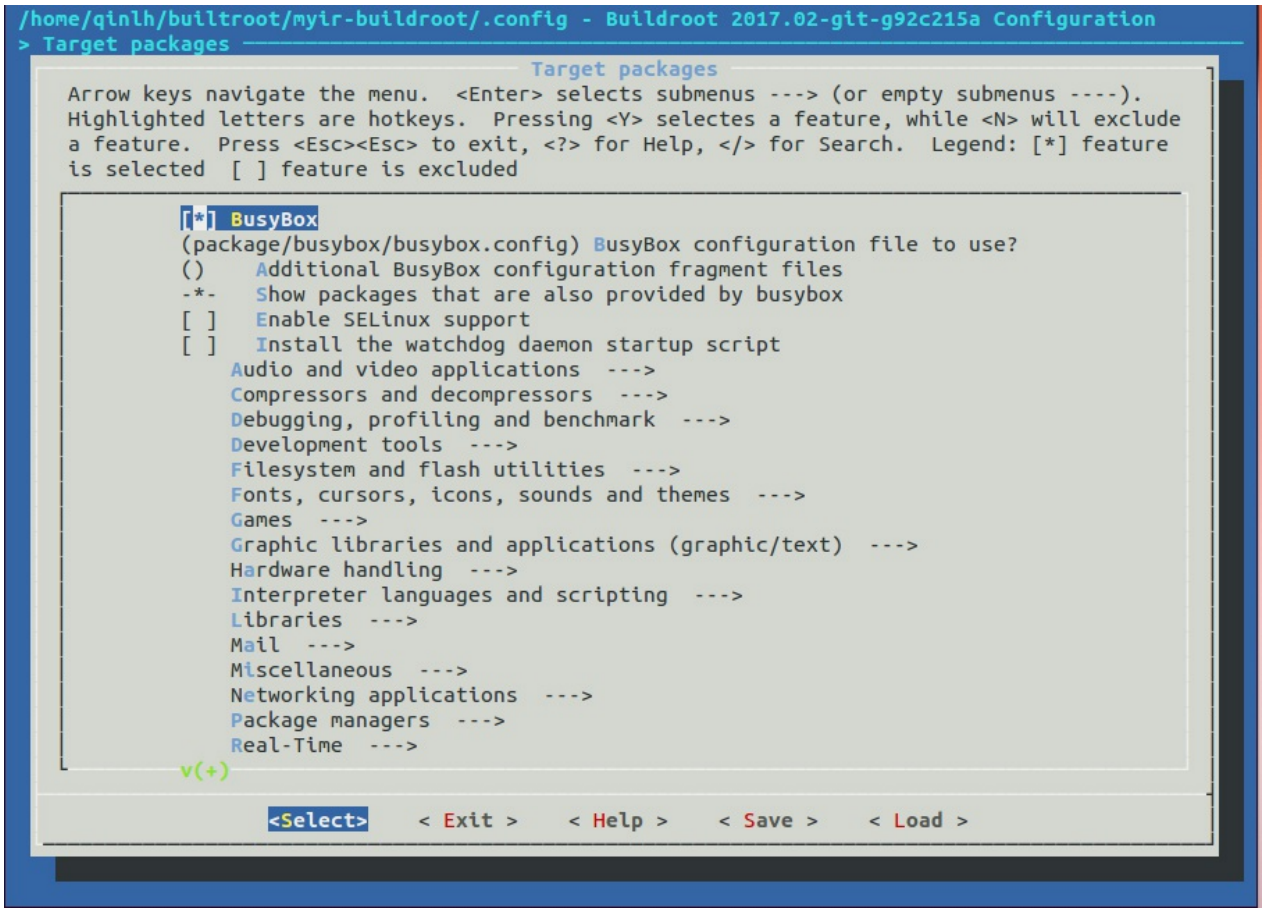

图3-3-6 工具包配置

## 3.3.3 开始构建

Buildroot构建的过程类似于Linux Kernel的构建，只需要简单的命令就可以完成。

```
$ make clean
$ make myd_c437x_evm_defconfig
$ make
```

编译过程中会生成一个output目录，最终生成的输出的文件位于 `<WORKDIR>/Filesystem/myir-buildroot/output/images` 目录。

```
$ls -al output/images
boot.vfat ramdisk.gz rootfs.cpio.uboot rootfs.tar.gz u-boot.img uEnv_ramdisk.txt
MLO readme.txt rootfs.ext2 rootfs.ubi uEnv uEnv.txt
myd_c437x_evm.dtb rootfs.cpio rootfs.ext4 rootfs.ubifs uEnv_hdmi.txt zImage
myd_c437x_evm_hdmi.dtb rootfs.cpio.gz rootfs.tar sdcard.img uEnv_mmc.txt
```

output/images 目录下的输出文件基本上包含了bootloader, kernel, 以及各种格式的文件系统镜像。这些文件在接下来的系统升级章节将会详细介绍。

## 3.3.4 Arago构建的文件系统

MYD-AM437X系列开发板也可以使用基于Arago构建的文件系统镜像，具体的构建方法可以参考TI官网WIKI页面. http://processors.wiki.ti.com/index.php/Processor_SDK_Building_The_SDK.

## 3.4 编译QT

QT的编译我们推荐使用Buildroot，Buildroot可以很容易地编译Qt5图形库，并且整合在文件系统中。同时，我们提供了配置文件，可以更方便地编译Qt5。

MEasy HMI演示系统的也是通过带QT5运行环境的buildroot配置文件生成的。关于MEasy HMI更多信息参考《MEasy HMI开发手册》。

进入上一节中解压好的myir-buildroot目录：

```
$ cd <WORKDIR>/Filesystem/myir-buildroot
```

开始编译Buildroot：

| 开发板 | 配置项 | 含义 |
| --- | --- | --- |
| MYD-C437X | myd_c437x_evm_defconfig | MYD-C437X不带qt5运行环境的的Buildroot配置 |
| MYD-C437X | myd_c437x_evm_qt5_defconfig | MYD-C437X带qt5运行环境的的Buildroot配置 |
| MYD-C437X-PRU | myd_c437x_idk_defconfig | MYD-C437X-PRU不带qt5运行环境的Buildroot配置 |
| MYD-C437X-PRU | myd_c437x_idk_qt5_defconfig | MYD-C437X-PRU带qt5运行环境的Buildroot配置 |

两种配置之间的差异如下:

```
BR2_PACKAGE_QT5=y
BR2_PACKAGE_QT5BASE_LICENSE_APPROVED=y
BR2_PACKAGE_QT5BASE_EXAMPLES=y
BR2_PACKAGE_QT5BASE_WIDGETS=y
BR2_PACKAGE_QT5BASE_LINUXFB=y
BR2_PACKAGE_QT5BASE_EGLFS=y
BR2_PACKAGE_QT5BASE_GIF=y
BR2_PACKAGE_QT5BASE_JPEG=y
BR2_PACKAGE_QT5BASE_PNG=y
BR2_PACKAGE_QT5BASE_DBUS=y
BR2_PACKAGE_QT5BASE_TSLIB=y
BR2_PACKAGE_QT5QUICKCONTROLS=y
```

编译完成之后的QT支持eglfs, linuxfb, minimal, offscreen等几种platform插件，默认使用的插件是eglfs。
如果我们使用的SOC为AM4372，AM4376或AM4377这几个没有SGX图形加速功能的型号之一，则linux内核设备树中，需要禁用sgx驱动，eglfs插件也就不能工作了。这时只能选择另外一种platform插件linuxfb。
在运行QT5应用程序的时候，通过--platform参数选择插件。例如：

```
$ ./helloqt5 --platform linuxfb:fb=/dev/fb0
```

下面以MYD-C473X开发板支持QT5的系统编译为例：

```
$ make clean
$ make myd_c437x_evm_qt5_defconfig
$ make
```

等待编译完成后，Qt5会打包在位于 `<WORKDIR>/Filesystem/myir-buildroot/output/images` 目录之下的文件系统镜像中。 `<WORKDIR>/Filesystem/myir-buildroot/output/host` 同样可作为Qt开发的SDK目录，其中包含了交叉编译工具, QT5应用程序构建工具qmake等，在后续章节中会详细介绍。

# 4. Linux应用开发

本节主要介绍设备驱动功能验证的一些方法，并提供一些基本的应用开发示例实现对设备的操作。

MYD-AM437X系列开发板提供了常用外设的演示程序，程序以及源码都位于 `<WORKDIR>/Examples/` , 请根据目录内的Makefile或README文件进行编译：

```
$ cd <WORKDIR>/Examples/
```

- 确保环境变量按下面的示例设置完成:

```
$ export ARCH=arm
$ export CROSS_COMPILE=arm-linux-gnueabihf-
$ export PATH=$PATH:<WORKDIR>/ToolChain/gcc-linaro-5.3-2016.02-x86_64_arm-linux-gnueabihf/bin
```

编译完Buildroot之后，在 `<WORKDIR>/Filesystem/myir-buildroot/output/host/usr/bin` 目录下也生成了一个交叉编译工具链，这里可以使用这个工具链替换上面linaro的工具链：

```
$ export CROSS_COMPILE=arm-linux-myir-gnueabihf-
$ export PATH=$PATH:<WORKDIR>/Filesystem/myir-buildroot/output/host/usr/bin
```

对于PRU开发板的测试程序，还需要安装一个PRU工具链。 这个工具可以在附带的发布包 `03Tools/ti_cgt_pru_2.1.3_linux_installer_x86.bin` 路径下找到。
下面是安装和设置环境变量的步骤:

```
$ cd <WORKDIR>/ToolChain/
$ ./ti_cgt_pru_2.1.3_linux_installer_x86.bin
$ export PRU_CGT=<WORKDIR>/ToolChain/ti-cgt-pru_2.1.3/
```

Camera和Audio例程的Makefile需要指定依赖库的目录和头文件的目录,下面以Camera的例程为例子：

```
$ cd <WORKDIR>/Examples/camera
$ cat Makefile
CC = $(CROSS_COMPILE)gcc
CFLAGS ?=-I /<WORKDIR>/Filesystem/myir-buildroot/output/host/usr/
                        arm-myir-linux-gnueabihf/sysroot/usr/include/libdrm/ \
                -I <WORKDIR>/Filesystem/myir-buildroot/output/host/usr/
                         arm-myir-linux-gnueabihf/sysroot/usr/include \
                -I <WORKDIR>/Filesystem/myir-buildroot/output/host/usr/
                arm-myir-linux-gnueabihf/sysroot/usr/include/omap
LDFLAGS ?=  -lpthread  -ljpeg -ldrm -ldrm_omap -L <WORKDIR>/Filesystem/
               myir-buildroot/output/host/usr/arm-myir-linux-gnueabihf/sysroot/usr/lib
TARGET = $(notdir $(CURDIR))_test
SRC =  $(shell ls *.c)
OBJS = $(patsubst %.c ,%.o ,$(SRC))
.PHONY: all
all: $(TARGET)
$(TARGET) : $(OBJS)
        $(CC) $(CFLAGS) $(LDFLAGS) -o $@ $^
%.o : %.c
        $(CC) $(CFLAGS) -c $< -o $@
clean:
        $(RM) *.o $(TARGET)
```

- 用户可以一次完成所有示例的编译，下面以MYD-C437X开发板为例:

```
$ cd <WORKDIR>/Examples/
$ make OPTION=MYD-C437X-EVM clean
$ make OPTION=MYD-C437X-EVM
$ make OPTION=MYD-C437X-EVM install
```

安装路径由Makefile里面的PREFIX变量指定默认为 `<WORKDIR>/Examples/rootfs` ，其中OPTION选项指定为MYIR不同系列的开发板，具体参见 `<WORKDIR>/Examples/` 目录下的README.md，MYD-C437X-PRU开发板的例程编译与MYD-C437X类似。

也可以一个一个的单独编译，例如:

```
$ cd <WORKDIR>/Examples/<APP_DIR>
$ make
```

在开发板上运行程序时需要注意权限，如果无法执行请使用如下命令：

```
# chmod +x *
```

- 不同的应用需要不同的设备树文件，设备树文件修改如下：

**SD**卡启动：编辑位于SD卡中的uEnv_ramdisk.txt的 `fdtfile` 为应用所需的设备树文件名，编辑完成后将文件名改为uEnv.txt。内容改为如下所示：

```
# This uEnv.txt file can contain additional environment settings that you
# want to set in U-Boot at boot time.  This can be simple variables such
# as the serverip or custom variables.  The format of this file is:
#    variable=value
# NOTE: This file will be evaluated after the bootcmd is run and the
#       bootcmd must be set to load this file if it exists (this is the
#       default on all newer U-Boot images.  This also means that some
#       variables such as bootdelay cannot be changed by this file since
#       it is not evaluated until the bootcmd is run.
#optargs=video=HDMI-A-1:800x600

# Uncomment the following line to enable HDMI display and disable LCD display.
fdtfile=myd_c437x_evm.dtb
#fdtfile=myd_c437x_evm_hdmi.dtb
devtype=mmc
devnum=0
bootdir=/
bootpart=0:1
uenvcmd=if run loadimage;
        then run loadfdt; run loadramdisk; echo Booting from mmc${mmcdev} ...;
        run ramargs; print bootargs; bootz ${loadaddr} ${rdaddr} ${fdtaddr}; fi
```

**EMMC**启动：通过EMMC方式启动开发板，进入到系统里面，挂着eMMC的boot分区到/mnt目录下面：

```
# mkdir /mnt/boot
# mount -t vfat /dev/mmcblk0p1 /mnt/boot/
# ls /mnt/boot/
MLO                     ramdisk.gz              uEnv.txt
myd_c437x_evm.dtb       u-boot.img              ws-calibrate.rules
myd_c437x_evm_hdmi.dtb  uEnv                    zImage
```

使用 `vi` 命令编辑位于*/mnt/boot*目录中的uEnv.txt的 `fdtfile` 为应用所需的设备树文件名，编辑完成后保存退出。内容修改为下面所示：

```
# This uEnv.txt file can contain additional environment settings that you
# want to set in U-Boot at boot time.  This can be simple variables such
# as the serverip or custom variables.  The format of this file is:
#    variable=value
# NOTE: This file will be evaluated after the bootcmd is run and the
#       bootcmd must be set to load this file if it exists (this is the
#       default on all newer U-Boot images.  This also means that some
#       variables such as bootdelay cannot be changed by this file since
#       it is not evaluated until the bootcmd is run.
#optargs=video=HDMI-A-1:800x600

# Uncomment the following line to enable HDMI display and disable LCD display.
fdtfile=myd_c437x_evm.dtb
#fdtfile=myd_c437x_evm_hdmi.dtb
```

使用umount命令卸载boot分区。重新启动开发板，使用指定的设备树来配置内核。

```
#umount /mnt/boot
```

**NFS**启动： 直接拷贝应用所需的设备树文件到 `tftp` 目录下面，然后进入uboot控制台通过设置环境变量来选择对应的设备树。操作如下：

```
setenv serverip 192.168.30.2
setenv ipaddr 192.168.30.214
setenv rootpath /home/qinlh/export/rootfs
setenv fdtfile myd_c437x_evm.dtb            //修改为应用所需的设备树
saveenv
run netboot
```

## 4.1 GPIO测试

本例程演示如何使用Linux API配置开发板上的GPIO，详情请参考源码。

测试硬件环境：

- MYD-AM437X系列开发板一块
- USB转TTL调试串口一根，连接MYD-AM437X系列开发板 Debug串口和PC, PC端波特率设置115200-8-n-1

| 接口 | **MYD-C437X** | **MYD-C437X-PRU** |
| --- | --- | --- |
| debug串口 | J16 UART0 | J25 Debug UART |

测试软件环境：

- Linux Kernel 4.1.18
- gpio_test 应用程序

| 开发板 | 设备树 |
| --- | --- |
| MYD-C437X | myd_c437x_evm.dtb、myd_c437x_evm_hdmi.dtb |
| MYD-C437X-PRU | myd_c437x_idk.dtb、myd_c437x_idk_lcd.dtb |

测试过程：

- 将目录 `<WORKDIR>/Examples/rootfs/usr/bin/` 中的可执行程序gpio_test拷贝至开发板/usr/bin/目录下, 执行该程序进行测试如下：

```
# gpio_test -h
Usage: gpio_test [options]

Version 1.0
Options:
-n | -- number gpio      gpio number.
-g | -- get            get gpio level.
-s | -- set  level     set gpio level. 0: low; 1: high
-h | --help                 Print this message
```

- GPIO3_7作为EEPROM的写保护控制脚，输出高则使能写保护，输出低则禁用写保护，下面通过gpio_test进行测试：

```
# gpio_test -n 103 -s 1
==gpio3_7  direction is out
==gpio3_7  level is high
Set gpio3_7 level high success!
# gpio_test -n 103 -s 0
==gpio3_7  direction is out
==gpio3_7  level is low
Set gpio3_7 level low success!
```

用户也可以通过shell脚本，使用echo和cat命令来实现对/sys/class/gpio下文件的访问。例如:set_eeprom.sh

```
#!/bin/bash

EEPROM_WP_GPIO_PIN=103

wait_gpio() {
        sleep 1
}

wp_init() {
  if [ ! -d "/sys/class/gpio/gpio$EEPROM_WP_GPIO_PIN" ]; then
      echo "$EEPROM_WP_GPIO_PIN" > /sys/class/gpio/export; wait_gpio
  fi

  echo "out" > /sys/class/gpio/gpio$EEPROM_WP_GPIO_PIN/direction; wait_gpio
}

up() {
    echo "0" > /sys/class/gpio/gpio$EEPROM_WP_GPIO_PIN/value; wait_gpio
}

down() {
    echo "1" > /sys/class/gpio/gpio$EEPROM_WP_GPIO_PIN/value; wait_gpio
}

if [ "$1" = "1" ]; then
    wp_init
    up
fi

if [ "$1" = "0" ]; then
    wp_init
    down
    echo "$EEPROM_WP_GPIO_PIN" > /sys/class/gpio/unexport; wait_gpio
fi
```

- 通过执行set_eeprom.sh脚本也可以实现对GPIO的控制。

```
# chmod 777 /usr/bin/set_eeprom.sh
# set_eeprom.sh 1                          -- 使能写保护
# set_eeprom.sh 0                          -- 禁止写保护
```

MYD-AM437X系列其它板型GPIO测试情况类似。

## 4.2 LCD测试

本例程通过对Linux的FrameBuffer操作，实现LCD的彩色画点，画线，以及区域填充的测试。 也可以使用Buildroot文件系统自带的fbv程序显示图片。

测试硬件环境：

- MYD-AM437X系列 开发板一块
- MY-TFT070CV2 连接MYD-AM437X系列开发板的LCD接口
- USB转TTL调试串口一根，连接MYD-AM437X系列开发板 Debug串口和PC, PC端波特率设置115200-8-n-1

| 接口 | **MYD-C437X** | **MYD-C437X-PRU** |
| --- | --- | --- |
| debug串口 | J16 UART0 | J25 Debug UART |
| LCD 16bit接口 | J8 LCD_16Bit | J20 LCD_16B |

测试软件环境：

- Linux Kernel 4.1.18
- framebuffer_test 应用程序
- fbv应用程序

| 开发板 | 设备树 |
| --- | --- |
| MYD-C437X | myd_c437x_evm.dtb |
| MYD-C437X-PRU | myd_c437x_idk_lcd.dtb |

测试过程：

- 编译并拷贝 `<WORKDIR>/Examples/rootfs/usr/bin/` 目录下的测试程序framebuffer_test至开发板/usr/bin目录。在Linux终端输入如下命令：

```
# chmod 777 /usr/bin/framebuffer_test
# framebuffer_test -h
Usage: framebuffer_test [options]

Version 1.0
Available options:
-d | --device name   framebuffer device name, default: /dev/fb0
-h | --help          Print this message

# framebuffer_test -d /dev/fb0
xres:800 >>> yres:480 >>> bpp:32>>>
```

运行程序后，终端显示屏幕信息，LCD屏幕会先后出现多种背景色，然后进行彩色画点，画线，以及区域填充的测试。

- 将一个32位颜色深度，分辨率800X480的BMP图像拷贝到开发板/media/1.bmp, 用fbv测试图片显示：

```
# fbv
Usage: fbv [options] image1 image2 image3 ...

Available options:
--help        | -h : Show this help
--alpha       | -a : Use the alpha channel (if applicable)
--dontclear   | -c : Do not clear the screen before and after displaying the image
--donthide    | -u : Do not hide the cursor before and after displaying the image
--noinfo      | -i : Supress image information
--stretch     | -f : Strech (using a simple resizing routine) the image to fit
                                                    ontoscreen if necessary
--colorstretch| -k : Strech (using a 'color average' resizing routine) the image
                                                  to fit onto screen if necessary
--enlarge     | -e : Enlarge the image to fit the whole screen if necessary
--ignore-aspect| -r : Ignore the image aspect while resizing
--delay <d>   | -s <delay> : Slideshow, 'delay' is the slideshow delay in tenths of seconds.

Keys:
 r            : Redraw the image
 a, d, w, x   : Pan the image
 f            : Toggle resizing on/off
 k            : Toggle resizing quality
 e            : Toggle enlarging on/off
 i            : Toggle respecting the image aspect on/off
 n            : Rotate the image 90 degrees left
 m            : Rotate the image 90 degrees right
 p            : Disable all transformations
Copyright (C) 2000 - 2004 Mateusz Golicz, Tomasz Sterna.
Error: Required argument missing.

# fbv /meida/1.bmp
fbv - The Framebuffer Viewer
/media/1.bmp
800 x 480
```

程序执行完毕，图片完整显示在LCD上。

MYD-AM437X系列其它板型的LCD测试情况类似。

## 4.3 Touch Screen 测试

本例程主要使用buildroot制作的文件系统自带TS_CALIBRATE测试程序，进行触摸屏的校准测试。

测试硬件环境：

- MYD-AM437X系列开发板一块
- 一块MY-TFT070CV2 连接MYD-AM437X系列开发板LCD接口
- 或者一块MY-TFT070RV2 连接MYD-AM437X系列开发板LCD接口
- USB转TTL调试串口一根，连接MYD-AM437X系列开发板 Debug串口和PC, PC端波特率设置115200-8-n-1

| 接口 | **MYD-C437X** | **MYD-C437X-PRU** |
| --- | --- | --- |
| debug串口 | J16 UART0 | J25 Debug UART |
| LCD 16bit接口 | J8 LCD_16Bit | J20 LCD_16B |

测试软件环境：

- Linux Kernel 4.1.18
- TS_CALIBRATE应用程序

| 开发板 | 设备树 |
| --- | --- |
| MYD-C437X | myd_c437x_evm.dtb |
| MYD-C437X-PRU | myd_c437x_idk_lcd.dtb |

测试过程：

由于设备树上的差异性，在不同的MYD-AM437X系列开发板上的电容触摸和电阻触摸对应的event事件编号会不一样，如下表所示：

| 触摸模组 | **MYD-C437X** | **MYD-C437X-PRU** |
| --- | --- | --- |
| 电阻触摸 | event1 | event2 |
| 电容触摸 | event3 | event3 |

- 断电，连接电容触摸模组，启动开发板，查看触摸设备对应的设备节点。

```
# ls /dev/input/
by-path  event0   event1   event2   event3   mice     mouse0   mouse1

# cat /sys/class/input/event0/device/name
volume_keys@0

# cat /sys/class/input/event1/device/name
ti-tsc

# cat /sys/class/input/event2/device/name
tps65218_pwrbutton

# cat /sys/class/input/event3/device/name
ft5x06_ts
```

从以上查询结果可知，电阻触摸对应的设备节点为/dev/input/event1, 电容触摸对应的设备节点为/dev/input/event3。测试步骤如下：

```
# export TSLIB_TSDEVICE=/dev/input/event3
# ts_calibrate
xres = 800, yres = 480
Took 4 samples...
Top left : X =   54 Y =   46
Took 4 samples...
Top right : X =  745 Y =   58
Took 4 samples...
Bot right : X =  745 Y =  421
Took 3 samples...
Bot left : X =   68 Y =  429
Took 3 samples...
Center : X =  394 Y =  245
-5.867981 1.023202 -0.019352
-2.867676 -0.003020 1.017846
Calibration constants: -384564 67056 -1268 -187936 -197 66705 65536
```

- 断电，连接电阻触摸模组，启动开发板，查看触摸设备对应的设备节点。

```
# cat /sys/class/input/
event0/ event1/ event2/ input0/ input1/ input2/ mice/   mouse0/

# cat /sys/class/input/event0/device/name
volume_keys@0

# cat /sys/class/input/event1/device/name
ti-tsc

# cat /sys/class/input/event2/device/name
tps65218_pwrbutton
```

从以上查询结果可知，电阻触摸对应的设备节点为/dev/input/event1, 测试步骤如下：

```
# export TSLIB_TSDEVICE=/dev/input/event1
# ts_calibrate
xres = 800, yres = 480
Took 60 samples...
Top left : X =  244 Y =  592
Took 72 samples...
Top right : X = 3674 Y =  604
Took 84 samples...
Bot right : X = 3695 Y = 3404
Took 2 samples...
Bot left : X =  275 Y = 3295
Took 11 samples...
Center : X = 2040 Y = 1881
-2.257812 0.204344 -0.001784
-25.022156 -0.002373 0.137957
Calibration constants: -147968 13391 -116 -1639852 -155 9041 65536
```

MYD-AM437X系列其它板型Touch Screen测试情况类似。

## 4.4 RTC 测试

本例程演示如何使用Linux API对开发板上的RTC进行时间设置与读取，详情请参考源码。
用户也可以使用Buildroot文件系统自带的date和hwclock配合使用对RTC进行测试。

测试硬件环境：

- MYD-AM437X系列开发板一块
- RTC所用CR2032 3V纽扣电池一块
- USB转TTL调试串口一根，连接MYD-AM437X系列开发板 Debug串口和PC, PC端波特率设置115200-8-n-1

| 接口 | **MYD-C437X** | **MYD-C437X-PRU** |
| --- | --- | --- |
| debug串口 | J16 UART0 | J25 Debug UART |

测试软件环境：

- Linux Kernel 4.1.18
- date, hwclock或rtc_test应用程序

| 开发板 | 设备树 |
| --- | --- |
| MYD-C437X | myd_c437x_evm.dtb、myd_c437x_evm_hdmi.dtb |
| MYD-C437X-PRU | myd_c437x_idk.dtb、myd_c437x_idk_lcd.dtb |

测试过程：

- 将目录 `<WORKDIR>/Examples/rootfs/usr/bin/` 中的可执行程序rtc_test拷贝至开发板/usr/bin/，执行以下命令将rtc当前时间设置为2016/11/18 17:58:18 并读取当前时间：

```
# chmod 777 /usr/bin/rtc_test
# rtc_test -h
Usage: rtc_test [options]

Version 1.0
Options:
-d | --device name   rtc device name, default: /dev/rtc0
-w | --write  time     time string with format MMDDhhmm[CCYY][.ss]. such as: 111817582016.18
-h | --help          Print this message

# rtc_test -d /dev/rtc1 -w 111817582016.18

  date/time is updated to:  18-11-2016, 17:58:18.
```

- 将开发板断电，经过一段时间之后，重新上电开机，再次通过rtc_test读出rtc的时间，如下：

```
# rtc_test -d /dev/rtc1

  Current RTC date/time is 18-11-2016, 17:59:12.
```

- 用户也可以使用date, hwclock配合，进行RTC的测试

```
# date  081518002016.30        -- 设置系统时间为2016年8月15日18:00:30
Mon Aug 15 18:00:30 UTC 2016
# date
Mon Aug 15 18:00:38 UTC 2016
# hwclock -w /dev/rtc1         -- 将系统时间写入到rtc1
```

- 将开发板断电，经过一段时间之后，重新上电开机，再次通过hwclock读出rtc的时间，如下：

```
# hwclock -r /dev/rtc1
Mon Aug 15 18:11:08 2016  0.000000 seconds
```

MYD-AM437X系列其它板型的RTC测试情况类似。

## 4.5 RS232测试

本例程演示如何使用Linux API配置开发板上的RS232并使其发送和接收数据，详情请参考源码。

测试硬件环境：

- MYD-AM437X系列开发板两块
- 数据线连接两块开发的RS232接口，GND<->GND，TXD<->RXD，RXD<->TXD, CTS<->RTS， RTS<->CTS
- USB转TTL调试串口线两根，分别连接两块开发板的Debug UART和PC , PC端波特率设置115200-8-n-1

| 接口 | **MYD-C437X** | **MYD-C437X-PRU** |
| --- | --- | --- |
| debug串口 | J16 UART0 | J25 Debug UART |
| RS232接口 | J17 UART3 | J12 RXD、TXD、RTS、CTS、GND |

测试软件环境：

- Linux Kernel 4.1.18
- tty_test 应用程序

| 开发板 | 设备树 |
| --- | --- |
| MYD-C437X | myd_c437x_evm.dtb、myd_c437x_evm_hdmi.dtb |
| MYD-C437X-PRU | myd_c437x_idk.dtb、myd_c437x_idk_lcd.dtb |

测试过程：

- 将目录 `<WORKDIR>/Examples/rootfs/usr/bin/` 中的可执行程序tty_test拷贝至开发板/usr/bin/目录下，执行 `tty_test -h` , 查看测试程序帮助信息，如下：

```
# tty_test -h
Usage: tty_test [options]
Version 1.0
Options:
-d | --device name   tty device name, default: /dev/tty0
-m | --mode  mode   operate mode. 0: RS232, 1: RS485  default mode: 0
-f | --flow                    flow control
-b | --baudrate baudrate  set baudrate, default baudrate: 115200
-l | --loop              operate circularly
-w | --write  frame     frame string. such as: 0123456789
-h | --help          Print this message
```

- MYD-C437X开发板的J17为一个带硬件流控的RS232串口，对应的设备节点/dev/ttyO3, 两块开发板一个作为发送端另一个作为接收端，先在一块开发板执行以下命令发送数据：

```
# tty_test -d /dev/ttyO3 -b 9600 -m 0  -w 0123456789 -f -l

 SEND:0123456789

 SEND:0123456789

 SEND:0123456789
```

- 再在另一开发板执行以下命令接收数据：

```
#tty_test -d /dev/ttyO3 -b 9600 -m 0 -f -l

RECV:0123456789, total:10

RECV:0123456789, total:10

RECV:0123456789, total:10
```

- 两块板互换一下角色，结果一样，也可以通过短接某一块开发板的RTS和CTS，TXD和RXD, 进行硬件回环测试，具体情况不再赘述。

MYD-AM437X系列其它板型RS232测试情况类似。

## 4.6 RS485测试

本例程演示如何使用Linux API配置开发板上的RS485并使其发送和接收数据，详情请参考源码。

测试硬件环境：

- MYD-AM437X系列开发板两块
- 数据线连接两块板的RS485接口，485A<->485A，485B<->485B，GND<->GND
- USB转TTL调试串口一根，连接MYD-AM437X系列开发板 Debug串口和PC, PC端波特率设置115200-8-n-1

| 接口 | **MYD-C437X** | **MYD-C437X-PRU** |
| --- | --- | --- |
| debug串口 | J16 UART0 | J25 Debug UART |
| RS485接口 | J15 485_A、485_B、GND | J10 485A、485B、GNDISO |

测试软件环境：

- Linux Kernel 4.1.18
- tty_test 应用程序

| 开发板 | 设备树 |
| --- | --- |
| MYD-C437X | myd_c437x_evm.dtb、myd_c437x_evm_hdmi.dtb |
| MYD-C437X-PRU | myd_c437x_idk.dtb、myd_c437x_idk_lcd.dtb |

测试过程：

| 开发板 | 串口设备 |
| --- | --- |
| MYD-C437X | ttyO3 |
| MYD-C437X-PRU | ttyO5 |

- 将目录 `<WORKDIR>/Examples/rootfs/usr/bin/` 中的可执行程序tty_test拷贝至开发板/usr/bin/目录下，执行 `tty_test -h` , 查看测试程序帮助信息，如下：

```
# tty_test -h
Usage: tty_test [options]
Version 1.0
Options:
-d | --device name   tty device name, default: /dev/tty0
-m | --mode  mode   operate mode. 0: RS232, 1: RS485  default mode: 0
-f | --flow                    flow control
-b | --baudrate baudrate  set baudrate, default baudrate: 115200
-l | --loop              operate circularly
-w | --write  frame     frame string. such as: 0123456789
-h | --help          Print this message
```

- 两块开发板一个作为发送端另一个作为接收端，先在一块开发板执行以下命令发送数据：

```
# tty_test -d /dev/ttyO5 -b 9600 -m 1  -w 0123456789 -l
 SEND:0123456789
```

- 再在另一开发板执行以下命令接收数据：

```
#  tty_test -d /dev/ttyO5 -b 9600 -m 1 -l
RECV:0, total:1
RECV:1, total:1
RECV:2, total:1
RECV:3, total:1
RECV:4, total:1
RECV:5, total:1
RECV:6, total:1
RECV:7, total:1
RECV:8, total:1
RECV:9, total:1
```

- 两块板互换一下角色，结果一样，不再赘述。

MYD-AM437X系列其它板型RS485测试情况类似。

## 4.7 CAN Bus 测试

本例程演示使用Linux API让开发板上的CAN实现点对点发送和接收数据，详情请参考源码。

测试硬件环境：

- MYD-AM437X系列开发板两块
- 数据线连接两块板的CAN接口，CANH0<->CANH0，CANL0<->CANL0，GND<->GND
- USB转TTL调试串口一根，连接MYD-AM437X系列开发板 Debug串口和PC, PC端波特率设置115200-8-n-1

| 接口 | **MYD-C437X** | **MYD-C437X-PRU** |
| --- | --- | --- |
| debug串口 | J16 UART0 | J25 Debug UART |
| CAN接口 | J15 CANH0、CANL1、GND、CANH1、CANL1 | J10 CANH、CANL、GNDISO |

测试软件环境：

- Linux Kernel 4.1.18
- can_test 应用程序
- Buildroot自带can_utils应用程序包
- ip link应用程序

| 开发板 | 设备树 |
| --- | --- |
| MYD-C437X | myd_c437x_evm.dtb、myd_c437x_evm_hdmi.dtb |
| MYD-C437X-PRU | myd_c437x_idk.dtb、myd_c437x_idk_lcd.dtb |

测试过程：

- 将目录 `<WORKDIR>/Examples/rootfs/usr/bin/` 中的可执行程序can_test拷贝至开发板/usr/bin/, 分别将两块开发板上的can0通信波特率都设置位50Kbps，并使能can0设备, 如下所示:

```
# can_test --help
Usage: can_test [options]

Version 1.0
Options:
-d | --device name   can device name: can0
-b | --baudrate baudrate   set baudrate, default baudrate:50000
-l | --loop         operate circularly, default not operate circularly!
-w | --write  frame     frame string with format ID#MESSAGE.
                             such as: 123#112233445566
-h | --help                Print this message
```

```
# ip link set can0 down
# ip link set can0 type can bitrate 50000 triple-sampling on
# ip link set can0 up
```

- 上述设置波特率的命令在can_test程序初始化时已经执行，不需再次手动执行。两块开发板一个作为发送端另一个作为接收端，先在一块开发板执行以下命令发送数据：

```
# chmod 777 /usr/bin/can_test
# can_test -d can0 -w 123#112233445566
[ 6862.997962] c_can_platform 481cc000.can can0: setting
BTR=1c1d BRPE=0000
====== write  frame: ======
 frame_id = 0x123
 frame_len = 6
 frame_data = 0x11 0x22 0x33 0x44 0x55 0x66
============================
```

- 再在另一开发板执行以下命令接收数据：

```
# chmod 777 /usr/bin/can_test
# can_test -d can0 -l
[ 6484.014811] c_can_platform 481cc000.can can0: setting BTR=1c1d
BRPE=0000 can0  0x123  [6]  0x11 0x22 0x33 0x44 0x55 0x66
```

- 如果使用-l参数，则会循环执行读写操作，两块板互换一下角色，结果一样，不再赘述。

注意, 如果程序运行过程中出现下述错误，可以修改tx_queue_len， 如下：

```
# can_test -d can0 -w 123#112233445566
  can raw socket write: No buffer space available

# echo 1000 > /sys/class/net/can0/tx_queue_len
```

- 用同样的方法测试两块板子的can1接口，结果一样，不再赘述。

MYD-AM437X系列其它板型CAN BUS测试情况类似。

## 4.8 KEY测试

本例程演示使用Linux User API读取按键信息，详情请参考源码。

测试硬件环境：

- MYD-AM437X系列 开发板一块
- USB转TTL调试串口一根，连接MYD-AM437X系列开发板 Debug串口和PC, PC端波特率设置115200-8-n-1

| 接口 | **MYD-C437X** | **MYD-C437X-PRU** |
| --- | --- | --- |
| debug串口 | J16 UART0 | J25 Debug UART |

测试软件环境：

- Linux Kernel 4.1.18
- hexdump或keypad_test应用程序

| 开发板 | 设备树 |
| --- | --- |
| MYD-C437X | myd_c437x_evm.dtb、myd_c437x_evm_hdmi.dtb |
| MYD-C437X-PRU | myd_c437x_idk.dtb、myd_c437x_idk_lcd.dtb |

测试过程：

MYIR-AM437X系列开发板提供了四个按键POWER(SW1)、RESET(SW2)、USER0(SW3)、USER1(SW4)，其中RESET键按下会复位系统所以不在本次测试中测试。
由于设备树上的差异性，在不同的MYIR-AM437X系列开发板上的POWER键对应的event事件编号会不一样，如下表所示：

| 按键 | **MYD-C437X** | **MYD-C437X-PRU** |
| --- | --- | --- |
| POWER | event2 | event1 |

- 将目录 `<WORKDIR>/Examples/rootfs/usr/bin/` 中的可执行程序keypad_test拷贝至开发板/usr/bin/目录下，执行该程序进行测试如下：

```
$ chmod 777 /usr/bin/keypad_test
$ keypad_test -h
Usage: keypad_test [options]

Version 1.0
Options:
-d | --device name   keypad device name, default: /dev/input/event0
-h | --help          Print this message
```

- 查看按键设备对应的设备节点，可以发现S3,S4两个按键对应/dev/input/event0， 而S1对应/dev/input/event1

```
#  ls /dev/input/
by-path  event0   event1   event2   event3   mice     mouse0   mouse1

# cat /sys/class/input/event0/device/name
volume_keys@0

# cat /sys/class/input/event1/device/name
tps65218_pwrbutton

# cat /sys/class/input/event2/device/name
ti-tsc

# cat /sys/class/input/event3/device/name
ft5x06_ts
```

- 测试S3和S4

```
# keypad_test  -d /dev/input/event0
Event: Code = 115, Type = 1, Value=1               -- 按下S3
Event: Code = 0, Type = 0, Value=0
Event: Code = 115, Type = 1, Value=2
Event: Code = 0, Type = 0, Value=1
Event: Code = 115, Type = 1, Value=0               -- 释放S3
Event: Code = 0, Type = 0, Value=0
Event: Code = 114, Type = 1, Value=1               -- 按下S4
Event: Code = 0, Type = 0, Value=0
Event: Code = 114, Type = 1, Value=0               -- 释放S4
Event: Code = 0, Type = 0, Value=0
```

- 测试S1

```
# kepad_test  -d /dev/input/event2
Event: Code = 116, Type = 1, Value=1               -- 按下SW1
Event: Code = 0, Type = 0, Value=0
Event: Code = 116, Type = 1, Value=0               -- 释放SW1
Event: Code = 0, Type = 0, Value=0
```

以上测试结果中返回的Event Code即为按键键值，和内核设备树中定义的键值必须一致。
用户也可以通过Buildroot文件系统中自带的hexdump程序进行按键的测试，然后分别按S3,S4这两个按键。 其中输出信息中第7列中的0073和0072即为设备树中定义的按键键值115和114，其它信息含义请参阅hexdump说明。

```
# hexdump /dev/input/event0
0000000 3912 57b2 cc9e 0007 0001 0073 0001 0000
0000010 3912 57b2 cc9e 0007 0000 0000 0000 0000
0000020 3912 57b2 056d 000b 0001 0073 0000 0000
0000030 3912 57b2 056d 000b 0000 0000 0000 0000

0000040 3915 57b2 1e47 0004 0001 0072 0001 0000
0000050 3915 57b2 1e47 0004 0000 0000 0000 0000
0000060 3915 57b2 6c14 0007 0001 0072 0000 0000
0000070 3915 57b2 6c14 0007 0000 0000 0000 0000
```

测试S1按键方法类似，如下：

```
# hexdump /dev/input/event2
0000000 937b 5956 879e 000b 0001 0074 0001 0000
0000010 937b 5956 879e 000b 0000 0000 0000 0000
0000020 937b 5956 89c0 000e 0001 0074 0000 0000
0000030 937b 5956 89c0 000e 0000 0000 0000 0000
```

MYD-AM437X系列其它板型的KeyPad测试情况类似。

## 4.9 LED测试

Linux系统下，LED的控制主要是通过访问sysfs文件系统下文件节点实现的，本例程演示如何使用echo命令和led_test应用对LED进行控制，验证开发板上的LED驱动程序。

测试硬件环境：

- MYD-AM437X系列开发板一块
- USB转TTL调试串口一根，连接MYD-AM437X系列开发板 Debug串口和PC, PC端波特率设置115200-8-n-1

| 接口 | **MYD-C437X** | **MYD-C437X-PRU** |
| --- | --- | --- |
| debug串口 | J16 UART0 | J25 Debug UART |
| LED_CPU0 | 核心板D36 | 核心板D36 |
| LED_HeartBeat | 底板D34 | 底板D20 |
| LED_mmc1 | 底板D35 | 底板D19 |
| LED_usr3 | 地板D36 | 底板D18 |

测试软件环境：

- Linux Kernel 4.1.18
- echo, led_test 应用程序

| 开发板 | 设备树 |
| --- | --- |
| MYD-C437X | myd_c437x_evm.dtb、myd_c437x_evm_hdmi.dtb |
| MYD-C437X-PRU | myd_c437x_idk.dtb、myd_c437x_idk_lcd.dtb |

测试过程：

查看led设备的设备节点

```
# ls /sys/class/leds/
myc:blue:cpu0       myd:blue:mmc1
myd:blue:heartbeat  myd:blue:usr3
```

- 用 `echo` 命令对LED进行控制

```
#echo "0" > /sys/class/leds/myc:blue:cpu0/brightness
#echo "1" > /sys/class/leds/myc:blue:cpu0/brightness
#echo "0" > /sys/class/leds/myc:blue:cpu0/brightness

#echo "1" > /sys/class/leds/myd:blue:mmc1/brightness
#echo "0" > /sys/class/leds/myd:blue:mmc1/brightness

#echo "1" > /sys/class/leds/myd:blue:heartbeat/brightness
#echo "0" > /sys/class/leds/myd:blue:heartbeat/brightness

#echo "1" > /sys/class/leds/myd:blue:usr3/brightness
#echo "0" > /sys/class/leds/myd:blue:usr3/brightness
```

- 将目录 `<WORKDIR>/Examples/rootfs/usr/bin/` 中的可执行程序led_test拷贝至开发板/usr/bin/目录下, 执行该程序进行测试如下：

```
# led_test -h
Usage: led_test [options]

Version 1.0
Options:
-d | --device name   led name myc:blue:cpu0
-l | --light brightness   led brightness. 0~255 0: off.
-h | --help                Print this message
# led_test -d myc:blue:cpu0 -l 0
Set led myc:blue:cpu0 off, brightness = 0
# led_test -d myc:blue:cpu0 -l 1
Set led myc:blue:cpu0 off, brightness = 1
```

- 观察LED的变化

> 注意：由于myc:blue:cpu0设置了触发器为cpu0, 所以不能直接控制，需要先往/sys/class/leds/myc:blue:cpu0/brightness文件节点写"0", 使触发器失效之后才能像其它LED一样正常控制。

MYD-AM437X系列其它板型LED测试情况类似。

## 4.10 EEPROM测试

本例程演示使用Linux User API对开发板上EEPROM进行读写测试，详情请参考源码。

测试硬件环境：

- MYD-AM437X系列开发板一块
- 请确认板上带有EEPROM芯24256E
- USB转TTL调试串口一根，连接MYD-AM437X系列开发板 Debug串口和PC, PC端波特率设置115200-8-n-1

| 接口 | **MYD-C437X** | **MYD-C437X-PRU** |
| --- | --- | --- |
| debug串口 | J16 UART0 | J25 Debug UART |

测试软件环境：

- Linux Kernel 4.1.18
- eeprom_test应用程序

| 开发板 | 设备树 |
| --- | --- |
| MYD-C437X | myd_c437x_evm.dtb、myd_c437x_evm_hdmi.dtb |
| MYD-C437X-PRU | myd_c437x_idk.dtb、myd_c437x_idk_lcd.dtb |

测试过程：

- 将目录 `<WORKDIR>/Examples/rootfs/usr/bin/` 中的可执行程序eeprom_test拷贝至开发板/usr/bin/目录下，执行该程序进行测试如下：

```
# chmod 777 /usr/bin/eeprom_test
# eeprom_test -h
Usage: eeprom_test [options]

Version 1.0
Options:
-d | --device name   i2c device name: /dev/i2c-0
-a | --address addr     eeprom i2c address, default 0x50
-s | --start addr       start offset to read/write
-r | --read  count    read byte count
-w | --write frame      write frame string. such as: 0123456789
-h | --help               Print this message
```

- 测试读写之前，需要先控制GPIO3_7，禁用EEPROM写保护。eeprom_test执行时已经禁用了EEPROM写保护，用户不需要再手动执行

```
# echo 103 > /sys/class/gpio/export
# echo "out" > /sys/class/gpio/gpio103/direction
# echo 0 > /sys/class/gpio/gpio103/value
```

- EEPROM读写测试

```
# eeprom_test -d /dev/i2c-0 -a 0x50 -w "hello world!"
WRITE:hello world!
WRITE SUCCESS!


# eeprom_test -d /dev/i2c-0 -a 0x50 -r 12
READ:hello world!
TOTAL 12 BYTES.
```

MYD-AM437X系列其它板型的EEPROM测试情况类似。

## 4.11 USB Host 测试

本例程演示通过相关命令来验证USB Host功能。

测试硬件环境：

- MYD-AM437X系列开发板一块
- U盘一个
- USB转TTL调试串口一根，连接MYD-AM437X系列开发板 Debug串口和PC, PC端波特率设置115200-8-n-1

| 接口 | **MYD-C437X** | **MYD-C437X-PRU** |
| --- | --- | --- |
| debug串口 | J16 UART0 | J25 Debug UART |
| USB Host | J13、J12 | J7 |

测试软件环境：

- Linux Kernel 4.1.18
- mount, umount 应用程序

| 开发板 | 设备树 |
| --- | --- |
| MYD-C437X | myd_c437x_evm.dtb、myd_c437x_evm_hdmi.dtb |
| MYD-C437X-PRU | myd_c437x_idk.dtb、myd_c437x_idk_lcd.dtb |

测试过程：

- 将U盘连接到开发板USB Host接口，并且执行mount, umount，读写，插拔等操作。插入U盘至USB Host接口时内核提示信息如下：

```
# [13752.969569] usb 1-1: new high-speed USB device number 2 using xhci-hcd
[13753.114361] usb 1-1: New USB device found, idVendor=0930, idProduct=6545
[13753.121504] usb 1-1: New USB device strings: Mfr=1, Product=2, SerialNumber=3
[13753.129056] usb 1-1: Product: DT 101 G2
[13753.133580] usb 1-1: Manufacturer: Kingston
[13753.138021] usb 1-1: SerialNumber: 001D6095CA1EEC2146A90004
[13753.179033] usb-storage 1-1:1.0: USB Mass Storage device detected
[13753.189488] scsi host0: usb-storage 1-1:1.0
[13753.197187] usbcore: registered new interface driver usb-storage
[13754.266687] scsi 0:0:0:0: Direct-Access  Kingston DT 101 G2  PMAP PQ: 0 ANSI: 0 CCS
[13755.539278] sd 0:0:0:0: [sda] 15240576 512-byte logical blocks: (7.80 GB/7.27 GiB)
[13755.547768] sd 0:0:0:0: [sda] Write Protect is off
[13755.553880] sd 0:0:0:0: [sda] No Caching mode page found
[13755.559903] sd 0:0:0:0: [sda] Assuming drive cache: write through
[13755.591585]  sda: sda1
[13755.602727] sd 0:0:0:0: [sda] Attached SCSI removable disk
```

- 将U盘挂载到文件系统/mnt目录，并执行基本的查看，读写操作。

```
#
# mount /dev/sda1 /mnt
[ 1301.201855] FAT-fs (sdb1): Volume was not properly unmounted. Some data may be corrupt.
Please run fsck.

# ls /mnt
u-boot.img          MLO             helloworld
```

- 拔下U盘，内核提示信息。

```
#
# [14018.109698] usb 1-1: USB disconnect, device number 2
```

- 其他USB host接口测试方法参照上面操作。

MYD-AM437X系列其它板型USB Host接口测试情况类似。

## 4.12 USB DEVICE测试

本实例演示通过相关命令来验证USB DEVICE的功能。使用USB数据线连接开发板的miniUSB接口与电脑端的USB接口，开发板实现一个读卡器功能。

测试硬件环境：

- MYD-AM437X系列开发板一块
- TF卡一个
- USB Type A to Mini 数据 线一根
- USB转TTL调试串口一根，连接MYD-AM437X系列开发板 Debug串口和PC, PC端波特率设置115200-8-n-1

| 接口 | **MYD-C437X** | **MYD-C437X-PRU** |
| --- | --- | --- |
| debug串口 | J16 UART0 | J25 Debug UART |
| USB DEVICE | J14 | J8 |
| SD卡接口 | J22 | J19 |

测试软件环境：

- Linux Kernel 4.1.18
- modprobe 应用程序

| 开发板 | 设备树 |
| --- | --- |
| MYD-C437X | myd_c437x_evm.dtb、myd_c437x_evm_hdmi.dtb |
| MYD-C437X-PRU | myd_c437x_idk.dtb、myd_c437x_idk_lcd.dtb |

测试过程：

- 开发板上的系统起来后，使用USB Type A to Mini B转接线连接开发板USB DEVICE接口与电脑端，其中USB mini B接口连接开发板，USB Type A接口连接电脑端，将TF插入到SD卡接口,在开发板上运行如下命令：

```
# modprobe g_mass_storage stall=0 file=/dev/mmcblk0p1 removable=1
[15151.225218] Mass Storage Function, version: 2009/09/11
[15151.231350] LUN: removable file: (no medium)
[15151.236837] LUN: removable file: /dev/mmcblk0p1
[15151.242148] Number of LUNs=1
[15151.245236] Number of LUNs=1
[15151.248511] g_mass_storage gadget: Mass Storage Gadget, version: 2009/09/11
[15151.256739] g_mass_storage gadget: userspace failed to provide iSerialNumber
[15151.264318] g_mass_storage gadget: g_mass_storage ready
[15151.270155] dwc3 48390000.usb: otg: gadget gadget registered
# [15152.172523] g_mass_storage gadget: high-speed config #1: Linux File-Backed Storage
```

- g_mass_storage 驱动加载成功之后， PC端会识别到一个可移动磁盘，磁盘内容正是TF上的内容，至此就实现了读卡器的功能

除此之外，通过加载不同的gadget模块驱动，开发板可以实现不同的功能，如g_ether实现RNDIS网卡的功能，g_serial实现串口的功能，等等。这里不一一列举。

MYD-AM437X系列其它板型USB Device接口测试情况类似。

## 4.13 CAMERA测试

本例程演示在linux下的通过V4L2视频框架 API接口实现摄像头的显示，详情请参考源码。

测试硬件环境：

- MYD-AM437X系列开发板一块
- MY-CAM011B摄像头模块两个
- MY-TFT070-K显示屏一块
- USB转TTL调试串口一根，连接MYD-AM437X系列开发板 Debug串口和PC, PC端波特率设置115200-8-n-1

| 接口 | **MYD-C437X** | **MYD-C437X-PRU** |
| --- | --- | --- |
| debug串口 | J16 UART0 | J25 Debug UART |
| camera接口 | J21、J23 | J18 |
| LCD接口 | J8 | J20 |

> 注意：对于MYD-C437X-PRU开发板camera接口和LCD接口引脚复用所以不能同时使用。

测试软件环境：

- Linux Kernel 4.1.18
- camera_test应用程序

| 开发板 | 设备树 |
| --- | --- |
| MYD-C437X | myd_c437x_evm.dtb、myd_c437x_evm_hdmi.dtb |
| MYD-C437X-PRU | myd_c437x_idk.dtb |

测试过程：

**MYD-C437X**测试过程：

- 开发板关机，将两个摄像头模块MY-CAM011B分别接入到camera接口中，MY-TFT070-K显示屏接入LCD接口，上电开机。通过命令查看摄像头驱动是否加载成功并生成设备：

```
# ls /dev/v*
/dev/vcs          /dev/vcsa         /dev/vga_arbiter  /dev/video1
/dev/vcs1         /dev/vcsa1        /dev/video0
```

可以看到驱动加载正常并成功找到两个摄像头设备video0、video1

- 将目录 `<WORKDIR>/Examples/rootfs/usr/bin/` 中的可执行程序camera_test拷贝至开发板/usr/bin/,执行下面命令：

```
#chmod +x /usr/bin/camera_test
# camera_test -h
Usage: camera_test [options]

Version 1.0
Options:
-h | --help               Print this message
```

可以看到本例程不需要其他参数，直接运行：

```
# camera_test
xres:800 >>> yres:480 >>> bpp:32>>>
CRTCs size: 800x480

Capture 0: Opened Channel

Capture 0: Capable of streaming

Capture 0: Init done successfully

Exported buffer fd = 7

Exported buffer fd = 9

Exported buffer fd = 11


Capture 1: Opened Channel

Capture 1: Capable of streaming

Capture 1: Init done successfully

Exported buffer fd = 14

Exported buffer fd = 16

Exported buffer fd = 18
```

运行成功后即可在7寸LCD显示屏上看到两个摄像头的画面

**MYD-C437X-PRU**测试过程：

- 开发板关机，将摄像头模块MY-CAM011B接入到camera接口中，上电开机。通过命令查看摄像头驱动是否加载成功并生成设备：

```
# ls /dev/v*
v4l/         vcs1         vcsa1        video0
vcs          vcsa         vga_arbiter
```

可以看到驱动加载正常并成功找到一个摄像头设备video0

- 插入SD卡到SD卡接口中，挂载SD卡到系统 `/mnt/sdcard` 目录下：

```
# [  221.139812] mmc0: host does not support reading read-only switch, assuming write-enable

# mkdir /mnt/sdcard
# mount -t vfat /dev/mmcblk0p1 /mnt/sdcard
```

> 注意：SD卡的设备名称，会根据系统启动方式有所变化：SD卡启动时，设备名称为/dev/mmcblk0p1 ；EMMC启动时，设备名称为/dev/mmcblk1p1

- 通过命令拍照并存储到sd卡中，具体操作如下：

```
# v4l2grab -d /dev/video0 -o /mnt/sdcard/test.jpg -W 800 -H 600
Unable to set frame interval.

# ls /mnt/sdcard/
test.jpg
#umount /mnt/sdcard
```

- 拔出SD卡，插入到PC的读卡器中，查看拍摄的照片。

MYD-AM437X系列其它板型camera测试可以参照本例程进行。

## 4.14 AUDIO测试

本例程演示通过linux下ALSA音频框架及其alsa-utils工具集来实现音频播放和录音的功能。

测试硬件环境：

- MYD-AM437X系列开发板一块
- 双头耳机一副
- USB转TTL调试串口一根，连接MYD-AM437X系列开发板 Debug串口和PC, PC端波特率设置115200-8-n-1

| 接口 | **MYD-C437X** | **MYD-C437X-PRU** |
| --- | --- | --- |
| debug串口 | J16 UART0 | J25 Debug UART |
| 耳机接口 | J19 LINE_OUT | 无 |
| 麦克风接口 | J18 LINE_IN | 无 |

测试软件环境：

- Linux Kernel 4.1.18
- alsa-utils工具集，audio_test应用程序

| 开发板 | 设备树 |
| --- | --- |
| MYD-C437X | myd_c437x_evm.dtb |

测试过程：

- 开发板开机，将双头耳机的耳机头插入开发板耳机接口，将麦克风头插入到开发板麦克风接口。
- 测试音频播放，将准备好的WAV格式的音频文件拷贝到开发板 `/media/test.wav` 目录下，用alsa-utils工具集里面的 `aplay` 来测试音频播放：

```
# cd media/
# ls
test.wav
# aplay test.wav
Playing WAVE 'test.wav' : Signed 16 bit Little Endian, Rate 44100 Hz, Stereo
```

此时通过耳机即可听到正在播放的音乐文件，如需调节音量可通过alsa-utils工具集里面的 `alsamixer` 来调节，这个命令执行然后就会出现字符形式的图形界面，使用四个方向键就可以进行调节：

```
+----------------------------- AlsaMixer v1.1.2 ------------------------------+
| Card: MYD-C437X-EVM                              F1:  Help                |
| Chip:                                            F2:  System information  |
| View: F3:[Playback] F4: Capture  F5: All         F6:  Select sound card   |
| Item: Headphone [dB gain: -4.00, -4.00]          Esc: Exit                |
|                                                                           |
|     +--+                      +--+     +--+                               |
|     |  |                      |aa|     |  |                               |
|     |  |                      |aa|     |  |                               |
|     |  |                      |aa|     |  |                               |
|     |  |                      |aa|     |  |                               |
|     |  |                      |aa|     |  |                               |
|     |  |                      |aa|     |  |                               |
|     |aa|                      |aa|     |  |                               |
|     |aa|                      |aa|     |  |                               |
|     |aa|                      |aa|     |  |                               |
|     |aa|                      |aa|     |  |                               |
|     |aa|                      |aa|     |  |                               |
|     +--+     DAC      +--+     +--+     +--+     +--+    MIC_IN    +--+      |
|                       |OO|                       |MM|              |OO|      |
|                       +--+                       +--+              +--+      |
|    50<>50                     100<>100    0                                |
|  <Headphon>Headphon Headphon   PCM      Mic    Capture  Capture  Capture     |
+---------------------------------------------------------------------------+
```

通过方向键左右切换项目，`Item: Headphone [dB gain: -4.50, -4.50]`项目为音量调节，通过方向键上下调节所需的音量大小。

- 测试音频输入，使用alsa-utils工具集里面的`arecord`来测试：

```
# arecord -c 2 -r 44100 -f S16_LE record.wav
Recording WAVE 'record.wav' : Signed 16 bit Little Endian, Rate 44100 Hz, Stereo
```

结束录音，通过`aplay`来播放刚才的录音文件：

```
# aplay record.wav
Playing WAVE 'record.wav' : Signed 16 bit Little Endian, Rate 44100 Hz, Stereo
```

用户也可通过audio_test应用程序来测试音频，此应用的功能为实时播放录音的内容。

- 将目录`<WORKDIR>/Examples/rootfs/usr/bin/`中的可执行程序audio_test拷贝至开发板/usr/bin/,执行下面命令：

```
# audio_test
rate set to 21999, expected 22000
Init capture successfully, rate: 21999, period_size: 128
rate set to 21998, expected 21999
Period size: 128 frames, buffer size: 256 bytes
^C38 frames
```

此时通过麦克风说话，耳机端也能同步听到声音。

- MYD-AM437X系列其它板型audio测试可以参照本例程进行。

## 4.15 HDMI测试

本例程演示通过linux的libdrm库提供的接口来访问内核DRM显示框架，实现HDMI接口屏的显示。

测试硬件环境：

- MYD-AM437X系列开发板一块
- HDMI连接线一根，HDMI接口的显示器一台
- USB转TTL调试串口一根，连接MYD-AM437X系列开发板 Debug串口和PC, PC端波特率设置115200-8-n-1

| 接口 | **MYD-C437X** | **MYD-C437X-PRU** |
| --- | --- | --- |
| debug串口 | J16 UART0 | J25 Debug UART |
| HDMI接口 | J9 HDMI | 不支持 |

测试软件环境：

- Linux Kernel 4.1.18
- modetest应用程序

| 开发板 | 设备树 |
| --- | --- |
| MYD-C437X | myd_c437x_evm_hdmi.dtb |

测试过程：

- 关机，把HDMI连接线一端插入到开发板的HDMI接口，另一端接入到显示器的HDMI接口。
- 上电进入到开发板，执行modetest程序如下：

```
# modetest -s 25@27:1024x768
trying to open device 'i915'...failed
trying to open device 'amdgpu'...failed
trying to open device 'radeon'...failed
trying to open device 'nouveau'...failed
trying to open device 'vmwgfx'...failed
trying to open device 'omapdrm'...done
setting mode 1024x768-60Hz@XR24 on connectors 25, crtc 27
```

显示屏上即可看到画面。

- MYD-AM437X系列其它板型HDMI测试可以参照本例程进行。

## 4.16 PRU测试

本例程演示Linux系统下如何在用户空间使用rpmsg实现和PRU的通讯，PRU-ICSS0实现LED控制的功能，通过rpmsg向PRU发送0~7这几个数字字符，PRU根据收到的数字对板上3个LED进行控制。
PRU-ICSS1实现工业以太网扩展的功能，下面分别加以说明。

测试硬件环境：

- MYD-C437X-PRU 开发板一块
- CAT5 网线三根,4口或以上交换机一台
- USB转TTL调试串口一根，连接MYD-AM437X系列开发板 Debug串口和PC, PC端波特率设置115200-8-n-1

| 接口 | **MYD-C437X** | **MYD-C437X-PRU** |
| --- | --- | --- |
| debug串口 | J16 UART0 | J25 Debug UART |
| PRU-ETH0 | 不支持 | J26 PRU-ETH0 |
| PRU-ETH1 | 不支持 | J27 PRU-ETH1 |

测试软件环境：

- Linux Kernel 4.1.18
- ifconfig, ping等命令
- pru_led_test 应用程序

| 开发板 | 设备树 |
| --- | --- |
| MYD-C437X-PRU | myd_c437x_idk.dtb |

测试过程：

- 测试PRU-ICSS1扩展工业以太网接口
  MYD-C437X-PRU开发板默认出厂固件为PRU Ethernet工作模式，实现一个千兆以太网接口J6和两个PRU-ICSS1扩展的百兆工业以太网接口(J26和J27)。 三个网口可以独立工作。将三个网口连接到交换机和PC一起组成一个小型网络，然后测试如下：

```
# ifconfig -a
can0      Link encap:UNSPEC  HWaddr 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00
          NOARP  MTU:16  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:10
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
          Interrupt:159

eth0      Link encap:Ethernet  HWaddr C4:BE:84:CB:13:AF
          inet addr:192.168.30.116  Bcast:192.168.30.255  Mask:255.255.255.0
          inet6 addr: fe80::c6be:84ff:fecb:13af/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:1992 errors:0 dropped:0 overruns:0 frame:0
          TX packets:11 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:138373 (135.1 KiB)  TX bytes:1392 (1.3 KiB)
          Interrupt:143

eth1      Link encap:Ethernet  HWaddr 4E:18:13:C8:60:0D
          BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:1730283589 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

eth2      Link encap:Ethernet  HWaddr 72:90:07:2B:6C:10
          BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:958943060 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
```

系统默认只使能了eth0, eth1和eth2可以通过如下命令手动设置IP地址并使能，如下：

```
# ifconfig eth1 192.168.30.117 netmask 255.255.255.0 up
[ 1361.576798]  remoteproc1: powering up 54434000.pru0
[ 1361.582309]  remoteproc1: Booting fw image ti-pruss/am437x-pru0-prueth-fw.elf,
size 3893
[ 1361.591546] pruss-rproc 54400000.pruss: configured system_events = 0x0000060000500000
intr_channels = 0x00000095 host_intr = 0x00000115
[ 1361.604692]  remoteproc1: remote processor 54434000.pru0 is now up
[ 1361.612466] net eth1: started
[ 1361.615797] IPv6: ADDRCONF(NETDEV_UP): eth1: link is not ready
# [ 1364.079947] eth1: Link is Up - 100Mbps/Full - flow control rx/tx
[ 1364.086377] IPv6: ADDRCONF(NETDEV_CHANGE): eth1: link becomes ready

# ifconfig eth2 192.168.30.118 netmask 255.255.255.0 up
[ 1442.906605]  remoteproc2: powering up 54438000.pru1
[ 1442.912123]  remoteproc2: Booting fw image ti-pruss/am437x-pru1-prueth-fw.elf,
size 3893
[ 1442.921365] pruss-rproc 54400000.pruss: configured system_events = 0x0060000000a00000
intr_channels = 0x0000012a host_intr = 0x0000022a
[ 1442.934397]  remoteproc2: remote processor 54438000.pru1 is now up
[ 1442.942090] net eth2: started
[ 1442.945432] IPv6: ADDRCONF(NETDEV_UP): eth2: link is not ready
# [ 1445.159906] eth2: Link is Up - 100Mbps/Full - flow control rx/tx
[ 1445.166334] IPv6: ADDRCONF(NETDEV_CHANGE): eth2: link becomes ready
```

以上Log信息显示PRU Ethernet连接正常，接下来可以在PC上使用ping命令和开发板上的网口依次进行连通测试，也可以进一步使用iperf工具进行测试，这里不详细介绍了。

- 测试PRU-ICSS控制GPIO LED

定制内核和设备树, 由于MYD-C437X-PRU默认发布的代码使用的是PRU-ICSS1以实现工业以太网接口的扩展，而板上的LED是通过PRU-ICSS0来控制的。
所以需要给内核打上相应的补丁进行切换。补丁打上之后，重新编译zImage和dtbs。补丁内容参见出厂附带资料04-Linux_sources/patches/0001-Enable-PRUSS0-instead-of-PRUSS1-on-the-MYD_C437x_PRU.
由于补丁内容较长，这里仅摘取关于pinmux配置的部分加以说明，如下：

```
// 修改pinmux模式，将AD21, AE22, AD22三个GPIO由arm控制改为由pru控制。
@@ -168,9 +148,10 @@
    leds_pins: leds_pins {
        pinctrl-single,pins = <
            0x244 ( PIN_OUTPUT_PULLUP | MUX_MODE7 ) /* (F23) gpio5_11.gpio5[11] */
-            0x1f0 ( PIN_OUTPUT_PULLUP | MUX_MODE7 ) /* (AD21) cam1_data2.gpio4[16] */
-            0x1f4 ( PIN_OUTPUT_PULLUP | MUX_MODE7 ) /* (AE22) cam1_data3.gpio4[17] */
-            0x1f8 ( PIN_OUTPUT_PULLUP | MUX_MODE7 ) /* (AD22) cam1_data4.gpio4[18] */
+            0x1f0 ( PIN_OUTPUT_PULLUP | MUX_MODE4 ) /* (AD21) cam1_data2.pr0_pru1_gpo[10] */
+            0x1f4 ( PIN_OUTPUT_PULLUP | MUX_MODE4 ) /* (AE22) cam1_data3.pr0_pru1_gpo[11] */
+            0x1f8 ( PIN_OUTPUT_PULLUP | MUX_MODE4 ) /* (AD22) cam1_data4.pr0_pru1_gpo[12] */
+
```

- 编译PRU端的firmware。PRU端的firmware使用ti-cgt-pru_2.1.3编译器进行编译，对应的代码参见04-Linux_Source\Examples\pru_led目录下的代码PRU_RPMsg_LED0_1。 编译之后得到PRU_RPMsg_LED0_1.out，将其拷贝到linux文件系统/lib/firmware/目录下，并重命名为am437x-pru0_1-fw

```
$ cd <WORKDIR>/ToolChain/
$ ./ti_cgt_pru_2.1.3_linux_installer_x86.bin
$ export PRU_CGT=<WORKDIR>/ToolChain/ti-cgt-pru_2.1.3/
$ make

************************************************************
Building project: PRU_RPMsg_LED0_1

Finished building project: PRU_RPMsg_LED0_1
************************************************************
```

- 重新启动系统，如果Linux系统下出现/dev/rpmsg_pru31设备节点，则说明PRU工作正常，且正确加载了firmware，才可以进行以下测试。 在zImage和dtbs不变的情况下，也可以通过重新加载pru_rproc.ko实现PRU firmware的加载

```
# rmmod pru_rproc -f
[ 5702.650841] pru-rproc 54478000.pru1: pru_rproc_remove: removing rproc 54478000.pru1
[ 5702.660895] pruss-rproc 54440000.pruss: unconfigured system_events = 0x0800000000000000
host_intr = 0x00000002
[ 5702.672138]  remoteproc2: stopped remote processor 54478000.pru1
[ 5702.678938]  remoteproc2: releasing 54478000.pru1
[ 5702.684928] pru-rproc 54474000.pru0: pru_rproc_remove: removing rproc 54474000.pru0
[ 5702.694806] pruss-rproc 54440000.pruss: unconfigured system_events = 0x1000000000000000
host_intr = 0x00000001
[ 5702.706273]  remoteproc1: stopped remote processor 54474000.pru0
[ 5702.713233]  remoteproc1: releasing 54474000.pru0
# modprobe pru_rproc
[ 5707.305735]  remoteproc1: 54474000.pru0 is available
[ 5707.314208]  remoteproc1: Note: remoteproc is still under development and considered
experimental.
[ 5707.324384]  remoteproc1: THE BINARY FORMAT IS NOT YET FINALIZED, and backward
compatibility isn't yet guaranteed.
[ 5707.342644]  remoteproc1: powering up 54474000.pru0
[ 5707.347860]  remoteproc1: Booting fw image am437x-pru0_0-fw, size 84928
[ 5707.355459] pruss-rproc 54440000.pruss: configured system_events = 0x1000000000000000
intr_channels = 0x00000001 host_intr = 0x00000001
[ 5707.368458]  remoteproc1: remote processor 54474000.pru0 is now up
[ 5707.375567] virtio_rpmsg_bus virtio0: rpmsg host is online
[ 5707.381484]  remoteproc1: registered virtio0 (type 7)
[ 5707.386895] pru-rproc 54474000.pru0: PRU rproc node /ocp/pruss@54440000/pru0@54474000
probed successfully
[ 5707.397446] virtio_rpmsg_bus virtio0: creating channel rpmsg-pru addr 0x1e
[ 5707.405191]  remoteproc2: 54478000.pru1 is available
[ 5707.415107] rpmsg_pru rpmsg4: new rpmsg_pru device: /dev/rpmsg_pru30
[ 5707.421988]  remoteproc2: Note: remoteproc is still under development and considered
experimental.
[ 5707.432231]  remoteproc2: THE BINARY FORMAT IS NOT YET FINALIZED, and backward
compatibility isn't yet guaranteed.
[ 5707.448155]  remoteproc2: powering up 54478000.pru1
[ 5707.453860]  remoteproc2: Booting fw image am437x-pru0_1-fw, size 84360
[ 5707.461062] pruss-rproc 54440000.pruss: configured system_events = 0x0800000000000000
 intr_channels = 0x00000002 host_intr = 0x00000002
[ 5707.474342]  remoteproc2: remote processor 54478000.pru1 is now up
[ 5707.481129] virtio_rpmsg_bus virtio1: rpmsg host is online
[ 5707.487010]  remoteproc2: registered virtio1 (type 7)
[ 5707.492891] pru-rproc 54478000.pru1: PRU rproc node /ocp/pruss@54440000/pru1@54438000
probed successfully
[ 5707.503186] virtio_rpmsg_bus virtio1: creating channel rpmsg-pru addr 0x1f
# [ 5707.517016] rpmsg_pru rpmsg5: new rpmsg_pru device: /dev/rpmsg_pru31
```

- pru_led_test程序对LED进行控制

```
# pru_led_test -h
Usage: pru_led_test [options]

Version 1.0
Options:
-d | --device name   pru rpmsg char device name /dev/rpmsg_pru31
-l | --loop          operate circularly, default not operate circularly!
-n | --number num    number send to pru! must be 0~7.
-h | --help          Print this message


# pru_led_test -d /dev/rpmsg_pru31 -n 7
Opened /dev/rpmsg_pru31, sending 1 messages

1 - Sent to PRU: 7
1 - Received from PRU:7

Received 1 messages, closing /dev/rpmsg_pru31
```

- 观察LED的变化, 会发现三个LED随着-n后面参数变化而变化。变化的规律就是数字转化为二进制之后，对应位为1的灯点亮，为0的熄灭。例如3的二进制位011b,则对应D18熄灭,D19,D20两个LED灯亮

# 5. Qt应用开发

在上一节中使用Buildroot构建系统时，我们得到了QT应用的构建工具qmake，可以用它来生成QT应用程序的Makefile和各种工程文件，然后通过交叉编译得到可执行的QT应用程序。
如果开发大型的应用，还需要用到专门的集成开发环境QtCreator。在我们产品发布资料中*03-Tools*目录下提供了一个Ubuntu 64位系统下的QtCreator安装包qt-creator-opensource-linux-x86_64-4.1.0.run。
下面的章节将会介绍集成开发环境QtCreator的安装，配置和开发的过程，并使用QtCreator创建一个简单的演示程序在MYD-C437X开发板上运行。

## 5.1 安装QtCreator

下面是在Ubuntu 64位系统上安装QtCreator的步骤：

```
$ cd <WORKDIR>
$ cp /media/cdrom/03-Tools/Qt/qt-creator-opensource-linux-x86_64-4.1.0.run .
$ chmod a+x qt-creator-opensource-linux-x86_64-4.1.0.run
$ sudo ./qt-creator-opensource-linux-x86_64-4.1.0.run
```

此安装过程类似Windows下应用的安装方法,点击下一步即可。

# 5.2 配置QtCreator

- 配置 QtCreator 开发环境:

运行 QtCreator 后,依次点击 `tools` -> `options` ,出现选项对话框,在左边点击 `Build & Run` ,右边选择 `Compilers` 标签。

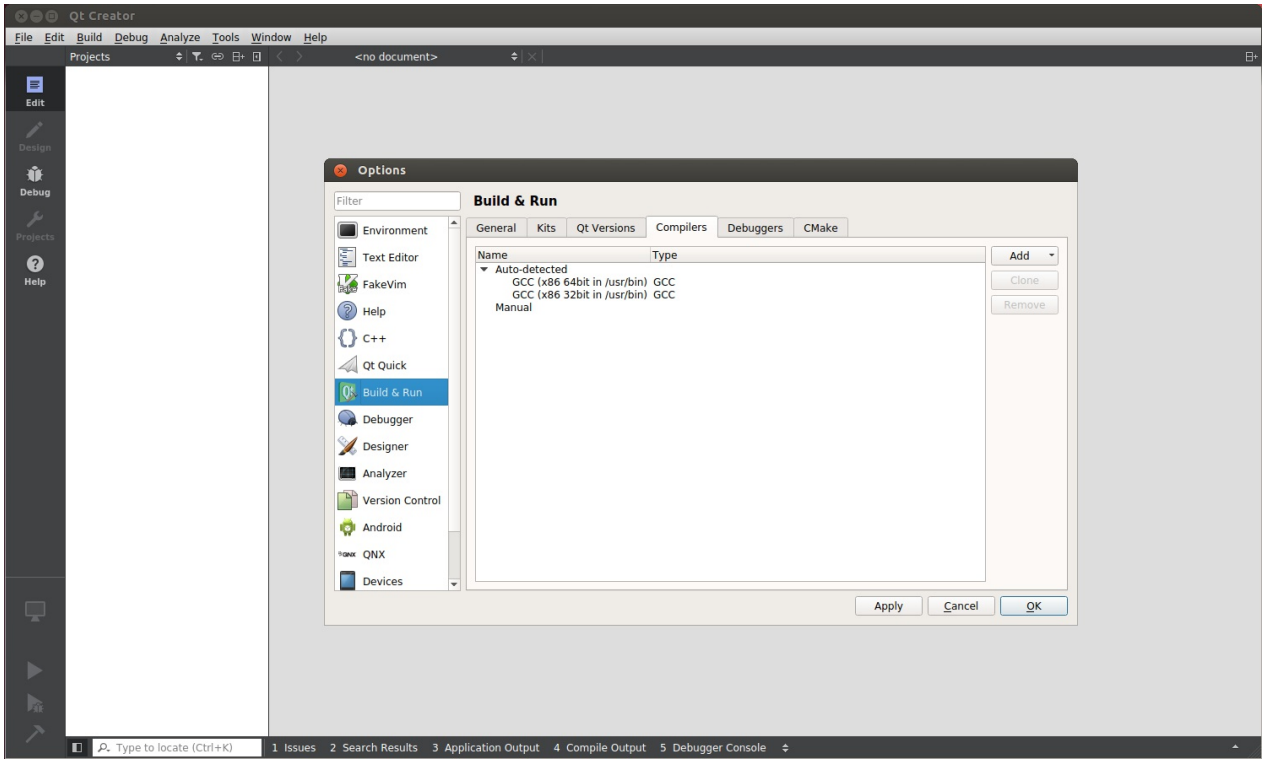

图5-2-1 编译器选择

点击右侧 `add` ,弹出下拉列表后,选择 `custom` ,在下侧填写以下内容: `Name` , `Compiler path` , `Make path` 和 `ABI` 。填写完成后,点击 `Apply` ,进行保存。

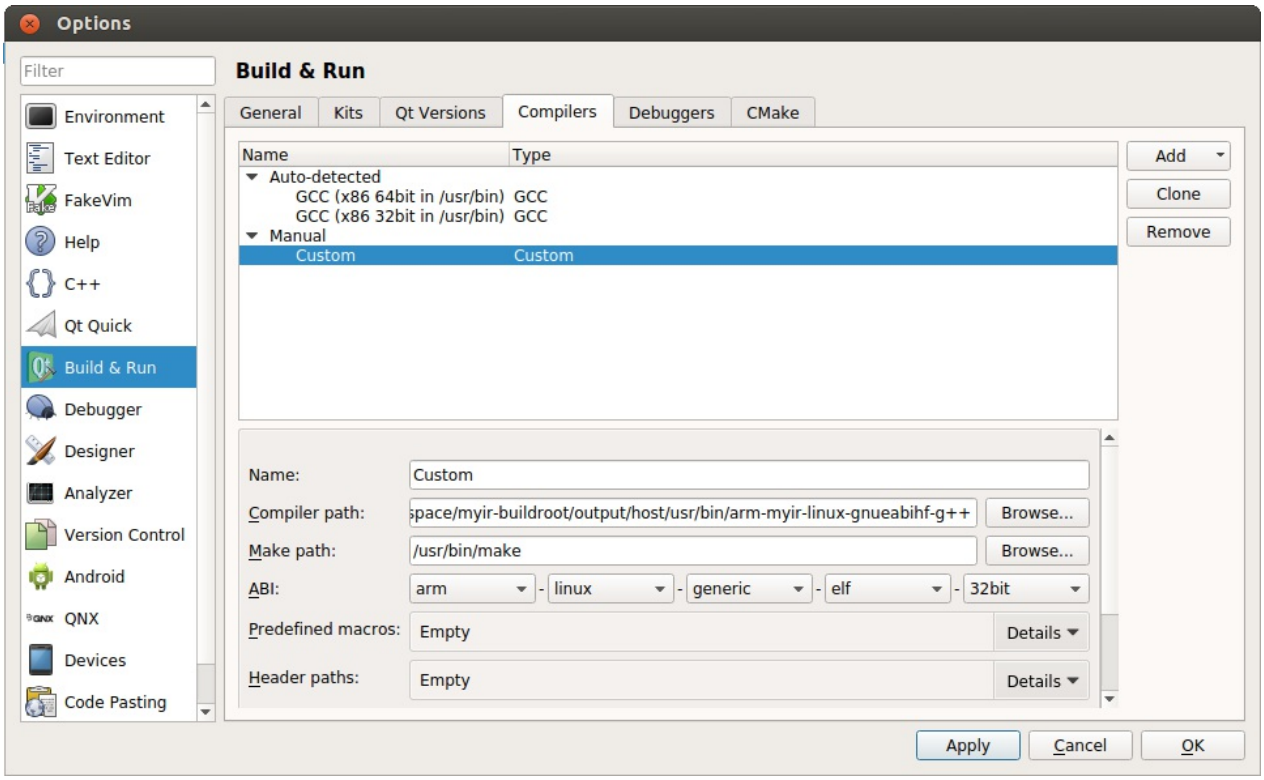

图5-2-2 添加编译器

在同一窗口下,选择 `Qt Version` 标签,在右侧点击 `Add...` ,会弹出对话框,切换目录到Chapter3-4-1中编译QT得到的SDK目录,选择 `qmake` 文件后,点击 `open` 按钮,设置完成之后，点击 `Apply` 按钮保存。

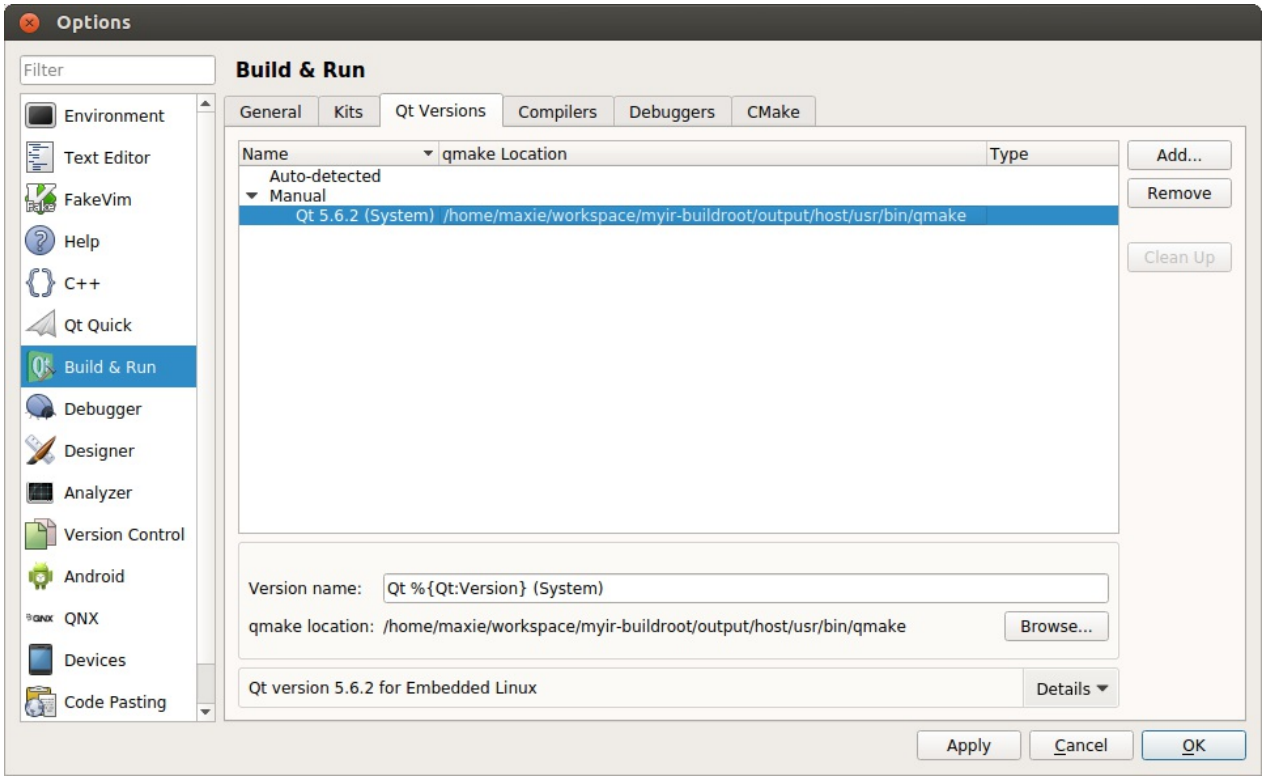

图5-2-3 选择qmake构建工具

在 `Build&Run` 窗口，继续选择上边的 `Kits` 标签,点击右侧 `Add` ,填写相应内容。其中 `Sysroot` 选择编译工具链的目录, `compiler` 选择之前填写的名称, `Debugger` 选择 `None` , `Qt version` 选择之前添加时的名称, `CMake Tool` 设置为默认。

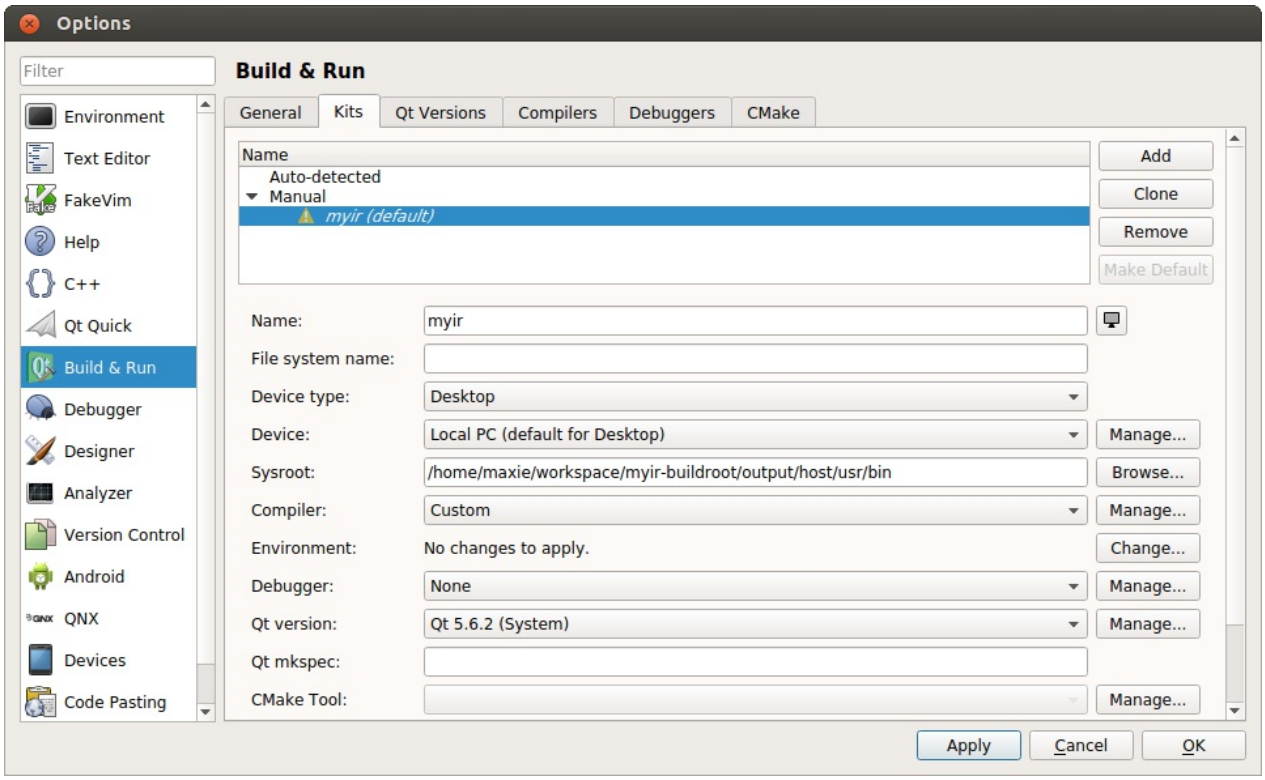


图5-2-4 添加Kits

- 创建 Helloworld 项目:

在菜单栏选择 `File -> New File or Project` ,在打开的对话框中,依次选择 `Application -> Qt Widgets Application` ,点击 `Choose...` , 如下图所示:

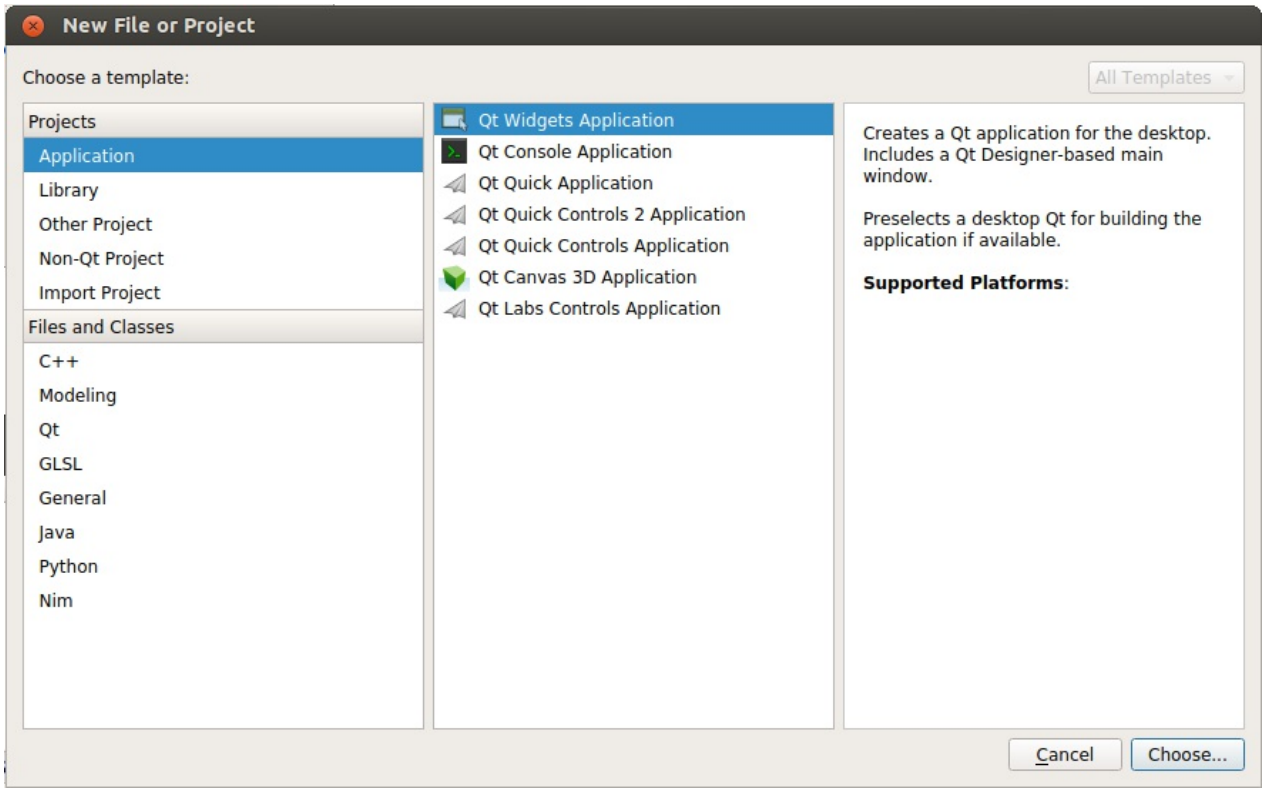


图5-2-5 创建新工程

在弹出的 `Qt Widgets Application` 对话框中,设置项目名称为 `helloworld` , `Create in` 一栏填写项目的存储路径, 如下：

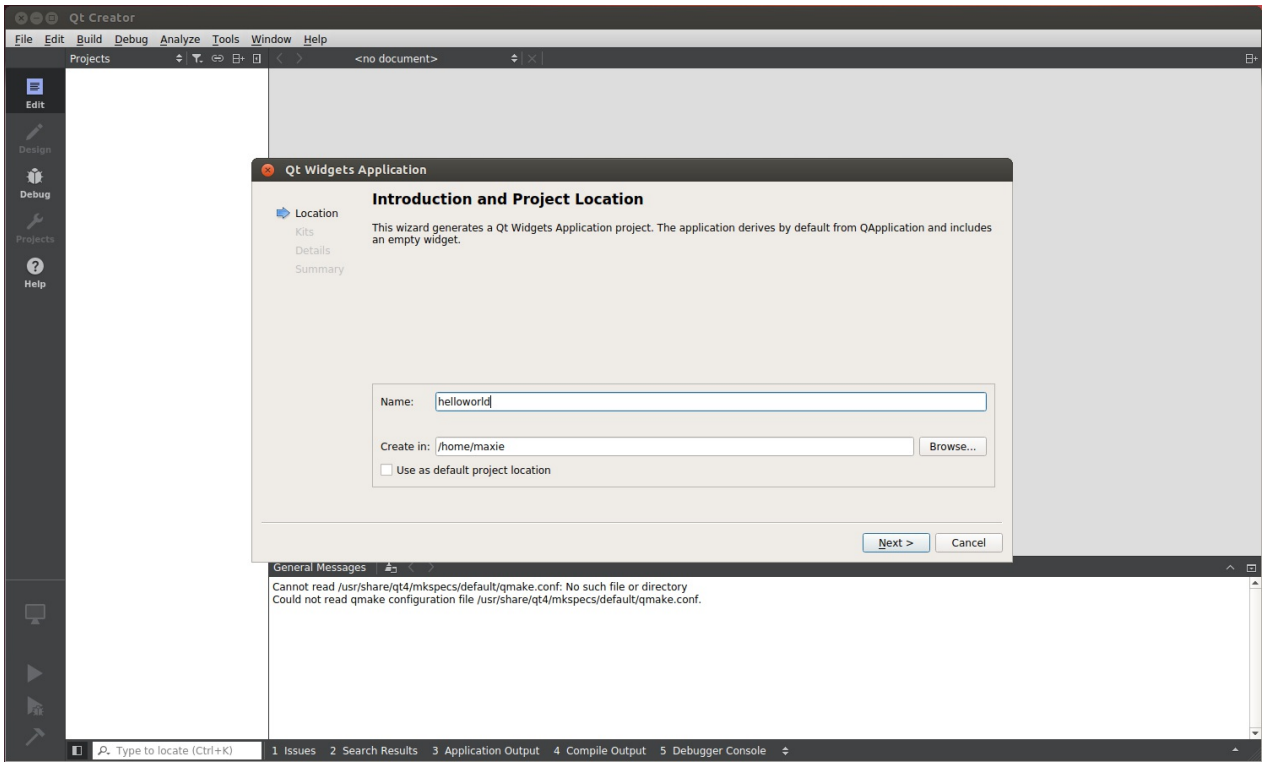

图5-2-6 设置新工程名称和路径

点击 `Next` 后,选择之前添加好的 `Kits` ,继续点击 `Next` , 如下:

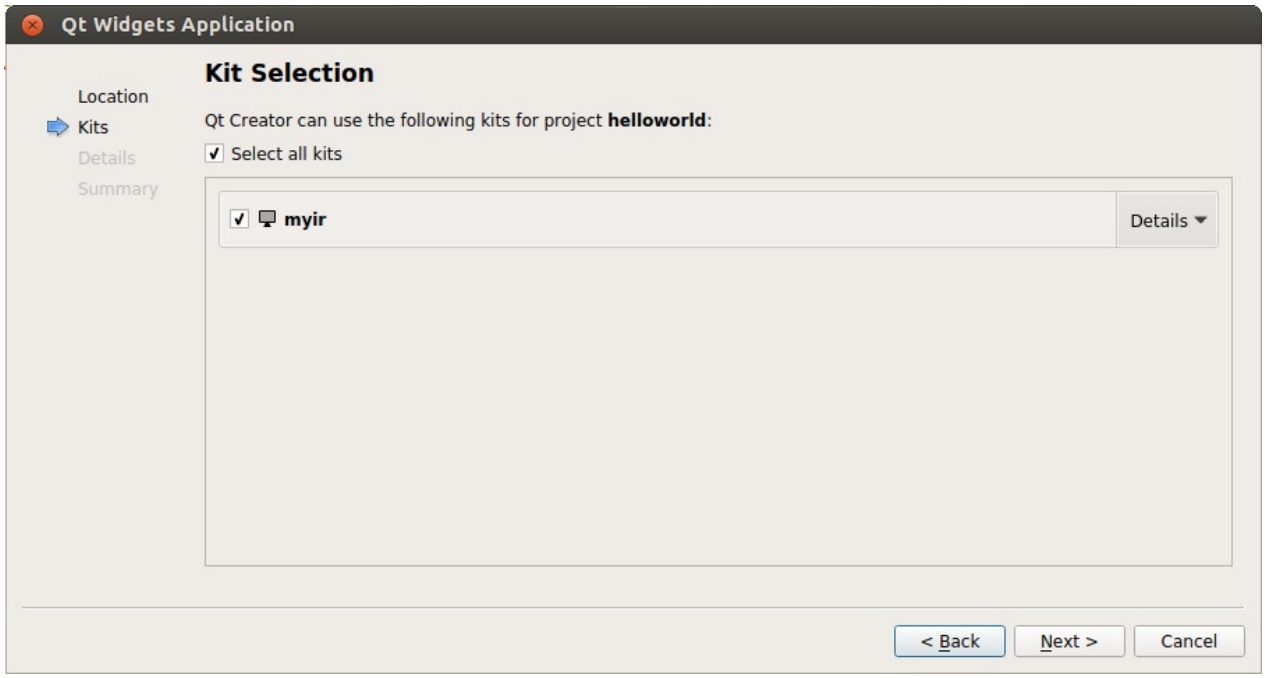

图5-2-7 设置新工程Kits

选择当前的的应用继承自哪种Widget,默认选择QMainWindow,然后点击 `Next` 进入下一步。

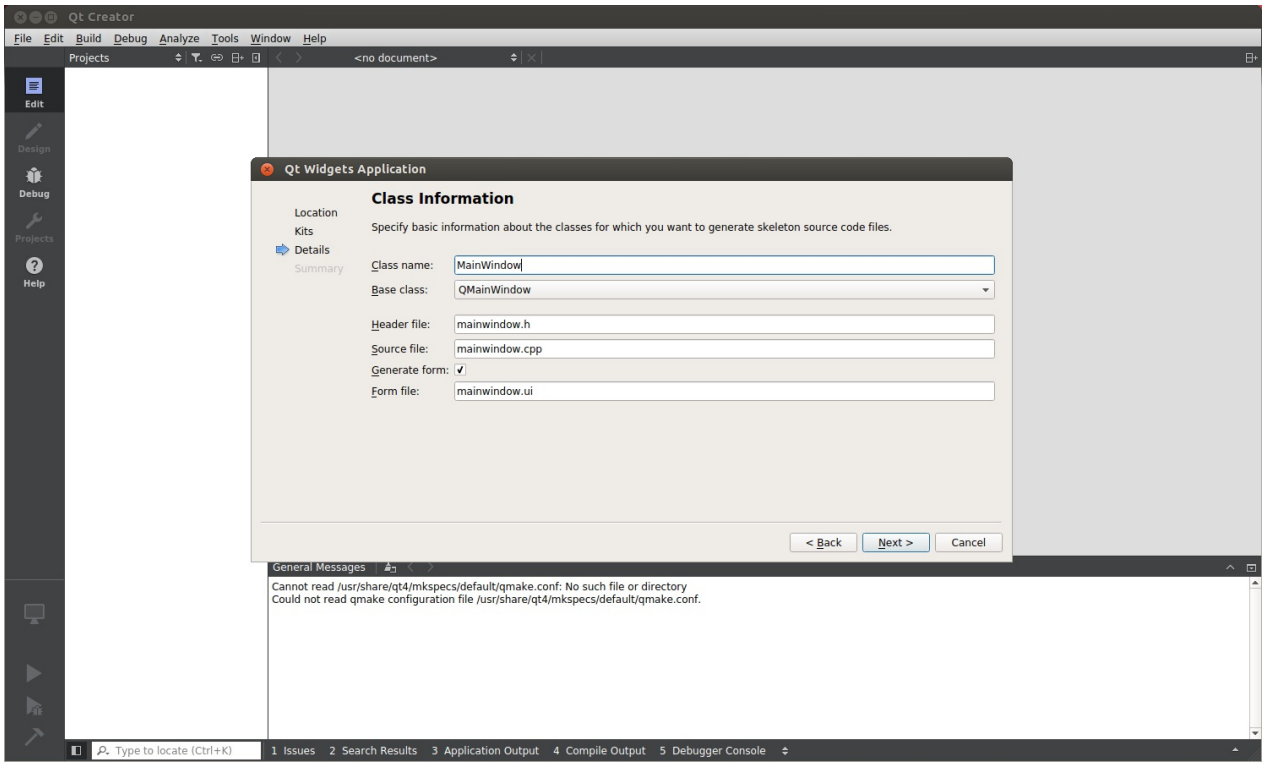


图5-2-8 选择应用程序的类型

以上信息填写完后,点击 `Finish` ,完成QtCreator工程的创建。

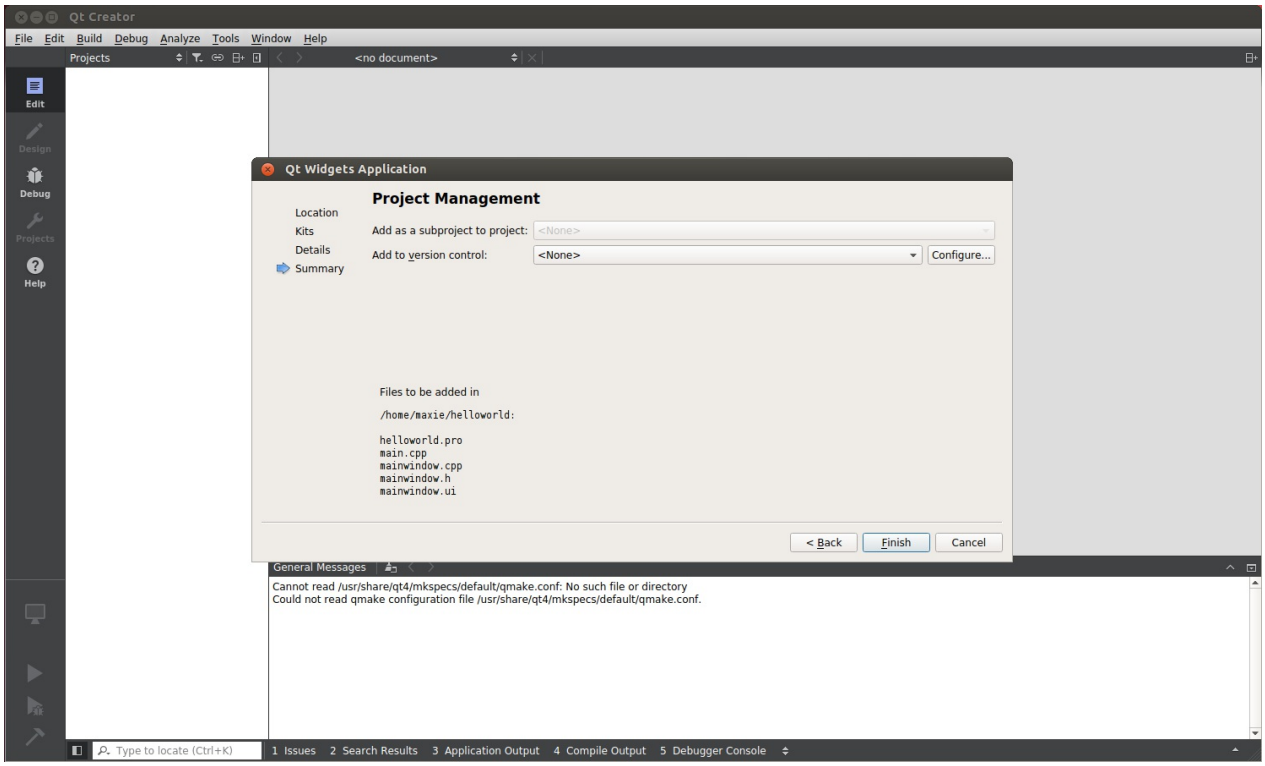


图5-2-9 完成工程创建

# 5.3 编译运行QT应用

以下是创建好的 `helloworld` 项目截图,左侧是 `QtCreator` 创建好的项目目录结构,右侧是代码编辑区。

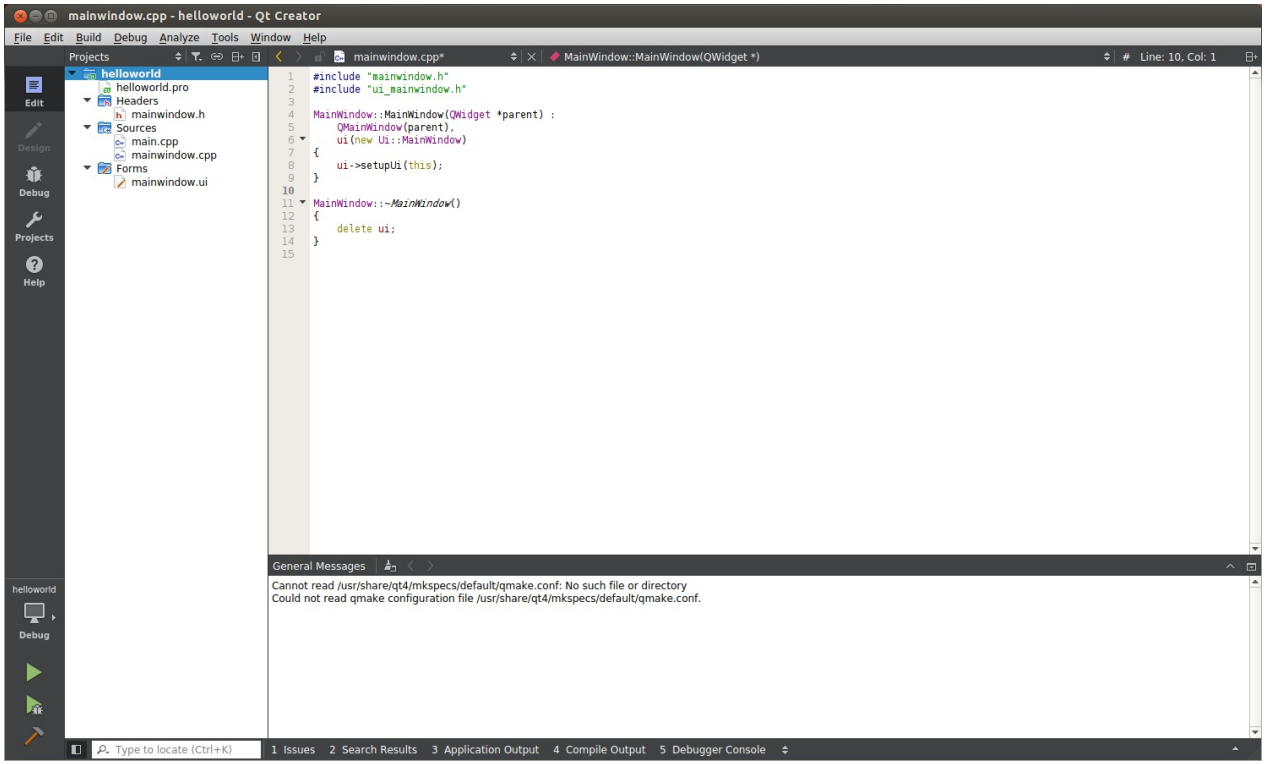

图5-3-1 项目文件管理窗口

双击左侧的 `Forms` 里的 `mainwindow.ui` 文件,打开 `Design` 视图,从左铡 `Display Widgets` 栏目下,拖动 `Label` 到中间的区域。双击后,修改内容为 `Hello,world!` 。

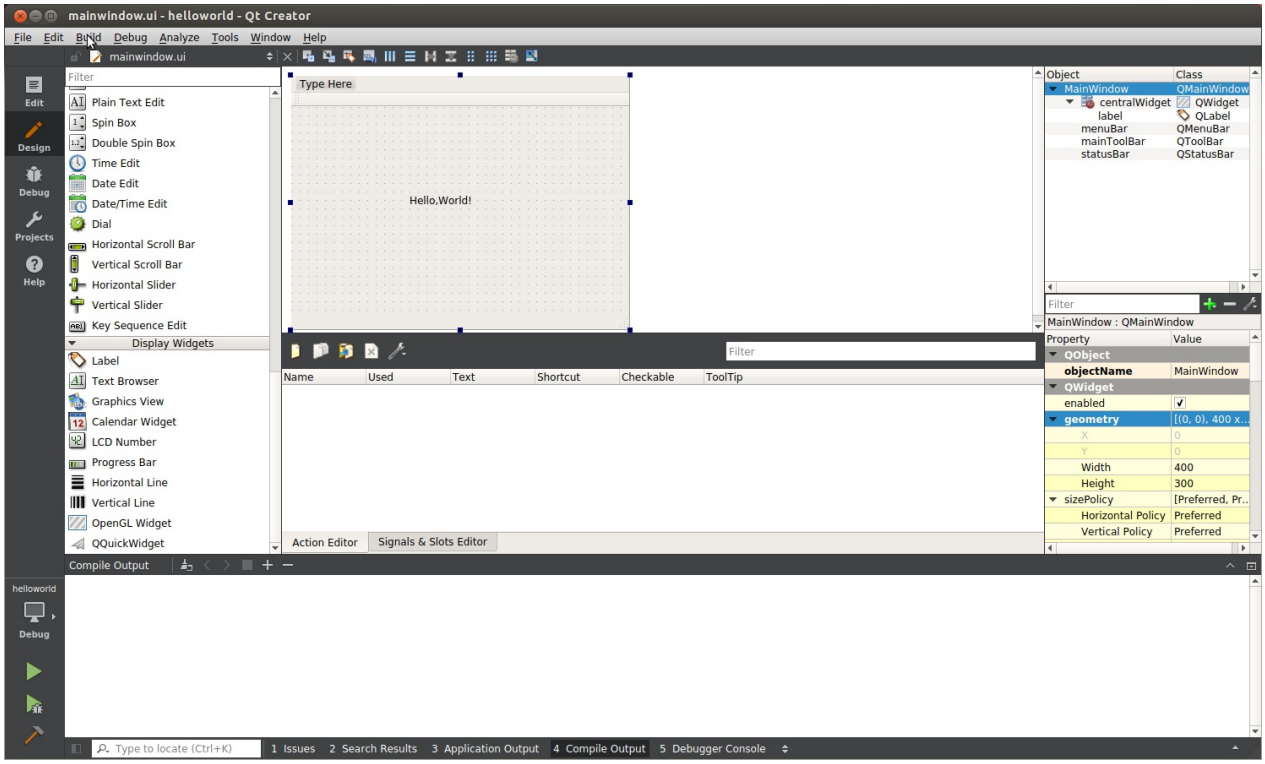

图5-3-2 可视化界面编辑

完成以上操作后,就可以点击菜单栏 `Build -> Build Project hellowrld` ,进行项目构建,此时下侧 `Compile Output` 会有编译信息输出。若有错误,请根据提示,修改正确后重新构建。

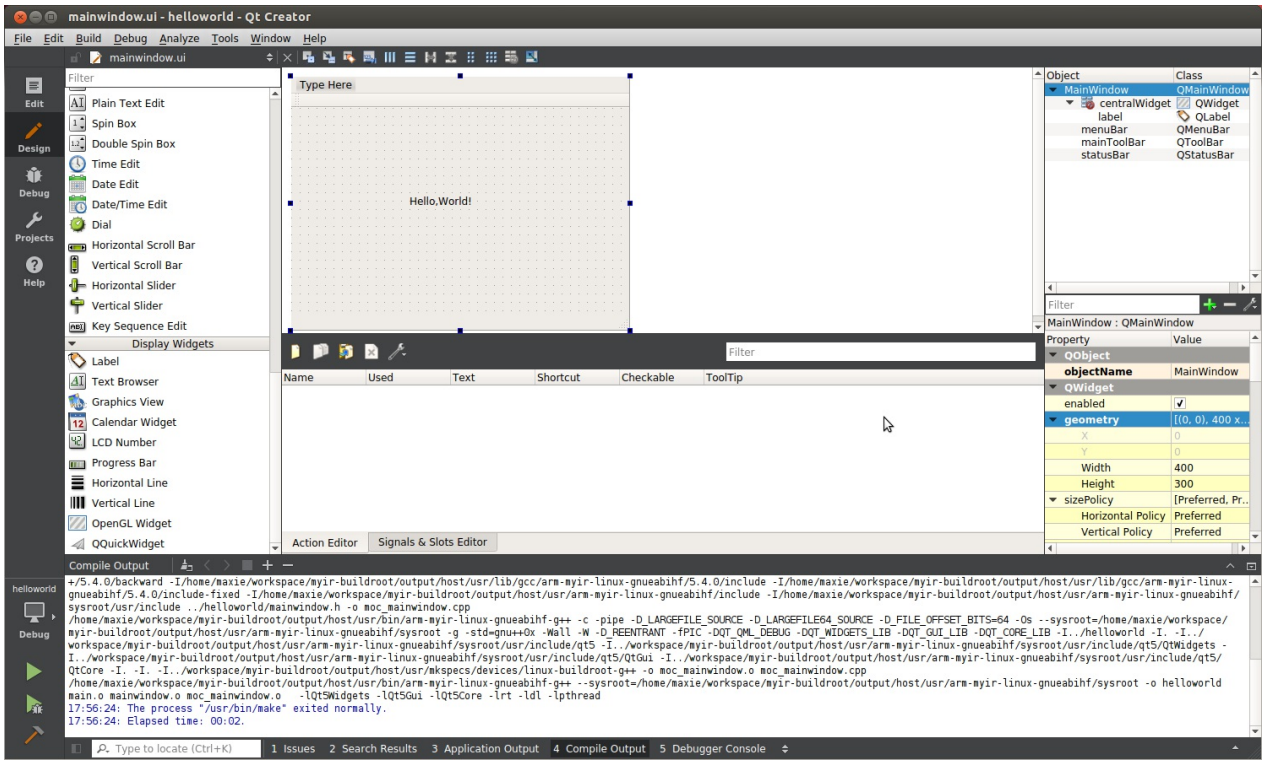


图5-3-3 编译项目

QtCreator 构建 helloworld 项目后,编译好的二进制文件存放在*~/build-helloworld-myir_dev_kit-Debug/*目录下,可以使用 file 命令查看,是否编译为 ARM 架构。

```
file helloworld
helloworld: ELF 32-bit LSB executable, ARM, version 1 (GNU/Linux), dynamically linked
 (uses shared libs), for GNU/Linux 4.1.0, not stripped
```

将生成的QT应用程序的可执行文件 `helloworld` 拷贝到开发板/usr/bin目录下,并在开发板上执行,如下:

```
# helloworld  --platform linuxfb:fb=/dev/fb0
```

将会在 LCD 屏幕上看到 `Hello,World!` 的 Qt 窗口。

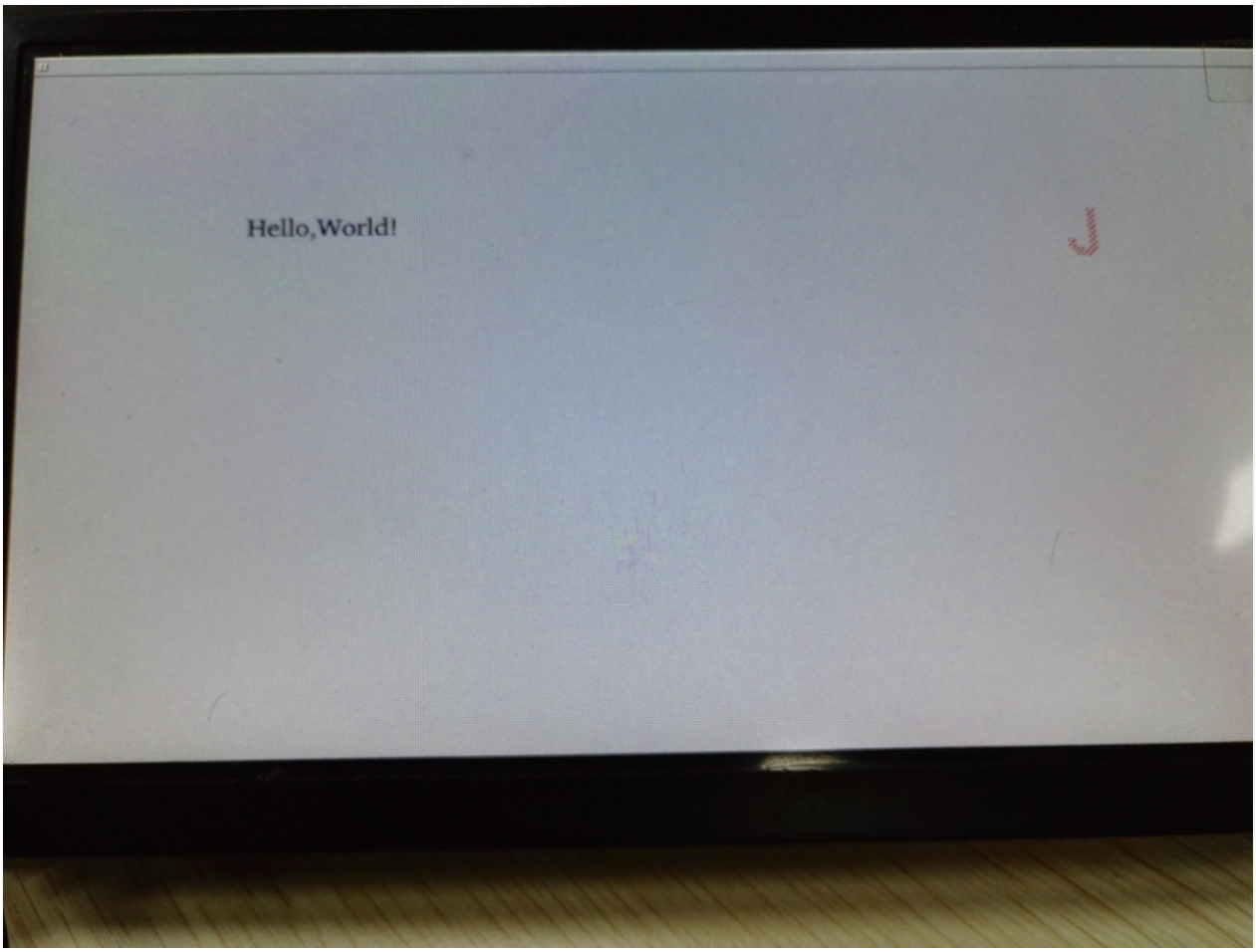

图5-3-4 QT应用程序执行

# 6. 系统更新

本节主要介绍嵌入式Linux系统构建完成之后，如何进行调试，固化和更新的操作。
MYD-AM437X系列每一块开发板出厂附带资料中都包含了对应开发板的出厂镜像文件。
用户可以直接使用这些文件进行系统更新的操作，也可以通过前面章节中介绍自行构建系统之后，再行升级。

MYD-AM437X系列开发板出厂镜像文件位于光盘02-Images目录下面，linux-images目录下为精简版文件系统镜像，MEasy_HMI-images目录下为支持MEasy_HMI演示系统的文件系统镜像。

表6-1 MYD-AM437X系列开发板预编译镜像文件列表

| 文件名 | 描述 |
| --- | --- |
| MLO | 一级引导（SPL） |
| u-boot.img | 二级引导程序 |
| myd_c437x_evm.dtb | MYD-C437X开发板LCD显示配置设备树 |
| myd_c437x_evm_hdmi.dtb | MYD-C437X开发板HDMI显示配置设备树 |
| myd_c437x_idk.dtb | MYD-C437X-PRU开发板PRU Ethernet配置设备树 |
| myd_c437x_idk_lcd.dtb | MYD-C437X-PRU开发板LCD配置设备树 |
| zImage | 内核镜像 |
| uEnv.txt | 默认环境变量 |
| uEnv_ramdisk.txt | 使用RAM DISK文件系统时的环境变量，实际使用时用该文件替换uEnv.txt放置到TF卡根目录 |
| ramdisk.gz | ramdisk文件系统压缩镜像 |
| rootfs.ubi | ubifs文件系统镜像 |
| rootfs.ext4 | ext4文件系统镜像 |
| rootfs.tar.gz | 文件系统目录压缩包 |
| sdcard.img | TF/SD/EMMC 镜像 |
| matrix-rootfs.tar.gz | TI官方文件系统目录压缩包 |

MYD-AM437X系列开发板目前提供了多种系统引导加载方案。

- 方案1: TF卡启动(EXT4文件系统)
- 方案2: TF卡启动(Ramdisk文件系统)
- 方案3: EMMC启动（EXT4文件系统)
- 方案4: NFS ROOT文件系统（主要用于文件系统调试）

其中方案3为出厂默认方案，下面分别对这几种方式的实现机制和烧写方法进行说明。

**U-Boot**环境变量说明

使用TF卡或EMMC启动的时候，经常会用到uEnv.txt文件对U-Boot进行环境变量的设置。通过修改uEnv.txt下面的fdtfile参数，可以在启动时加载不同的设备树文件，从而适配不同的硬件配置。
例如MYD-C437X开发板，可以通过uEnv.txt设置不同的fdtfile来实现。

```
fdtfile=myd_c437x_evm.dtb          # 使用LCD
#fdtfile=myd_c437x_evm_hdmi.dtb    # 使用HDMI
```

## 6.1 TF卡启动(EXT4文件系统)

Buildroot编译完成之后生成sdcard.img，它是整个TF卡的磁盘镜像文件，其中包含两个分区，分区一为FAT格式，包含了MLO, u-boot.img, zImage和设备树，uEnv.txt等文件;分区二为EXT4格式，为该启动模式下的根文件系统。

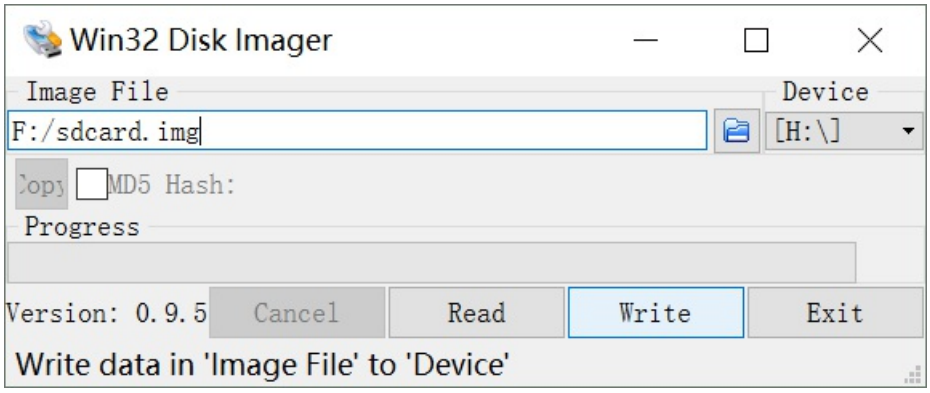


图6-1-1 使用win32diskimager烧写sdcard.img

- 把TF卡插入USB读卡器，然后将读卡器跟windows主机连接。
- 使用win32diskimager.exe烧写02-Image\linux-images\sdcard.img到TF卡上，如图6-1所示。
- 烧写完成之后，将TF卡插入开发板TF卡槽，设置为TF卡启动模式，上电即可从TF卡引导启动，并加载位于TF卡上的EXT4根文件系统。

注意：烧写过程会清除TF/SD卡原有内容，请注意备份。

## 6.2 TF卡启动(Ramdisk文件系统)

Buildroot编译完成之后生成ramdisk.gz，它是压缩后的ramdisk根文件系统镜像。

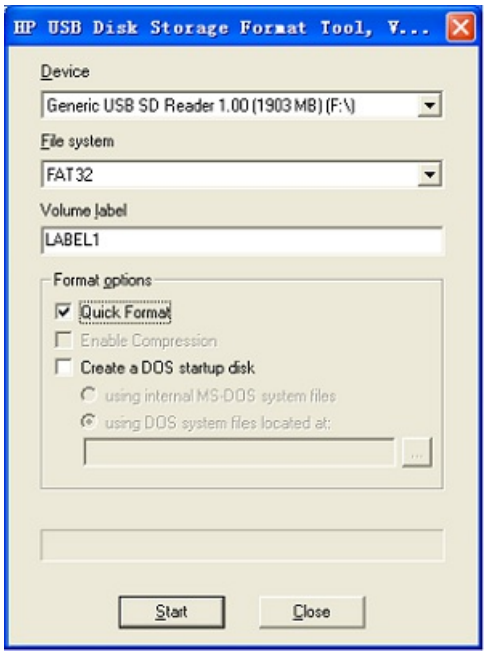


图6-2-1 格式化TF卡

(请使用光盘目录03-Tools目录下的HP USB Disk Storage Format Tool 2.0.6工具来格式化TF卡。)

- 把MMC/SD卡插入USB读卡器，然后将读卡器跟电脑连接
- 格式化TF卡，如图6-2所示。

注意：HP USB Disk Storage Format Tool会将清除TF卡的分区。若需要保留分区，请使用电脑系统自带的格式软件。

- 将02-Image\linux-images目录下所有文件拷贝到TF卡上，并将uEnv_ramdisk.txt重命名为uEnv.txt，替换原来的uEnv.xt。
- 将TF卡插入到开发板上的TF卡插槽，设置对应开发板的启动模式为TF卡启动，上电重启开发板即可从TF卡启动并加载ramdisk.gz作为根文件系统。

## 6.3 EMMC启动（EXT4文件系统)

Buildroot编译完成之后生成的rootfs.tar.gz同样可以烧写到EMMC上，用于EMMC启动。

- 首先, 使用TF卡启动，加载ramdisk文件系统镜像。
- 系统启动完成之后，进入linux控制台运行updatesys.sh将TF卡上的rootfs.tar.gz写入到EMMC,如下。
- 烧写完成之后，拔下TF卡，设置开发板为EMMC启动模式，上电即可从EMMC引导启动，并加载位于EMMC上的EXT4文件系统。

下面以MYD-C437X开发板为例说明 `updatesys.sh` 的执行过程，MYD-C437X-PRU开发板的执行过程与MYD-C437X类似。

```
#cd /
# ./updatesys.sh
All data on eMMC now will be destroyed! Continue? [y/n]
y           //此处为用户输入，确认是否擦除EMMC所有数据
[  138.794247] FAT-fs (mmcblk0p1): Volume was not properly unmounted. Some data may be
                corrupt. Please run fsck.
1024+0 records in
1024+0 records out
DISK SIZE - 3867148288 bytes
mkfs.fat 4.0 (2016-05-06)
mkfs.fat: warning - lowercase labels might not work properly with DOS or Windows
mkfs.fat 4.0 (2016-05-06)
mkfs.fat: warning - lowercase labels might not work properly with DOS or Windows
umount: /dev/mmcblk1p2: not mounted
mke2fs 1.43.3 (04-Sep-2016)
/dev/mmcblk1p2 contains a ext4 file system labelled 'rootfs'
        last mounted on /root/rootfs on Mon Jul  3 16:48:15 2017
Proceed anyway? (y,n) y //此处为用户输入，确认下面操作是否继续
Discarding device blocks: done
Creating filesystem with 261615 4k blocks and 65408 inodes
Filesystem UUID: ec25c446-3589-4f51-a6ff-d092053157e5
Superblock backups stored on blocks:
        32768, 98304, 163840, 229376

Allocating group tables: done
Writing inode tables: done
Creating journal (4096 blocks): done
Writing superblocks and filesystem accounting information: done

===> Update loader to emmc...
===> Updating kernel and devicetree to emmc...
===> Update uEnv to emmc...
===> Updating filesystem to emmc...


 Update system completed, The board can be booted from eMMC now
```

用户如果需要修改root分区的大小也可以通过修改 `updatesys.sh` 脚本来实现，其中emmc_partition()函数为eMMC的分区功能。

```
emmc_partition()
{
    #
    # Format the eMMC, the partition table were be deleted
    #
    umount $EMMC_DRIVE"p1" > /dev/null 2>&1
    umount $EMMC_DRIVE"p2" > /dev/null 2>&1
    umount $EMMC_DRIVE"p3" > /dev/null 2>&1

    dd if=/dev/zero of=$EMMC_DRIVE bs=1024 count=1024
    if [ $? -ne 0 ]; then
        echo "===> Format emmc failed"
        exit 1
    fi

    SIZE=`fdisk -l $EMMC_DRIVE | grep Disk | awk '{print $5}'`

    echo DISK SIZE - $SIZE bytes

    CYLINDERS=475
    {
    echo ,395352,0x0C,*
    echo ,2092920,,-
    echo ,,,-
    } | sfdisk -u S $EMMC_DRIVE >/dev/null 2>&1

    if [ $? -ne 0 ]; then
        echo "===> eMMC partition failed"
        exit 1
    fi

    umount $EMMC_DRIVE"p1" > /dev/null 2>&1
    sleep 1
    mkfs.fat -F 32 -n "boot" "$EMMC_DRIVE"p1
    if [ $? -ne 0 ]; then
        echo "===> Creating boot partition failed"
        exit 1
    fi

    umount $EMMC_DRIVE"p3" > /dev/null 2>&1
    sleep 1
    mkfs.fat -F 32 -n "extented" "$EMMC_DRIVE"p3
    if [ $? -ne 0 ]; then
        echo "===> Create extended partition failed"
        exit 1
    fi

    umount $EMMC_DRIVE"p2" >> /dev/null
    sleep 1
    mkfs.ext4 -L "rootfs" "$EMMC_DRIVE"p2
    if [ $? -ne 0 ]; then
        echo "===> Creating rootfs partition failed"
        exit 1
    fi
```

```
    mkdir $EMMC_BOOT_MP
    mount $EMMC_DRIVE"p1" $EMMC_BOOT_MP
    mkdir $EMMC_ROOTFS_MP
    mount -t ext4 $EMMC_DRIVE"p2" $EMMC_ROOTFS_MP
}
```

分区主要是使用 `sfdisk` 命令来实现，格式：起始、大小、ID，默认分区信息如下所示，用户可根据自己的需求进行修改

```
{
echo ,395352,0x0C,*   //分区P1，boot分区
echo ,2092920,,-      //分区P2，root分区
echo ,,,-             //分区P3，extended分区
} | sfdisk -u S $EMMC_DRIVE >/dev/null 2>&1
```

分区完成后在终端使用 `fdisk -l` 即可看到eMMC的分区信息

```
Device         Boot   Start     End Sectors  Size Id Type
/dev/mmcblk0p1 *       2048  397399  395352  193M  c W95 FAT32 (LBA)
/dev/mmcblk0p2       399360 2492279 2092920 1022M 83 Linux
/dev/mmcblk0p3      2492416 7553023 5060608  2.4G 83 Linux
```

## 6.4 NFS ROOT启动(挂载NFS ROOT文件系统)

Buildroot编译完成之后生成的rootfs.tar.gz，可以解压之后放到NFS服务器上作为NFS ROOT文件系统进行加载，便于文件系统的调试。
使用NFS ROOT方式启动，首先需要配置TFTP和NFS服务，下面以ubuntu系统为例加以说明。

- 安装TFTP服务端:

```
$ sudo apt-get install tftp-hpa tftpd-hpa
```

- 配置TFTP服务:

创建TFTP服务器工作目录,并打开TFTP服务配置文件,如下:

```
$ mkdir -p <WORKDIR>/tftpboot
$ chmod 777 <WORKDIR>/tftpboot
$ sudo vi /etc/default/tftpd-hpa
```

修改或添加以下字段：

```
TFTP_DIRECTORY="<WORKDIR>/tftpboot"
TFTP_OPTIONS="-l -c -s"
```

重启TFTP服务:

```
$ sudo service tftpd-hpa restart
```

将出厂镜像或者自行编译的MLO, u-boot.img, zImage和设备树文件以及ramdisk.gz等拷贝到TFTP服务器的 `<WORKDIR>/tftpboot` 目录，在U-Boot命令行，即可以使用tftpboot命令加载TFTP服务器上 `<WORKDIR>/tftpboot` 目录下的文件,例如：

```
>#  help tftpboot
tftpboot - boot image via network using TFTP protocol

Usage:
tftpboot [loadAddress] [[hostIPaddr:]bootfilename]
>#  tftpboot ${loadaddr} 192.168.1.111:zImage
```

- 安装NFS服务:

NFS即网络文件系统，允许主机直接通过网络实现文件系统挂载，还可以在Linux系统启动的时候将NFS服务器上的目录挂载为开发板的根文件系统。下面以ubuntu文件系统为例，介绍NFS服务的安装和配置。

```
$ sudo apt-get install nfs-kernel-server
```

- 配置NFS服务:

编辑/etc/exports文件，添加NFS服务导出的工作目录

```
$ sudo vi /etc/exports
```

添加NFS ROOT根文件系统目录，下面以/home/myir/rootfs为例，将其添加到/etc/exports文件中, 如下：

```
/home/myir/rootfs *(rw,nohide,insecure,no_subtree_check,async,no_root_squash)
```

重启NFS服务

```
$ cd /home/myir/rootfs
$ sudo tar zxvf  <WORKDIR>/images/rootfs.tar.gz
$ sudo service nfs-kernel-server restart
```

在NFS服务器本机上测试NFS服务：

```
$ sudo mount -t nfs 127.0.0.1:/home/myir/rootfs /mnt
```

若本机NFS挂载成功，接下来将开发板和NFS服务器接入到同一网络，在开发板上挂载NFS。例如，NFS服务器IP为192.168.1.111.
在U-boot中设置开发板IP为192.168.1.112, 如下:

```
>#  setenv ipaddr 192.168.1.112
```

在U-boot中使用ping命令测试开发板与NFS服务器是否连通:

```
# ping 192.168.1.111
```

U-Boot控制台下执行netboot命令，挂载NFS ROOT文件系统，如下:

```
>#  setenv serverip 192.168.1.111
>#  setenv ipaddr 192.168.1.112
>#  ping 192.168.1.111                        -- 测试网络是否连通
>#  setenv rootpath /home/myir/rootfs
>#  run findfdt
>#  echo $fdtfile                             -- 查看fdtfile名称是否正确
>#  run netboot
```

## 6.5 Matrix-rootfs使用

目前只支持EMMC启动和NFS启动，仅限AM4378和AM4379 cpu使用。

**EMMC**启动：

更新EMMC上的文件系统

- 首先, 采用方案2: TF卡启动(Ramdisk文件系统)，确认TF卡中有matrix-rootfs.tar.gz压缩包，启动开发板。
- 系统启动完成之后，进入linux控制台，修改updatesys.sh中的FILE_FILESYSTEM的值，修改过程如下：

```
# cd /
# vi updatesys.sh


# This script to update myd_c437x_evm or Rico Board system
# This script will respectively update the u-boot, device tree, zImage to
# QSPI.U_BOOT, QSPI.U-BOOT-DEVICETREE, QSPI.KERNEL, and update the filesystem
# to emmc
#
# Author: MYiR
# Email: support@myirtech.com
# Date: 2015.1.21
#       Initial Version
# Date: 2017.05.24
#       update to kernel4.1.18 and u-boot201605

#!/bin/sh

# The path sdcard mounted
SD_MOUNT_POINT="/media/mmcblk1p1"
# The rootfs partition would be mounted on current 'rootfs' directory
EMMC_BOOT_MP="boot"
EMMC_ROOTFS_MP="rootfs"

FILE_MLO="MLO"
if [ "$1" = "loader2qspi" ]; then
        FILE_UBOOT="u-boot.bin"
else
        FILE_UBOOT="u-boot.img"
fi
FILE_ZIMAGE="zImage"
FILE_DEVICETREE="myd_c437x_evm.dtb"
FILE_FILESYSTEM="matrix-rootfs.tar.gz"
FILE_RAMDISK="ramdisk.gz"
FILE_UBOOTENV="u-boot-env.bin"
FILE_UENV="uEnv"
FILE_DEFAULT_UENV="uEnv/uEnv.txt"
- updatesys.sh 1/286 0%
```

将其中的FILE_FILESYSTEM的值修改为matrix-rootfs.tar.gz，保存并退出，然后执行updatasys.sh这个脚本，将更新EMMC上的文件系统为matrix-rootfs。

- 烧写完成之后，拔下TF卡，设置开发板为EMMC启动模式，上电即可从EMMC引导启动，并加载位于EMMC上的matrix-rootfs文件系统。

**NFS**启动：

主要流程参照前面的方案4: NFS ROOT文件系统（主要用于文件系统调试），这里要将matrix-rootfs.tar.gz解压到NFS ROOT根文件系统目录。操作如下：

```
$ cd /home/myir/rootfs
$ sudo tar zxvf  <WORKDIR>/images/matrix-rootfs.tar.gz
$ sudo service nfs-kernel-server restart
```

操作完成后按照NFS启动的方式启动开发板，即可进入TI matrix-rootfs演示系统。

# 附录一 联系方式

销售联系方式

- 网址： www.myir-tech.com
- 邮箱：sales.cn@myirtech.com

深圳总部

- 负责区域：广东 / 四川 / 重庆 / 湖南 / 广西 / 云南 / 贵州 / 海南 / 香港 / 澳门
- 电话：0755-25622735 0755-22929657
- 传真：0755-25532724
- 邮编：518020
- 地址：深圳市龙岗区坂田街道发达路云里智能园2栋6楼04室

上海办事处

- 负责区域：上海 / 湖北 / 江苏 / 浙江 / 安徽 / 福建 / 江西
- 电话：021-60317628 15901764611
- 传真：021-60317630
- 邮编：200062
- 地址：上海市普陀区中江路106号北岸长风I座1402

北京办事处

- 负责区域：北京 / 天津 / 陕西 / 辽宁 / 山东 / 河南 / 河北 / 黑龙江 / 吉林 / 山西 / 甘肃 / 内蒙古 / 宁夏
- 电话：010-84675491 13269791724
- 传真：010-84675491
- 邮编：102218
- 地址：北京市昌平区东小口镇中滩村润枫欣尚2号楼1009

技术支持联系方式

- 电话：027-59621648
- 邮箱：support.cn@myirtech.com

如果您通过邮件获取帮助时，请使用以下格式书写邮件标题：

```
[公司名称/个人--开发板型号]问题概述
```

这样可以使我们更快速跟进您的问题，以便相应开发组可以处理您的问题。

# 附录二 售后服务与技术支持

凡是通过米尔科技直接购买或经米尔科技授权的正规代理商处购买的米尔科技全系列产品，均可享受以下权益：

1、6个月免费保修服务周期

2、终身免费技术支持服务

3、终身维修服务

4、免费享有所购买产品配套的软件升级服务

5、免费享有所购买产品配套的软件源代码，以及米尔科技开发的部分软件源代码

6、可直接从米尔科技购买主要芯片样品，简单、方便、快速；免去从代理商处购买时，漫长的等待周期

7、自购买之日起，即成为米尔科技永久客户，享有再次购买米尔科技任何一款软硬件产品的优惠政策

8、OEM/ODM服务

如有以下情况之一，则不享有免费保修服务：

1、超过免费保修服务周期

2、无产品序列号或无产品有效购买单据

3、进液、受潮、发霉或腐蚀

4、受撞击、挤压、摔落、刮伤等非产品本身质量问题引起的故障和损坏

5、擅自改造硬件、错误上电、错误操作造成的故障和损坏

6、由不可抗拒自然因素引起的故障和损坏

产品返修：用户在使用过程中由于产品故障、损坏或其他异常现象，在寄回维修之前，请先致电米尔科技客服部，与工程师进行沟通以确认问题，避免故障判断错误造成不必要的运费损失及周期的耽误。

维修周期：收到返修产品后，我们将即日安排工程师进行检测，我们将在最短的时间内维修或更换并寄回。一般的故障维修周期为3个工作日（自我司收到物品之日起，不计运输过程时间），由于特殊故障导致无法短期内维修的产品，我们会与用户另行沟通并确认维修周期。

维修费用：在免费保修期内的产品，由于产品质量问题引起的故障，不收任何维修费用；不属于免费保修范围内的故障或损坏，在检测确认问题后，我们将与客户沟通并确认维修费用，我们仅收取元器件材料费，不收取维修服务费；超过保修期限的产品，根据实际损坏的程度来确定收取的元器件材料费和维修服务费。

运输费用：产品正常保修时，用户寄回的运费由用户承担，维修后寄回给用户的费用由我司承担。非正常保修产品来回运费均由用户承担。