MYD-AM335X Linux 4.1.18 开发手册

MYiR™ Make Your Idea Real

MYD-AM335X

MYD-AM335X-Y

MYD-AM335X-J

目录

MYD-AM335X-Linux-4.1.18开发手册

前言

本小节概述文档所覆盖的范围，包含的内容，适合的读者，文档版本历史记录，适用的硬件版本。

本文主要讲述如何在MYD-AM335X系列开发板上安装运行Linux系统和嵌入式Linux应用程序及驱动的开发流程。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。

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	2017.04.01
V1.1	1. 增加Buildroot中配置内核和U-boot代码仓库的说明 2. 增加MEasy HMI演示系统的说明	2018.07.01
V1.2	1. 增加对使用EMMC作为存储设备的核心板的支持	2018.07.30
V1.3	1. 增加米尔外设模块使用章节 2. MYD-AM335X插针板网卡1限制为最高100BASE-T	2018.08.28
V1.4	1. 内核中增加对以太网PHY芯片YT8511的支持 2. Buildroot出厂镜像兼容电阻和电容触摸 3. 提供github代码下载链接，请参照https://github.com/MYiR-Dev/myir-ti-buildroot/releases/tag/MYD-AM335X_V20R5_20211118	2021.11.18

硬件版本

MYD-AM335X	MYD-AM335X-Y	MYD-AM335X-J
插针板	邮票孔	金手指

> 注意： 本文档适用以上三个版本的硬件，请用户自行根据相应板型选择对应的操作，本文用**MYD-AM335X**系列表示上面三块板型。

1. 软件资源介绍

本小节以表格的形式描述MYD-AM335X系列开发板的软件资源。

表 1-1软件资源列表

类别	名称	描述	源码	MYD-AM335X	MYD-AM335XY
Bootloader	U-boot	引导程序，负责系统初始化和引导内核	YES	√	√
内核	Linux 4.1.18	专为MYD-AM335X系列的硬件制定的Linux内核	YES	√	√
驱动	LCD	LCD屏驱动，可支持4.3寸、7寸液晶屏	YES	√	√
驱动	Touch Panel	电阻触摸屏驱动	YES	√	√
驱动	Touch Panel	电容触摸屏驱动	YES	√	√
驱动	RTC	内置RTC时钟驱动	YES	√	√
驱动	UART	串口驱动	YES	√	√
驱动	SDIO WiFi	WiFi驱动	YES	×	√
驱动	RS485	RS485驱动，提供源码	YES	√	√
驱动	CAN Bus	CAN驱动	YES	√	√

驱动	Ethernet	以太网驱动	YES	√	√
驱动	MMC/SD	MMC/SD卡驱动	YES	√	√
驱动	NAND Flash	NAND Flash驱动	YES	√	√
驱动	Audio	音频驱动	YES	√	√
驱动	GPIO-LED	GPIO-LED驱动	YES	√	√
驱动	I2C	I2C驱动	YES	√	√
驱动	HDMI	HDMI驱动，提供源码	YES	√	×
驱动	PMU	电源管理驱动	YES	√	√
驱动	USB Host	USB Host驱动	YES	√	√
驱动	USB Device	USB Device驱动（Gadget）	YES	√	√
文件系统	Rootfs	基于buildroot定制的文件系统	二进制	√	√
文件系统	Rootfs-qt	基于buildroot定制的Qt文件系统	二进制	√	√
文件系统	UBI	提供镜像文件	YES	√	√
文件系统	Ramdisk.gz	Ramdisk文件系统	YES	√	√
应用程序	LCD	LCD测试程序	YES	√	√
应用程序	RTC	RTC时钟测试实验	YES	√	√

应用程序	RS485	RS485测试程序	YES	√	√
应用程序	CAN	CAN 测试程序	YES	√	√
应用程序	Ethernet	网络测试程序	YES	√	√
应用程序	NAND Flash	NAND Flash测试实验	YES	√	√
应用程序	GPIO-LED	GPIO-LED测试实验	YES	√	√
应用程序	Audio	Audio测试程序	YES	√	√
应用程序	Qt	Qt环境验证Demo程序	YES	√	√

2. 部署开发环境

本小节主要介绍相关工具和环境，工具链的设置，开发环境验证。

pc开发环境：Ubuntu12.04/14.04/16.04 64位桌面版

交叉编译器：gcc5.3(Linaro GCC 2016.02)

硬件调试环境搭建

- MYD-AM335X硬件调试环境搭建：

把调试串口J12连到PC上并将PC串口的波特率设为115200，数据位为8，停止位为1，无奇偶校验。 具体如图2-1：

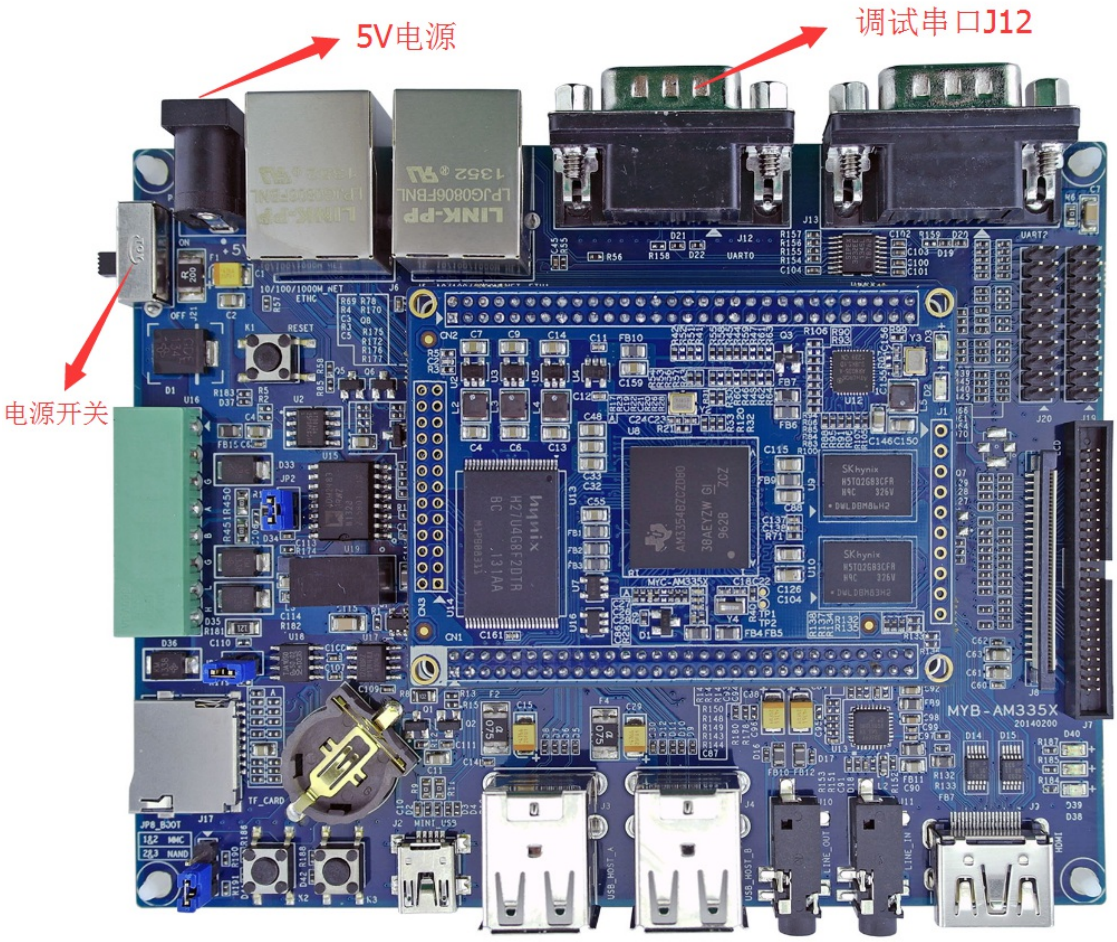

图2-1 MYD-AM335X硬件调试接口

MYD-AM335X-Y硬件调试环境搭建：

把调试串口J10连到PC上并将PC串口的波特率设为115200，数据位为8，停止位为1，无奇偶校验。 具体如下2-2：

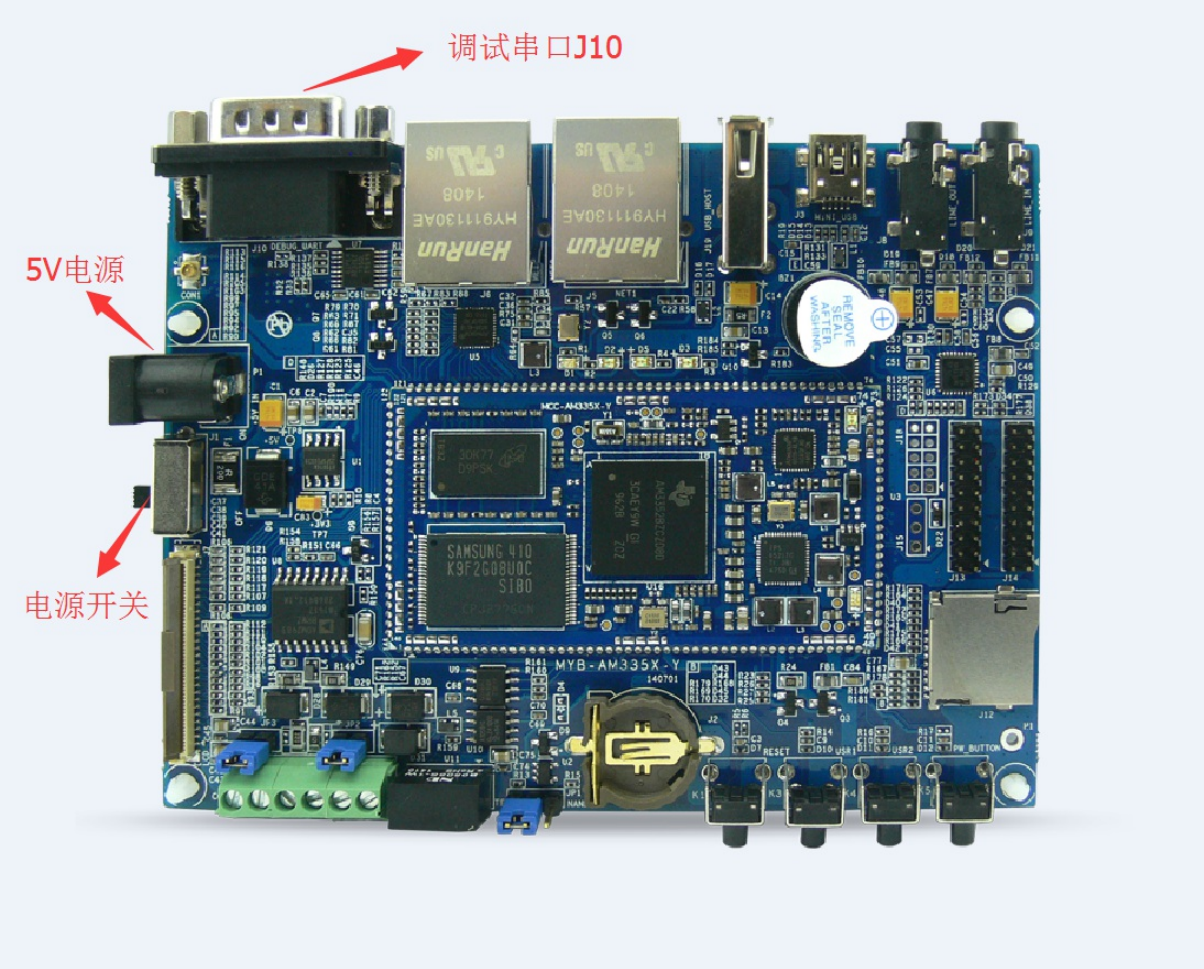

图2-2 MYD-AM335X-Y硬件调试接口

MYD-AM335X-J硬件调试环境搭建：

把调试串口J3连到PC上并将PC串口的波特率设为115200，数据位为8，停止位为1，无奇偶校验。 具体如下图2-3：

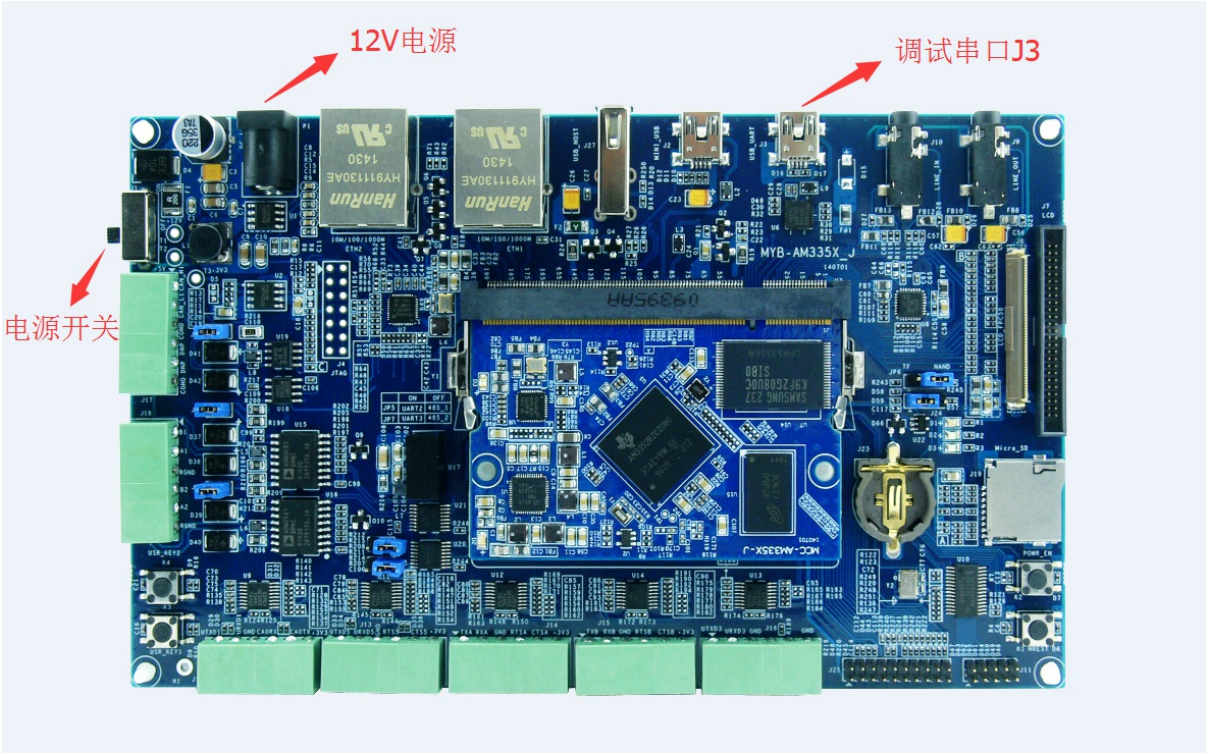

图2-3 MYD-AM335X-J硬件调试接口

建立工作目录:

创建工作目录, 并拷贝MYD AM335X系列开发板出厂附带资料04-Linux_Source目录到Linux开发主机中，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/  Patches/ ToolChain/
```

2.1 安装工具

安装其它开发工具依赖包　　下面安装一些linux开发常用的工具，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。

Ubuntu 12.04 64位桌面版

```
$ sudo apt-get install build-essential git-core libncurses5-dev
$ sudo apt-get install flex bison texinfo zip unzip zlib1g-dev g
ettext
$ 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 64位桌面版

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

Ubuntu 16.04 64位桌面版

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

```
$ sudo apt-get install g++ xz-utils
$ sudo apt-get install subversion
```

需要注意的是在64位的系统上，还需要安装32位兼容的库。

```
$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

```
$ cd <WORKDIR>/Toolchain
$ tar xvf gcc-linaro-5.3-2016.02-x86_64_arm-linux-gnueabihf.tar.
xz
$ export PATH=$PATH:<WORKDIR>/Toolchain/gcc-linaro-5.3-2016.02-x
86_64_arm-linux
-gnueabihf/bin
$ export ARCH=arm
$ export CROSS_COMPILE=arm-linux-gnueabihf-
```

执行完“export”命令后输入arm按Tab键来检查是否设置成功，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，例如修改 `~/.profile` 。

```
vi ~/.profile
```

在行尾添加：

```
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
```

```
source ~/.profile
```

测试环境变量：

```
$echo $ARCH
```

```
arm
$echo $CROSS_COMPILE
arm-linux-gnueabihf-
```

交叉编译器的验证：

```
arm-linux-gnueabihf-gcc -v
$ 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，这些框架极大地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构建文件系统和编译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-AM335X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

编译U-Boot,进入Bootloader目录，解压U-boot源码：

```
$ cd <WORKDIR>/Bootloader
$ tar -jxvf myir-u-boot.tar.bz2
$ cd myir-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。

表 3-1-1 U-boot配置文件列表

开发板	编译选项
MYD-AM335X(NAND)	myd_c335x_defconfig
MYD-AM335X(EMMC)	myd_c335x_emmc_defconfig
MYD-AM335X-Y(NAND)	myd_y335x_defconfig
MYD-AM335X-Y(EMMC)	myd_y335x_emmc_defconfig
MYD-AM335X-J(NAND)	myd_j335x_defconfig
MYD-AM335X-J(EMMC)	myd_j335x_emmc_defconfig

下面以MYD-AM335X开发板为例，说明u-boot的编译过程：

```
$ make distclean
$ make myd_c335x_defconfig
$ make
```

其中第二步make中的配置选项见上表,编译完成后，在u-boot目录下会生成MLO和u-boot.img文件。MYD-AM335X-Y和MYD-AM335X-J的编译和MYD-AM335X类似。

3.2 编译Linux内核

进入Kernel目录，解压内核源码：

```
$ cd <WORKDIR>/Kernel
$ tar -jxvf myir-kernel.tar.bz2
$ 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。

表 3-2-1 Kernel配置文件列表

开发板	编译选项
MYD-AM335X	myd_c335x_defconfig
MYD-AM335X-Y	myd_y335x_defconfig
MYD-AM335X-J	myd_j335x_defconfig

如果用户想编译和安装内核模块，需要先设置 `INSTALL_MOD_PATH` 环境变量，在使用NFS调试内核的时候会比较有用。需要注意的是内核模块包含一个 `Version Magic` ，它必须与内核 `zImage` 匹配，否则内核模块会加载失败，出现类似下面的错误：

```
[ 2750.480576] ti_am335x_adc: disagrees about version of symbol
dev_warn
[ 2750.487670] ti_am335x_adc: Unknown symbol dev_warn (err -22)
[ 2750.493977] ti_am335x_adc: disagrees about version of symbol
dev_err
[ 2750.502474] ti_am335x_adc: Unknown symbol dev_err (err -22)
```

所以内核修改之后，内核模块需要和内核 `zImage` 一起编译，下面以MYD-AM335X开发板为例，说明kernel的编译过程：

```
$ export INSTALL_MOD_PATH=$HOME/export/rootfsa/
$ make distclean
$ make myd_c335x_defconfig
$ make zImage dtbs
$ make modules
$ make modules_install
```

其中第二步make中的内核配置文件选项见上表。编译完成后，会在arch/arm/boot目录下生成zImage文件,会在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，例如屏幕尺寸大小可以修改dts文件。MYD-AM335X-Y和MYD-AM335X-J的编译和MYD-AM335X类似。

上面配置中默认为7寸屏，256M NAND 256M DDR，禁用SGX功能，用户可以通过 `<WORKDIR>/Patches` 中的补丁文件生成配置的设备树文件，然后编译内核使用设备树文件即可，补丁只针对使用NAND的情况，使用EMMC的用户请参照补丁内容手动修改。以下是相关补丁的列表说明：

板型	MYD-AM335X	MYD-AM335X-Y	MYD-AM335X-J
HDMI补丁	myd_c335x_hdmi_display.diff	无扩展	无扩展
4.3寸屏补丁	myd-am335x-lcd4.3.diff	myd-am335x-y-lcd4.3.diff	myd-am335x-j-lcd4.3.diff
SGX补丁	myd-am335x-sgx.diff	myd-am335x-y-sgx.diff	myd-am335x-j-sgx.diff

如果用户希望使用QT中的QML功能，则必须在内核设备树中使能SGX。例如，MYD-AM335X-Y使能SGX, 4.3寸屏打补丁如下：

```
$ patch -p1 < <WORKDIR>/Patches/myd-am335x-y-lcd4.3.diff

$ patch -p1 < <WORKDIR>/Patches/myd-am335x-y-sgx.diff
```

编译设备树文件,并用arch/arm/boot/dts目录下的设备树文件去替换烧录的image中的设备树文件。

```
make dtbs
```

3.3 构建文件系统

本节主要介绍使用Buildroot进行文件系统的制作。

3.3.1 准备编译buildroot

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 -al <WORKDIR>/Filesystem/myir-buildroot
arch    CHANGES           configs     dl    linux            outp
ut    support
board   Config.in         COPYING     docs  Makefile         pack
age   system
boot    Config.in.legacy  DEVELOPERS  fs    Makefile.legacy  READ
ME    toolchain
```

关于 `buildroot` 的目录结构可以参照https://buildroot.org/downloads/manual/manual.html.其中和 `MYD-AM335X` 系列开发板相关的部分主要位于 `<WORKDIR>/Filesystem/myir-buildroot/board/myir/myd_c335x` , `<WORKDIR>/Filesystem/myir-buildroot/board/myir/myd_y335x` 和 `<WORKDIR>/Filesystem/myir-buildroot/board/myir/myd_j335x` 目录。

3.3.2 配置说明

MYD-AM335X系列平台默认的配置文件位于 `<WORKDIR>/Filesystem/myir-buildroot/configs/` .

Table 3-3-1 MYIR AM335x系列开发板Buildroot配置文件列表

Config File	Description

myd_c335x_defconfig	Buildroot configuration without QT5 for MYC-AM335X CPU Module with NAND Flash
myd_c335x_emmc_defconfig	Buildroot configuration without QT5 for MYC-AM335X CPU Module With EMMC
myd_c335x_qt5_defconfig	Buildroot configuration with QT5 for MYD-AM335X development board with NAND
myd_j335x_defconfig	Buildroot configuration without QT5 for MYC-AM335X-J CPU Module with NAND Flash
myd_j335x_emmc_defconfig	Buildroot configuration without QT5 for MYC-AM335X-J CPU Module with EMMC
myd_j335x_qt5_defconfig	Buildroot configuration with QT5 for MYD-AM335X-J development board with NAND Flash
myd_y335x_defconfig	Buildroot configuration without QT5 for MYC-AM335X-Y CPU Module with NAND Flash
myd_y335x_defconfig	Buildroot configuration without QT5 for MYC-AM335X-Y CPU Module with EMMC
myd_y335x_qt5_defconfig	Buildroot configuration with QT5 for MYD-AM335X-Y development board with NAND Flash

配置文件中包含bootloader， kernel，根文件系统相关的配置，用户也可以根据不同的应用场景需要编写自己的配置文件。先加载基本配置文件:

```
$ cd <WORKDIR>/Filesystem/myir-buildroot/
$ make myd_y335x_defconfig
```

再通过 `make menuconfig` 进入配置界面。然后可以跟内核配置一样，对当前的配置进行一些修改。下面以 `MYD-AM335X-Y` 平台为例，加以说明。

> 注意：使用**Buildroot**编译**u-boot**和**kernel**需要用户自行建好代码**git**仓库，然后用建好的代码**git**仓库路径替换下面**bootloader**和**Kernel**配置中的**git**路径，具体见下面的**bootloader**和**Kernel**的配置

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/` 。

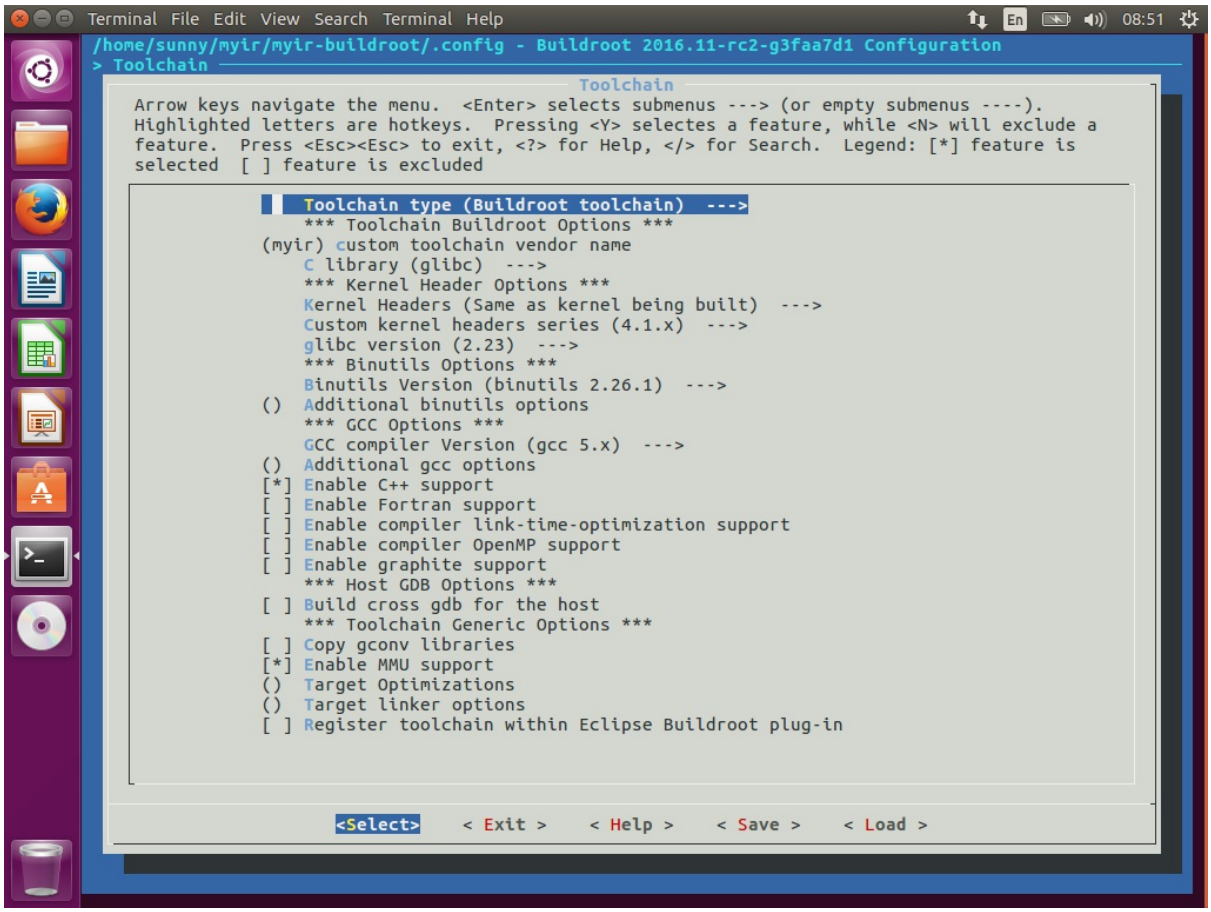

图3-3-1 配置buildroot交叉编译工具链

系统配置

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。

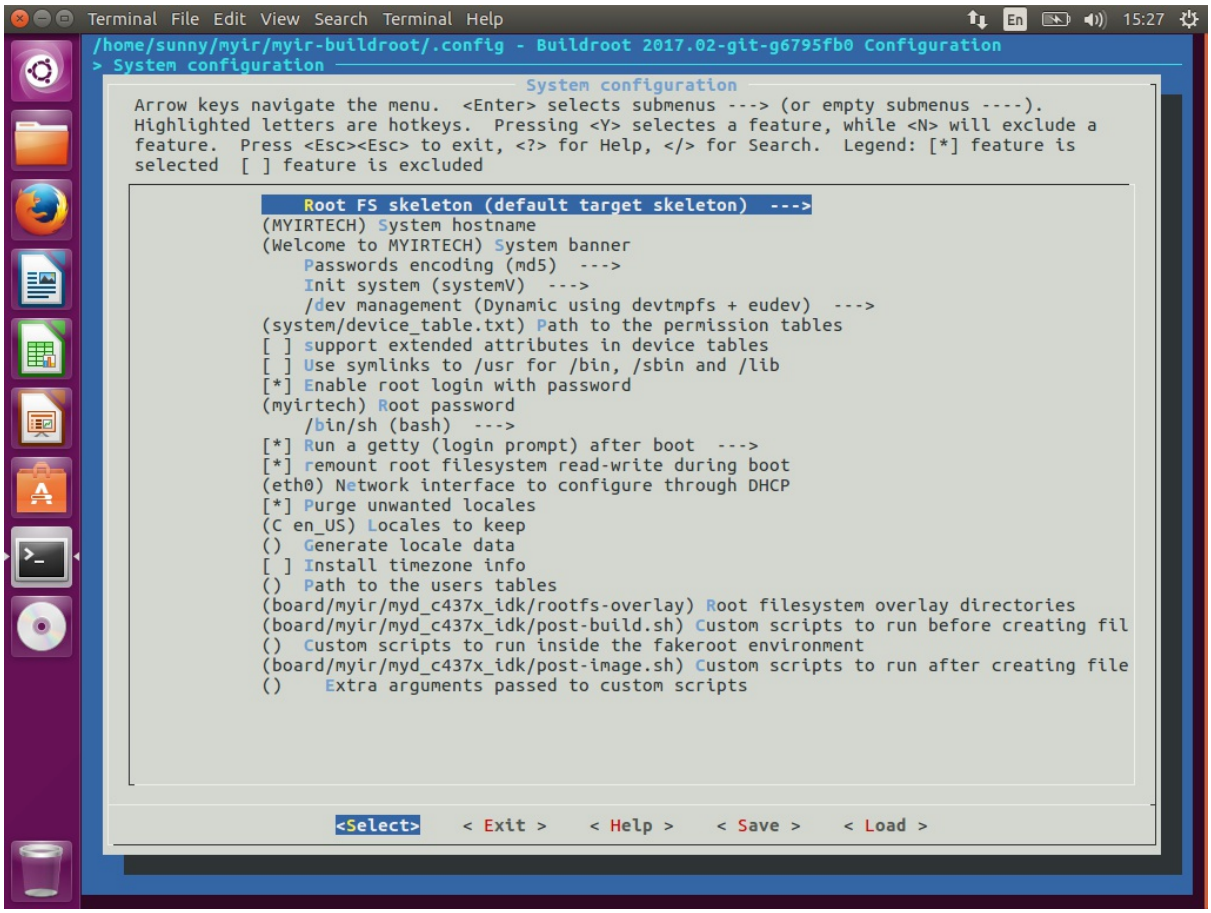

图3-3-2 系统配置

bootloader配置

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。

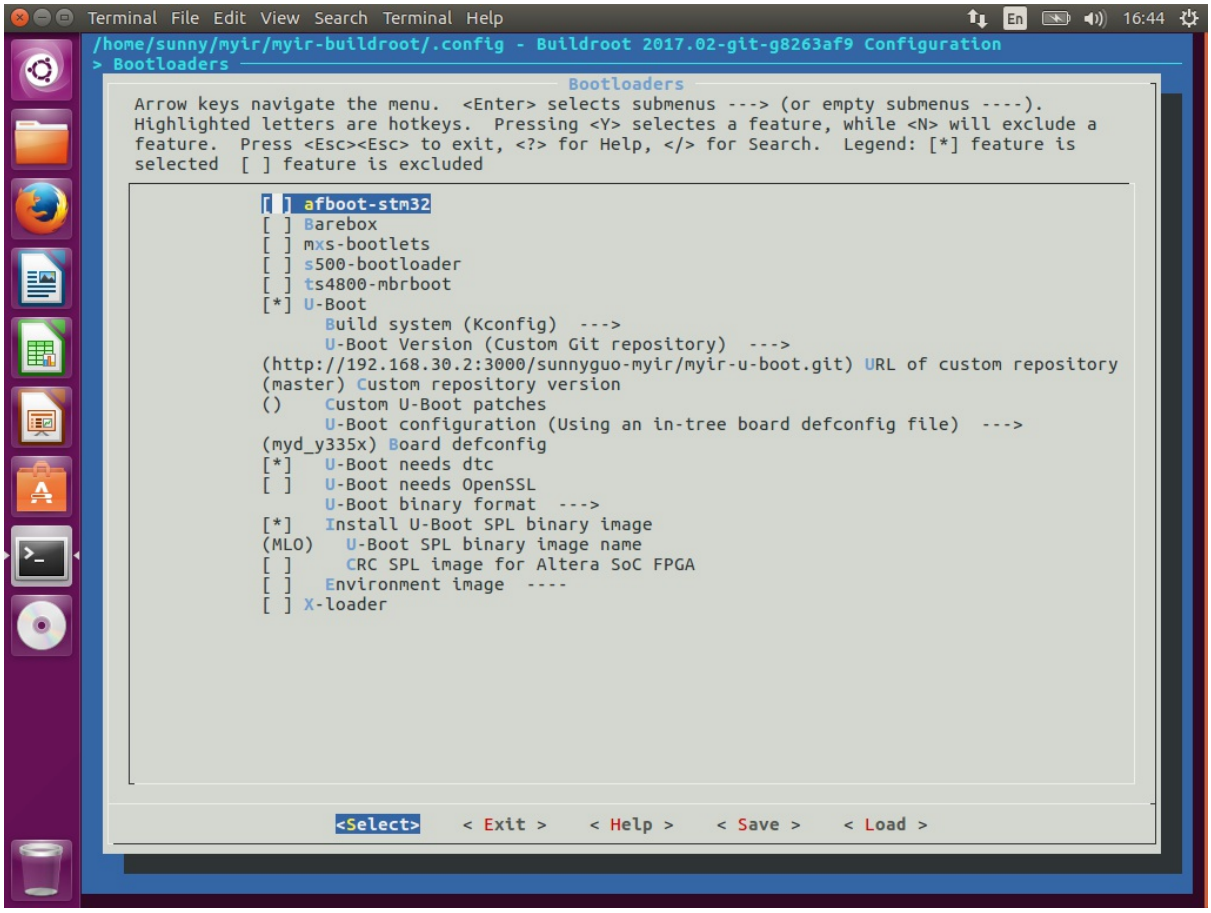

图3-3-3 bootloader配置

图中采用的是git协议从内网的git服务器获取代码。用户需要自行建好代码git仓库，然后用建好的代码git仓库路径替换这里设置的路径。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。

下面根据开发板附带资料中的代码，创建自己的git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 /Filesystem/buildroot/configs/myd_y335x_defconfig 配置文件和/Filesystem/buildroot/configs/myd_y335x_qt5_defconfig ，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"
```

用户也可以直接使用开发板附带资料中的压缩包作为Buildroot中的U-boot代码来源,将其拷贝到 `<WORKDIR>/Filesystem/buildroot/board/myir/` 目录下。然后修改位于 /Filesystem/buildroot/configs/myd_y335x_defconfig 配置文件和/Filesystem/buildroot/configs/myd_y335x_qt5_defconfig ，将下面内容改为如下所示：

```
BR2_TARGET_UBOOT_CUSTOM_TARBALL=y

# BR2_TARGET_UBOOT_CUSTOM_GIT is not set

BR2_TARGET_UBOOT_CUSTOM_TARBALL_LOCATION="file://$(TOPDIR)/board
/myir/myir-u-boot.tar.gz"
BR2_TARGET_UBOOT_VERSION="custom"
```

> 注意：修改了u-boot的代码库中代码之后，buildroot并不能自动重新获取和编译。需要手动删除 `<WORKDIR>/Filesystem/myir-buildroot/dl/uboot-master.tgz` 和 `<WORKDIR>/Filesystem/myir-buildroot/output/build/uboot-master` 目录。

Kernel配置

Kernel的配置和bootloader类似，也是配置内核的代码来源，和kernel的配置，编译，安装，如下图所示。

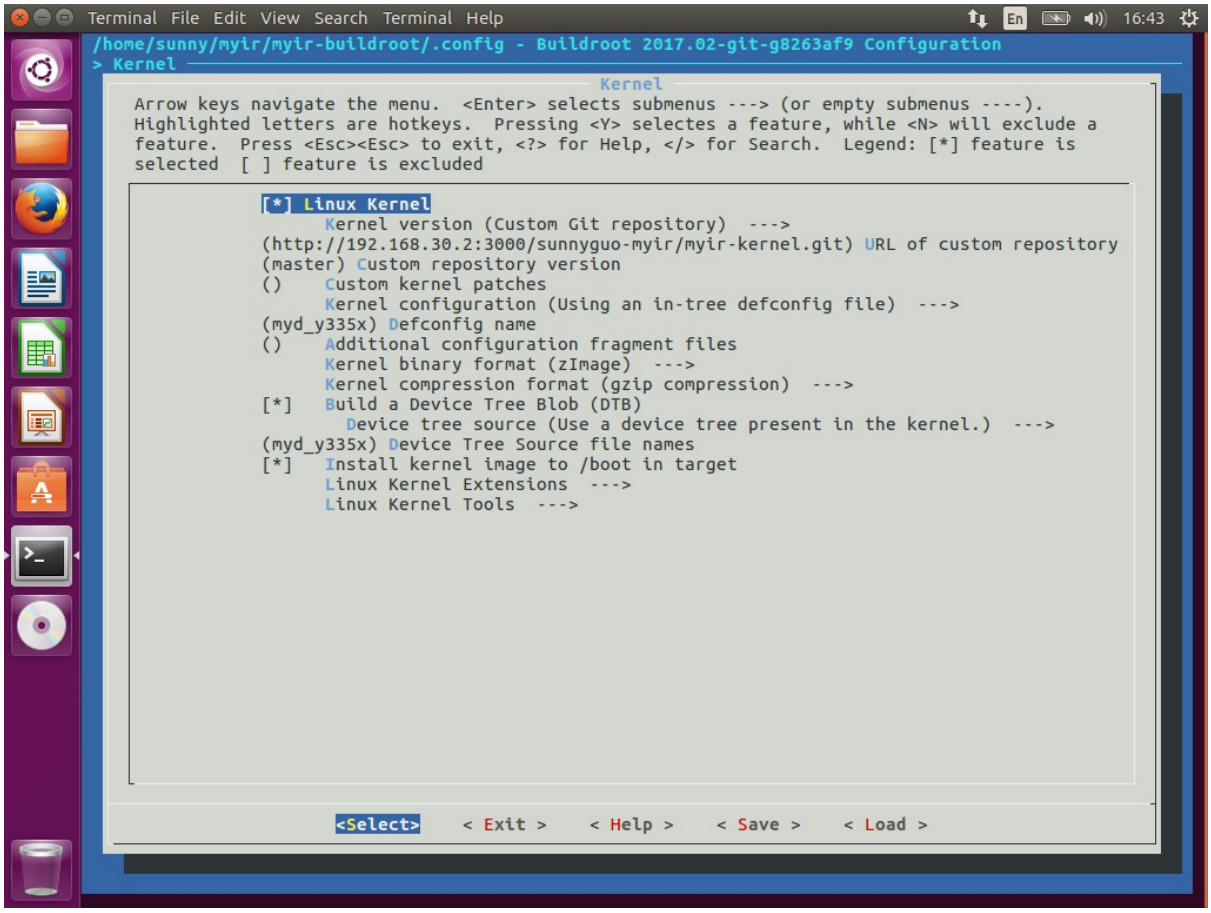

图3-3-4 Kernel配置

图中采用的是git协议从内网的git服务器获取代码。用户需要自行建好代码git仓库，然后用建好的代码git仓库路径替换这里设置的路径。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。

下面根据开发板附带资料中的代码，创建自己的git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 /Filesystem/buildroot/configs/myd_y335x_defconfig 配置文件和/Filesystem/buildroot/configs/myd_y335x_qt5_defconfig ，将下面两项的内容改为如下所示：

```
BR2_TARGET_KERNEL_CUSTOM_REPO_URL="~/myir-kernel/.git"
BR2_TARGET_KERNEL_CUSTOM_REPO_VERSION="master"
```

用户也可以直接使用开发板附带资料中的压缩包作为Buildroot中的Kernel代码来源,将其拷贝到 `<WORKDIR>/Filesystem/buildroot/board/myir/` 目录下。然后修改位于 /Filesystem/buildroot/configs/myd_y335x_defconfig 配置文件和/Filesystem/buildroot/configs/myd_y335x_qt5_defconfig ，将下面内容改为如下所示：

```
BR2_LINUX_KERNEL_CUSTOM_TARBALL=y
# BR2_LINUX_KERNEL_CUSTOM_GIT is not set
BR2_LINUX_KERNEL_CUSTOM_TARBALL_LOCATION="file://$(TOPDIR)/board
/myir/myir-kernel.tar.gz"
BR2_LINUX_KERNEL_VERSION="custom"
```

> 注意：修改了kernel的代码库中代码之后，buildroot并不能自动重新获取和编译。需要手动删除 `<WORKDIR>/Filesystem/myir-buildroot/dl/linux-master.tgz` 和 `<WORKDIR>/Filesystem/myir-buildroot/output/build/linux-master` 目录。

文件系统配置

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)。

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

```
$ export PATH=$PATH:<WORKDIR>/Filesystem/myir-buildroot/output/h
ost/usr/sbin
$ tar zxvf rootfs.tar.gz
$ 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 20
48 ubinize.cfg
```

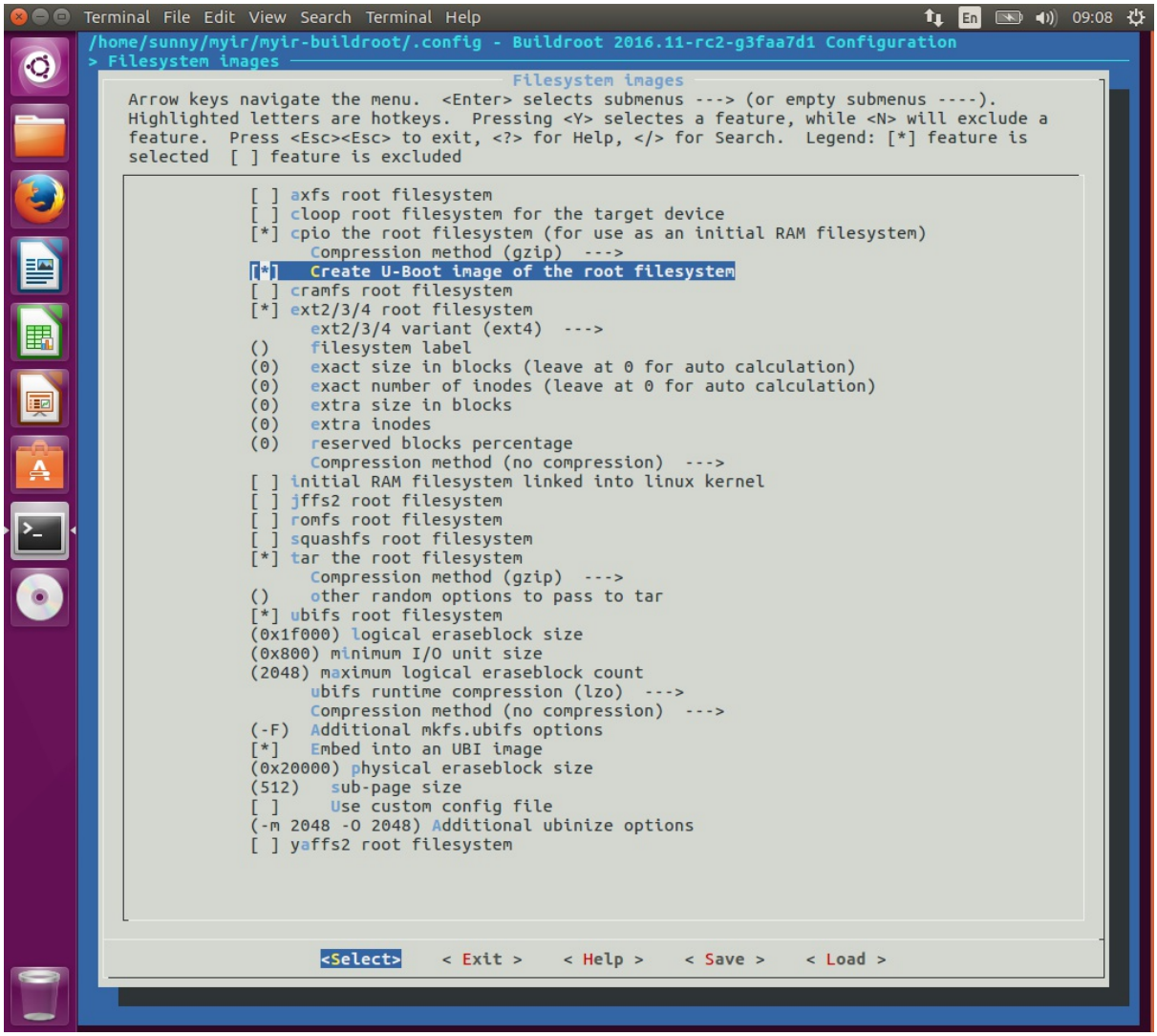

图3-3-5 文件系统配置

工具包配置

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。

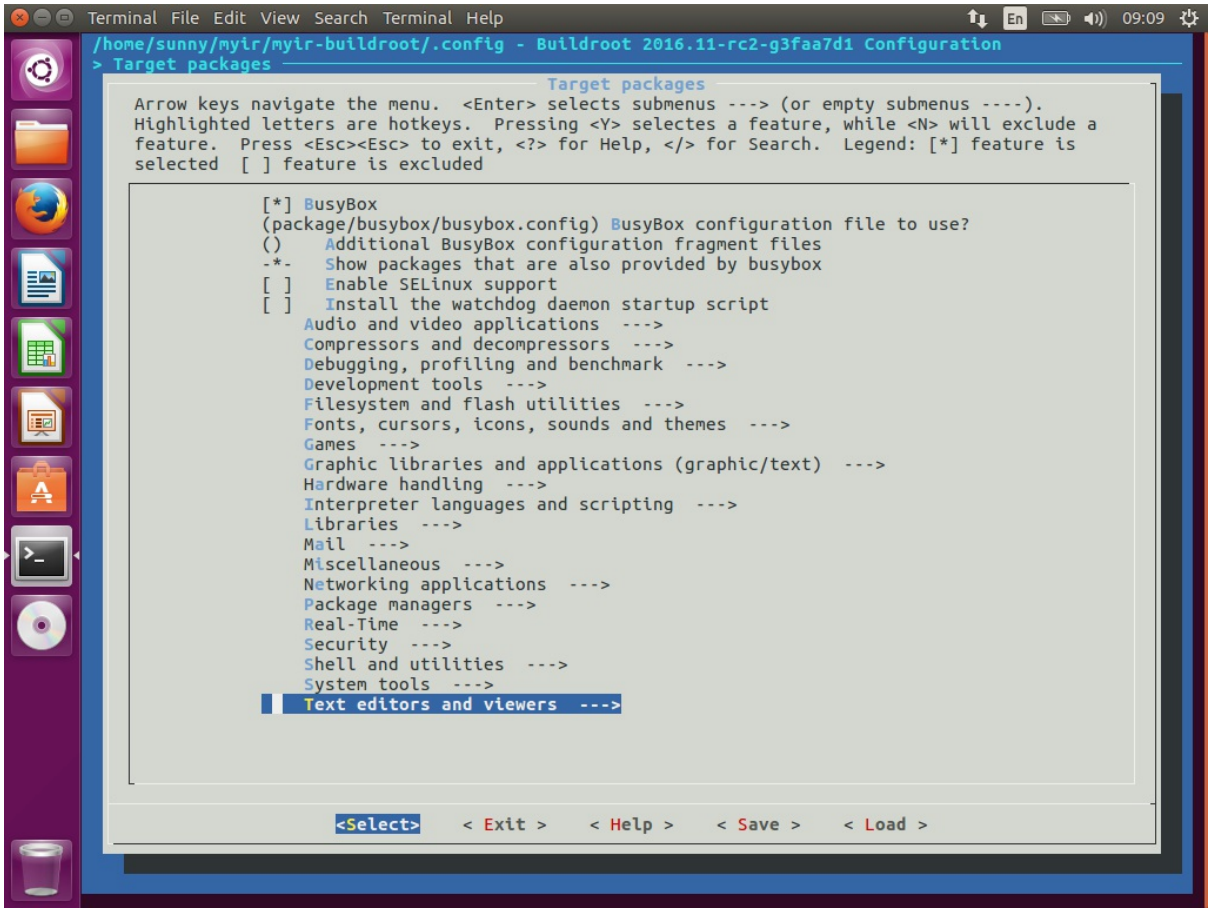

图3-3-6 工具包配置

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
```

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  MLO_usbmsc          rootfs.cpio        rootfs.tar
 u-boot_emmc.img  u-boot_usbmsc.img    uEnv.txt
MLO        myd_y335x.dtb       rootfs.cpio.gz     rootfs.tar.gz
```

```
 u-boot.img       uEnv_mmc.txt         uEnv_usbmsc_ramdisk.txt
MLO_emmc   myd_y335x_emmc.dtb  rootfs.cpio.uboot  rootfs.ubi
 u-boot_nand.img  uEnv_ramdisk.txt     uEnv_usbmsc.txt
MLO_nand   ramdisk.gz           rootfs.ext2        rootfs.ubifs
 u-boot_sd.img    uEnv_sd_ramdisk.txt  zImage
MLO_sd     readme.txt           rootfs.ext4        sdcard.img
 u-boot-spl.bin   uEnv_sd.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

在MYD-AM335X系列开发板支持基于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。

进入上一节中解压好的myir-buildroot目录：

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

开始编译buildroot

以MYD-AM335X为例进行的带qt5运行时库的文件系统编译,其它配置参见表3-3-1：

```
$ make myd_c335x_qt5_defconfig
$ make menuconfig  (需要修改配置的时候执行)
$ make
```

为了兼容AM3352系列的产品，Buildroot配置中默认没有使能QML, 用户如果希望使用QT5中的QML功能，需要自行修改配置，添加相应的功能。除此之外，还需要根据3.2节的说明使能SGX。

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Qt5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。

使用myd_c335x_qt5_defconfig配置编译生成的文件系统镜像包含米尔电子基于QT5开发的MEasy HMI演示系统。关于MEasy HMI的更多信息可以查阅《MEasy HMI开发手册》。

4. Linux应用程序

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-AM335X系列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 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-
```

编译完Buildroot之后，在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-myir-linux-gnueabihf-
$ export PATH=$PATH:<WORKDIR>/Filesystem/myir-buildroot/output/h
ost/usr/bin
```

用户可以一次完成所有示例的编译，如下:

```
$ cd <WORKDIR>/Examples/
$ make OPTION=MYD-AM335X-SERIES
```

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 *
```

4.1 LCD

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：

硬件调试环境搭建见部署开发环境中的硬件调试环境搭建部分。

板型	MYD-AM335X	MYD-AM335X-Y	MYD-AM335X-J
LCD屏接口	(MY-TFT070CV2) 7寸电容屏连接J8	(MY-TFT070CV2) 7寸电容屏连接J7	(MY-TFT070CV2) 7寸电容屏连接J8

> 注意：本例程中所用的7寸电容屏资料参见**LCD 屏**。

测试软件环境：

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

测试过程：　　编译并拷贝 `<WORKDIR>/Examples/framebuffer` 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。

重启开发板后，将一个24位颜色深度，分辨率800X480的BMP图像拷贝到开发板/media/, 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 d
isplaying the image
 --donthide    | -u : Do not hide the cursor before and after di
splaying the image
 --noinfo      | -i : Supress image information
 --stretch     | -f : Strech (using a simple resizing routine) t
he image to fit onto
                                              screen if necessary
 --colorstretch| -k : Strech (using a 'color average' resizing r
outine) 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 slidesho
w 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/800-480.bmp
fbv - The Framebuffer Viewer
/media/800-480.bmp
800 x 480
```

程序执行完毕，图片完整显示在LCD上。

4.2 Touch Panel

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：

硬件调试环境搭建见部署开发环境中的硬件调试环境搭建部分。

板型	MYD-AM335X	MYD-AM335X-Y	MYD-AM335X-J
触摸屏接口	(MY-TFT070RV2) 7寸电阻屏连接J8	(MY-TFT070CV2) 7寸电容屏/(MY-TFT070RV2) 7寸电阻屏连接J7	(MY-TFT070CV2) 7寸电容屏/(MY-TFT070RV2) 7寸电阻屏连接J8

测试软件环境：

- Linux Kernel 4.1.18
- TS_CALIBRATE应用程序

测试过程:　　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  event1   event3   mice     mouse1
event0   event2   event4   mouse0
# cat /sys/class/input/event0/device/name
beeper
# cat /sys/class/input/event1/device/name
tps65217_pwrbutton
# cat /sys/class/input/event2/device/name
ft5x06_ts
# cat /sys/class/input/event3/device/name
ti-tsc
```

从以上查询结果可知，电阻触摸对应的设备节点为/dev/input/event3, 电容触摸对应的设备节点为/dev/input/event2,测试步骤如下：

```
# export TSLIB_TSDEVICE=/dev/input/event2
# ts_calibrate
xres = 800, yres = 480
Took 4 samples...
Top left : X =   57 Y =   56
Took 2 samples...
Top right : X =  743 Y =   64
Took 4 samples...
Bot right : X =  742 Y =  438
Took 4 samples...
Bot left : X =   61 Y =  443
Took 4 samples...
Center : X =  398 Y =  246
-8.802551 1.024123 -0.004216
-8.078796 -0.002245 0.998305
Calibration constants: -576884 67116 -276 -529452 -147 65424 655
36
#
```

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。

```
# ls /dev/input
by-path  event0   event1   event2   event3   mice     mouse0
# cat /sys/class/input/event0/device/name
beeper
# cat /sys/class/input/event1/device/name
ti-tsc
# cat /sys/class/input/event2/device/name
tps65217_pwrbutton
# cat /sys/class/input/event3/device/name
volume_keys@0
```

```
#
```

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 32 samples...
Top left : X =  299 Y =  619
Took 21 samples...
Top right : X = 3689 Y =  659
Took 29 samples...
Bot right : X = 3732 Y = 3463
Took 27 samples...
Bot left : X =  280 Y = 3423
Took 18 samples...
Center : X = 2009 Y = 2072
-7.786255 0.204611 -0.000881
-34.248169 -0.001587 0.135514
Calibration constants: -510280 13409 -57 -2244488 -103 8881 6553
6
```

4.3 RTC

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。 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：

- 硬件调试环境搭建见部署开发环境中的硬件调试环境搭建部分。
- RTC所用CR1220 3V纽扣电池一块

板型	**MYD-AM335X**	**MYD-AM335X-Y**	**MYD-AM335X-J**
CR1220电池座子	J1	J2	J23

测试软件环境：

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

测试过程：　　将目录 `<WORKDIR>/Examples/rtc` 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][.s
s].
```

```
such as: 111817582016.18
-h | --help          Print this message

# rtc_test -d /dev/rtc -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/rtc

  Current RTC date/time is 18-11-2016, 18:04:32.
```

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/rtc                                     -- 将系统时间写
入到rtc
```

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/rtc
Mon Aug 15 18:11:08 2016  0.000000 seconds
```

4.4 RS485

本例程演示MYD-AM335X系列同类型2块开发板如何使用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：

- 硬件调试环境搭建见部署开发环境中的硬件调试环境搭建部分。

板型	MYD-AM335X	MYD-AM335X-Y	MYD-AM335X-J
RS485接口	U16的4，5脚分别连接同类型开发板的4，5脚	CON2的5，4脚分别连接同类型开发板的5，4脚	短接JP2和JP3，断开JP5和JP7，J18的1，2，4，5脚分别连接同类型开发板的1，2，4，5脚

测试软件环境：

- Linux Kernel 4.1.18
- tty_test 应用程序

板型	MYD-AM335X	MYD-AM335X-Y	MYD-AM335X-J
设备节点	ttyO1	ttyO2	ttyO2 ttyO3

测试过程：　　将目录 `<WORKDIR>/Examples/tty` 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 mo
de: 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/ttyO1 -b 9600 -w "123456789" -m 1 -l
SEND:123456789
SEND:123456789
SEND:123456789
```

再在另一开发板执行以下命令接收数据：

```
# tty_test -d /dev/ttyO2 -b 9600 -m 1  -l
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
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。

4.5 CAN BUS

本例程演示MYD-AM335X系列同类型2块开发板使用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：

- 硬件调试环境搭建见部署开发环境中的硬件调试环境搭建部分。

板型	MYD-AM335X	MYD-AM335X-Y	MYD-AM335X-J
CAN接口	U16的7，8脚分别连接同类型开发板的7，8脚	CON2的1，2脚分别连接同类型开发板的1，2脚	断开跳线帽JP7，短接JP4，J17的1，2脚分别连接同类型开发板的1，2脚

测试软件环境：

- Linux Kernel 4.1.18
- can_test 应用程序
- ip link应用程序

板型	MYD-AM335X	MYD-AM335X-Y	MYD-AM335X-J
操作节点	can1	can1	can0

测试过程：　　将目录 `<WORKDIR>/Examples/can` 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can1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. suc
h as: 123#112233445566
-h | --help                Print this message
```

```
# ip link set can1 down
# ip link set can1 type can bitrate 50000 triple-sampling on
# ip link set can1 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
[ 5783.823623] c_can_platform 481cc000.can can0: setting BTR=1c1
d BRPE=0000
[ 5786.888723] can: controller area network core (rev 20120528 a
bi 9)
[ 5786.895565] NET: Registered protocol family 29
[ 5786.952090] can: raw protocol (rev 20120528)
====== 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 can1 -l
[ 5888.821956] c_can_platform 481d0000.can can1: setting BTR=1c1
d BRPE=0000
```

```
[ 5891.884726] can: controller area network core (rev 20120528 a
bi 9)
[ 5891.898711] NET: Registered protocol family 29
[ 5891.952878] can: raw protocol (rev 20120528)
can1 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
```

两块板互换一下角色，结果一样，不再赘述。

4.6 Ethernet

本例程演示使用Linux API让开发板上的网口发送和接收数据，详情请参考源码。

测试硬件环境：

硬件调试环境搭建见部署开发环境中的硬件调试环境搭建部分。

板型	MYD-AM335X	MYD-AM335X-Y	MYD-AM335X-J
网口接口	网线连接J5	网线连接J5	网线连接J5

测试软件环境：

- Linux Kernel 4.1.18
- server应用程序
- client应用程序

测试过程：　　编译并拷贝 `<WORKDIR>/Examples/network` 目录下的测试程序client至开发板/usr/bin目录，server拷贝至虚拟机。假设虚拟机的ip是192.168.30.114。

配置开发板的ip：

```
ifconfig eth0 192.168.30.122 up
```

配置好网络后，将开发板上eth0用网线连到PC，以虚拟机为服务器端，开发板为客户端，先在虚拟机上执行以下命令：

```
$./server 192.168.30.122
```

再在开发板执行以下命令可看到所发送的信息：

```
# ./client 192.168.30.114
```

```
send messages: 1234567890 to 192.168.30.114
send messages: 1234567890 to 192.168.30.114
send messages: 1234567890 to 192.168.30.114
send messages: 1234567890 to 192.168.30.114
send messages: 1234567890 to 192.168.30.114
send messages: 1234567890 to 192.168.30.114
```

同时可以看到虚拟机接收到开发板发送的数据：

```
$ ./server 192.168.30.122
received messages: 1234567890@ from 192.168.30.122
received messages: 1234567890@ from 192.168.30.122
received messages: 1234567890@ from 192.168.30.122
received messages: 1234567890@ from 192.168.30.122
received messages: 1234567890@ from 192.168.30.122
received messages: 1234567890@ from 192.168.30.122
```

4.7 NAND Flash

本例程演示如何使用Linux API擦除、写入和读取开发板上的NAND Flash，详情请参考源码。

测试硬件环境：

- 硬件调试环境搭建见部署开发环境中的硬件调试环境搭建部分。
- MYD-AM335x系列开发板一块

测试软件环境：

- Linux Kernel 4.1.18
- mtd_test 应用程序

测试过程：　　编译并拷贝 `<WORKDIR>/Examples/mtd` 目录下的测试程序mtd_test至开发板/usr/bin目录。在Linux终端输入如下命令可对nandflash进行擦除、写入和读取数据：

```
# ./mtd_test /dev/mtd7

MTD Type: 4
MTD total size: 131072 bytes
MTD erase size: 131072 bytes
MTD write size: 2048 bytes

erase the last block at 0

erase done!

writing 16 bytes data to flash...
0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0a 0x0b 0x0c
0x0d 0x0e 0x0f
write done!

reading data from flash...
```

```
0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0a 0x0b 0x0c
0x0d 0x0e 0x0f
read done!
```

4.8 KeyPad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：

- 硬件调试环境搭建见部署开发环境中的硬件调试环境搭建部分。
- MYD-AM335x系列开发板一块

测试软件环境：

- Linux Kernel 4.1.18
- hexdump或keypad_test应用程序

测试过程：　　将目录 `<WORKDIR>/Examples/keypad` 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/eve
nt0
-h | --help          Print this message
```

查看MYD-AM335X按键设备对应的设备节点，可以发现K2,K3两个按键对应/dev/input/event1。

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

# cat /sys/class/input/event0/device/name
ti-tsc
```

```
# cat /sys/class/input/event1/device/name
volume_keys@0
```

测试MYD-AM335X K2和K3。

```
# keypad_test -d /dev/input/event1
Event: Code = 115, Type = 1, Value=1                -- 按下K2
Event: Code = 0, Type = 0, Value=0
Event: Code = 115, Type = 1, Value=0                -- 释放K2
Event: Code = 0, Type = 0, Value=0
Event: Code = 114, Type = 1, Value=1                -- 按下K3
Event: Code = 0, Type = 0, Value=0
Event: Code = 114, Type = 1, Value=2
Event: Code = 0, Type = 0, Value=1
Event: Code = 114, Type = 1, Value=0                -- 释放K3
Event: Code = 0, Type = 0, Value=0
```

查看MYD-AM335X-Y按键设备对应的设备节点，可以发现K3,K4两个按键对应/dev/input/event3，K5对应/dev/input/event2。

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

# cat /sys/class/input/event0/device/name
beeper
# cat /sys/class/input/event1/device/name
ti-tsc
# cat /sys/class/input/event2/device/name
tps65217_pwrbutton
# cat /sys/class/input/event3/device/name
volume_keys@0
```

测试MYD-AM335X-Y K3和K4。

```
# keypad_test -d /dev/input/event3
Event: Code = 115, Type = 1, Value=1                -- 按下K3
Event: Code = 0, Type = 0, Value=0
Event: Code = 115, Type = 1, Value=0                -- 释放K3
Event: Code = 0, Type = 0, Value=0
Event: Code = 114, Type = 1, Value=1                -- 按下K4
Event: Code = 0, Type = 0, Value=0
Event: Code = 114, Type = 1, Value=0                -- 释放K4
Event: Code = 0, Type = 0, Value=0
```

测试MYD-AM335X-Y K5。

```
# keypad_test -d /dev/input/event2
Event: Code = 116, Type = 1, Value=1                -- 按下K5
Event: Code = 0, Type = 0, Value=0
Event: Code = 116, Type = 1, Value=0                -- 释放K5
Event: Code = 0, Type = 0, Value=0
```

查看MYD-AM335X-J按键设备对应的设备节点，可以发现K3,K4两个按键对应/dev/input/event3，K2对应/dev/input/event1。

```
# ls /dev/input/
by-path  event0   event1   event2   event3   mice     mouse0   m
ouse1

# cat /sys/class/input/event0/device/name
ti-tsc
# cat /sys/class/input/event1/device/name
tps65217_pwrbutton
# cat /sys/class/input/event2/device/name
ft5x06_ts
# cat /sys/class/input/event3/device/name
volume_keys@0
```

测试MYD-AM335X-J K3和K4。

```
# keypad_test -d /dev/input/event3
Event: Code = 115, Type = 1, Value=1            -- 按下K3
Event: Code = 0, Type = 0, Value=0
Event: Code = 115, Type = 1, Value=0            -- 释放K3
Event: Code = 0, Type = 0, Value=0
Event: Code = 114, Type = 1, Value=1            -- 按下K4
Event: Code = 0, Type = 0, Value=0
Event: Code = 114, Type = 1, Value=0            -- 释放K4
Event: Code = 0, Type = 0, Value=0
```

测试MYD-AM335X-J K2。

```
# keypad_test -d /dev/input/event1
Event: Code = 116, Type = 1, Value=1            -- 按下K2
Event: Code = 0, Type = 0, Value=0
Event: Code = 116, Type = 1, Value=0            -- 释放K2
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，下面以MYD-AM335X为例，其它版型的KeyPad测试情况类似。分别按K2,K3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/event1
0000000 022b 0000 2a83 0005 0001 0073 0001 0000
0000010 022b 0000 2a83 0005 0000 0000 0000 0000
0000020 022b 0000 78b5 0007 0001 0073 0000 0000
0000030 022b 0000 78b5 0007 0000 0000 0000 0000

0000040 0231 0000 8c69 0001 0001 0072 0001 0000
```

```
0000050 0231 0000 8c69 0001 0000 0000 0000 0000
0000060 0231 0000 fc11 0003 0001 0072 0000 0000
0000070 0231 0000 fc11 0003 0000 0000 0000 0000
```

4.9 GPIO-LED

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：

- 硬件调试环境搭建见部署开发环境中的硬件调试环境搭建部。
- MYD-AM335x系列开发板一块

测试软件环境：

- Linux Kernel 4.1.18
- echo, led_test 应用程序

测试过程：　　编译并拷贝 `<WORKDIR>/Examples/led` 目录下的测试程序led_test至开发板/usr/bin目录。

查看MYD-AM335X LED的设备节点。

```
# ls /sys/class/leds
myc:green:user1  myd:green:user2  myd:green:user3
```

用'echo'命令对MYD-AM335X LED进行控制。

```
D3：
#echo "1" > /sys/class/leds/myc\:green\:user1/brightness
#echo "0" > /sys/class/leds/myc\:green\:user1/brightness
D39：
#echo "1" > /sys/class/leds/myd\:green\:user2/brightness
#echo "0" > /sys/class/leds/myd\:green\:user2/brightness
D40：
#echo "1" > /sys/class/leds/myd\:green\:user3/brightness
#echo "0" > /sys/class/leds/myd\:green\:user3/brightness
```

查看MYD-AM335X-Y LED的设备节点。

```
# ls /sys/class/leds
myc:green:user1  myd:green:user2
```

用'echo'命令对MYD-AM335X-Y LED进行控制。

```
D3：
#echo "1" > /sys/class/leds/myc\:green\:user1/brightness
#echo "0" > /sys/class/leds/myc\:green\:user1/brightness
D2：
#echo "1" > /sys/class/leds/myd\:green\:user2/brightness
#echo "0" > /sys/class/leds/myd\:green\:user2/brightness
```

查看MYD-AM335X-J LED的设备节点。

```
# ls /sys/class/leds
myc:green:user1  myd:green:user2
```

用'echo'命令对MYD-AM335X-J LED进行控制。

```
D3：
# echo "1" > /sys/class/leds/myc\:green\:user1/brightness
# echo "0" > /sys/class/leds/myc\:green\:user1/brightness
D2：
# echo "1" > /sys/class/leds/myd\:green\:user2/brightness
# echo "0" > /sys/class/leds/myd\:green\:user2/brightness
```

用'led_test'应用程序对LED进行控制。

```
# 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:green:user1 -l 0
Set led myc:green:user1 off, brightness = 0
# led_test -d myc:green:user1 -l 1
Set led myc:green:user1 on, brightness = 1
```

依照不同的板型更换对应的设备节点可以用led_test实现不同LED的控制。

4.10 Audio

本例程演示如何使用Linux API控制音频的输入与输出，详情请参考源码。

测试硬件环境：　　硬件调试环境搭建见部署开发环境中的硬件调试环境搭建部分。

板型	MYD-AM335X	MYD-AM335X-Y	MYD-AM335X-J
Audio接口	音频输入J11，耳机连接J10	音频输入J9，耳机连接J8	音频输入J10，耳机连接J9

测试软件环境：

- Linux Kernel 4.1.18
- audio_test 应用程序

测试过程：　　编译并拷贝 `<WORKDIR>/Examples/audio` 目录下的测试程序audio_test至开发板/usr/bin目录。将耳机插到音频输出口，音频输入到音频输入口，在Linux终端输入如下命令：

```
# 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
```

4.11 USB Host

本例程演示通过相关命令来验证USB Host驱动的可行性，实现读写U盘的功能。

测试硬件环境：

- 硬件调试环境搭建见部署开发环境中的硬件调试环境搭建部分。
- U盘一个

板型	MYD-AM335X	MYD-AM335X-Y	MYD-AM335X-J
USB Host接口	J3或J4	J19	J27

测试软件环境：

- Linux Kernel 4.1.18
- mount, umount 应用程序

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：

```
# [  334.568567] usb 2-1.4: new high-speed USB device number 3 u
sing musb-hdrc
[  334.688922] usb 2-1.4: New USB device found, idVendor=1908, i
dProduct=0226
[  334.696305] usb 2-1.4: New USB device strings: Mfr=0, Product
=0, SerialNumber=0
[  334.743502] usb-storage 2-1.4:1.0: USB Mass Storage device de
tected
[  334.751563] scsi host0: usb-storage 2-1.4:1.0
[  334.762053] usbcore: registered new interface driver usb-stor
age
[  334.788766] cpu cpu0: clk_voltdm_notifier_handler: Failed to
```

```
scale voltage(1100000)
: -22
[  334.797531] cpu cpu0: clk_voltdm_notifier_handler: Failed to
scale voltage(950000)
: -22
[  334.806827] cpu cpu0: failed to set clock rate: -16
[  334.812093] cpufreq: __target_index: Failed to change cpu fre
quency: -16
[  335.759638] scsi 0:0:0:0: Direct-Access     Generic  Mass-Sto
rage     1.11 PQ: 0
ANSI: 2
[  336.488565] sd 0:0:0:0: [sda] 15605760 512-byte logical block
s: (7.99 GB/7.44 GiB)
[  336.497237] sd 0:0:0:0: [sda] Write Protect is off
[  336.503459] sd 0:0:0:0: [sda] Mode Sense: 03 00 00 00
[  336.509699] sd 0:0:0:0: [sda] No Caching mode page found
[  336.515366] sd 0:0:0:0: [sda] Assuming drive cache: write thr
ough
[  336.531208]  sda: sda1
[  336.552003] 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
[  429.892793] FAT-fs (sda1): Volume was not properly unmounted.
 Some data may be
corrupt. Please run fsck.
# cd /mnt
# ls
1.bmp                mtd                  rootfs.ubi
MLO                  myd_c335x.dtb        spi
audio                myir-linux-examples  u-boot.img
audio1               network              zImage
#
```

拔下U盘，内核提示信息。

```
# [  493.899143] usb 2-1.4: USB disconnect, device number 3
```

4.12 USB Device

本实例演示通过相关命令来验证USB DEVICE驱动的可行性。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：

- 硬件调试环境搭建见部署开发环境中的硬件调试环境搭建部分。
- TF卡一个
- USB miniB to mini A 线一根

板型	MYD-AM335X	MYD-AM335X-Y	MYD-AM335X-J
MINI USB接口	J2	J3	J3
TF卡接口	J17	J12	J19

测试软件环境：

- Linux Kernel 4.1.18
- modprobe 应用程序

测试过程：

开发板上的系统起来后，使用USB mini B to USB A转接线连接开发板与电脑端，其中USB mini B接口连接开发板，USB A接口连接电脑端，插入TF卡,在开发板上运行如下命令：

```
# modprobe g_mass_storage stall=0 file=/dev/mmcblk0p1 removable=
1
[  687.171803] udc musb-hdrc.0.auto: registering UDC driver [g_m
ass_storage]
[  687.179455] Mass Storage Function, version: 2009/09/11
[  687.184933] LUN: removable file: (no medium)
```

```
[  687.192157] lun0: open backing file: /dev/mmcblk0p1
[  687.197379] LUN: removable file: /dev/mmcblk0p1
[  687.202952] Number of LUNs=1
[  687.206057] g_mass_storage gadget: adding config #1 'Linux Fi
le-Backed Storage'
/bf2a45cc
[  687.215260] Number of LUNs=1
[  687.219060] g_mass_storage gadget: I/O thread pid: 185
[  687.224576] g_mass_storage gadget: adding 'Mass Storage Funct
ion'/dc896b00 to
config 'Linux File-Backed Storage'/bf2a45cc
[  687.236802] g_mass_storage gadget: cfg 1/bf2a45cc speeds: hig
h full
[  687.243534] g_mass_storage gadget:   interface 0 = Mass Stora
ge Function/dc896b00
[  687.252005] g_mass_storage gadget: Mass Storage Gadget, versi
on: 2009/09/11
[  687.259466] g_mass_storage gadget: userspace failed to provid
e iSerialNumber
[  687.266948] g_mass_storage gadget: g_mass_storage ready
[  687.273141] g_mass_storage musb-hdrc.0.auto: usb_gadget_udc_s
tart
```

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网卡的功能等等。这里不一一列举。

5. Qt开发

在第3.3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-AM335X 系列开发板上运行。

5.1 安装QtCreator

- 安装开发环境

```
$ 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win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。

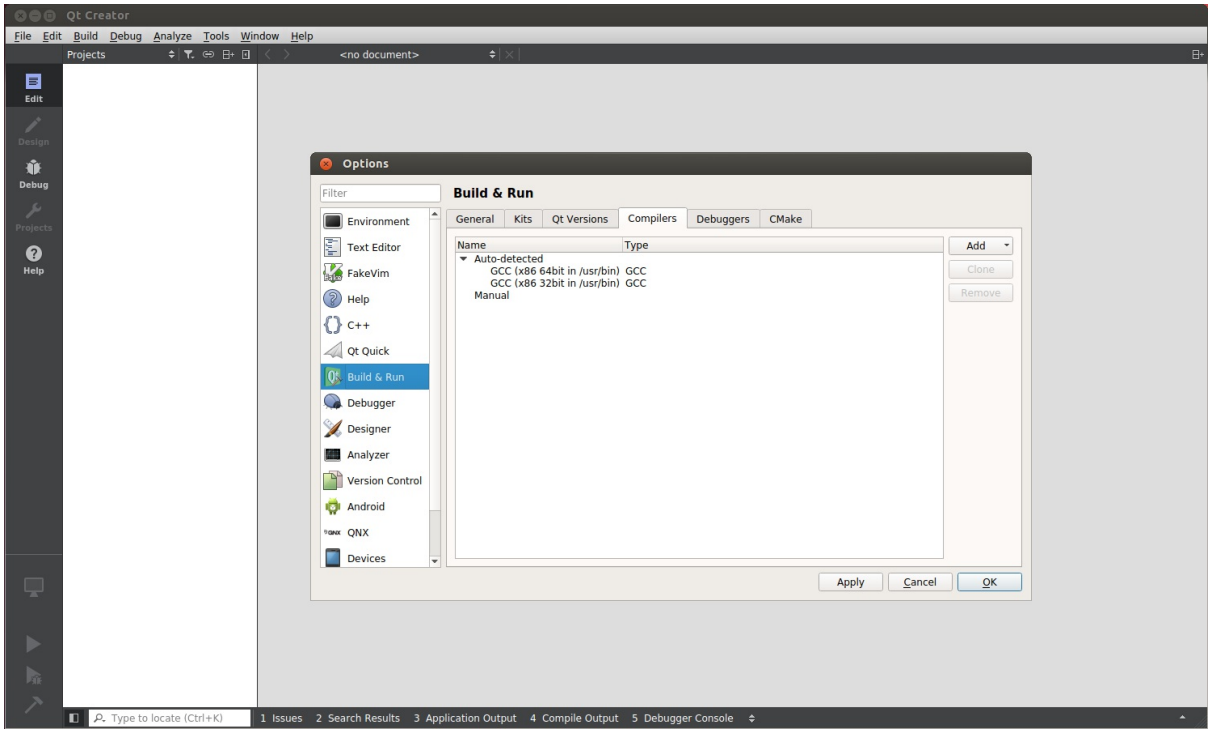

图5-2-1 编译器选择

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。

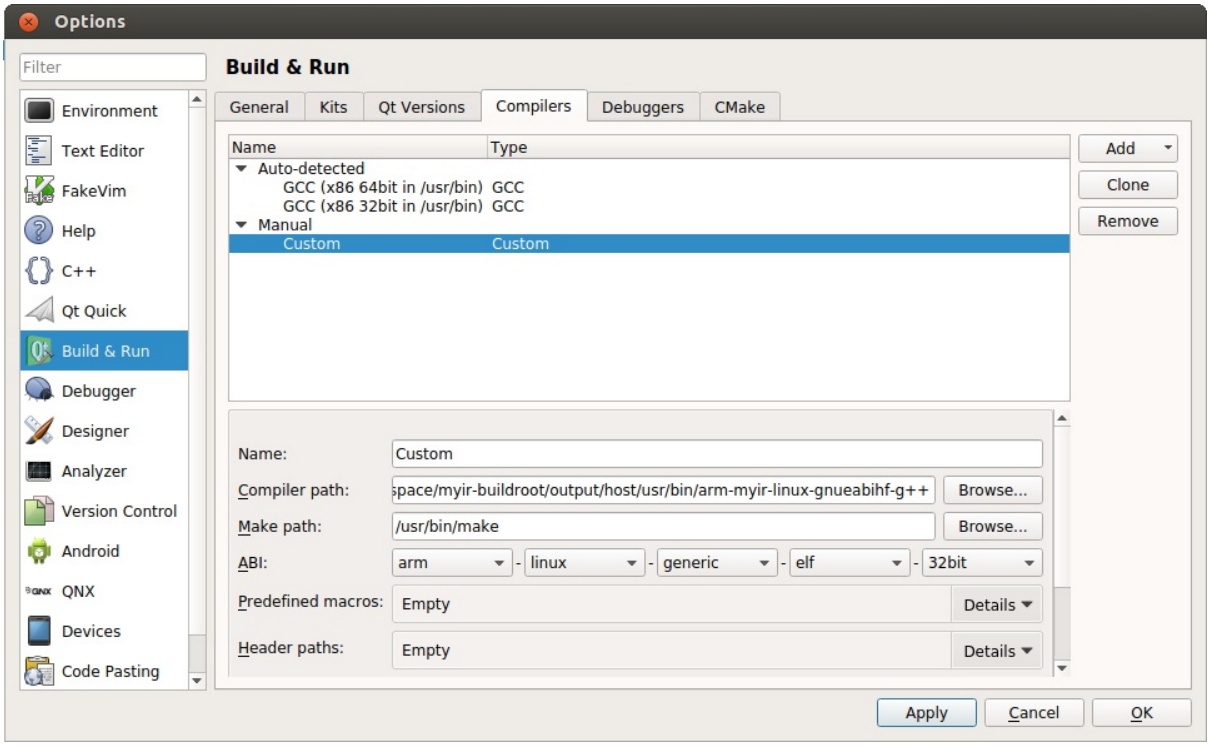

图5-2-2 添加编译器

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。

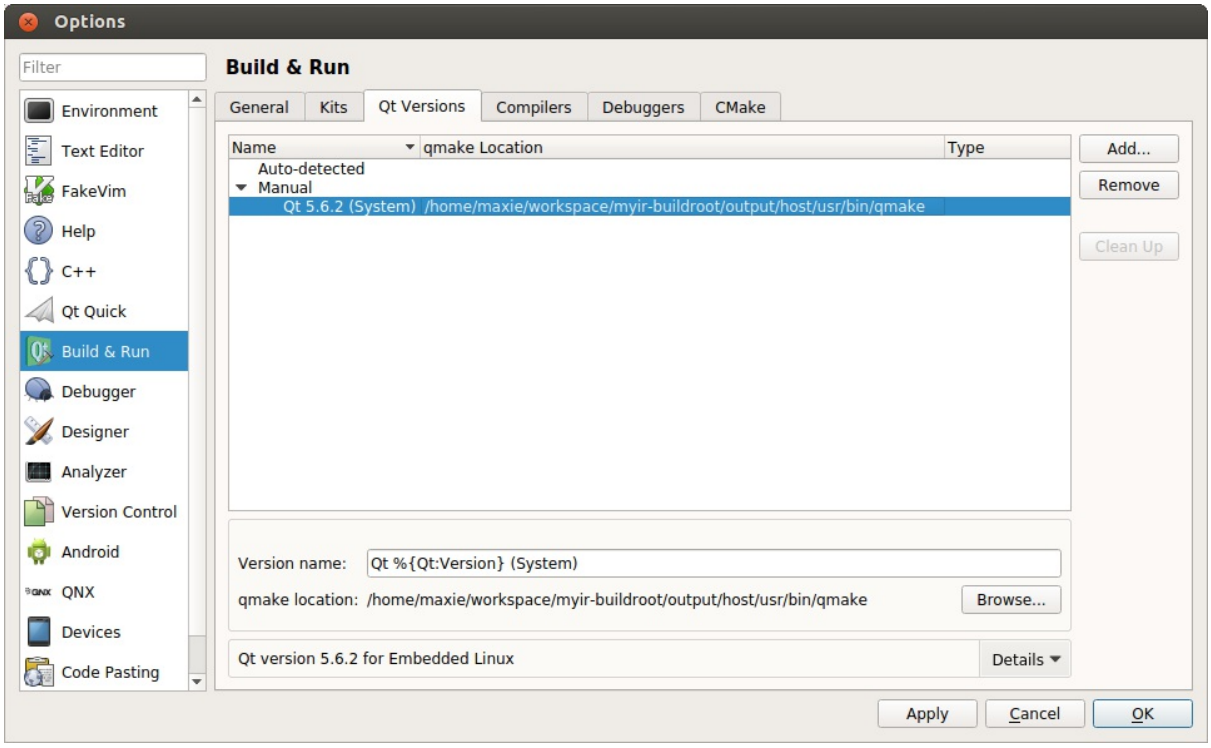

图5-2-3 选择qmake构建工具

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。

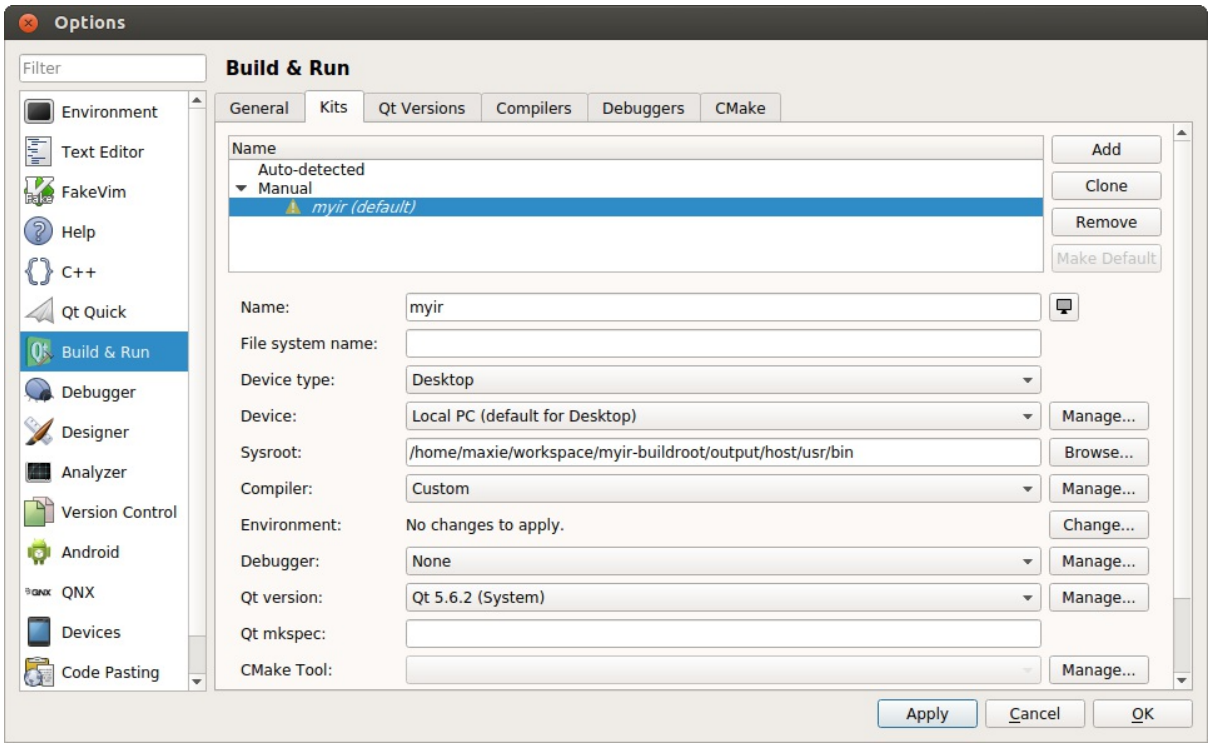

图5-2-4 添加Kits

- 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:

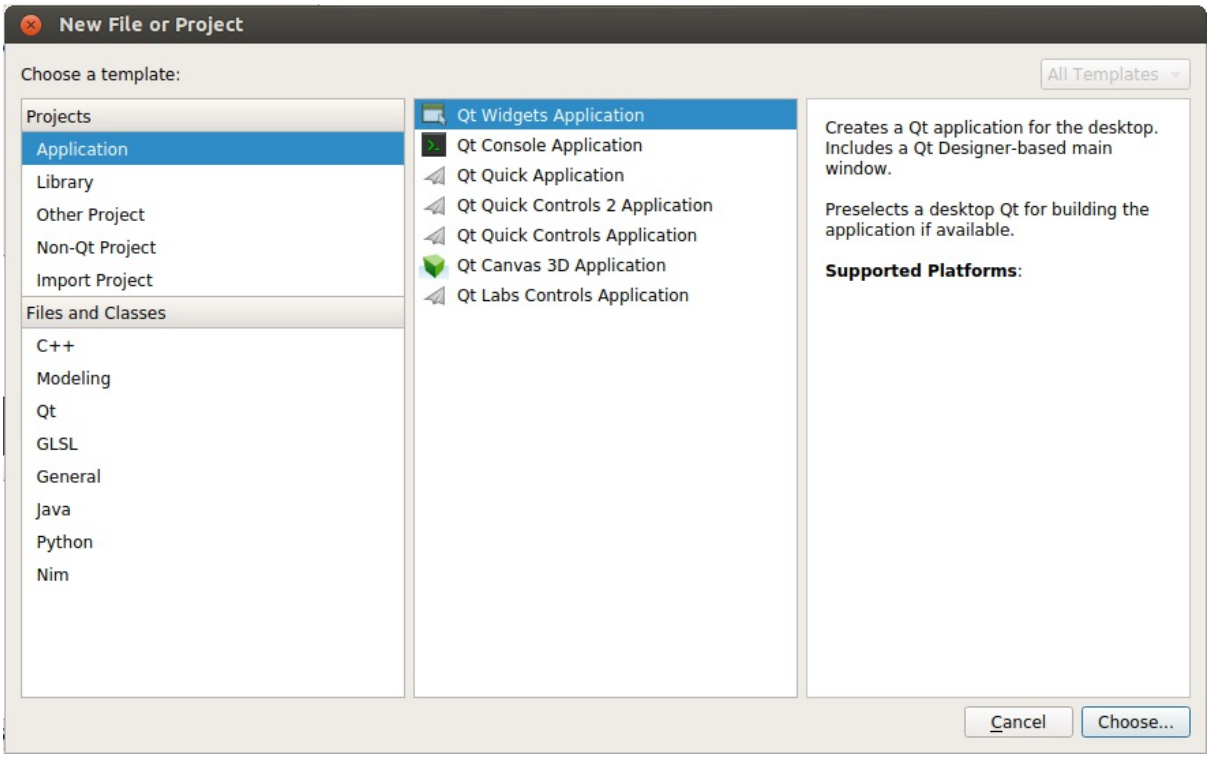

图5-2-5 创建新工程

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：

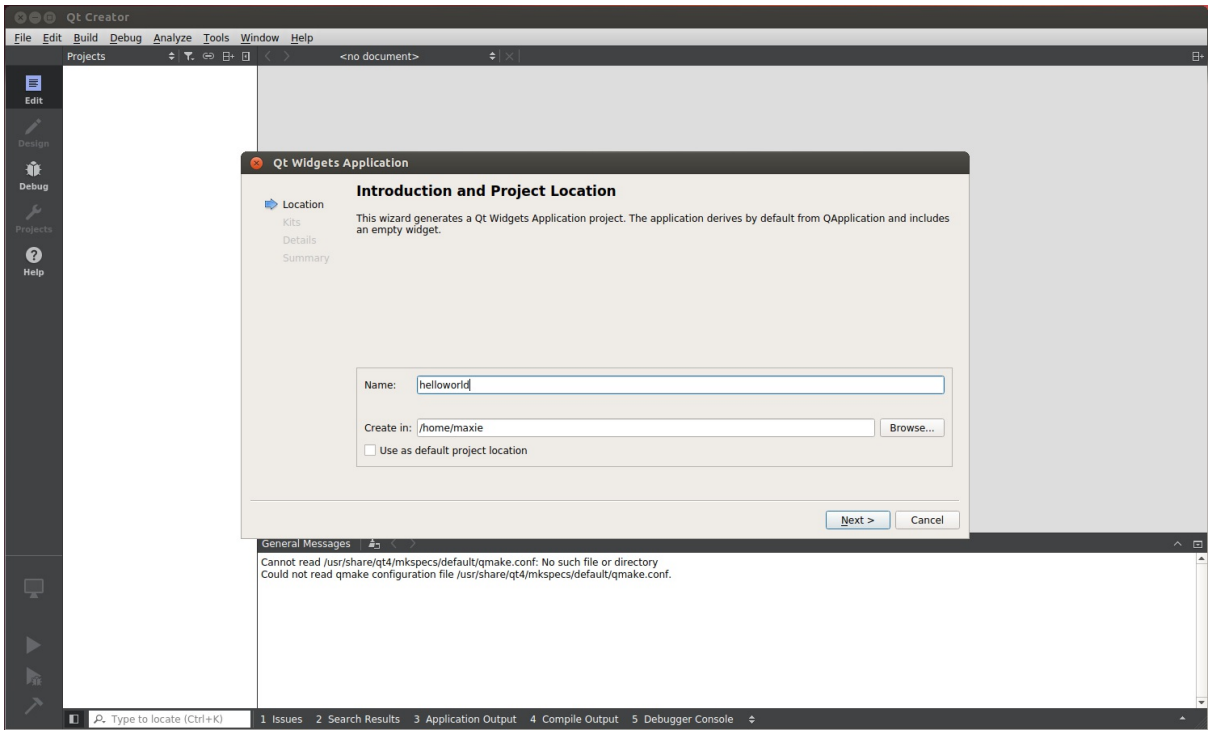

图5-2-6 设置新工程名称和路径

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

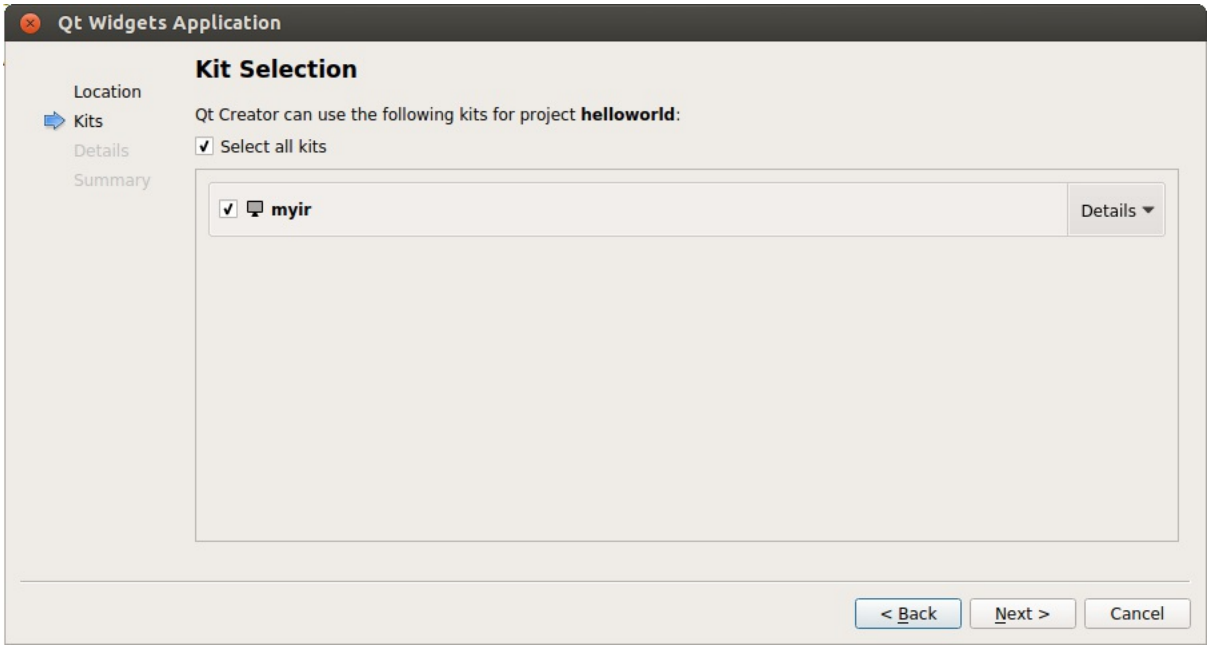

图5-2-7 设置新工程Kits

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。

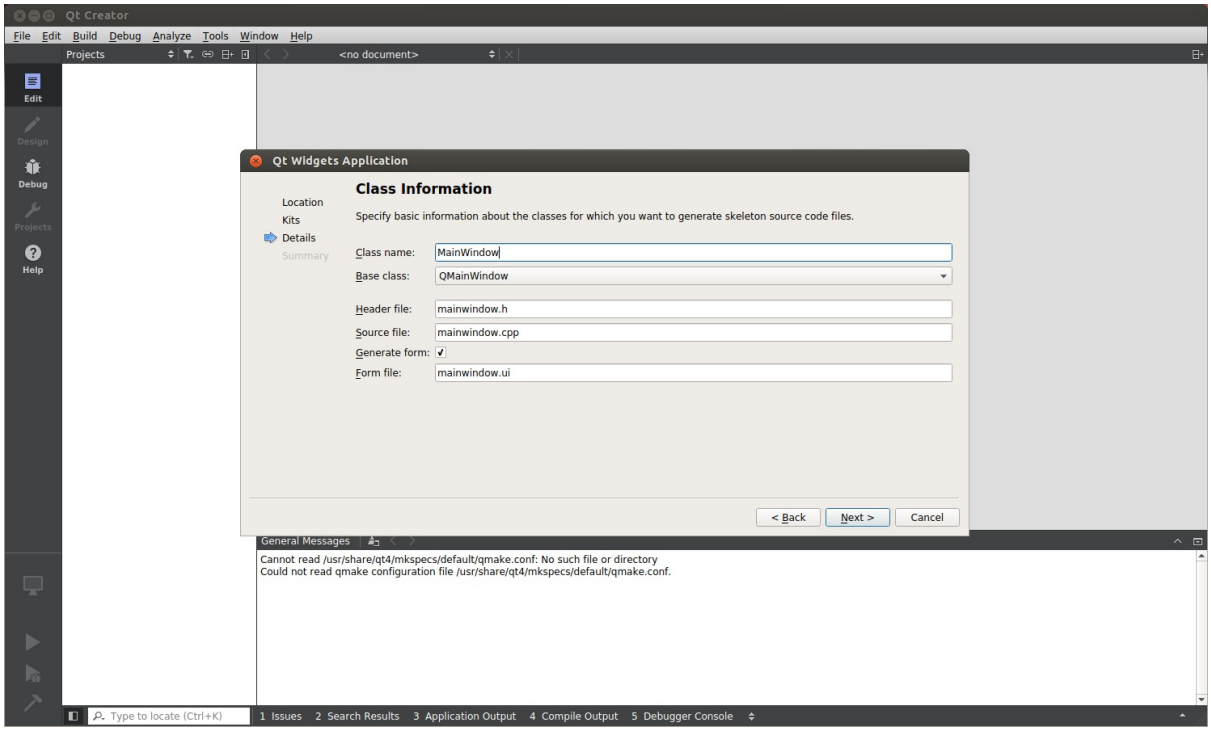

图5-2-8 选择应用程序的类型

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

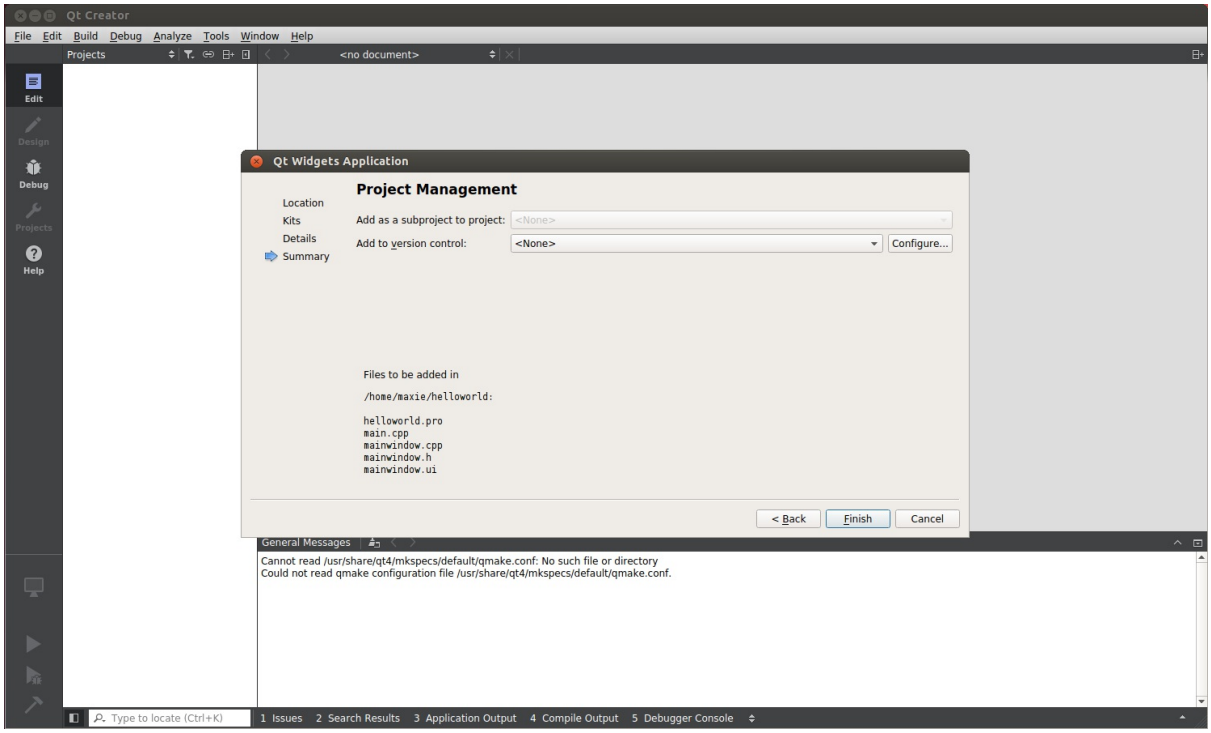

图5-2-9 完成工程创建

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。

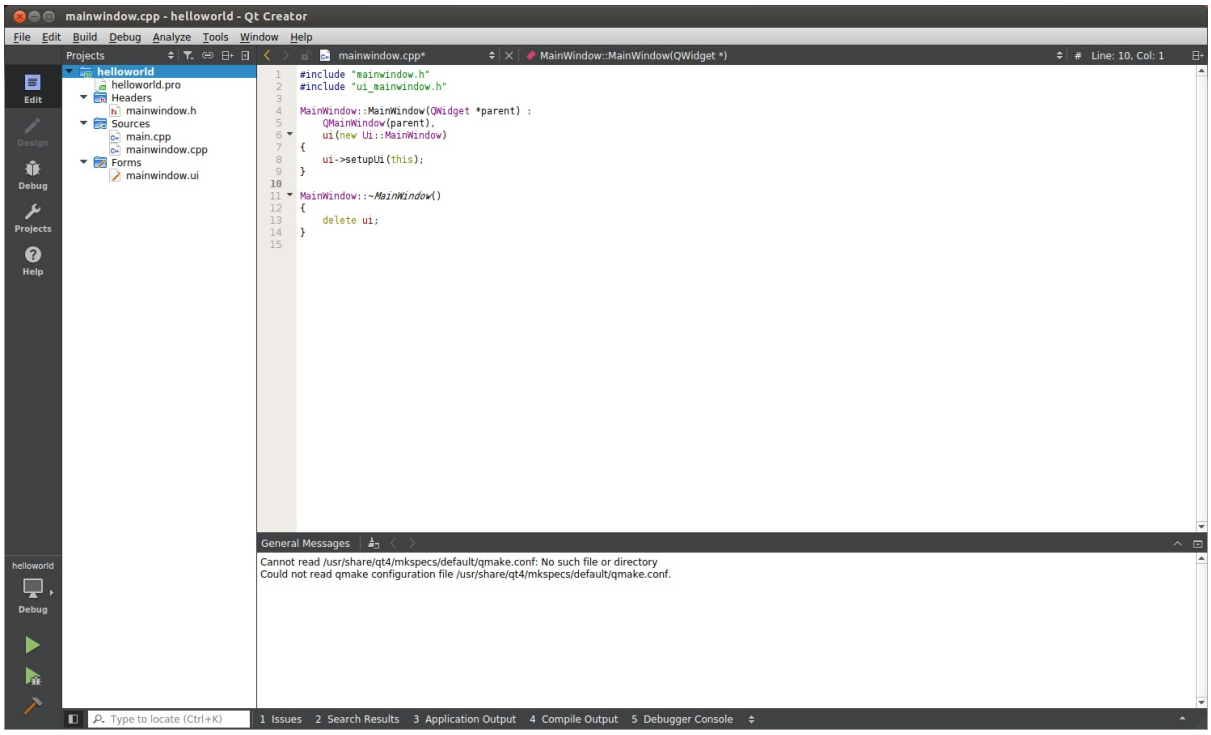

图5-3-1 项目文件管理窗口

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

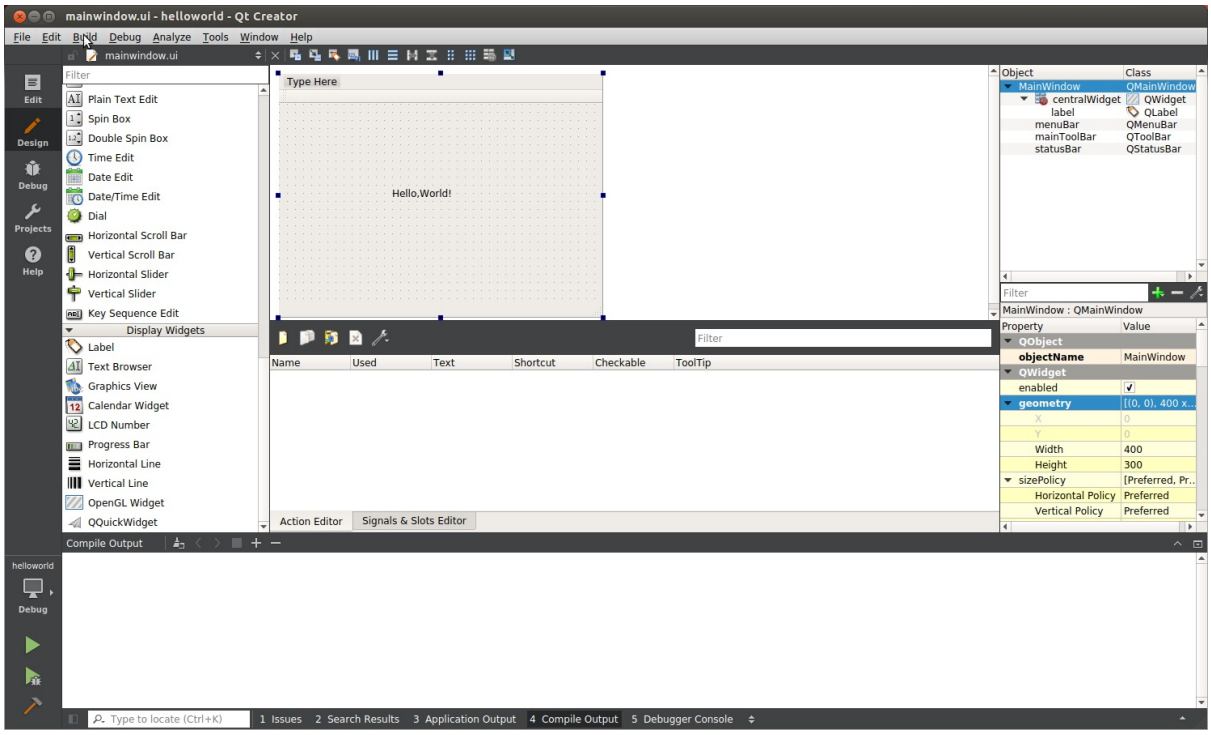

图5-3-2 可视化界面编辑

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。

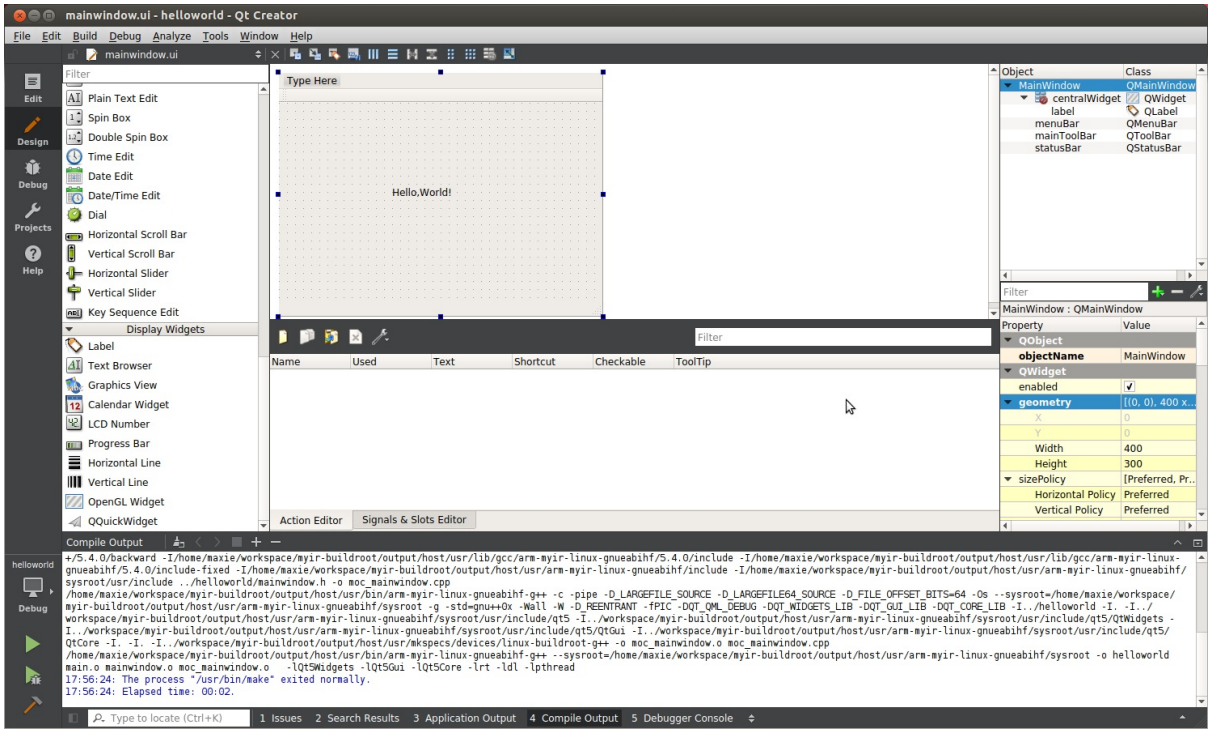

图5-3-3 编译项目

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。

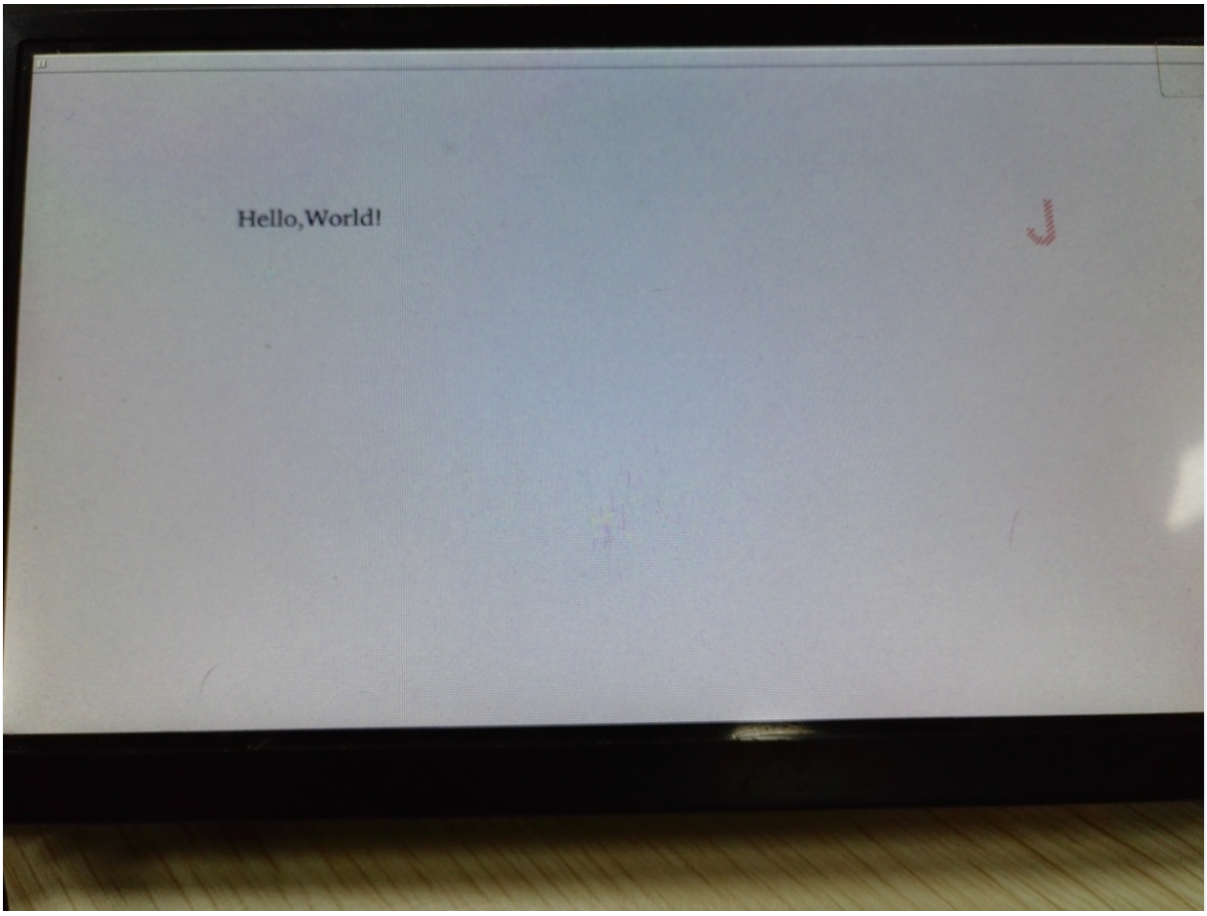

图5-3-4 QT应用程序执行

6. Linux 系统更新

本小节主要介绍Linux系统的tf卡启动，NAND Flash烧录，NFS ROOT文件系统挂载。

光盘镜像文件说明：

表 6-1 预编译镜像文件列表

File Name	Description
MLO	SPL(First Stage Bootloader),第一阶段启动引导程序，由U-boot代码编译生成
MLO_nand	SPL(First Stage Bootloader),第一阶段启动引导程序，用于使用NAND Flash的核心板，预编译镜像位于"board/myir/myd_c335x/"，使用时需要改名为MLO
MLO_sd	SPL(First Stage Bootloader),第一阶段启动引导程序，用于使用NAND Flash的核心板，预编译镜像位于"board/myir/myd_c335x/"，使用时需要改名为MLO
MLO_emmc	SPL(First Stage Bootloader),第一阶段启动引导程序，用于使用EMMC的核心板，预编译镜像位于"board/myir/myd_c335x/"，使用时需要改名为MLO
MLO_usbmsc	SPL(First Stage Bootloader),第一阶段启动引导程序，用于使用NAND Flash的核心板，预编译镜像位于"board/myir/myd_c335x/"，使用时需要改名为MLO
u-boot.img	第二阶段启动引导程序，由U-boot代码编译生成
u-boot_nand.img	第二阶段启动引导程序，用于使用NAND Flash的核心板，预编译镜像位于"board/myir/myd_c335x/"，使用时需要

	改名为u-boot.img
u-boot_sd.img	第二阶段启动引导程序，用于使用NAND Flash的核心板，预编译镜像位于"board/myir/myd_c335x/"，使用时需要改名为u-boot.img
u-boot_emmc.img	第二阶段启动引导程序，用于使用EMMC的核心板，预编译镜像位于"board/myir/myd_c335x/"，使用时需要改名为u-boot.img
u-boot_usbmsc.img	第二阶段启动引导程序，用于使用NAND Flash的核心板，预编译镜像位于"board/myir/myd_c335x/"，使用时需要改名为u-boot.img
uEnv.txt	U-boot 默认环境变量
uEnv_ramdisk.txt	从TF卡启动引导ramdisk文件系统的环境变量，使用时需要改名为uEnv.txt
uEnv_sd.txt	从TF卡启动引导Ext4文件系统的环境变量，使用时需要改名为uEnv.txt
uEnv_sd_ramdisk.txt	从TF卡启动引导ramdisk文件系统的环境变量，使用时需要改名为uEnv.txt
uEnv_usbmsc.txt	从USB Mass Storage设备启动引导Ext4文件系统的环境变量，使用时需要改名为uEnv.txt
uEnv_usbmsc_ramdisk.txt	从USB Mass Storage设备启动引导ramdisk文件系统的环境变量，使用时需要改名为uEnv.txt
uEnv_mmc.txt	从EMMC启动引导EXT4文件系统的环境变量，使用时需要改名为uEnv.txt
zImage	压缩内核镜像文件
myd_c335x.dtb	用于使用NAND Flash的MYD-AM335X核心板
myd_c335x_emmc.dtb	用于使用EMMC的MYD-AM335X核心板
myd_y335x.dtb	用于使用NAND Flash的MYD-AM335X-Y核心板
myd_y335x_emmc.dtb	用于使用EMMC的MYD-AM335X-Y核心板

myd_j335x.dtb	用于使用NAND Flash的MYD-AM335X-J核心板
myd_j335x_emmc.dtb	用于使用EMMC的MYD-AM335X-J核心板
rootfs.tar.gz	ramdisk filesystem compressed by gzip
rootfs.ubi	UBIFS filesystem image
ramdisk.gz	ramdisk filesystem compressed by gzip
sdcard.img	TF/SD/EMMC disk image

`uEnv*.txt`是文件文件，它们定义了U-boot启动时的一些环境变量，通过这些环境变量可以决定U-boot加载和引导内核文件系统的行为，以RAMDISK方式启动所用的`uEnv_ramdisk.txt`为例：`fdtfile`定义要使用的设备树文件名称，`devtype`定义存放镜像文件的设备类型，`devnum`定义存放镜像文件的设备序号(0: TF Card, 1: EMMC).`bootdir`定义镜像文件在设备上的存放目录，`bootpart`定义镜像文件在设备上的存放分区(0:1 表示设备0上的第一个分区)。`uenvcmd`定义了一个命令脚本，由U-boot执行，用于加载内核，设备树，最后启动引导整个系统。

```
# This uEnv.txt file can contain additional environment settings
 that you
# want to set in U-Boot at boot time.  This can be simple variab
les 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 tha
t some
#       variables such as bootdelay cannot be changed by this fi
le 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 disabl
e LCD display.
fdtfile=myd_c335x.dtb
devtype=mmc
devnum=0
bootdir=/
bootpart=0:1
uenvcmd=if run loadimage; then run loadfdt; run loadramdisk; ech
o Booting from mmc${mmcdev} ...; run ramargs; print bootargs; bo
otz ${loadaddr} ${rdaddr} ${fdtaddr}; fi;
```

> 注意： **MYD-AM335X**的启动开发板之前需要注意**JP8**跳线的连接，连接**JP8**的**1-2**脚时将会从**SD**卡启动系统，连接**JP8**的**2-3**脚时将会从**NandFlash**启动系统。改变**JP8**跳线的连接之后需要给开发板重新上电启动才能生效。**MYD-AM335X-Y**的为**jp1**，**MYD-AM335X-J**的为**jp6**。

`MYD-AM335X` 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: NAND Flash启动（ubi文件系统，可选用于使用NAND Flash的核心板).
- 方案4：EMMC启动(EXT4文件系统，可选用于使用EMMC的核心板)
- 方案5: NFS ROOT文件系统（主要用于文件系统调试）.

TF卡启动(EXT4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。

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。

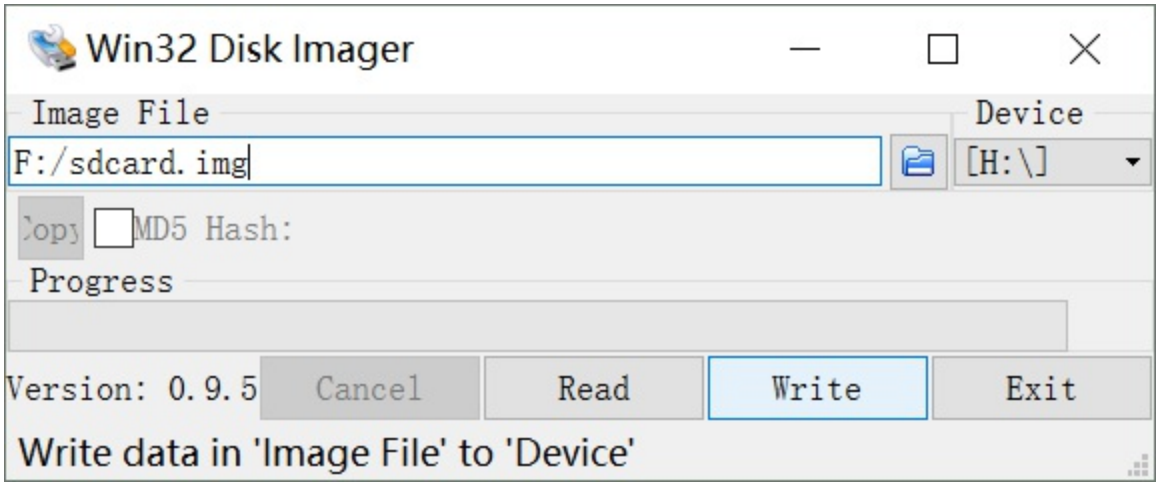

图6-1 使用win32diskimager烧写sdcard.img

- 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。

TF卡启动(Ramdisk文件系统)

> 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。

- TF卡格式化
  (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。)
  1.把MMC/SD卡插入USB读卡器，然后将读卡器跟电脑连接 2.打开HP USB Disk Storage Format Tool，出现类似提示如下：

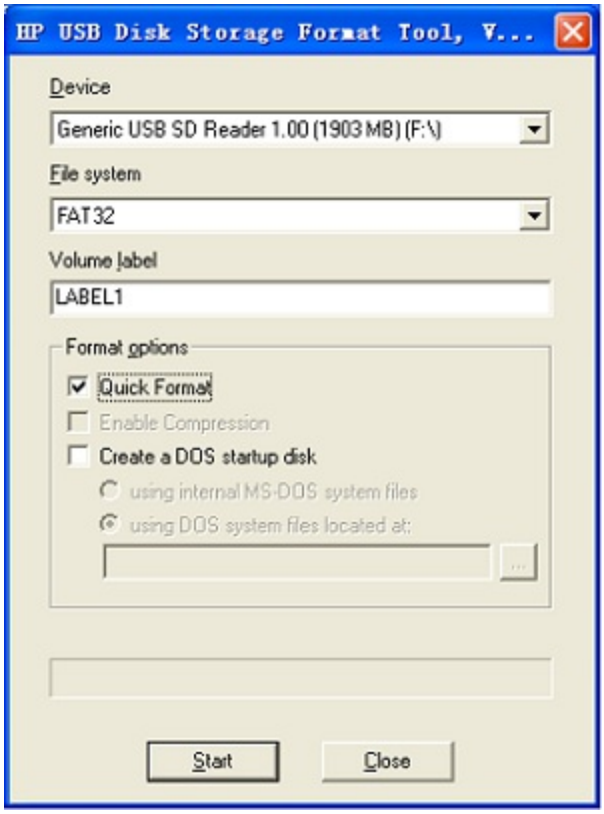

图6-2 格式化TF卡

```
3.选择“FAT32”系统格式
4.勾选Quick format（可选）
5.点击“Start”
6.等待格式化完成，点击“OK”
```

- 映像更新

将02-Images\Linux-image目录下的所有文件拷贝到TF卡上，备份uEnv.txt并重命名uEnv_ramdisk.txt为uEnv.txt，然后将TF卡插入到开发板上的SD卡插槽，连接对应板子设置启动方式的跳线帽的1-2脚，上电重启开发板，输入账户名 `root` 登录。

NAND Flash启动（ubi文件系统，可选用于使用NAND Flash的核心板)

Nand启动映像的更新需要借助于u-boot来完成。不管NAND Flash是否有数据，都可以利用TF卡启动的u-boot对NAND Flash更新映像。

- 准备

  1.用光盘目录03-Tools目录下的HP USB Disk Storage Format Tool 2.0.6工具将TF卡格式化为FAT或FAT32文件系统

  2.将光盘目录02-Images\Linux-image目录下的映像文件拷贝到TF卡中

- 更新

将带有系统映象的TF卡插入开发板，连接对应板子设置启动方式的跳线帽的1-2脚，上电启动，在u-boot 的提示读秒处，按下键盘上任意键进入u-boot：

```
U-Boot 2016.05 (Jan 09 2017 - 19:37:43 +0800)

       Watchdog enabled
I2C:   ready
DRAM:  512 MiB
NAND:  512 MiB
MMC:   OMAP SD/MMC: 0, OMAP SD/MMC: 1
*** Warning - bad CRC, using default environment

Net:   cpsw
Press SPACE to abort autoboot in 2 seconds
MYIR>#
```

进入u-boot命令行后，在命令行上输入“run updatesys”，开始自动更新系统。如果u-boot分区有调整，需要擦除整个NAND Flash,如下所示：

```
MYIR># nand erase.chip
NAND erase.chip: device 0 whole chip
Erasing at 0x1ffe0000 -- 100% complete.
OK
```

```
MYIR># run updatesys
switch to partitions #0, OK
mmc0 is current device
reading MLO
55092 bytes read in 10 ms (5.3 MiB/s)

NAND write: device 0 offset 0x0, size 0xd734
 55092 bytes written: OK
reading myd_c335x.dtb
39229 bytes read in 10 ms (3.7 MiB/s)

NAND write: device 0 offset 0x80000, size 0x993d
 39229 bytes written: OK
reading u-boot.img
321300 bytes read in 34 ms (9 MiB/s)

NAND write: device 0 offset 0xc0000, size 0x4e714
 321300 bytes written: OK
reading zImage
4480016 bytes read in 396 ms (10.8 MiB/s)

NAND write: device 0 offset 0x200000, size 0x445c10
 4480016 bytes written: OK
reading rootfs.ubi
24248320 bytes read in 2111 ms (11 MiB/s)

NAND write: device 0 offset 0xa00000, size 0x1720000
 24248320 bytes written: OK
MYIR>#
```

系统更新完后，会自动u-boot的命令行模式，断电拔出TF卡，连接对应板子设置启动方式的跳线帽的2-3脚，给开发板重新上电(改变启动模式之后必须重新上电)后即可从Nand Flash启动，输入帐户名 `root` 登录。

EMMC启动(EXT4文件系统，可选用于使用EMMC的核心板)

米尔AM335X系列核心板都支持NAND Flash和EMMC两种存储设备，一些需要使用比较大容量存储空间的用户会选择使用EMMC的核心板。目前U-boot中还不支持直接写入镜像到EMMC,用户需要通过一个运行于TF卡的ramdisk系统来进行EMMC的烧写和更新，具体步骤如下:

- 准备工作
    1. 使用HP USB Disk Storage Format Tool对TF卡进行格式化
    2. 将MLO, MLO_emmc, u-boot.img, u-boot_emmc.img, uEnv_ramdisk.txt, uEnv_mmc.txt, zImage, myd_c335x_emmc.dtb, ramdisk.gz, rootfs.tar.gz拷贝到TF卡
    3. 将uEnv_ramdisk.txt中的fdtfile参数修改为"fdtfile=myd_c335x_emmc.dtb"，然后改名为uEnv.txt
    4. 按照第一种方式启动进入ramdisk文件系统(账户为root，密码为空)
- 执行升级
    1. 执行命令"/etc/modules-load.myir/updatesys.sh loader2emmc sd"将TF卡上的镜像写入到EMMC上
    2. 改变启动模式为EMMC,重新上电启动
- 自动升级

自动升级主要用于批量生产过程中，不需要人工输入命令直接进行升级，提高生产效率。用户只需要在RAMDISK启动的uEnv.txt中增加一个变量 `optargs=updatesys_from_sd_to_emmc` 即可实现自动升级, 修改后的uEnv.txt内容如下。

```
# This uEnv.txt file can contain additional environment settings
 that you
# want to set in U-Boot at boot time.  This can be simple variab
les 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 tha
t some
#       variables such as bootdelay cannot be changed by this fi
le 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 disabl
e LCD display.
fdtfile=myd_c335x_emmc.dtb
devtype=mmc
devnum=0
bootdir=/
bootpart=0:1
optargs=updatesys_from_sd_to_emmc
uenvcmd=if run loadimage; then run loadfdt; run loadramdisk; ech
o Booting from mmc${mmcdev} ...; run ramargs; print bootargs; bo
otz ${loadaddr} ${rdaddr} ${fdtaddr}; fi;
```

optargs变量在系统启动的时候会传递给内核，进入ramdisk系统之后, 通过 `/etc/init.d/S98preboot` 调用 `/etc/modules-load.myir/preboot` . 在进入系统之后检测内核启动参数 `/proc/cmdline` , 从而决定升级的方式。

```
# file: /etc/modules-load.myir/preboot
#! /bin/sh
if grep "updatesys_from_usbmsc_to_emmc" /proc/cmdline > /dev/nul
l;then
        echo
        echo "*********************************************"
        echo "*** SYSTEM UPDATE FROM USBMSC TO EMMC *********"
        echo "*********************************************"
```

```
        echo
        yes | /etc/modules-load.myir/updatesys.sh loader2emmc us
bmsc
fi
if grep "updatesys_from_sd_to_emmc" /proc/cmdline > /dev/null;th
en
        echo
        echo "*********************************************"
        echo "*** SYSTEM UPDATE FROM SDCARD TO EMMC *********"
        echo "*********************************************"
        echo
        yes | /etc/modules-load.myir/updatesys.sh loader2emmc sd
fi
#! /bin/sh
if grep "updatesys_from_usbmsc_to_nand" /proc/cmdline > /dev/nul
l;then
        echo
        echo "*********************************************"
        echo "*** SYSTEM UPDATE FROM USBMSC TO NAND *********"
        echo "*********************************************"
        echo
        yes | /etc/modules-load.myir/updatesys.sh loader2nand us
bmsc
fi
if grep "updatesys_from_sd_to_nand" /proc/cmdline > /dev/null;th
en
        echo
        echo "*********************************************"
        echo "*** SYSTEM UPDATE FROM SDCARD TO NAND *********"
        echo "*********************************************"
        echo
        yes | /etc/modules-load.myir/updatesys.sh loader2nand sd
fi
```

升级成功之后，核心板上LED灯会慢闪；升级失败，核心板上LED灯会快闪，具体行为可以查看 `/etc/modules-load.myir/updatesys.sh` 内容，以下是通过 `updatesys.sh` 升级成功的log信息。

```
*********************************************
*** SYSTEM UPDATE FROM SDCARD TO EMMC *********
*********************************************

All data on eMMC now will be destroyed! Continue? [y/n]
===> found sdcard /dev/mmcblk0  on  mmc0:0007
[   18.631313] FAT-fs (mmcblk0p1): Volume was not properly unmou
nted. Some data may be corrupt. Please run fsck.
===> found emmc /dev/mmcblk1  on  mmc1:0001
1024+0 records in
1024+0 records out
DISK SIZE - 3909091328 bytes
===> 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
```

NFS ROOT文件系统（主要用于文件系统调试）

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
```

挂载**NFS ROOT**文件系统

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 ROOT文件系统。　　先断电，然后修改启动方式跳线帽为nandflash方式启动，nandflash上必须要先烧写uboot。拷贝zImage和

dtb文件到/tftpboot目录，如下：

```
cp <WORKDIR>/Filesystem/myir-buildroot/output/images/zImage <WOR
KDIR>/tftpboot
cp <WORKDIR>/Filesystem/myir-buildroot/output/images/myd_c335x.d
tb <WORKDIR>/tftpboot
```

解压rootfs.tar.gz到/home/myir/rootfs目录，如下：

```
cd /home/myir/rootfs
sudo tar -xvf <WORKDIR>/Filesystem/myir-buildroot/output/images/
rootfs.tar.gz ./
```

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
>#  setenv serverip 192.168.1.111
```

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
```

设置开发板nfs挂载目录和设备树文件名，如下：

```
>#  setenv rootpath /home/myir/rootfs    -- 使用该方式前将rootfs.ta
r.gz解压到该nfs目录
>#  setenv fdtfile myd_c335x.dtb         -- 这里以MYD-AM335X板为例
，使用不同的板型要用对应的dtb文件
>#  echo $fdtfile                        -- 查看fdtfile名称是否正确
```

保存相关环境变量.如下：

```
>#  saveenv
```

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

```
>#  run netboot
```

如果ping serverip正常，但运行run netboot时tftp失败，请检查server端tftp service是否正常，如下：

```
$ tftp 192.168.1.111
tftp> get myd_c335x.dtb
tftp> quit                                        -- 退出tftp
```

如果失败，重启tftp service，如下：

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

7.外设模块使用

本章节主要讲述米尔外设模块在开发板上的软件硬件环境搭建及测试过程。

MYD-AM335X系列开发板的外设模块支持情况，如下表所示：

模块名称	描述	**MYD-AM335X**	**MYD-AM335X-Y**	**MYD-AM335X-J**	参考章节
MY-TFT043RV2	4.3寸电阻屏V2	√	√	√	7.1 4.3寸电阻屏
MY-TFT070CV2	7寸电容屏V2	√	√	√	4.1 LCD测试
MY-TFT070RV2	7寸电阻屏V2	√	√	√	4.1 LCD测试
MY-WF003U	USB WIFI模块	√	√	√	7.2 WIFI模块
MY-GPS008C	GPS模块	√	√	√	7.4 GPS模块
MY-CAM002U	USB摄像头模块	√	√	√	7.3 USB摄像头模块
MY-GPRS007C	GPRS模块	√	√	√	7.5 GPRS模块
MY-WF004S	SDIO WIFI模块	√	√	√	7.2 WIFI模块

MY-CAM011B	数字摄像头模块	×	×	×	无

7.1 4.3寸电阻屏

本章讲述4.3寸电阻屏在开发板的上的环境搭建和测试过程。

硬件环境：

- MYD-AM335X系列开发板一块
- MY-TFT043RV2 连接MYD-AM335X系列开发板的LCD接口
- USB转TTL调试串口一根，连接MYD-AM335X系列开发板 Debug串口和PC, PC端波特率设置115200-8-n-1

接口	**MYD-AM335X**	**MYD-AM335X-Y**	**MYD-AM335X-J**
Debug串口	J12 UART0	J10 Debug UART	J3 USB_UART
LCD 接口	J8	J7	J8

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-AM335X	myd_c335x.dtb
MYD-AM335X-Y	myd_y335x.dtb
MYD-AM335X-J	myd_j335x.dtb

出厂软件默认是支持7寸电容和电阻屏直接显示的，现在要使用4.3寸电阻屏，需要修改设备树来进行支持。

设备树中关于4.3寸屏的配置默认是注释的，在设备树中搜索*/* 4.3 inch, 480x272 resolution LCD, MYiR */*这个关键字释放被注释的代码，同时将7寸屏的配置注释了，修改完成后使用命令make dtbs来编译设备树，然后拷贝

编译好的设备到SD卡中，使用第6章的方法来更新开发板的软件。

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:480>>> yres:272 >>> 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480X272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 di
splaying the image
--donthide    | -u : Do not hide the cursor before and after dis
playing the image
--noinfo      | -i : Supress image information
```

```
--stretch     | -f : Strech (using a simple resizing routine) th
e image to fit
                                                      ontoscreen if
 necessary
--colorstretch| -k : Strech (using a 'color average' resizing ro
utine) the image
                                                   to fit onto scre
en if necessary
--enlarge     | -e : Enlarge the image to fit the whole screen i
f 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
480 x 272
```

程序执行完毕，图片完整显示在LCD上。

7.2 WIFI 模块

本章讲述USB WIFI和SIDO WIFI在开发板的上的环境搭建和测试过程。

硬件环境：

- MYD-AM335X系列开发板一块
- MY-WF003U 模块、MY-WF004S 模块
- USB转TTL调试串口一根，连接MYD-AM335X系列开发板 Debug串口和PC, PC端波特率设置115200-8-n-1

接口	**MYD-AM335X**	**MYD-AM335X-Y**	**MYD-AM335X-J**
Debug串口	J12 UART0	J10 Debug UART	J3 USB_UART
USB接口	J3 J4	J19 USB_HOST	J27 USB_HOST
SDIO接口	J17 TF_CARD	J12	J19 Micro_SD

软件环境：

- Linux Kernel 4.1.18
- wpa_supplicant 应用程序
- hostapd 应用程序
- iptables 应用程序

> 注意：包含**MEASY-HMI**的系统中**connman**与**network manager**冲突，如要测试**WIFI,**需先禁用**connman**。可以将**/etc/init.d/S45connman**重命名为**/etc/init.d/K45connman**之后重启系统。

开发板	设备树
MYD-AM335X	myd_c335x.dtb
MYD-AM335X-Y	myd_y335x.dtb

MYD-AM335X-J	myd_j335x.dtb

测试过程：

将MY-WF003U 模块插入开发板USB接口或者MY-WF004S 模块插入开发板TF卡接口，使用命令 `ifconfig -a` 即可看到模块已加载驱动并生成了WLAN设备。

```
# ifconfig -a


eth0      Link encap:Ethernet  HWaddr 68:9E:19:BC:1C:84
          UP BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
          Interrupt:240

eth1      Link encap:Ethernet  HWaddr 68:9E:19:BC:1C:86
          BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

wlan0     Link encap:Ethernet  HWaddr 00:1D:43:A0:04:11
          BROADCAST MULTICAST  MTU:1500  Metric:1
```

```
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
```

WIFI STA模式

1.修改*/etc/wap_supplicant.conf*配置文件中ssid和psk的值，ssid为测试环境中的wifi ap名称，psk为测试环境中wifi ap的密码。

```
# cat /etc/wpa_supplicant.conf
ctrl_interface=/var/run/wpa_supplicant
ap_scan=1
p2p_disabled=1

network={
 ssid="MYIR_TECH"
 scan_ssid=1
 proto=WPA RSN
 pairwise=CCMP TKIP NONE
 key_mgmt=WPA-EAP WPA-PSK IEEE8021X NONE
 group=TKIP CCMP
 psk="myir2016"
 priority=10
}
```

2.连接wifi ap。

```
# wpa_supplicant -i wlan0 -c /etc/wpa_supplicant.conf -B
```

3.获取IP地址及DNS。

```
#udhcpc -b -i wlan0
udhcpc: started, v1.25.1
udhcpc: sending discover
udhcpc: sending select for 192.168.30.115
```

```
udhcpc: lease of 192.168.30.115 obtained, lease time 3600
deleting routers
adding dns 223.5.5.5
adding dns 201.104.111.114
```

4.ping测试。

```
# ping www.baidu.com
PING www.baidu.com (61.135.169.125): 56 data bytes
64 bytes from 61.135.169.125: seq=0 ttl=56 time=29.253 ms
64 bytes from 61.135.169.125: seq=1 ttl=56 time=32.218 ms
64 bytes from 61.135.169.125: seq=2 ttl=56 time=24.717 ms
64 bytes from 61.135.169.125: seq=4 ttl=56 time=141.210 ms
64 bytes from 61.135.169.125: seq=5 ttl=56 time=64.064 ms
64 bytes from 61.135.169.125: seq=11 ttl=56 time=133.112 ms
```

WIFI SoftAP模式

1.修改*/etc/hostapd.conf*配置文件中ssid和wpa_passphrase的值，ssid为wifi模块生成的ap热点的名称，wpa_passphrase为ap热点的密码。

```
# cat /etc/hostapd.conf
interface=wlan0
ssid=myirAP
hw_mode=g
channel=6
macaddr_acl=0
auth_algs=1
ignore_broadcast_ssid=0
wpa=2
wpa_passphrase=123456789
wpa_key_mgmt=WPA-PSK
wpa_pairwise=TKIP
rsn_pairwise=CCMP
```

2.根据需求修改/etc/dhcp/dhcpd.conf 配置文件中所内容，这个配置文件主要是为连上wifi ap设备提供DHCP服务。

```
# cat /etc/dhcp/dhcpd.conf
ddns-update-style interim;
ignore client-updates;

subnet 192.168.43.0 netmask 255.255.255.0{
range 192.168.43.100 192.168.43.150;
default-lease-time 86400;
max-lease-time 86400;
option routers 192.168.43.1;
option broadcast-address 192.168.43.255;
option subnet-mask 255.255.255.0;
option domain-name "redpinesignals.com";
option domain-name-servers 114.114.114.114;
host VAP_0.redpinesignals.com{
        hardware ethernet 00:23:a7:3a:11:d1;
        fixed-address 192.168.43.1;
        }
}
```

3.创建ap热点。

```
# hostapd /etc/hostapd.conf -B
Configuration file: /etc/hostapd.conf
Using interface wlan0 with hwaddr 00:1d:43:a0:04:11 and ssid "my
irAP"
random: Only 15/20 bytes of strong random data available from /d
ev/random
random: Not enough entropy pool available for secure operations
WPA: Not enough entropy in random pool for secure operations - u
pdate keys later when the
first station connects
wlan0: interface state UNINITIALIZED->ENABLED
wlan0: AP-ENABLED
```

4.配置ap热点的ip地址和子网掩码，这个IP地址需要根据/etc/dhcp/dhcpd.conf中routers的值来进行设置。

```
# ifconfig wlan0 192.168.43.1 netmask 255.255.255.0 up
```

5.启动wlan0的DHCP服务。

```
# dhcpd wlan0
Internet Systems Consortium DHCP Server 4.3.5
Copyright 2004-2016 Internet Systems Consortium.
All rights reserved.
For info, please visit https://www.isc.org/software/dhcp/
WARNING: Host declarations are global.  They are not limited to
the scope you declared
them in.
Config file: /etc/dhcp/dhcpd.conf
Database file: /var/lib/dhcp/dhcpd.leases
PID file: /var/run/dhcpd.pid
Wrote 0 deleted host decls to leases file.
Wrote 0 new dynamic host decls to leases file.
Wrote 0 leases to leases file.
Listening on LPF/wlan0/00:1d:43:a0:04:11/192.168.43.0/24
Sending on   LPF/wlan0/00:1d:43:a0:04:11/192.168.43.0/24
Sending on   Socket/fallback/fallback-net
```

完成上面五个步骤已经可以使用其他WIFI STA模式的设备来连接所创建的AP热点了，不过只能进行局域网通信，如需进行外网通信还要完成下面步骤。

1.将eth0接入互联网络，并获取IP地址。

```
# udhcpc eth0
udhcpc: started, v1.25.1
```

```
udhcpc: sending discover
udhcpc: sending select for 192.168.30.160
udhcpc: lease of 192.168.30.160 obtained, lease time 3600
deleting routers
adding dns 223.5.5.5
adding dns 201.104.111.114
```

2.打开IP报文转发。

```
echo 1 > /proc/sys/net/ipv4/ip_forward
```

3.配置eth0为所有报文的出口。

```
iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
```

执行完上述3个步骤，WIFI设备就可通过这个AP热点访问互联网了。

7.3 USB摄像头模块

本章讲述USB 摄像头在开发板的上的环境搭建和测试过程。

硬件环境：

- MYD-AM335X系列开发板一块
- MY-CAM002U 模块
- USB转TTL调试串口一根，连接MYD-AM335X系列开发板 Debug串口和PC, PC端波特率设置115200-8-n-1

接口	**MYD-AM335X**	**MYD-AM335X-Y**	**MYD-AM335X-J**
Debug串口	J12 UART0	J10 Debug UART	J3 USB_UART
USB接口	J3 J4	J19 USB_HOST	J27 USB_HOST

软件环境：

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

开发板	设备树
MYD-AM335X	myd_c335x.dtb
MYD-AM335X-Y	myd_y335x.dtb
MYD-AM335X-J	myd_j335x.dtb

测试过程：

- 将摄像头模块插入开发板的USB接口，系统会自动匹配驱动并在创建设备头设备/dev/video0。
- 使用测试程序fswebcam来进行拍照。

```
fswebcam -d /dev/video0 --no-banner -r 640x480 image.jpg
```

- 拷贝image.jpg到PC上来显示，如果开发板接了LCD屏也可以直接使用命令来将拍摄的照片输出到LCD上进行显示。

```
# fbv image.jpg
fbv - The Framebuffer Viewer
image.jpg
640 x 480
```

7.4 GPS模块

本章讲述GPS模块在开发板的上的环境搭建和测试过程。

硬件环境：

- MYD-AM335X系列开发板一块
- MY-GPS008C 模块
- USB转TTL调试串口一根，连接MYD-AM335X系列开发板 Debug串口和PC, PC端波特率设置115200-8-n-1

接口	**MYD-AM335X**	**MYD-AM335X-Y**	**MYD-AM335X-J**
Debug串口	J12 UART0	J10 Debug UART	J3 USB_UART
扩展接口	J20	J13	J25

软件环境：

- Linux Kernel 4.1.18
- microcom应用程序

开发板	设备树
MYD-AM335X	myd_c335x.dtb
MYD-AM335X-Y	myd_y335x.dtb
MYD-AM335X-J	myd_j335x.dtb

测试过程：

- 如果开发板有扩展接口，直接将GPS模块与扩展接口对接，没有扩展接口的开发板直接使用开发板的空闲的串口来进行对接。
- 使用microcom工具来读取GPS模块返回的数据。

```
# microcom -s 9600 /dev/ttyO1
$GPGSA,A,2,01,04,11,17,28,,,,,,,,2.8,2.7,0.9*3C
```

```
$GPGSV,3,1,09,01,59,035,53,03,31,127,21,04,30,037,50,06,06,219,3
3*75
$GPGSV,3,2,09,11,45,032,42,17,24,285,54,19,09,059,35,28,40,334,5
1*7C
$GPGSV,3,3,09,32,21,084,27*4B
$GPGGA,061102.0,2233.150278,N,11356.386896,E,1,05,2.7,83.5,M,-1.
0,M,,*79
$GPVTG,,T,0.0,M,0.0,N,0.0,K,A*0D
$GPRMC,061102.0,A,2233.150278,N,11356.386896,E,0.0,,230715,0.0,E
,A*2E
```

7.5 GPRS模块

本章讲述GPRS模块在开发板的上的环境搭建和测试过程。

硬件环境：

- MYD-AM335X系列开发板一块
- MY-GPRS007C 模块
- USB转TTL调试串口一根，连接MYD-AM335X系列开发板 Debug串口和PC, PC端波特率设置115200-8-n-1

接口	**MYD-AM335X**	**MYD-AM335X-Y**	**MYD-AM335X-J**
Debug串口	J12 UART0	J10 Debug UART	J3 USB_UART
扩展接口	J20	J13	J25

软件环境：

- Linux Kernel 4.1.18
- pppd 应用程序

开发板	设备树
MYD-AM335X	myd_c335x.dtb
MYD-AM335X-Y	myd_y335x.dtb
MYD-AM335X-J	myd_j335x.dtb

测试过程：

- 如果开发板有扩展接口，直接将GPRS模块与扩展接口对接，没有扩展接口的开发板直接使用开发板的空闲的串口来进行对接。
- 修改/etc/ppp/peers/gprs这个配置文件中/dev/ttyUSB1为当前所对接的串口设备，例如/dev/ttyO3。

```
#/etc/ppp/peers/gprs
# Usage:   root>pppd call gprs
/dev/ttyO3
115200
#crtscts
modem
noauth
debug
nodetach
#hide-password
usepeerdns
noipdefault
defaultroute
user "cmnet"
0.0.0.0:0.0.0.0
#ipcp-accept-local
#ipcp-accept-remote
#lcp-echo-failure  12
#lcp-echo-interval 3
noccp
#novj
#novjccomp
persist
connect '/usr/sbin/chat -s -v -f /etc/ppp/peers/gprs-connect-cha
t'
#connect '/bin/chat -v -s -f /etc/ppp/gprs-connect-chat'
#disconnect '/bin/chat -v -f /etc/ppp/gprs-disconnect-chat'
```

- 使用命令进行拨号连接。

```
#pppd call gprs &
```

- ping测试。

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

```
PING www.baidu.com (61.135.169.125): 56 data bytes
64 bytes from 61.135.169.125: seq=0 ttl=56 time=29.253 ms
64 bytes from 61.135.169.125: seq=1 ttl=56 time=32.218 ms
64 bytes from 61.135.169.125: seq=2 ttl=56 time=24.717 ms
64 bytes from 61.135.169.125: seq=4 ttl=56 time=141.210 ms
```

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
- 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302

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：北京市大兴区荣华中路8号院力宝广场10号楼901室

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。