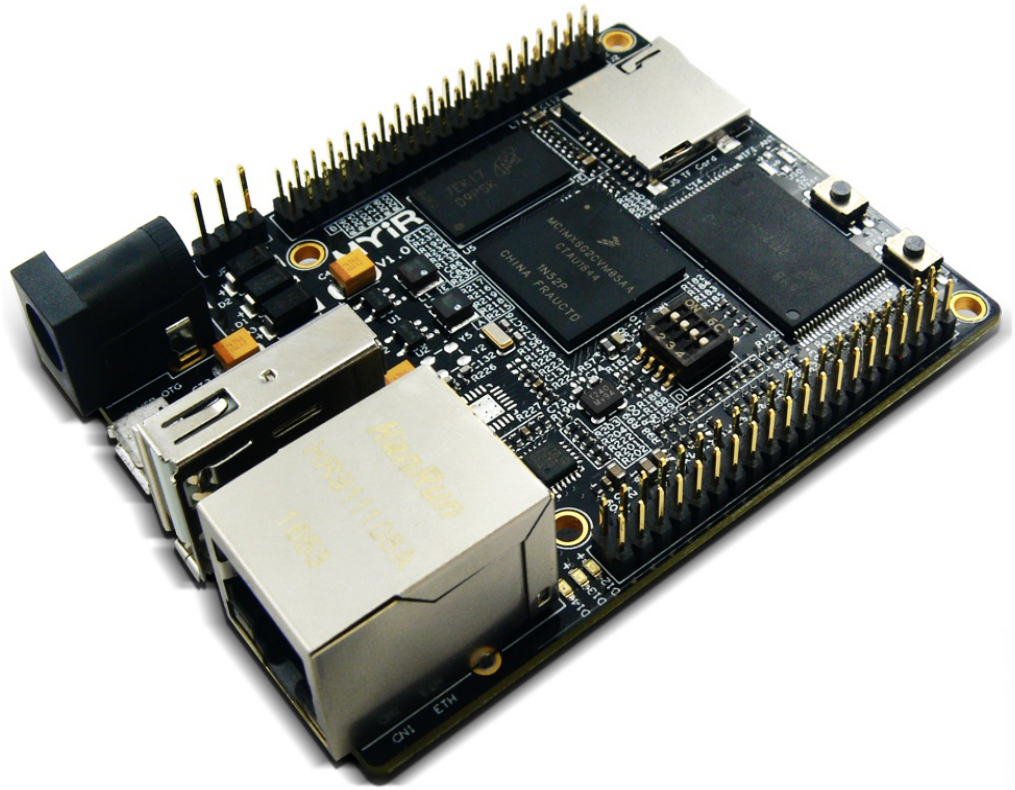
MYS-6ULX Linux 开发手册

MYiR™ Make Your Idea Real



目錄

前言 0
1 软件资源介绍 1
2 部署开发环境 2
3 构建系统 3
    3.1 U-Boot 3.1
    3.2 Linux Kernel 3.2
    3.3 构建文件系统 3.3
        3.3.1 Yocto构建Linux系统 3.3.1
        3.3.2 Yocto构建SDK工具 3.3.2
4 Linux应用开发 4
    4.1 LCD测试 4.1
    4.2 触摸测试 4.2
    4.3 Ethernet测试 4.3
    4.4 GPIO-KEY测试 4.4
    4.5 GPIO-LED测试 4.5
    4.6 USB Host测试 4.6
    4.7 USB Device测试 4.7
    4.8 RS485测试 4.8
    4.9 CAN bus测试 4.9
    4.10 Audio测试 4.10
    4.11 Camera测试 4.11
    4.12 WiFi测试 4.12
5 QT应用开发 5
    5.1 安装QtCreator 5.1
    5.2 配置QtCreator 5.2
    5.3 测试Qt应用 5.3
6 系统更新 6
附录A 7
际录B 8

MYS-6ULX Linux开发手册

本文档介绍基于MYS-6ULX系列开发板和MYB-6ULX扩展板，进行Linux系统的编译和安装，硬件接口的使用，Qt应用开发等。

历史版本

版本号	描述	时间
V1.0	初始版本	2017.04.20
V1.1	添加MYB-6ULX支持	2017.07.11
V1.2	添加LCD 7.0吋屏幕的支持	2017.08.03

硬件版本

本手册适合于以下型号的开发板：

- MYS-6ULX-IND
- MYS-6ULX-IoT
- MYB-6ULX

1 软件资源

MYS-6ULX搭载基于Linux 4.1.15内核的操作系统，提供了丰富的系统资源和软件资源。部分资源需要配合MYB-6ULX扩展板才能使用。以下是软件资源列表：

类别	名称	描述信息	源码
引导程序	U-boot	u-boot.imx引导启动程序	YES
Linux内核	Linux 4.1.15	基于官方imx_4.1.15_2.0.0_ga版本	YES
设备驱动	USB Host	USB Host驱动	YES
设备驱动	USB OTG	USB OTG驱动	YES
设备驱动	I2C	I2C总线驱动	YES
设备驱动	Ethernet	10/100Mbps以太网驱动	YES
设备驱动	MMC	MMC/eMMC/TF卡存储驱动	YES
设备驱动	LCD	显示驱动，支持4寸和7寸液晶屏	YES
设备驱动	RTC	实时时钟驱动	YES
设备驱动	Touch	电容(FT5316)，电阻触摸	YES
设备驱动	USART	串口驱动	YES
设备驱动	LED	GPIO LED驱动	YES
设备驱动	KEY	GPIO KEY驱动	YES
设备驱动	Audio	WM8904驱动	YES
设备驱动	CAN bus	CAN总线驱动	YES
设备驱动	RS485	RS485总线驱动	YES
设备驱动	Camera	ov2659驱动	YES
文件系统	Debian rootfs	基于Debian构建带X11的文件系统	BINARY
文件系统	Yotcto rootfs	基于Yocto构建带Qt 5.6的文件系统	YES
文件系统	Yotcto rootfs	基于Yocto构建终端型的通用文件系统	YES
应用程序	GPIO KEY	GPIO按键指示例程	YES
应用程序	GPIO LED	按键指示灯例程	YES
应用程序	NET	TCP/IP Sokect C/S例程	YES
应用程序	RTC	实时时钟例程	YES
应用程序	RS232	RS232例程	YES
应用程序	Audio	Audio例程	YES
应用程序	LCD	显示屏例程	YES

编译工具链	Cross compiler	Linaro GCC 4.9 Hardfloat	BINARY
编译工具链	Cross compiler	Yocto GCC 5.3 Hardfloat	BINARY

表1-1软件资源列表

2 部署开发环境

开发前需要安装好Linux操作系统，推荐使用Ubuntu 16.04 64bit发行版，连接网线并配置好网络，后续操作需要连接互联网安装或下载相关软件包。

开发板与计算机连接

1. 计算机使用USB转TTL串口转接线与开发板的DEBUG串口(JP1)连接
2. 运行串口调试应用程序，并选择对应的串口设备

计算机端的串口配置参数如下：

- 波特率：115200
- 数据位: 8bit
- 校验方式：None
- 停止位：1bit
- 流控：Disable

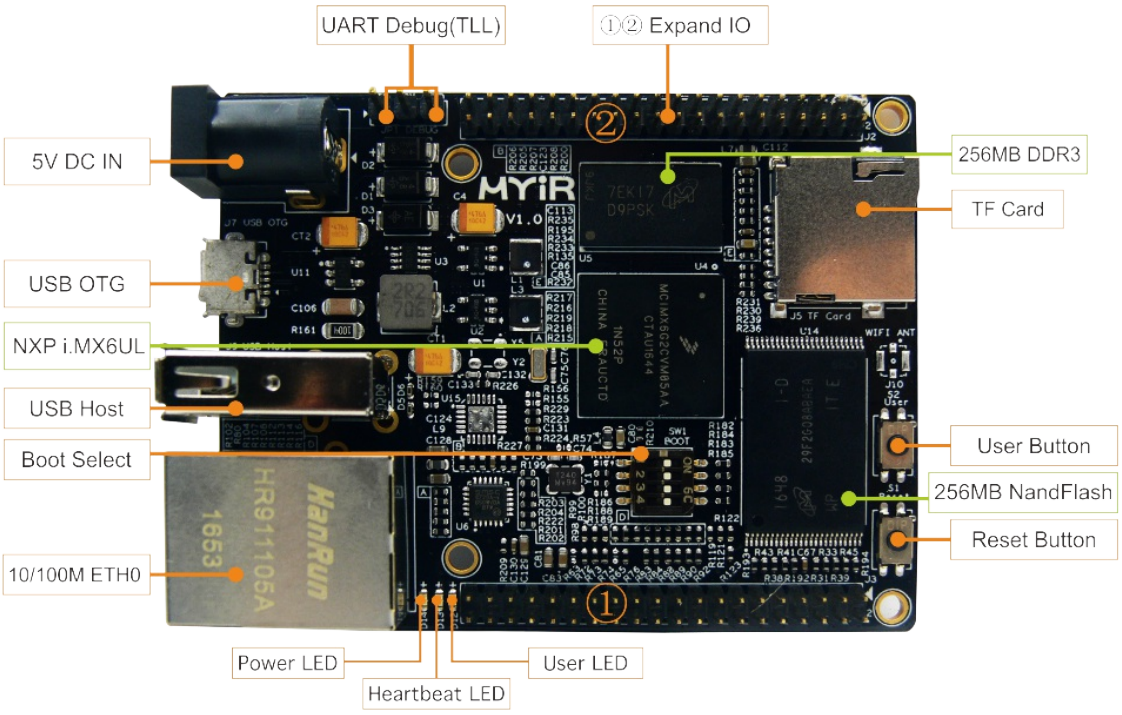


图2-1 MYS-6ULX-IND接口定义

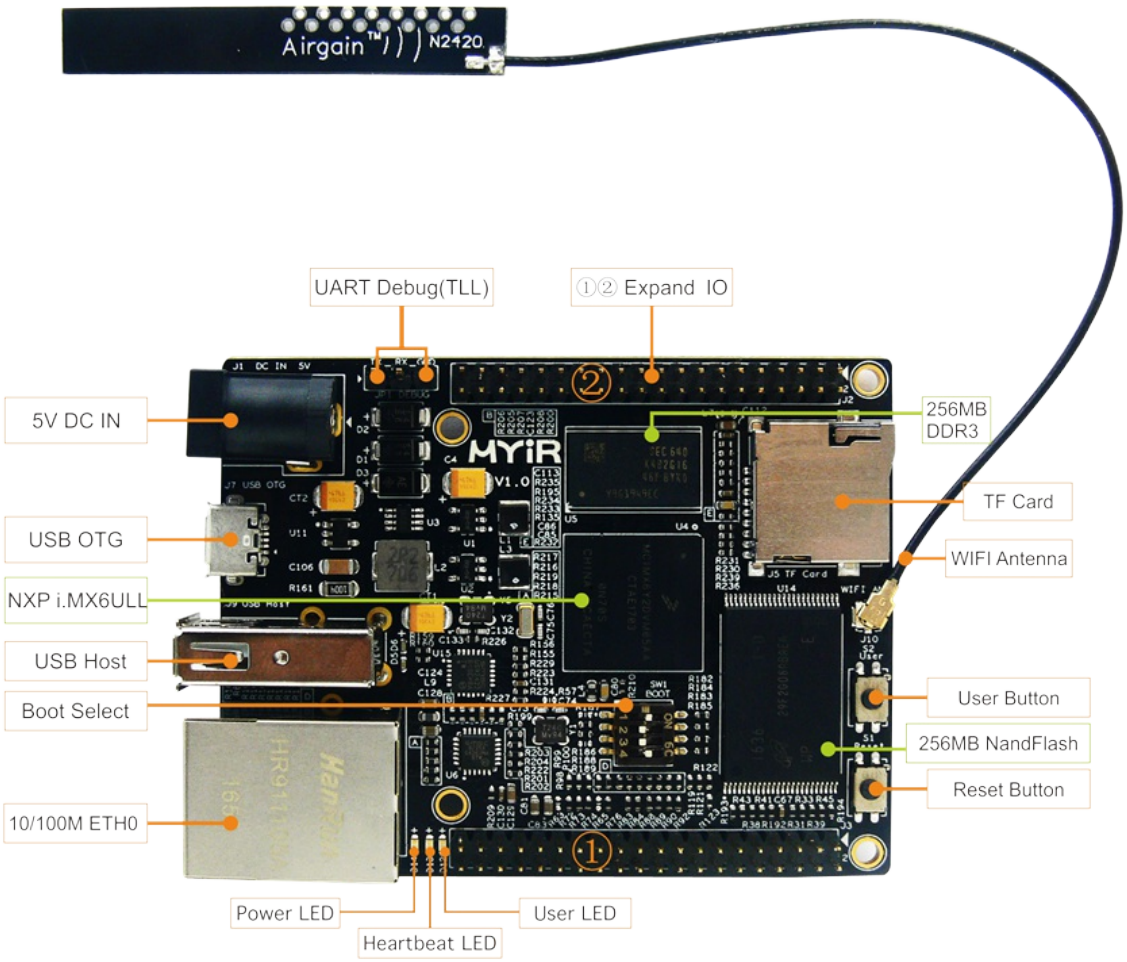


图2-2 MYS-6ULX-IoT接口定义

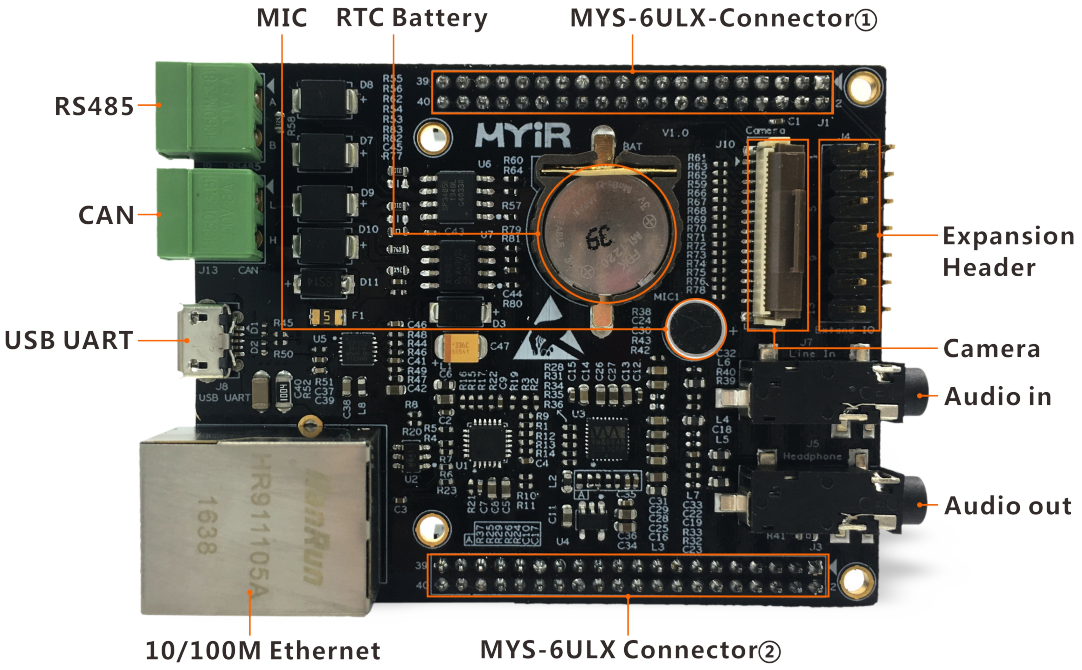


图2-3 MYB-6ULX接口定义

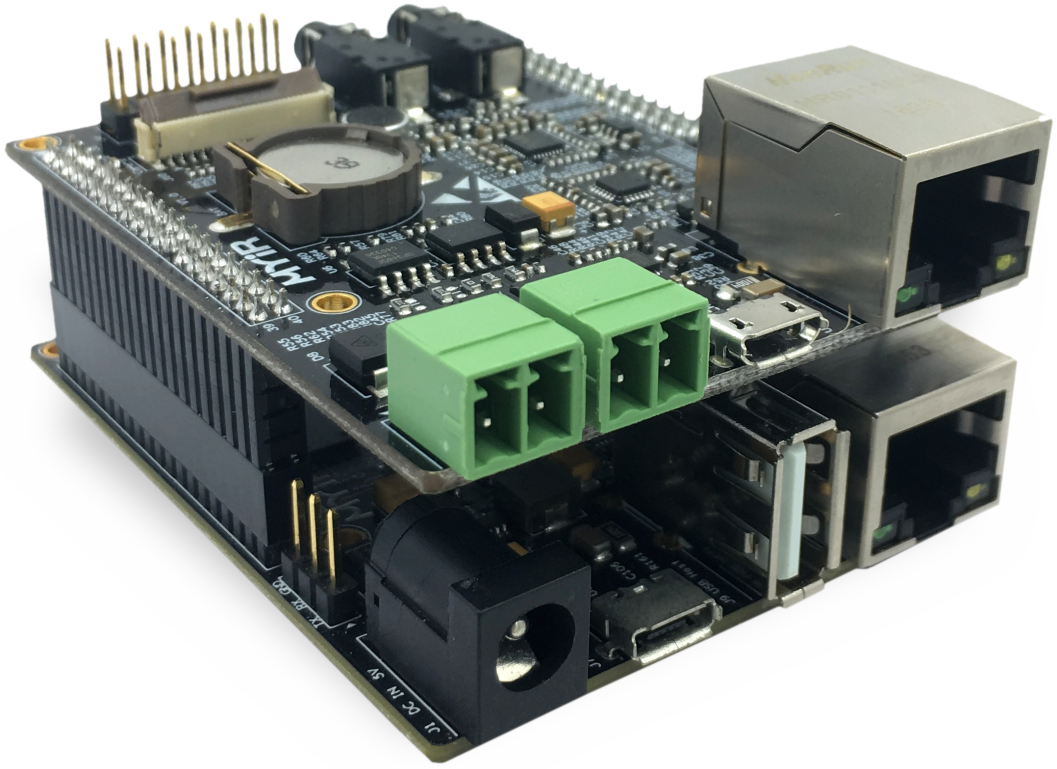


图2-4 MYS-6ULX安装MYB-6ULX侧面图

安装必备软件包

```
sudo apt-get install build-essential git-core libncurses5-dev \
flex bison texinfo zip unzip zlib1g-dev gettext u-boot-tools \
g++ xz-utils mtd-utils gawk diffstat gcc-multilib python git \
make gcc g++ diffstat bzip2 gawk chrpath wget cpio texinfo lzop
```

建立工作目录

建立工作目录，方便设置统一的环境变量路径。拷贝产品光盘中的源码到工作目录下，同时设置DEV_ROOT变量，方便后续步骤的路径访问。

```
mkdir -p ~/MYS6ULx-devel
export DEV_ROOT=~/MYS6ULx-devel
cp -r <DVDROM>/02-Images $DEV_ROOT
cp -r <DVDROM>/03-Tools $DEV_ROOT
cp -r <DVDROM>/04-Source $DEV_ROOT
```

配置编译工具

- Linaro交叉编译器: gcc version 4.9.3 20141031 (prerelease) (Linaro GCC 2014.11)
- Yocto交叉编译器: gcc version 5.3.0 (GCC)

这里有两个编译器，一个是Linaro提供，另一个是由Yocto构建的，建议使用Yocto提供的，以便和文件系统统一。

Linaro编译器

```
cd $DEV_ROOT
tar -xvf 03-Tools/Toolchain/\
gcc-linaro-4.9-2014.11-x86_64_arm-linux-gnueabihf.tar.xz
export PATH=$PATH:$DEV_ROOT/\
gcc-linaro-4.9-2014.11-x86_64_arm-linux-gnueabihf/bin
export CROSS_COMPILE=arm-linux-gnueabihf-
export ARCH=arm
```

执行完上述命令后输入"arm-linux-gnueabihf-gcc -v"，若有输出版本信息，说明设置成功，以上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。

```
$ arm-linux-gnueabihf-gcc -v
Using built-in specs.
COLLECT_GCC=arm-linux-gnueabihf-gcc
COLLECT_LTO_WRAPPER=/home/kevinchen/MYS6ULx/\
gcc-linaro-4.9-2014.11-x86_64_arm-linux-gnueabihf/bin/../
libexec/gcc/arm-linux-gnueabihf/4.9.3/lto-wrapper
Target: arm-linux-gnueabihf
Configured with: /home/buildslave/workspace/BinaryRelease/label/
x86_64/target/arm-linux-gnueabihf/snapshots/
gcc-linaro-4.9-2014.11/configure SHELL=/bin/bash --with-bugurl=
https://bugs.linaro.org
--with-mpc=/home/buildslave/workspace/BinaryRelease/label/x86_64
/target/arm-linux-gnueabihf/_build/builds/destdir/x86_64-unknown
-linux-gnu --with-mpfr=/home/buildslave/workspace/
BinaryRelease/label/x86_64/target/arm-linux-gnueabihf/_build/bui
lds/destdir/x86_64-unknown-linux-gnu
--with-gmp=/home/buildslave/workspace/BinaryRelease/label/x86_64
/target/arm-linux-gnueabihf/_build/builds/destdir/x86_64-unknown
-linux-gnu --with-gnu-as --with-gnu-ld --disable-libstdcxx-pch -
-disable-libmudflap --with-cloog=no --with-ppl=no --with-isl=no
```

```
--disable-nls --enable-multiarch --disable-multilib --enable-c99
--with-tune=cortex-a9 --with-arch=armv7-a --with-fpu=vfpv3-d16
--with-float=hard --with-mode=thumb --disable-shared --enable-
static --with-build-sysroot=/home/buildslave/workspace/BinaryRel
ease/label/x86_64/target/arm-linux-gnueabihf/_build/sysroots/arm
-linux-gnueabihf --enable-lto --enable-linker-build-id --enable-
long-long --enable-shared --with-sysroot=/home/buildslave/worksp
ace/BinaryRelease/label/x86_64arm-linux-gnueabihf/_build/builds/
destdir/x86_64-unknown-linux-gnu/libc --enable-languages=c,c++,
fortran,lto -enable-fix-cortex-a53-835769 --enable-checking=rele
ase --with-bugurl=https://bugs.linaro.org
--with-pkgversion='Linaro GCC 2014.11' --build=x86_64-unknown-li
nux-gnu --host=x86_64-unknown-linux-gnu --target=arm-linux-gnuea
bihf
--prefix=/home/buildslave/workspace/BinaryRelease/label/x86_64/t
arget/arm-linux-gnueabihf/_build/builds/destdir/x86_64-unknown-l
inux-gnu
Thread model: posix
gcc version 4.9.3 20141031 (prerelease) (Linaro GCC 2014.11)
```

Yocto编译工具链

Yocto提供的工具链有两种，一种是底层开发的meta-toolchain，另一种是用于应用开发的工具链。前者和Linaro类似，后者包含应用开发中的相关库，可以直接使用pkg-config工具来解决头文件或库文件的依赖关系。MYS-6ULX的资源包中有提供两种工具链。

工具链文件名	描述
myir-imx-fb-glibc-x86_64-fsl-image-qt5-cortexa7hf-neon-toolchain-4.1.15-2.0.1.sh	包含Qt5相关库的应用工具链
myir-imx-fb-glibc-x86_64-core-image-base-cortexa7hf-neon-toolchain-4.1.15-2.0.1.sh	不包含Qt5的应用工具链
myir-imx-fb-glibc-x86_64-meta-toolchain-cortexa7hf-neon-toolchain-4.1.15-2.0.1.sh	meta-toolchain

Yocto编译器是以SDK工具包方式来提供，需要先安装SDK包后，才可以使用。安装方法如下：

以普通用户权限执行shell脚本，运行中会提示安装路径，默认在/opt目录下，同时会提示输入用户密码以便有写入目录的权限。安装完成后，可以使用"source"或"."命令加载工链接环境到当前终端。

例子把应用开发工具链安装在了/opt/myir-imx6ulx-fb/4.1.15-2.0.1目录下。

```
./myir-imx-fb-glibc-x86_64-fsl-image-qt5-cortexa7hf-\
neon-toolchain-4.1.15-2.0.1.sh
Freescale i.MX Release Distro SDK installer version 4.1.15-2.0.1
==========================================================================
Enter target directory for SDK (default: /opt/myir-imx-fb/4.1.15
-2.0.1):
/opt/myir-imx6ulx-fb/4.1.15-2.0.1
Do You are about to install the SDK to "/opt/myir-imx6ulx-fb/4.1
.15-2.0.1". Proceed[Y/n]? Y
[sudo] password for kevinchen:
Extracting SDK.....................................................
.....................................................................
..............done
Setting it up...done
SDK has been successfully set up and is ready to be used.
Each time you wish to use the SDK in a new shell session, you ne
ed to source the environment setup script e.g.
```

```
. /opt/myir-imx6ulx-fb/4.1.15-2.0.1/environment-setup-cortexa7hf\
-neon-poky-linux-gnueabi
```

验证SDK工具链是否安装正确，先使用"source"命令加载Yocto的环境配置文件，然后查看编译器版本。

```
source /opt/myir-imx6ulx-fb/4.1.15-2.0.1/environment-setup-corte
xa7hf-neon-poky-linux-gnueabi
arm-poky-linux-gnueabi-gcc -v
Using built-in specs.
COLLECT_GCC=arm-poky-linux-gnueabi-gcc
COLLECT_LTO_WRAPPER=/opt/myir-imx6ulx-fb/4.1.15-2.0.1/sysroots/x
86_64-pokysdk-linux/usr/libexec/arm-poky-linux-gnueabi/gcc/arm-p
oky-linux-gnueabi/5.3.0/lto-wrapper
Target: arm-poky-linux-gnueabi
Configured with: ../../../../../../work-shared/gcc-5.3.0-r0/gcc-
5.3.0/configure --build=x86_64-linux --host=x86_64-pokysdk-linux
--target=arm-poky-linux-gnueabi --prefix=/opt/myir-imx6ulx-fb/4.
1.15-2.0.1/sysroots/x86_64-pokysdk-linux/usr --exec_prefix=/opt/
myir-imx6ulx-fb/4.1.15-2.0.1/sysroots/x86_64-pokysdk-linux/usr
--bindir=/opt/myir-imx6ulx-fb/4.1.15-2.0.1/sysroots/x86_64-pokys
dk-linux/usr/bin/arm-poky-linux-gnueabi
--sbindir=/opt/myir-imx6ulx-fb/4.1.15-2.0.1/sysroots/x86_64-poky
sdk-linux/usr/bin/arm-poky-linux-gnueabi --libexecdir=/opt/myir-
imx6ulx-fb/4.1.15-2.0.1/sysroots/x86_64-pokysdk-linux/usr/libexe
c/arm-poky-linux-gnueabi --datadir=/opt/myir-imx6ulx-fb/4.1.15-2
.0.1/sysroots/x86_64-pokysdk-linux/usr/share --sysconfdir=/opt/m
yir-imx6ulx-fb/4.1.15-2.0.1/sysroots/x86_64-pokysdk-linux/etc
--sharedstatedir=/opt/myir-imx6ulx-fb/4.1.15-2.0.1/sysroots/x86_
64-pokysdk-linux/com
--localstatedir=/opt/myir-imx6ulx-fb/4.1.15-2.0.1/sysroots/x86_6
4-pokysdk-linux/var --libdir=/opt/myir-imx6ulx-fb/4.1.15-2.0.1/s
ysroots/x86_64-pokysdk-linux/usr/lib/
arm-poky-linux-gnueabi --includedir=/opt/myir-imx6ulx-fb/4.1.15-
2.0.1/sysroots/x86_64-pokysdk-linux/usr/include --oldincludedir=
/opt/myir-imx6ulx-fb/4.1.15-2.0.1/sysroots/x86_64-pokysdk-linux/
usr/include --infodir=/opt/myir-imx6ulx-fb/4.1.15-2.0.1/
sysroots/x86_64-pokysdk-linux/usr/share/info
--mandir=/opt/myir-imx6ulx-fb/4.1.15-2.0.1/sysroots/x86_64-pokys
dk-linux/usr/share/man
--disable-silent-rules --disable-dependency-tracking
--with-libtool-sysroot=/home/kevinchen/mys-imx6ul/fsl-release-yo
cto/build/tmp/sysroots/x86_64-nativesdk-pokysdk-linux --with-gnu
-ld --enable-shared
--enable-languages=c,c++ --enable-threads=posix --enable-multili
b --enable-c99
--enable-long-long --enable-symvers=gnu --enable-libstdcxx-pch \
--program-prefix=arm-poky-linux-gnueabi- --without-local-prefix
--enable-lto --enable-libssp
--enable-libitm --disable-bootstrap --disable-libmudflap --with-
system-zlib
--with-linker-hash-style=gnu --enable-linker-build-id --with-ppl
=no --with-cloog=no
--enable-checking=release --enable-cheaders=c_global --without-i
sl
--with-gxx-include-dir=/not/exist/usr/include/c++/5.3.0
--with-build-time-tools=/home/kevinchen/mys-imx6ul/fsl-release-y
octo/build/tmp/sysroots/x86_64-linux/usr/arm-poky-linux-gnueabi/
bin --with-sysroot=/not/exist
--with-build-sysroot=/home/kevinchen/mys-imx6ul/fsl-release-yoct
o/build/tmp/sysroots/mys6ul14x14
--enable-poison-system-directories --with-mpfr=/home/kevinchen/m
ys-imx6ul/fsl-release-yocto/build/tmp/sysroots/x86_64-nativesdk-
pokysdk-linux --with-mpc=/home/kevinchen/mys-imx6ul/fsl-release-
```

```
yocto/build/tmp/sysroots/
x86_64-nativesdk-pokysdk-linux --enable-nls --with-arch=armv7-a
Thread model: posix
gcc version 5.3.0 (GCC)
```

同样方法请自行安装底层开发的工具链meta-toolchain。安装两个工具链，请指定不同目录，请勿使用相同目录，出现文件相互覆盖情形。

3 构建系统

本章主要介绍MYS-6ULX开发板上, Linux操作系统相关部件的编译和使用。MYS-6ULX的Linux系统包含以下部件:

- U-Boot: 引导程序，支持不同方式启动内核。
- Linux Kernel: 适用于MYS-6ULX开发板的Linux 4.1.15内核，同时包含支持板载外设的驱动。
- Yocto: 一个开源协作项目，提供丰富的模板、工具和方法来支持构建出面向嵌入式产品的自定义Linux系统。

本章中用到的代码存放在资源包04-Source目录下，部分软件包的文件名中有标识号，后文不再特别说明。

编前u-boot和Linux内核代码前，请先安装meta-toolchain并加载环境变量到当前shell。

3.1 编译U-Boot

进入Bootloader目录，解压U-boot源码：

```
cd $DEV_ROOT/04-Source/
tar -xvf MYiR-iMX-uboot.tar.gz
cd MYiR-iMX-uboot
```

开始编译：

```
make distclean
make <config>
make
```

这里的是配置选项名称，不同的启动模式需使用不同的配置选项，MYS-6ULX开发板四种选项：

启动模式	编译选项
MYS-6ULX-IND NAND Flash	mys_imx6ul_14x14_nand_defconfig
MYS-6ULX-IND eMMC Flash	mys_imx6ul_14x14_emmc_defconfig
MYS-6ULX-IoT NAND Flash	mys_imx6ull_14x14_nand_defconfig
MYS-6ULX-IoT eMMC Flash	mys_imx6ull_14x14_emmc_defconfig

u-boot启动时默认会先检测"boot.scr"文件，这个是u-boot上的脚本镜像文件，用于临时改变启动设备或从Micro SD启动时，可以使用此文件。以下是"mys-imx6ul-boot-sdcard.txt"文件生成"boot.scr"文件的例子，mkimage工具是在u-boot的tools目录下，u-boot编译完成后，mkimage也会被编译出来，直接使用即可。

```
cat mys-imx6ul-boot-sdcard.txt
setenv mmcroot '/dev/mmcblk0p2 rootwait rw rootdelay=5 mem=256M'
setenv mmcargs 'setenv bootargs console=${console},${baudrate} \
root=${mmcroot} mtdparts=gpmi-nand:5m(boot),10m(kernel),1m(dtb),\
-(rootfs)'
run mmcargs
fatload mmc 0 0x83000000 zImage
fatload mmc 0 0x84000000 mys-imx6ul-14x14-evk-emmc.dtb
bootz 0x83000000 - 0x84000000

./tool/mkimage -A arm -T script -O linux -d \
mys-imx6ul-boot-sdcard.txt boot.scr
```

3.2 Linux Kernel

进入Kernel目录，解压内核源码：

```
cd $DEV_ROOT/Kernel
tar -xvf MYiR-iMX-Linux.tar.gz
cd MYiR-iMX-Linux
```

开始编译：

```
make distclean
make mys_imx6_defconfig
make zImage dtbs modules
```

编译完成后在"arch/arm/boot"目录会生成内核镜像文件zImage，在"arch/arm/boot/dts"目录会生成DTB文件。

DTB文件	备注
mys-imx6ul-14x14-evk-emmc.dtb	MYS-6ULX-IND eMMC
mys-imx6ul-14x14-evk-gpmi-weim.dtb	MYS-6ULX-IND NAND
mys-imx6ul-14x14-evk-emmc-myb6ulx.dtb	MYS-6ULX-IND eMMC 上安装MYB-6ULX扩展板
mys-imx6ul-14x14-evk-gpmi-weim-myb6ulx.dtb	MYS-6ULX-IND NAND 上安装MYB-6ULX扩展板
mys-imx6ull-14x14-evk-emmc.dtb	MYS-6ULX-IoT eMMC
mys-imx6ull-14x14-evk-gpmi-weim.dtb	MYS-6ULX-IoT NAND
mys-imx6ull-14x14-evk-emmc-myb6ulx.dtb	MYS-6ULX-IoT eMMC 上安装MYB-6ULX扩展板
mys-imx6ull-14x14-evk-gpmi-weim-myb6ulx.dtb	MYS-6ULX-IoT NAND 上安装MYB-6ULX扩展板

MYS-6ULX板上的Micro SD卡槽是连接mmc0控制器，所有的dtb文件都是默认启用mmc0控制器。

添加有"myb6ulx"标识的dtb文件，是用于支持MYB-6ULX扩展板，已配置好MYB-6ULX上的CAN，RS485，Ethernet，Camera和Audio功能，使用"myb6ulx"标识的dtb文件可以直接启动并使用相关功能。使用前请先安装好MYB-6ULX扩展板至MYS-6ULX-IOT或MYS-6ULX-IND开发板上。

SD卡或eMMC方式启动时，U-Boot默认查找的文件是mys-imx6ul-14x14-evk-emmc.dtb或mys-imx6ull-14x14-evk-emmc.dtb文件。因此，带有"myb6ulx"标识的文件，修改编译后需要重新命名为对应文件名。由于NAND启动是从地址读数据，不受影响。

3.3 构建文件系统

Linux系统平台上有许多开源的系统构建框架，这些框架方便了开发者进行嵌入式系统的构建和定制化开发，目前比较常见的有Buildroot, Yocto, OpenEmbedded等等。其中Yocto项目使用更强大和系统化的方法，来构建出适合嵌入式产品的Linux系统。

Yocto不仅仅是一个制做文件系统工具，同时提供整套的基于Linux的开发和维护工作流程，使底层嵌入式开发者和上层应用开发者在统一的框架下开发，解决了传统开发方式下零散和无管理的开发形态。

Yocto是一个开源的“umbrella”项目，意指它下面有很多个子项目，Yocto只是把所有的项目整合在一起，同时提供一个参考构建项目Poky，来指导开发人员如何应用这些项目，构建出嵌入式Linux系统。它包含Bitbake, OpenEmbedded-Core, 板级支持包，各种软件包的配置文件。通过Poky，可以构建出不同类需求的系统，如最小的系统core-image-minimal、带GUI的图形系统fsl-image-gui、带Qt5图形库的fsl-image-qt5。

NXP i.MX6UL/i.MX6ULL芯片提供符合Yocto的构建配置文件，通过这些文件可以构建出NXP定制的镜像文件。我们提供了符合MYS-6ULX的配置文件，帮助开发者构建出可烧写在MYS-6ULX板上的Linux系统镜像。

Yocto还提供了丰富的开发文档资源，让开发者学习并定制自己的系统。由于篇幅有限，不能完整介绍Yocto的使用方法，建议开发者先阅读以下文档后，再开始动手构建。

Yocto Project Quick start

Bitback User Manual

Yocto Project Reference Manual

Yocto Project Development Manual

Yocto Project Complete Documentation Set

3.3.1 Yocto构建Linux系统

本节适合需要对文件系统进行深度定制的开发者，希望从Yocto构建出符合MYS-6ULX系列开发板的文件系统，同时基于它的定制需求。初次体验使用或无特殊需要的开发者可以直接使用MYS-6ULX已经提供的文件系统。

由于Yocto构建前需要下载文件系统中所有软件包到本地，为了快速构建，MYS-6ULX已经把相关的软件打包好，可以直接解压使用，减少重复下载的时间。

这里提供了两种方式使用Yocto:

- 使用由MYS-6ULX资源包中的Yocto和相关文件
- 从NXP官网下载Yocto

初次使用Yocto的用户，推荐使用第一种方式。

注意：构建Yocto不需要加载工具链环境变量，请创建新shell或打开新的终端窗口。

MYS-6ULX提供的Yocto

解压Yocto源码包，同时解压Yocto-downloads.tar.xz软件包至Yocto目录下。Yocto-downloads.tar.xz是把MYS-6ULX构建中用到的第三方软件包打包，免除用户再次下载。

注意：由于Yocto-downloads.tar.xz文件较大，无法与MYS-6ULX打包在同一文件内，请访问网页下载: http://down.myir-tech.com/MYS-6ULX/。

```
cd $DEV_ROOT
tar xvf 04-Source/fsl-release-yocto.tar.xz
tar xvf 04-Source/Yocto-downloads.tar.xz -C fsl-release-yocto
cd fsl-release-bsp
```

还需要将Linux内核和U-Boot代码放在用户家目录下，方便开发和Yocto编译。

```
cd $DEV_ROOT
tar xvf $DEV_ROOT/04-Source/MYiR-iMX-Linux.tar.gz -C ~/
tar xvf $DEV_ROOT/04-Source/MYiR-iMX-uboot.tar.gz -C ~/
```

NXP官方提供的Yocto

Yocto下的项目比较多，为了便于管理使用与Android相同的代码管理工具repo。通过repo下载代码前，需要配置好git的用户名和邮箱地址。然后使用repo的命令从NXP官方仓库下载代码

- 设置repo

```
mkdir -p ~/bin
curl http://commondatastorage.googleapis.com/git-repo-downloads/repo \
> ~/bin/repo
chmod a+x ~/bin/repo
export PATH=~/bin:$PATH
```

- 设置Yocto

```
git config --global user.name "Your Name"
git config --global user.email "Your email address"
```

```
cd $DEV_ROOT
mkdir fsl-release-bsp
cd fsl-replease-bsp
repo init -u git://git.freescale.com/imx/fsl-arm-yocto-bsp.git \
-b imx-4.1-krogoth
repo sync
```

同步完成后，需要把meta-myir-imx6ulx拷贝到fsl-release-bsp/source目录下。同时，在后面的初始化构建目录后，还需要在conf/bblayers.conf中添加BBLAYERS += " ${BSPDIR}/sources/meta-myir-imx6ulx "到最后行。

也需要将Linux内核和U-Boot代码放在用户家目录下，方便开发和Yocto编译。

```
tar xvf $DEV_ROOT/04-Source/MYiR-iMX-Linux.tar.gz -C ~/
tar xvf $DEV_ROOT/04-Source/MYiR-iMX-uboot.tar.bz -C ~/
```

初始化**Yocto**构建目录

使用NXP提供的fsl-setup-release.sh脚本，会创建一个工作空间，然后在此空间下构建镜像。执行脚本后会先要求阅读并同意版权声明后才会进入构盡过目录。同时，脚本会默认创建并进入build目录。如果需要特定目录名称，可以使用-b参数，如"-b myir"。 这里的MACHINE参数有两种设备，"mys6ull14x14"对应于MYS-6ULX-IoT和"mys6ul14x14"对应于MYS-6ULX-IND版本。

```
DISTRO=myir-imx-fb MACHINE=mys6ul14x14 source fsl-setup-release.sh \
-b build
tree conf/
conf/
├── bblayers.conf
├── bblayers.conf.org
├── local.conf
├── local.conf.org
├── local.conf.sample
├── sanity_info
└── templateconf.cfg
```

build/conf目录下是当前构建的配置文件。上面在初始化时，构建适合"mys6ul14x14"的镜像，也可以在local.conf文件中修改MACHINE变量来构建适合"mys6ull14x14"的镜像。Yocto支持在同一个构建任务下构建多个设备。

构建**GUI Qt5**版的系统

第一次构建时，会需要很长时间，请耐心等待。

```
bitbake fsl-image-qt5
```

构建非**GUI**版的系统

第二次构建时，如果是同设备，不需要修改其它文件，直接编译即可。

```
bitbake core-image-base
```

Image名称	描述	用途

core-image-minimal	minimal版本的文件系统	用于MYS-6ULX的升级或更新系统
core-image-base	base版本的终端更多功能的镜像	通用的文件系统
fsl-image-qt5	构建基于Qt5的镜像	带Qt5的通用文件系统

构建文件系统完成后，会在输出目录下有manifest文件，这个文件里包含了对应文件系统中已安装的软件包。

Yocto第一次构建会需要很长时间，取决于计算机的CPU核心数和硬件读写速度。Yocto建议可以使用八核和SSD硬盘可以加速构建速度。第一次构建完成后会生成缓存，后面修改的构建，时间会减少很多。

櫬建完成后在会"tmp/deploy/images/mys6ul14x14/"或"tmp/deploy/images/mys6ull14x14/"目录下生成不同的文件，以下是构建后的一个例子：

```
ls -lh tmp/deploy/images/mys6ul14x14/
total 1.8G
-rw-r--r-- 1 kevinchen kevinchen  56M Apr 16 23:05
core-image-minimal-mys6ul14x14-20170416150516.rootfs.ext4
-rw-r--r-- 1 kevinchen kevinchen 2.1K Apr 16 23:05
core-image-minimal-mys6ul14x14-20170416150516.rootfs.manifest
-rw-r--r-- 1 kevinchen kevinchen  72M Apr 16 23:05
core-image-minimal-mys6ul14x14-20170416150516.rootfs.sdcard
-rw-r--r-- 1 kevinchen kevinchen  11M Apr 16 23:05
core-image-minimal-mys6ul14x14-20170416150516.rootfs.tar.bz2
-rw-r--r-- 1 kevinchen kevinchen 7.3M Apr 16 23:05
core-image-minimal-mys6ul14x14-20170416150516.rootfs.tar.xz
lrwxrwxrwx 1 kevinchen kevinchen   57 Apr 16 23:05
core-image-minimal-mys6ul14x14.ext4 -> core-image-minimal-
mys6ul14x14-20170416150516.rootfs.ext4
lrwxrwxrwx 1 kevinchen kevinchen   61 Apr 16 23:05
core-image-minimal-mys6ul14x14.manifest -> core-image-minimal-
mys6ul14x14-20170416150516.rootfs.manifest
lrwxrwxrwx 1 kevinchen kevinchen   59 Apr 16 23:05
core-image-minimal-mys6ul14x14.sdcard -> core-image-minimal-
mys6ul14x14-20170416150516.rootfs.sdcard
lrwxrwxrwx 1 kevinchen kevinchen   60 Apr 16 23:05
core-image-minimal-mys6ul14x14.tar.bz2 -> core-image-minimal-
mys6ul14x14-20170416150516.rootfs.tar.bz2
lrwxrwxrwx 1 kevinchen kevinchen   59 Apr 16 23:05
core-image-minimal-mys6ul14x14.tar.xz -> core-image-minimal-
mys6ul14x14-20170416150516.rootfs.tar.xz
-rw-r--r-- 1 kevinchen kevinchen 780M Apr 16 23:08
fsl-image-qt5-mys6ul14x14-20170416150603.rootfs.ext4
-rw-r--r-- 1 kevinchen kevinchen  35K Apr 16 23:08
fsl-image-qt5-mys6ul14x14-20170416150603.rootfs.manifest
-rw-r--r-- 1 kevinchen kevinchen 796M Apr 16 23:08
fsl-image-qt5-mys6ul14x14-20170416150603.rootfs.sdcard
-rw-r--r-- 1 kevinchen kevinchen 166M Apr 16 23:08
fsl-image-qt5-mys6ul14x14-20170416150603.rootfs.tar.bz2
-rw-r--r-- 1 kevinchen kevinchen 105M Apr 16 23:09
fsl-image-qt5-mys6ul14x14-20170416150603.rootfs.tar.xz
lrwxrwxrwx 1 kevinchen kevinchen   52 Apr 16 23:08
fsl-image-qt5-mys6ul14x14.ext4 -> fsl-image-qt5-
mys6ul14x14-20170416150603.rootfs.ext4
lrwxrwxrwx 1 kevinchen kevinchen   56 Apr 16 23:08
fsl-image-qt5-mys6ul14x14.manifest -> fsl-image-qt5-
mys6ul14x14-20170416150603.rootfs.manifest
lrwxrwxrwx 1 kevinchen kevinchen   54 Apr 16 23:09
fsl-image-qt5-mys6ul14x14.sdcard -> fsl-image-qt5-
mys6ul14x14-20170416150603.rootfs.sdcard
lrwxrwxrwx 1 kevinchen kevinchen   55 Apr 16 23:09
fsl-image-qt5-mys6ul14x14.tar.bz2 -> fsl-image-qt5-
```

```
mys6ul14x14-20170416150603.rootfs.tar.bz2
lrwxrwxrwx 1 kevinchen kevinchen   54 Apr 16 23:09
fsl-image-qt5-mys6ul14x14.tar.xz -> fsl-image-qt5-
mys6ul14x14-20170416150603.rootfs.tar.xz
-rw-r--r-- 2 kevinchen kevinchen 657K Apr 16 23:04
modules--4.1.15-r0-mys6ul14x14-20170416150349.tgz
lrwxrwxrwx 1 kevinchen kevinchen   49 Apr 16 23:04
modules-mys6ul14x14.tgz -> modules--4.1.15-r0-
mys6ul14x14-20170416150349.tgz
-rw-r--r-- 2 kevinchen kevinchen  294 Apr 16 23:07
README_-_DO_NOT_DELETE_FILES_IN_THIS_DIRECTORY.txt
-rwxr-xr-x 2 kevinchen kevinchen 375K Apr 16 22:53
u-boot-emmc-2016.03-r0.imx
lrwxrwxrwx 1 kevinchen kevinchen   26 Apr 16 22:53
u-boot.imx -> u-boot-emmc-2016.03-r0.imx
lrwxrwxrwx 1 kevinchen kevinchen   26 Apr 16 22:53
u-boot.imx-emmc -> u-boot-emmc-2016.03-r0.imx
lrwxrwxrwx 1 kevinchen kevinchen   26 Apr 16 22:53
u-boot.imx-nand -> u-boot-nand-2016.03-r0.imx
lrwxrwxrwx 1 kevinchen kevinchen   24 Apr 16 22:53
u-boot.imx-sd -> u-boot-sd-2016.03-r0.imx
lrwxrwxrwx 1 kevinchen kevinchen   26 Apr 16 22:53
u-boot-mys6ul14x14.imx -> u-boot-emmc-2016.03-r0.imx
lrwxrwxrwx 1 kevinchen kevinchen   26 Apr 16 22:53
u-boot-mys6ul14x14.imx-emmc -> u-boot-emmc-2016.03-r0.imx
lrwxrwxrwx 1 kevinchen kevinchen   26 Apr 16 22:53
u-boot-mys6ul14x14.imx-nand -> u-boot-nand-2016.03-r0.imx
lrwxrwxrwx 1 kevinchen kevinchen   24 Apr 16 22:53
u-boot-mys6ul14x14.imx-sd -> u-boot-sd-2016.03-r0.imx
-rwxr-xr-x 2 kevinchen kevinchen 427K Apr 16 22:53
u-boot-nand-2016.03-r0.imx
-rwxr-xr-x 2 kevinchen kevinchen 375K Apr 16 22:53
u-boot-sd-2016.03-r0.imx
lrwxrwxrwx 1 kevinchen kevinchen   48 Apr 16 23:04
zImage -> zImage--4.1.15-r0-mys6ul14x14-20170416150349.bin
-rw-r--r-- 2 kevinchen kevinchen 6.5M Apr 16 23:04
zImage--4.1.15-r0-mys6ul14x14-20170416150349.bin
-rw-r--r-- 2 kevinchen kevinchen  36K Apr 16 23:04
zImage--4.1.15-r0-mys-imx6ul-14x14-evk-20170416150349.dtb
-rw-r--r-- 2 kevinchen kevinchen  37K Apr 16 23:04
zImage--4.1.15-r0-mys-imx6ul-14x14-evk-emmc-20170416150349.dtb
-rw-r--r-- 2 kevinchen kevinchen  37K Apr 16 23:04 zImage
--4.1.15-r0-mys-imx6ul-14x14-evk-gpmi-weim-20170416150349.dtb
lrwxrwxrwx 1 kevinchen kevinchen   48 Apr 16 23:04 zImage-
mys6ul14x14.bin -> zImage--4.1.15-r0-mys6ul14x14-
20170416150349.bin
lrwxrwxrwx 1 kevinchen kevinchen   57 Apr 16 23:04 zImage-
mys-imx6ul-14x14-evk.dtb -> zImage--4.1.15-r0-mys-
imx6ul-14x14-evk-20170416150349.dtb
lrwxrwxrwx 1 kevinchen kevinchen   62 Apr 16 23:04 zImage-
mys-imx6ul-14x14-evk-emmc.dtb -> zImage--4.1.15-r0-mys-
imx6ul-14x14-evk-emmc-20170416150349.dtb
lrwxrwxrwx 1 kevinchen kevinchen   67 Apr 16 23:04 zImage-
mys-imx6ul-14x14-evk-gpmi-weim.dtb -> zImage--4.1.15-r0-
mys-imx6ul-14x14-evk-gpmi-weim-20170416150349.dtb
```

生成的文件中，有一些是链接文件，下面是不同文件的用途：

文件名	用途
*.rootfs.manifest	文件系统内的软件列表
*.rootfs.ext4	打包成ext4格式的文件系统

*.rootfs.sdcard	可直接写入SD卡，从SD卡启动的镜像
*.rootfs.tar.bz2	打包成tar.bz2格式的文件系统
*.rootfs.tar.xz	打包成tar.xz格式的文件系统
u-boot-emmc-2016.03-r0.imx	适合从eMMC启动的u-boot镜像
u-boot-nand-2016.03-r0.imx	适合从NAND启动的u-boot镜像

Bitbake常用命令

Bitbake 参数	描述
-c fetch	从recipe中定义的地址，拉取软件到本地
-c cleanall	清空整个构建目录
-c deploy	部署镜像或软件包到目标rootfs内
-k	有错误发生时也继续构建
-c compile	重新编译镜像或软件包

更多Yocto使用方法，请参考NXP官方Yocto使用文档《i.MX Yocto Project User's Guide》。

3.3.2 Yocto构建SDK工具

Yocto提供可构建出SDK工具的功能，用于底层或上层应用开发者使用的工具链和相关的头文件或库文件，免去用户手动制做或编译依赖库。SDK工具有两种，一种是适合底层开发的工具链，用于编译u-boot和linux内核代码，另外一种是应用开发工具链，附带目标系统的头文件和库文件，方便应用开发者移植应用在目标设备上。两种SDK工具都是shell自解压文件，执行后，默认安装在/opt目录下。

构建底层工具连

```
bitbake meta-toolchain
```

构建完成后，在"tmp/deploy/sdk"目录下有三个文件：

```
ls tmp/deploy/sdk/ -lh
-rw-r--r-- 1 kevinchen kevinchen 9.5K Apr 17 00:00 myir-imx6ulx-fb-
glibc-x86_64-meta-toolchain-cortexa7hf-neon-toolchain-4.1.15-2.0.1.
host.manifest
-rwxr-xr-x 1 kevinchen kevinchen  76M Apr 17 00:01 myir-imx6ulx-fb-
glibc-x86_64-meta-toolchain-cortexa7hf-neon-toolchain-4.1.15-2.0.1.sh
-rw-r--r-- 1 kevinchen kevinchen 1.6K Apr 17 00:00 myir-imx6ulx-fb-
glibc-x86_64-meta-toolchain-cortexa7hf-neon-toolchain-4.1.15-2.0.1.
target.manifest
```

这里有两个manifest文件，host.manifest是工具链中包含主机端的软件包的列表，target.manifest是包含目标设备端的软件包列表。

构建应用层工具链

应用层工具链是和Image名称是统一的，这里可以使用"fsl-image-qt5"和"core-iamge-base"两种参数。

```
bitbake -c populate_sdk <image name>
```

构建完成后，同样在"tmp/deploy/sdk/"目录下有三个文件：

```
ls tmp/deploy/sdk/ -lh
-rw-r--r-- 1 kevinchen kevinchen 9.5K Apr 17 07:54 myir-imx6ulx-fb-
glibc-x86_64-fsl-image-qt5-cortexa7hf-neon-toolchain-4.1.15-2.0.1.
host.manifest
-rwxr-xr-x 1 kevinchen kevinchen 587M Apr 17 07:59 myir-imx6ulx-fb-
glibc-x86_64-fsl-image-qt5-cortexa7hf-neon-toolchain-4.1.15-2.0.1.sh
-rw-r--r-- 1 kevinchen kevinchen  70K Apr 17 07:54 myir-imx6ulx-fb-
glibc-x86_64-fsl-image-qt5-cortexa7hf-neon-toolchain-4.1.15-2.0.1.
target.manifest
```

"*.host.manifest"*文件表示工具链中包含主机端的软件包列表，".target.manifest"表示工具链中包含目标设德端的软件包列表。"myir-imx6ulx-fb-glibc-x86_64-fsl-image-qt5-cortexa7hf-neon-toolchain-4.1.15-2.0.1.sh"文件是构建出的SDK工具链，可以直接安装在其他Linux系统中，开发和编译目标端设备的二进制程序。

4 Linux应用开发

本章主要介绍MYS-6ULX开发板底板外围硬件设备应用例程的使用。

使用前，需要先安装Yocto提供的SDK工具链，再编译所有例程代码，并拷贝至开发板目录下。

编译应用例程

加载工具链到当前终端后，可以查看gcc的版本信息，确认当前环境已正确加载。

```
. /opt/myir-imx6ulx-fb/4.1.15-2.0.1/environment-setup-cortexa7hf-neon-\
poky-linux-gnueabi
arm-poky-linux-gnueabi-gcc -v
Using built-in specs.
COLLECT_GCC=arm-poky-linux-gnueabi-gcc
COLLECT_LTO_WRAPPER=/opt/myir-imx6ulx-fb/4.1.15-2.0.1/sysroots/
x86_64-pokysdk-linux/usr/libexec/arm-poky-linux-gnueabi/gcc/arm-
poky-linux-gnueabi/5.3.0/lto-wrapper
Target: arm-poky-linux-gnueabi
Configured with: ../../../../../../work-shared/gcc-5.3.0-r0/gcc-
5.3.0/configure --build=x86_64-linux --host=x86_64-pokysdk-linux
--target=arm-poky-linux-gnueabi --prefix=/opt/myir-imx6ulx-fb/4.1
.15-2.0.1/sysroots/x86_64-pokysdk-linux/usr --exec_prefix=/opt/my
ir-imx6ulx-fb/4.1.15-2.0.1/sysroots/x86_64-pokysdk-linux/usr --bi
ndir=/opt/myir-imx6ulx-fb/4.1.15-2.0.1/sysroots/x86_64-pokysdk-li
nux/usr/bin/arm-poky-linux-gnueabi --sbindir=/opt/myir-imx6ulx-fb
/4.1.15-2.0.1/sysroots/x86_64-pokysdk-linux/usr/bin/arm-poky-linu
x-gnueabi --libexecdir=/opt/myir-imx6ulx-fb/4.1.15-2.0.1/sysroots
/x86_64-pokysdk-linux/usr/libexec/arm-poky-linux-gnueabi --datadi
r=/opt/myir-imx6ulx-fb/4.1.15-2.0.1/sysroots/x86_64-pokysdk-linux
/usr/share --sysconfdir=/opt/myir-imx6ulx-fb/4.1.15-2.0.1/sysroot
s/x86_64-pokysdk-linux/etc --sharedstatedir=/opt/myir-imx6ulx-fb/
4.1.15-2.0.1/sysroots/x86_64-pokysdk-linux/com --localstatedir=/o
pt/myir-imx6ulx-fb/4.1.15-2.0.1/sysroots/x86_64-pokysdk-linux/var
--libdir=/opt/myir-imx6ulx-fb/4.1.15-2.0.1/sysroots/x86_64-pokysd
k-linux/usr/lib/arm-poky-linux-gnueabi --includedir=/opt/myir-imx
6ulx-fb/4.1.15-2.0.1/sysroots/x86_64-pokysdk-linux/usr/include --
oldincludedir=/opt/myir-imx6ulx-fb/4.1.15-2.0.1/sysroots/x86_64-p
okysdk-linux/usr/include --infodir=/opt/myir-imx6ulx-fb/4.1.15-2.
0.1/sysroots/x86_64-pokysdk-linux/usr/share/info --mandir=/opt/my
ir-imx6ulx-fb/4.1.15-2.0.1/sysroots/x86_64-pokysdk-linux/usr/shar
e/man --disable-silent-rules --disable-dependency-tracking --with
-libtool-sysroot=/home/kevinchen/mys-imx6ul/fsl-release-yocto/bui
ld/tmp/sysroots/x86_64-nativesdk-pokysdk-linux --with-gnu-ld --en
able-shared --enable-languages=c,c++ --enable-threads=posix --ena
ble-multilib --enable-c99 --enable-long-long --enable-symvers=gnu
--enable-libstdcxx-pch --program-prefix=arm-poky-linux-gnueabi- -
-without-local-prefix --enable-lto --enable-libssp --enable-libit
m --disable-bootstrap --disable-libmudflap --with-system-zlib --w
ith-linker-hash-style=gnu --enable-linker-build-id --with-ppl=no
--with-cloog=no --enable-checking=release --enable-cheaders=c_glo
bal --without-isl --with-gxx-include-dir=/not/exist/usr/include/c
++/5.3.0 --with-build-time-tools=/home/kevinchen/mys-imx6ul/fsl-r
elease-yocto/build/tmp/sysroots/x86_64-linux/usr/arm-poky-linux-g
nueabi/bin --with-sysroot=/not/exist --with-build-sysroot=/home/b
lackrose/mys-imx6ul/fsl-release-yocto/build/tmp/sysroots/mys6ul14
x14 --enable-poison-system-directories --with-mpfr=/home/blackros
e/mys-imx6ul/fsl-release-yocto/build/tmp/sysroots/x86_64-nativesd
k-pokysdk-linux --with-mpc=/home/kevinchen/mys-imx6ul/fsl-release
-yocto/build/tmp/sysroots/x86_64-nativesdk-pokysdk-linux --enable
-nls --with-arch=armv7-a
Thread model: posix
```

```
gcc version 5.3.0 (GCC)

cd $DEV_ROOT/04-Sources
tar xvf example.tar.bz2
cd example
make
```

4.1 LCD测试

本例程演示对Linux的FrameBuffer设备操作，实现液晶输出显示RGB颜色和颜色合成测试。例程基于Linux FrameBuffer API接口开发。测试前需要把LCD连接至J8接口上。米尔科技提供两种LCD模块，分别是7寸的MY-TFT070CV2和4.3寸的MY-TFT043RV2。提供的prebuilt镜像是默认为4.3寸液晶的。

执行程序后，LCD液晶屏会出现相应颜色,以下是终端输出信息:

```
./framebuffer_test
The framebuffer device was opened successfully.
vinfo.xres=480
vinfo.yres=272
vinfo.bits_per_bits=16
vinfo.xoffset=0
vinfo.yoffset=0
red.offset=11
green.offset=5
blue.offset=0
transp.offset=0
finfo.line_length=960
finfo.type = PACKED_PIXELS
The framebuffer device was mapped to memory successfully.
color: red   rgb_val: 0000F800
color: green   rgb_val: 000007E0
color: blue   rgb_val: 0000001F
color: r & g   rgb_val: 0000FFE0
color: g & b   rgb_val: 000007FF
color: r & b   rgb_val: 0000F81F
color: white   rgb_val: 0000FFFF
color: black   rgb_val: 00000000
```

支持MY-TFT070RV2的配置方法

MYS-6ULX开发板中提供的Linux代码已经支持该模块的显示和触摸功能。MY-TFT070RV2的触摸功能是通过ADC采样方式，dts代码中已配置好,只需要启用相应功能即要可。

- MYS-6ULX-IND 第一步，编辑"arch/arm/boot/dts/mys-imx6ul-14x14-evk.dts"文件，修改tsc的status属性为okay。

  ```
  &tsc {
     pinctrl-names = "default";
     pinctrl-0 = <&pinctrl_tsc>;
     xnur-gpio = <&gpio1 3 GPIO_ACTIVE_LOW>;
     measure-delay-time = <0xffffff>;
     pre-charge-time = <0xffff>;
     status = "okay";
  };
  ```

  第二步，将默认的4.3寸屏莫的配置注释，并打开7.0寸的配置。找到lcfif节点下的display-timings节点，修改如下：

  ```
        display-timings {
            native-mode = <&timing0>;
  /*
             timing0: timing0 {
             clock-frequency = <9200000>
             hsync-len = <41>;
  ```

```
            vback-porch = <2>;
            vfront-porch = <4>;
            vsync-len = <10>;

            hsync-active = <0>;
            vsync-active = <0>;
            de-active = <1>;
            pixelclk-active = <0>;
            };
*/
            timing0: timing0 {
            clock-frequency = <33000000>;
            hactive = <800>;
            vactive = <480>;
            hfront-porch = <210>;
            hback-porch = <46>;
            hsync-len = <1>;
            vback-porch = <22>;
            vfront-porch = <23>;
            vsync-len = <20>;

            hsync-active = <0>;
            vsync-active = <0>;
            de-active = <1>;
            pixelclk-active = <1>;
            };

      };
```

- MYS-6ULX-IoT MYS-6ULX-IoT的修改方法和MYS-6ULX-IND的相同，编辑的文件为"arch/arm/boot/dts/mys-imx6ull-14x14-evk.dts"。

支持**MY-TFT070CV2**的配置方法

MY-TFT070CV2模块的触摸使用的是I2C方式通讯，从设备已添加到i2c2控制器上。使用前禁用tsc控制器，再启用7寸屏的配置参数即可。

- MYS-6ULX-IND 第一步，编辑"arch/arm/boot/dts/mys-imx6ul-14x14-evk.dts"文件，修改tsc的status属性为disabled。

```
&tsc {
   pinctrl-names = "default";
   pinctrl-0 = <&pinctrl_tsc>;
   xnur-gpio = <&gpio1 3 GPIO_ACTIVE_LOW>;
   measure-delay-time = <0xffffff>;
   pre-charge-time = <0xffff>;
   status = "disabled";
};
```

第二步，将默认的4.3寸屏莫的配置注释，并打开7.0寸的配置。找到lcfif节点下的display-timings节点，修改如下：

```
      display-timings {
          native-mode = <&timing0>;
/*
           timing0: timing0 {
           clock-frequency = <9200000>
           hsync-len = <41>;
           vback-porch = <2>;
           vfront-porch = <4>;
           vsync-len = <10>;
```

```
            hsync-active = <0>;
            vsync-active = <0>;
            de-active = <1>;
            pixelclk-active = <0>;
            };
*/
            timing0: timing0 {
            clock-frequency = <33000000>;
            hactive = <800>;
            vactive = <480>;
            hfront-porch = <210>;
            hback-porch = <46>;
            hsync-len = <1>;
            vback-porch = <22>;
            vfront-porch = <23>;
            vsync-len = <20>;

            hsync-active = <0>;
            vsync-active = <0>;
            de-active = <1>;
            pixelclk-active = <1>;
            };

      };
```

- MYS-6ULX-IoT MYS-6ULX-IoT的修改方法和MYS-6ULX-IND的相同，编辑的文件为"arch/arm/boot/dts/mys-imx6ull-14x14-evk.dts"。

4.2 触摸测试

MYS-6ULX支持两种触摸方式，电容和电阻。米尔科技可提供两种带触摸的液晶，7寸电容MY-TFT070CV2和4.3寸电阻MY-TFT043RV2。

触摸测试可以使用ts_calibrate和ts_test命令，ts_calibrate用于触摸校验，ts_test用于测试。其中，命令需要变量"TSLIB_TSDEVICE"来找到触摸设备，不同触摸方式的设备节点不一定相同，请对应设备填写。

```
export TSLIB_TSDEVICE=/dev/input/event1
# ts_calibrate

# ts_test
```

4.6 Ethernet 测试

本例使用Linux sokect API，实现简单的C/S结构的程序，两个程序通过TCP/IP协议栈通信。将可执行程序arm_client拷贝至开发板，pc_server拷贝至PC，将开发板和PC接入网络。

MYS-6ULX 网口

在 PC 上配置IP并运行服务程序:

```
sudo ifconfig eth0 192.168.1.111
./pc_server
REC FROM: 192.168.1.222
```

在开发板上运行客户程序，将看到所发送的信息:

```
ifconfig eth0 192.168.1.222
./arm_client 192.168.1.111
form server: Make Your idea Real!
```

MYB-6ULX 网口

注意：本例程需要在MYS-6ULX-IOT或MYS-6ULX-IND上安装MYB-6ULX扩展板，同时使用支持MYB-6ULX的dtb文件启动Linux系统

MYS-6ULX-IOT或MYS-6ULX-IND安装MYB-6ULX扩展板后，系统中的eth0设备是MYB-6ULX网口，eth1设备是MYS-6ULX-IOT或MYS-6ULX-IND网口。

两个网口测试时，可以分别连接至两个不同网段的路由器或交换机。

测试双网口通讯，设置两个网段的IP地址，使用ping命令测试网络连通性。

```
ifconfig eth0 192.168.1.100
ifconfig eth1 192.168.2.100
ping 192.168.1.101
ping 192.168.2.101
```

4.4 GPIO-KEY 测试

本例演示如何在Linux用户空间读取按键状态和键值。运行gpio_key程序后，按下或释放S2按键，串口会输出相应按键的状态信息。按下"Ctrl-C"可退出程序。

在开发板的控制终端上执行程序:

```
# ./gpio_key /dev/input/event2
Hit any key on board ......
key 2 Pressed
key 2 Released
key 2 Pressed
key 2 Released
```

4.5 GPIO-LED 测试

本例程演示使用Linux系统API操作开发板上的LED灯，D12。运行程序后，D12闪烁。按下"Ctrl-C"可结束程序。

```
# ./gpio_led /sys/class/leds/user/brightness
```

4.6 USB Host 测试

使用USB存储设备插入USB HOST(J9)接口，调试串口会输出检测设备信息。同时，使用将此存储设备挂载至linux系统下对其读写。

```
# usb 1-2: USB disconnect, device number 6
usb 1-2: new high-speed USB device number 7 using atmel-ehci
usb 1-2: New USB device found, idVendor=0bda, idProduct=0316
usb 1-2: New USB device strings: Mfr=1, Product=2, SerialNumber=3
usb 1-2: Product: USB3.0-CRW
usb 1-2: Manufacturer: Generic
usb 1-2: SerialNumber: 20120501030900000
usb-storage 1-2:1.0: USB Mass Storage device detected
scsi host5: usb-storage 1-2:1.0
scsi 5:0:0:0: Direct-Access     Generic- SD/MMC           1.00 PQ:
0 ANSI: 4
sd 5:0:0:0: [sda] 31116288 512-byte logical blocks: (15.9 GB/14.8
GiB)
sd 5:0:0:0: [sda] Write Protect is off
sd 5:0:0:0: [sda] Write cache: disabled, read cache: enabled, doesn't
support DPO or FUA
 sda: sda1 sda2
 sd 5:0:0:0: [sda] Attached SCSI removable disk

# mount /dev/sda1 /mnt/
# echo "hello" > /mnt/hello.txt
# cat /mnt/hello.txt
hello
```

4.7 USB Device测试

本例程演示使用开发板的Micro USB接口(J7)作为Device模式，可以将指定的文件或设备模拟为设备，连接到其它USB HOST设备。这里把内存作为存储设备提供给HOST设备。

- 开发板上使用modprobe加载驱动:

```
mkfs.vfat /dev/ram1
modprobe g_mass_storage.ko file=/dev/ram1 removable=1 \
iSerialNumber="1234"

[ 3048.950498] Mass Storage Function, version: 2009/09/11
[ 3048.982245] LUN: removable file: (no medium)
[ 3048.997849] LUN: removable file: /dev/ram1
[ 3049.000674] Number of LUNs=1
[ 3049.002272] Number of LUNs=1
[ 3049.023990] g_mass_storage gadget: Mass Storage Gadget,
version: 2009/09/11
[ 3049.029682] g_mass_storage gadget: g_mass_storage ready
[ 3094.766373] g_mass_storage gadget: high-speed config
#1: Linux File-Backed Storage
```

- Linux PC机上查看到有USB设备接入，SerialNumber为"1234"，Manufacturer是内核构建版本号:

```
dmesg | tail -n 20
[2872436.778616] usb 1-1: USB disconnect, device number 102
[2872436.779156] sd 3:0:0:0: [sdb] Synchronizing SCSI cache
[2872436.779201] sd 3:0:0:0: [sdb] Synchronize Cache(10)
failed: Result: hostbyte=DID_NO_CONNECT driverbyte=DRIVER_OK
[2872442.508567] usb 1-1: new high-speed USB device number 103
using xhci_hcd
[2872442.650549] usb 1-1: New USB device found, idVendor=0525,
idProduct=a4a5
[2872442.650551] usb 1-1: New USB device strings: Mfr=3, Produ
ct=4, SerialNumber=5
[2872442.650552] usb 1-1: Product: Mass Storage Gadget
[2872442.650553] usb 1-1: Manufacturer: Linux 4.1.15-1.2.0+g8d9
8da6 with 2184000.usb
[2872442.650554] usb 1-1: SerialNumber: 1234
[2872442.657827] usb-storage 1-1:1.0: USB Mass Storage device
detected
[2872442.657895] usb-storage 1-1:1.0: Quirks match for vid 0525
pid a4a5: 10000
[2872442.657923] scsi host3: usb-storage 1-1:1.0
[2872443.669426] scsi 3:0:0:0: Direct-Access     Linux    File-
Stor Gadget 0401 PQ: 0 ANSI: 2
[2872443.669886] sd 3:0:0:0: Attached scsi generic sg1 type 0
[2872443.670820] sd 3:0:0:0: [sdb] 131072 512-byte logical blocks:
(67.1 MB/64.0 MiB)
[2872443.779976] sd 3:0:0:0: [sdb] Write Protect is off
[2872443.779979] sd 3:0:0:0: [sdb] Mode Sense: 0f 00 00 00
[2872443.890093] sd 3:0:0:0: [sdb] Write cache: enabled, read cache:
enabled, doesn't support DPO or FUA
[2872444.110372]  sdb:
[2872444.330074] sd 3:0:0:0: [sdb] Attached SCSI removable disk
```

4.8 RS485 测试

注意：本例程需要在MYS-6ULX-IOT或MYS-6ULX-IND上安装MYB-6ULX扩展板，同时使用支持MYB-6ULX的dtb文件启动Linux系统

本例程演示如何使用 Linux Serial API编程，使用MYB-6ULX扩展板上的 RS485接口，实现数据发送和接收，详细请参考源码。

硬件连接

MYB-6ULX扩展板上有一个RS485接口(J9)，可以将A，B信号线与另外的RS485设备相连接。或者是USB转RS485的转换器设备。

软件测试

将编译出来的可执行程序拷贝至MYS-6ULX开发板系统内。MYS-6ULX作为发送端执行以下命令，另外一端的设备可以接数据。

```
./rs485_write -d /dev/ttymxc2 -b 4800 -e 1
SEND[20]: 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0a 0x0b 0x0c 0x0d 0x0e 0x0f 0x10
SEND[20]: 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0a 0x0b 0x0c 0x0d 0x0e 0x0f 0x10
SEND[20]: 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0a 0x0b 0x0c 0x0d 0x0e 0x0f 0x10
SEND[20]: 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0a 0x0b 0x0c 0x0d 0x0e 0x0f 0x10
```

MYS-6ULX作为接收端：

```
./rs485_read -d /dev/ttymxc2 -b 4800 -e 1
RECV[20]: 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0a 0x0b 0x0c 0x0d 0x0e 0x0f 0x10
RECV[20]: 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0a 0x0b 0x0c 0x0d 0x0e 0x0f 0x10
RECV[20]: 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0a 0x0b 0x0c 0x0d 0x0e 0x0f 0x10
RECV[20]: 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0a 0x0b 0x0c 0x0d 0x0e 0x0f 0x10
```

4.9 CAN Bus 测试

注意：本例程需要在MYS-6ULX-IOT或MYS-6ULX-IND上安装MYB-6ULX扩展板，同时使用支持MYB-6ULX的dtb文件启动Linux系统

本例程演示使用Linux socket CAN API，使用MYB-6ULX扩展板上的CAN总线接口发送和接收数据。将can_send和can_receive拷贝至开发板。执行以下步骤:

硬件连接

MYB-6ULX开发板有一个CAN总线接口(J11)，将H，L信号线与另外的CAN通讯设备或USB CAN转换器相连接。

软件测试

配置开发板上CAN0通信波特率都设置为 50kbps，并使能CAN0设备

Linux上可以使用两种工具来配置CAN设备，canconfig和ip，MYS-6ULX附带的系统默认使用ip命令。 canconfig命令配置：

```
canconfig can0 bitrate 50000 ctrlmode triple-sampling on
canconfig can0 start
```

ip命令配置

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

CAN收发测试可以使用系统附带的cansend、candump命令，也可以使用资源包中CAN收发例程。

- MYB-6ULX扩展板作为发送端：

使用cansend发送数据到CAN总线：

```
cansend can0 100#01.02.03.04.05.06.07.08
```

can_send例程运行后会一直发送数据，直到ctrl + c结束。

```
./can_send -d can0 -i 100 0x11 0x22 0x33 0x44 0x55 0x66 0x77 0x88
```

- MYB-6ULX扩展板作为接收端：

使用candump接收CAN数据：

```
candump can0
can0  100   [8]  01 02 03 04 05 06 07 08
```

can_receive例程接收来自CAN总线的数据：

```
can0  0x100  [8]  0x11 0x22 0x33 0x44 0x55 0x66 0x77 0x88
can0  0x100  [8]  0x11 0x22 0x33 0x44 0x55 0x66 0x77 0x88
can0  0x100  [8]  0x11 0x22 0x33 0x44 0x55 0x66 0x77 0x88
```

```
can0  0x100  [8]  0x11 0x22 0x33 0x44 0x55 0x66 0x77 0x88
```

4.10 Audio 测试

注意：本例程需要在MYS-6ULX-IOT或MYS-6ULX-IND上安装MYB-6ULX扩展板，同时使用支持MYB-6ULX的dtb文件启动Linux系统

硬件连接

本例程演示使用Linux系统中的arecord/aplay命令对音频接口录音和放音。需要使用两头3.5mm的音频AUX线，从电脑音频输出孔和开发板的LINE IN(J7)接口连接，HEADERPHONE(J5)连接耳机。

软件操作

在电脑中播放音频文件，执行arecord命令会先将LINE IN中的音频录制并保存为test.wav文件。运行一分钟后再按ctrl + c来停止。

```
arecord -f cd test.wav
```

执行aplay命令来播放上面录制好的音频文件。

```
aplay test.wav
```

4.11 Camera 测试

注意：本次测试需要在MYS-6ULX-IOT或MYS-6ULX-IND上安装MYB-6ULX扩展板和MY-CAM011B，同时使用支持MYB-6ULX的dtb文件启动Linux系统

MYB-6ULX扩展板上提供一个并行Camera接口(J10)，可以连接MY-CAM011B型号的Camera模块，模块之间使用FPC线连接。由于信号序列影响，请勿直接将其它型号的Camera的模块插入，否则会引起模块或开发板的损坏。

本例程演示使用一款开源的视频流软件uvc_stream，可以将Camera设备捕捉的数据显示在web页面。

硬件连接

使用FPC数据线将MYB-CAM011B模块和MYB-6ULX板上的J10接口相连接。

软件操作

uvc_stream是通过的网络传输数据，需要先设置好MYS-6ULX板的以太网IP地址，对应系统中的eth1设备。Linux系统中的MY-CAM011B模块的设备，可通通过v4l2-ctl命令来查询到，输出信息的i.MX6S_CSI表示Camera控制器，对应设备是/dev/video1。uvc_stream参数中'-y'是使用yuyv方式，'-P'后面是设置web界面的登录密码，用户名默认为uvc_user。'-r'是指定分辨率，当前仅支持800x600。可以用ctrl + c来停止。

```
ifconfig eth1 192.168.1.42
v4l2-ctl --list-devices
i.MX6S_CSI (platform:21c4000.csi):
    /dev/video1

    pxp (pxp_v4l2):
        /dev/video0

./uvc_stream -d /dev/video1 -y -P 123456 -r 800x600
```

uvc_stream提供两种web功能，snapshot和streaming。snapshot的请求URL是snapshot.jpeg，streaming的请求URL是stream.mjpeg。 PC和开发板在同一网络内时，打开流览器，输入地址http://192.168.1.42:8080/stream.mjpeg，可以看到有登录框，输入用户名为uvc_user，密码为123456，就可以看到从MY-CAM011B实时采集到的图像了。

4.12 WiFi 测试

注意：本例程适用于MYS-6ULX-IOT开发板

MYS-6ULX-IOT开发板的背面提供一个WiFi模块(U13)，支持Client和AP模式，本节分别测试两种场景下的使用方法。

硬件连接

将附带I-PEX接口的天线安装在开发板的J10位置。

Client模式

Client模式是用于将WiFi模块作为客户访问设备，主动连接于路由器或其它提供无线热点的设备。

系统中已经加入WiFi模块的驱动，启动后会自动加载相应驱动，可以使用lsmod命令来确认。驱动加载成功后会出现对应的wlan0网络设备，使用ifconfig命令来确认。

```
lsmod
Module                  Size  Used by
8188eu                758318  0

ifconfig wlan0
wlan0     Link encap:Ethernet  HWaddr a0:2c:36:60:ee:e0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:3388 errors:0 dropped:10 overruns:0 frame:0
          TX packets:37 errors:0 dropped:3 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:395459 (386.1 KiB)  TX bytes:6074 (5.9 KiB)
```

下面使用wpa_passphrase生成对应WiFi热点SSID的密码，然后由wpa_supplicant命令实现WiFi模块与WiFi热点的连接。

```
wpa_passphrase "MYiRTech" >> wifi.conf
12345678

cat wifi.conf
# reading passphrase from stdin
network={
    ssid="MYiRTech"
    #psk="12345678"
    psk=b96d9a5de2d9480ad5f987857e20216b47a0c4bf43397825ba909438bc52aaff
}

wpa_supplicant -D wext -B -i wlan0 -c wifi.conf
Successfully initialized wpa_supplicant
rfkill: Cannot open RFKILL control device
R8188EU: Firmware Version 11, SubVersion 1, Signature 0x88e1
MAC Address = a0:2c:36:60:ee:e0
IPv6: ADDRCONF(NETDEV_UP): wlan0: link is not ready
ioctl[SIOCSIWAP]: Operation not permitted
R8188EU: INFO indicate disassoc
```

连接成功后，使用udhcpc获取IP地址后，就可以使用了。

```
udhcpc -b -i wlan0 -R
ifconfig wlan0
```

```
wlan0     Link encap:Ethernet  HWaddr a0:2c:36:60:ee:e0
          inet addr:192.168.1.211  Bcast:192.168.1.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:5577 errors:0 dropped:15 overruns:0 frame:0
          TX packets:46 errors:0 dropped:3 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:651690 (636.4 KiB)  TX bytes:7472 (7.2 KiB)
```

AP模式

AP模式是由软件和硬件同时支持才可以实现，MYS-6ULX-IOT板上的WiFi模块可以支持AP模式。软件上使用Hostapd来实现热点配置功能。

文件系统中给出AP模式需要的配置文件,存放在/etc/wifi-conf/目录

udhcpcd.conf内容如下:

```
# the start and end of the IP lease block
start        192.168.10.10
end            192.168.10.254
# the interface that udhcpd will use
interface    wlan0

option    subnet    255.255.255.0
opt    router    192.168.10.1
option    domain    local
option    lease    864000
```

hostapd.conf内容如下:

```
# WPA2-PSK authentication with AES encryption
interface=wlan0
driver=nl80211
ctrl_interface=/var/run/hostapd
ssid=MYIR-WIFI-AP
channel=6
ieee80211n=1
hw_mode=g
ignore_broadcast_ssid=0
wpa=2
wpa_passphrase=MYIR-TECH
wpa_key_mgmt=WPA-PSK
rsn_pairwise=CCMP
```

然后将wlan0作为路由：

```
udhcpd /etc/wifi-conf/udhcpd.conf
hostapd -B /etc/wifi-conf/hostapd.conf
```

使用手机或其他WiFi设备连接SSID为:"MYIR-WIFI-AP"热点,密码:"MYIR-TECH",连接后即可分配到正确的IP地址。

5 QT应用开发

Qt是一个跨平台的图形应用开发框架，被应用在不同尺寸设备和平台上，同时提供不同版权版本供用户选择。MYS-6ULX使用Qt 5.6.2版本进行应用开发。

在Qt应用开发中，推荐使用QtCreator集成开发环境，可以在Linux PC下开发Qt应用，自动化地交叉编译为开发板的ARM架构。

本章使用Yocto构建的SDK工具作为交叉编译系统，配合QtCreator快速开发图形类应用程序。开始本章前，请先完成第三章的Yocto构建过程。或者使用光盘中提供的预编译好的SDK工具包。本章开始前，请安装好应用SDK开发工具。

5.1 安装QtCreator

QtCreator安装包是一个二进制程序，直接执行就可以完成安装。

```
cd $DEV_ROOT
chmod a+x 03-Tools/qt-creator-opensource-linux-x86_64-4.1.0.run
sudo 03-Tools/Qt/qt-creator-opensource-linux-x86_64-4.1.0.run
```

执行安装程序后，一直点击下一步即可完成。默认安装目录在"/opt/qtcreator-4.1.0"。

安装完成后，为了让QtCreator使用Yocto的SDK工具，需要对QtCreator加入环境变量。修改"/opt/qtcreator-4.1.0/bin/qtcreator.sh"文件，在"#! /bin/sh"前加入Yocto的环境配置即可，参考如下：

```
source /opt/myir-imx6ulx-fb/4.1.15-2.0.1/\
environment-setup-cortexa7hf-neon-poky-linux-gnueabi
#! /bin/sh

# Use this script if you add paths to LD_LIBRARY_PATH
# that contain libraries that conflict with the
# libraries that Qt Creator depends on.
```

使用QtCreator时，请从终端执行"qtcreator.sh"来启动QtCreator，参考如下：

```
/opt/qtcreator-4.1.0/bin/qtcreator.sh &
```

5.2 配置QtCreator

第一步，运行 QtCreator 后，依次点击"Tool"->"Options"，出现选项对话框，在左侧点击"Build & Run"，右边选择"Compilers"标签。 点击右侧"Add"按钮，弹出下拉列表后，选择"GCC"，在下面填写"Name"为"MYS6ULx-GCC"，"Compiler path"点击旁边的"Browse.."按钮选 择到arm-poky-linux-gnueabi-g++的路径，例子中的路径是"/opt/myir-imx6ulx-fb/4.1.15-2.0.1/sysroots/x86_64-pokysdk-linux/usr/bin/arm-poky-linux-gnueabi/arm-poky-linux-gnueabi-g++"。填写完成后，点击"Apply"。

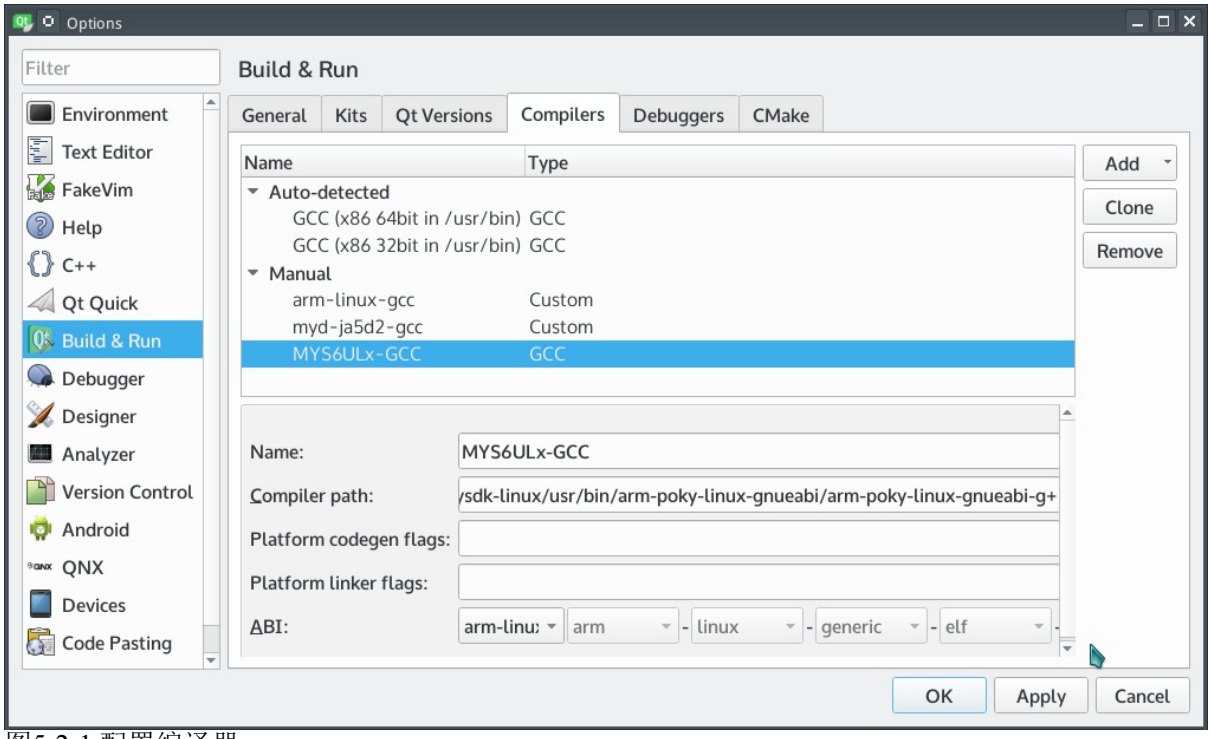


图5-2-1 配置编译器

第二步，选择"Qt Version"标签，在右侧点击"Add..."，会弹出qmake路径选择对话框，这里以"/opt/myir-imx6ulx-fb/4.1.15-2.0.1/sysroots/x86_64-pokysdk-linux/usr/bin/qt5/qmake"为例子。选择"qmake"文件后，点击"Open"按钮。"Version name"改为"Qt %{Qt:Version} (mys6ulx-qt5)"。然后点击"Apply"按钮。

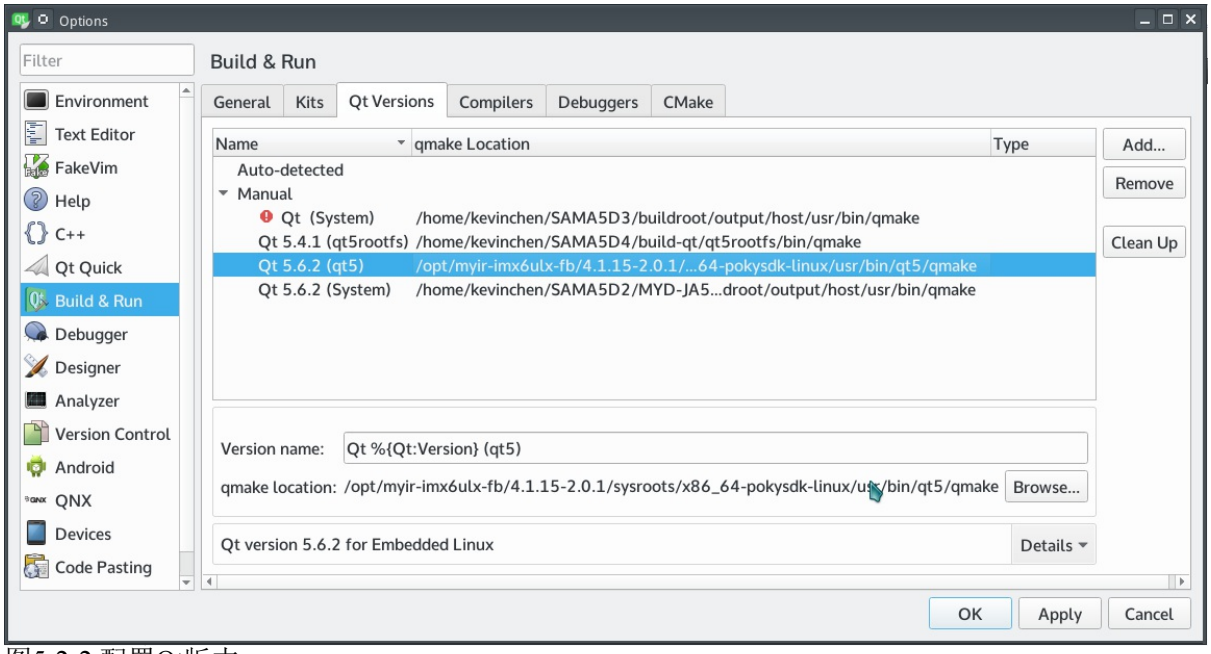


图5-2-2 配置Qt版本

第三步，选择左侧"Device"，点击右边的"Add..."按钮，填写内容"Name"为"MYS6ULx Board"，"Host name"为开发板的IP地址(可以暂时填写任意一个地址)，"Username"为"root"，然后点击"Apply"。

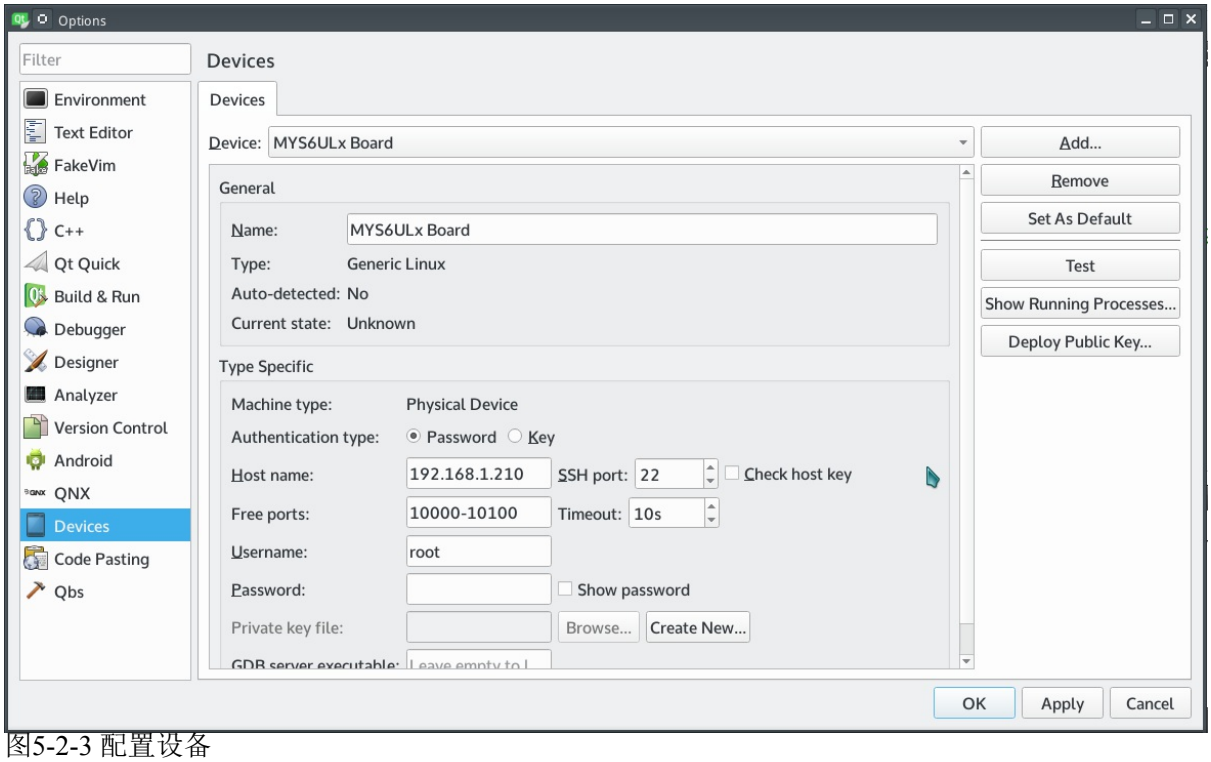
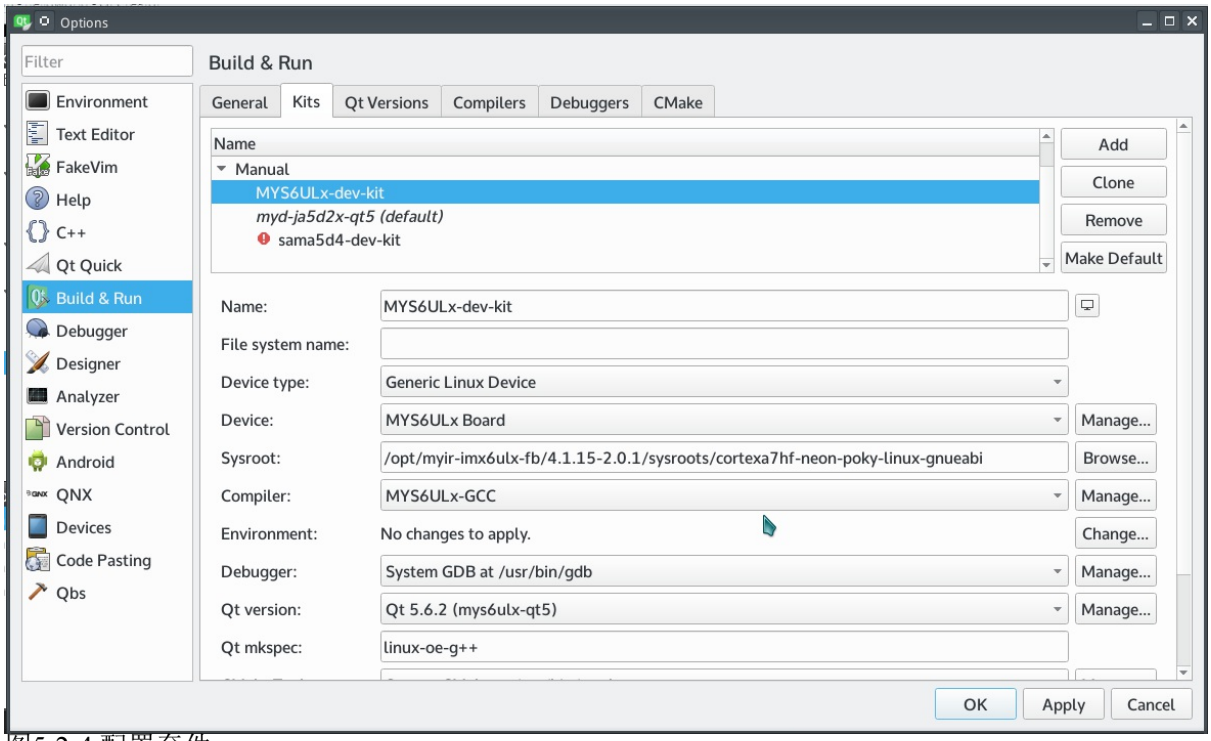


图5-2-3 配置设备

第四步，点击左侧"Build & Run"回到"Kits"标签下，"Name"为"MYS6ULx-dev-kit"，"Device"选择"MYS6ULx Board"选项了。"Sysroot"选择目标设备的系统目录，这里以"/opt/myir-imx6ulx-fb/4.1.15-2.0.1/sysroots/cortexa7hf-neon-poky-linux-gnueabi"为例。"Compiler"选择之前配置的名称"MYS6ULx-GCC"，"Qt version"选择之前配置的名称"Qt 5.6.2 (mys6ulx-qt5)"，"Qt mkspec"填写为"linux-oe-g++"。其它默认即可，最后点击"Apply"和"OK"按钮。


Options
Filter
Environment
Text Editor
FakeVim
Help
C++
Qt Quick
Build & Run
Debugger
Designer
Analyzer
Version Control
Android
QNX
Devices
Code Pasting
Qbs
Build & Run
General
Kits
Qt Versions
Compilers
Debuggers
CMake
Name
Manual
MYS6ULx-dev-kit
myd-ja5d2x-qt5 (default)
sama5d4-dev-kit
Add
Clone
Remove
Make Default
Name:
MYS6ULx-dev-kit
File system name:
Device type:
Generic Linux Device
Device:
MYS6ULx Board
Manage...
Sysroot:
/opt/myir-imx6ulx-fb/4.1.15-2.0.1/sysroots/cortexa7hf-neon-poky-linux-gnueabi
Browse...
Compiler:
MYS6ULx-GCC
Manage...
Environment:
No changes to apply.
Change...
Debugger:
System GDB at /usr/bin/gdb
Manage...
Qt version:
Qt 5.6.2 (mys6ulx-qt5)
Manage...
Qt mkspec:
linux-oe-g++
OK
Apply
Cancel
图5-2-4 配置套件

5.3 测试Qt应用

为了方便测试之前的配置是否正确，这里提供了一个Qt例程，打开项目后，配置为相应的编译工具套件，就可以编译此例程。

第一步，在菜单栏选择"File"->"Open File or Project"，在打开的对话框中，浏览到"helloword"例程的目录下，选择"helloworld.pro"文件，点击"Open"按钮。

第二步，项目打开后，在左侧菜单列中，选择"Projects"图标，右侧界面切换为"helloworld"项目的"Build & Run"标签下，点击"Add kit"下拉列表，选择"MYS6ULx-dev-kit"选项，这样"helloworld"项目就会使用"MYS6ULx-dev-kit"的相关配置构建应用。

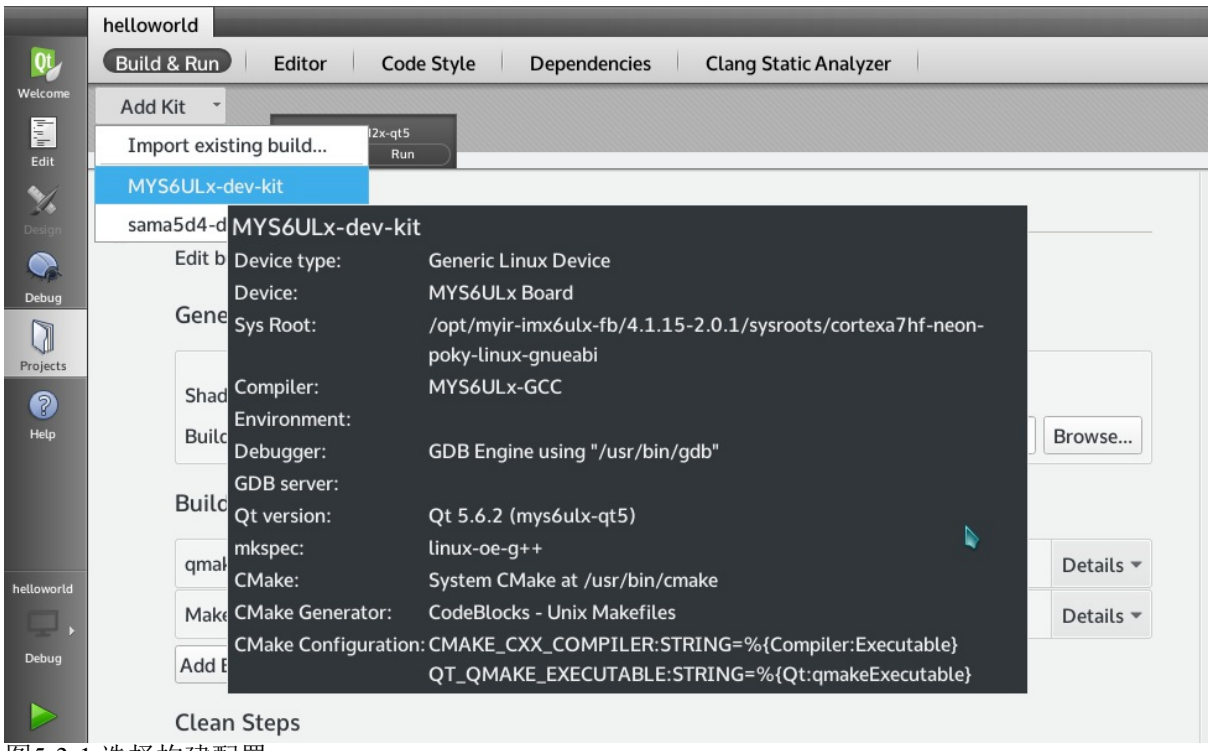


图5-3-1 选择构建配置

第三步，点击菜单栏"Build"->"Build Project hellowrld"按钮，即可完成项目的编译，同时下侧会有编译过程输出。

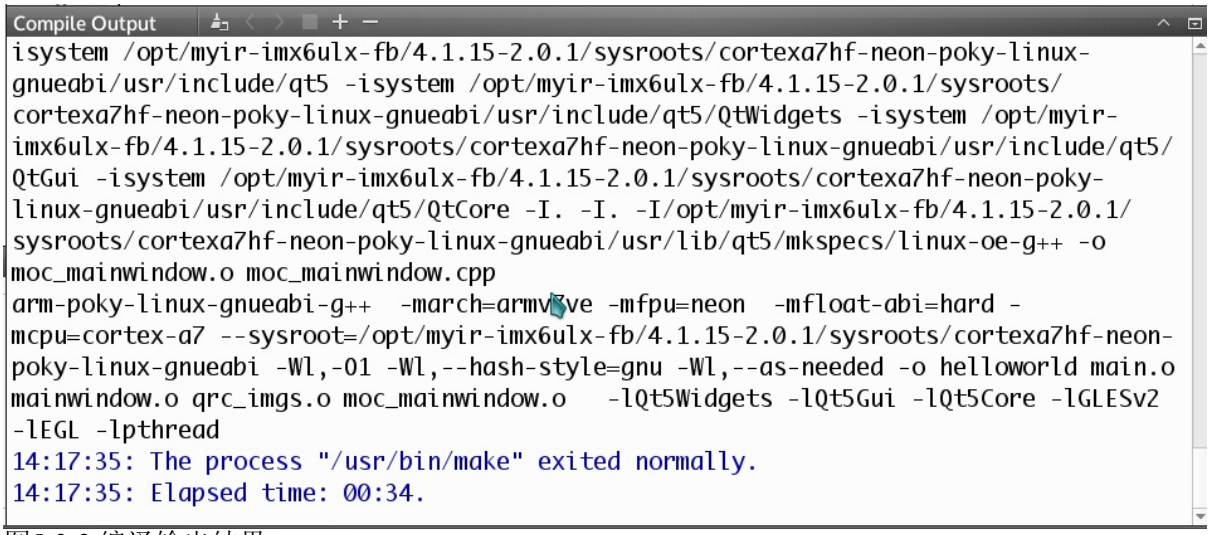


图5-3-2 编译输出结果

QtCreator 构建 helloworld 项目后，编译好的二进制文件存放在"~/build-helloworld-MYS6ULx_dev_kit-Debug/"目录下，可以使用 file 命令查看，是否编译为 ARM 架构。

```
file ~/build-helloworld-MYS6ULx_dev_kit-Debug/helloworld
/home/kevinchen/build-helloworld-MYS6ULx_dev_kit-Debug/helloworld:
ELF 32-bit LSB executable, ARM, EABI5 version 1 (GNU/Linux),
dynamically linked, interpreter /lib/ld-linux-armhf.so.3,
for GNU/Linux 2.6.32,
BuildID[sha1]=9c5f22deb1d8272c2a81528c171d215896112784, not stripped
```

然后将 helloworld 文件拷贝到开发板下运行即可。

```
# ./helloworld -platform linuxfb
```

将会在 LCD 屏幕上看到Qt 窗口中多个QLabel的文本内容。

图5-3-3 例程运行结果

6 系统更新

MYS-6ULX系列板的系统更新使用两种方法，MfgTool更新和SD卡更新。

MfgTool更新系统

安装工具

烧写工具是由NXP公司提供的MfgTool 2.7.0版本，光盘中路径"03-Tools/ManufactoryTool"目录下，支持Windows和Linux操作系统。解压后的目录中有多个vbs文件，这些是配置好的烧写脚本。执行后即可启动MfgTool程序。

VBS文件	描述
core-image-base-mys-6ulx-emmc.vbs	烧写core-image-base系统至MYS-6ULX开发板eMMC版本
core-image-base-mys-6ulx-nand.vbs	烧写core-image-base系统至MYS-6ULX开发板NAND版本
fsl-image-qt5-mys-6ulx-emmc.vbs	烧写fsl-image-qt5系统至MYS-6ULX开发板eMMC版本
fsl-image-qt5-mys-6ulx-nand.vbs	烧写fsl-image-qt5系统至MYS-6ULX开发板NAND版本

更新步骤如下(顺序不可颠倒):

注意：请勿将DV 5V电源适配器和Micro USB接口同时供电

- 拨动启动拨码开关(SW1)的第3位为ON，第4位为OFF。
- 使用USB转接线(Type-A转Micro-B)连接PC机USB端口与开发板MicroUSB接口(J7)。
- 双击MfgTool目录下的"mfgtool2-yocto-mx6ul-evk-nand.vbs"文件，此时可以看到MfgTool界面已识别到开发板。
- 点击MfgTool界面上的"Start"按钮，MfgTool就开始自动更新系统至板载存储设备。

更新成功后底部的总进度条会显示为绿色。若失败则为红色时，可以查看"MfgTool.log"文件的错误提示信息。或者使用USB转TTL串口线连接至JP1，再重新更新系统，就可以从串口查看更新过程并分析失败的原因。

更新MfgTool

MfgTool的文件更新有两个部分，firmware和files。files目录下为烧写的目标镜像文件，路径为"MYS-6ULX-mfgtools/Profiles/Linux/OS Firmware/files/"。 firmware是烧写系统的镜像文件，路径为"MYS-6ULX-mfgtools/Profiles/Linux/OS Firmware/firmware/"。当更新系统的分区大小或烧写方式时才需要更新firmware中的文件。

files目录下MYS-6ULX-IOT开发板的文件说明：

文件名	描述
core-image-base-mys6ull14x14.rootfs.tar.bz2	core-image-base文件系统
fsl-image-qt5-mys6ull14x14.rootfs.tar.bz2	fsl-image-qt5文件系统
u-boot-imx6ull14x14evk-ddr256_emmc.imx	支持DDR 256MB和eMMC启动的uboot
u-boot-imx6ull14x14evk-ddr256_nand.imx	支持DDR 256MB和NAND启动的uboot

zImage-imx6ull	内核镜像
zImage-imx6ull-14x14-evk-emmc.dtb	支持eMMC的设备树文件
zImage-imx6ull-14x14-evk-gpmi-weim.dtb	支持NAND的设备树文件

files目录下MYS-6ULX-IND开发板的文件说明：

文件名	描述
core-image-base-mys6ul14x14.rootfs.tar.bz2	core-image-base文件系统
fsl-image-qt5-mys6ul14x14.rootfs.tar.bz2	fsl-image-qt5文件系统
u-boot-imx6ul14x14evk-ddr256_emmc.imx	支持DDR 256MB和eMMC启动的uboot
u-boot-imx6ul14x14evk-ddr256_nand.imx	支持DDR 256MB和NAND启动的uboot
zImage-imx6ul	内核镜像
zImage-imx6ul-14x14-evk-emmc.dtb	支持eMMC的设备树文件
zImage-imx6ul-14x14-evk-gpmi-weim.dtb	支持NAND的设备树文件

Micro SD卡更新系统

MYS-6ULX开发板提供了sdcard.gz后缀的镜像文件，用于烧写至SD卡后，从SD卡更新系统至板载Flash存储器。sdcard.gz镜像文件需要使用特殊的磁盘操作工具才可以写入Micro SD卡内，Linux系统用户可以直接使用gzip和dd命令，Windows系统用户使用Usb Image Tool工具，或者先解压后得到sdcard文件再用Win32ImageWriter工具写入到SD卡。

制做SD启动更新的镜像

若对系统Linux kernel，U-Boot或者文件系统有修改，需要使用工具将这些二进制文件更新的开发板上。MYS-6ULX开发板提供了一个可以制做SD更新镜像的工具MYS-6ULX-mkupdate-sdcard，存放在04-Tools/ManufactoryTool目录。

build-sdcard.sh脚本用于制做从SD卡更新系统的镜像，分为两个部分：更新系统和目标文件。firmware目录下是更新系统，一般情况下不需要修改。 'mfgimages-*'是目标文件，里面的文件最终会烧写进板载的NAND或者eMMC存储芯片内。如果修改u-boot, kernel后，需要把相应的文件替换到目标文件内即可。

'mfgimage-*'目录内的文件名遵循以下方式命名，错误的文件名称在更新系统时不会被识别，会出现升级失败。 这些文件的名称被定义在Manifest文件内，命名规则如下：

```
ubootfile="u-boot.imx"
envfile="boot.scr"
kernelfile="zImage"
dtbfile="mys-imx6ull-14x14-evk-emmc.dtb"
rootfsfile="core-image-base.rootfs.tar.xz"
```

变量'envfile' 仅用于eMMC版本。 更新程序启动后会根据Manifest文件加载需要的文件，以将它们写入到目标Flash。

解压后就可以开始制做镜像了。

```
sudo ./build-sdcard.sh -p mys6ull -n -d mfgimages-mys-imx6ul-ddr256m-nand256m
```

build-sdcard.sh提供了四种参数：

- '-p' 表示平台，可用参数为"mys6ull"代表MYS-6ULX-IOT和"mys6ul"代表MYS-6ULX-IND
- '-n' 表示板上存储芯片是NAND
- '-e' 表示板上存储芯片是eMMC
- '-d' 表示更新文件的目录

注意：'-n'和'-e'不能同时使用，只能使用一种。

运行结束后会生成一个sdcard后缀的文件，如'mys6ull-update-nand-20170825150819.rootfs.sdcard'。

制做可更新系统的SD卡

MYS-6ULX资源包内提供了用于更新系统的sdcard镜像文件，可以直接使用，也可以使用上一步制做的sdcard文件。MYS-6ULX提供好的sdcard文件在02-Images目录内。 有了用于更新的SD卡镜像文件，就可以把镜像文件写入到SD卡。为了方便使用，建议把Micro SD插入USB读卡器，再插入电脑USB端口。

注意: 02-Images目录内的文件名的时间标识部分可能与如下示例代码有差异，请以实际为主。

- Linux系统

通常Linux下的存储设备名为"sd[x][n]"形式，x表示第几个存储设备，一般使用字母a~z表示。n表示存储设备的分区，一般使用数字，从1开始。插入后可以使用"dmesg | tail"命令查看新设备的设备名称。这里以"/dev/sdb"设备为例，sdb后面不写任务分区数字。

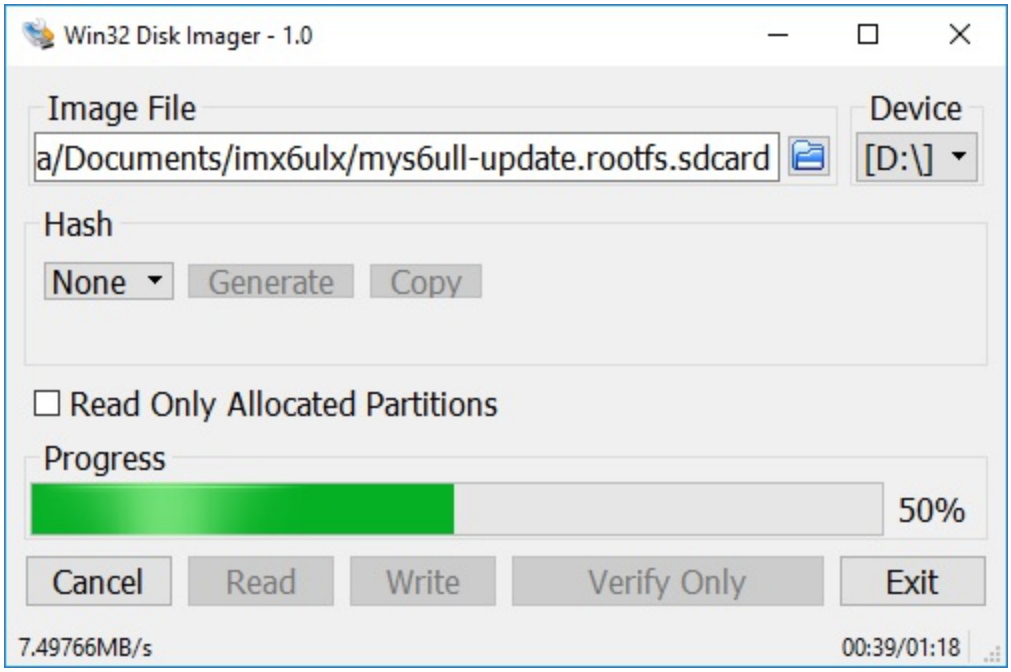
```
sudo dd if=mys6ull-update-nand-20170919090957.rootfs.sdcard of=/dev/sdb conv=fsync
```

写入的速度与USB和Micro SD卡的速率有关，如果对速度有要求，建议选用更高速度等级的Micro SD存储卡。

- Windows系统

Windows用户可以使用USB Image Tool工具把sdcard.gz镜像写入到Micro SD，或者先解压后再使用Win32DiskImager工具把sdcard镜像写入Micro SD里。工具在"03-Tools"目录下，解压后，双击"Win32DskImager.exe"应用程序。启动后的界面中，右侧的"Device"是选择要写入的设备盘符。左侧的"Image File"是选择将要写的镜像文件，点击旁边的文件夹图标，选中要写入的文件即可(注意：文件选择对话框中默认是过滤"*.img"文件，切换成"*.*"，就可以显示到sdcard后缀的文件)。

写入前请再次确认目标磁盘和文件是否正确，避免写入到系统磁盘，损坏Windows系统分区。



等待进度条结束后，就可以拔出USB读卡器。

把制做好的Micro SD卡插入开发板的卡槽(J5)，启动位拨码开关(SW1)配置为SDCARD启动方式。

NAND版本的SDCard启动配置如下：

启动位	状态
Bit1	OFF
Bit2	ON
Bit3	OFF
Bit4	ON

eMMC版本的SDCard启动配置如下：

启动位	状态
Bit1	OFF
Bit2	OFF
Bit3	OFF
Bit4	ON

连接USB转TTL串口线至调试口(JP1)，配置好电脑端的串口终端软件，连接USB数据线至USB OTG端口(J7)作为电源(也可以使用直接电源连接至J1接口)。通过串口可以看到系统从Micro SD卡启动，并执行更新脚本，把镜像写入NAND存储芯片内。

也可以通过用户LED灯(D12)来判断当前更新状态，更新中为闪烁状态，更新成功后会常亮，失败则会熄灭。

从板载**Flash**启动

两种方式重更新完成后断电，配置启动位拨码开关为对应板载的Flash启动方式。

切换为**NAND**启动方式

启动位	状态
Bit1	ON
Bit2	OFF
Bit3	OFF
Bit4	ON

切换为**eMMC**启动方式

启动位	状态
Bit1	ON
Bit2	ON
Bit3	OFF
Bit4	ON

再次连接电源，开发板即可从NAND或eMMC启动系统了。

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：0755-25622735
- 邮箱：support@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。