QY-IMX6S Linux 用户手册

版本号 V2.0

2014 年 11 月

目　录

版本说明

版本	硬件平台	描述	日期	修订人
1.0	QY-IMX6S-V1.1	初始版本，首次发布	2014-02-21	wujj
2.0	QY-IMX6S-V1.2	更新硬件版本	2014-11-11	wangwx

前言

欢迎使用浙江启扬智能科技有限公司产品 QY-IMX6S，本产品 Linux 部分包含 4 份手册：

《QY-IMX6S Linux 用户手册.pdf》
《QY-IMX6S 硬件说明手册.pdf》
《QY-IMX6S Linux 功能说明与测试手册.pdf》
《QY-IMX6S Linux 系统镜像烧写手册.pdf》

硬件相关部分可以参考《QY-IMX6S 硬件说明手册.pdf》
主板测试可以参考《QY-IMX6S Linux 功能说明与测试手册.pdf》
镜像烧写参考《QY-IMX6S 系统镜像烧写手册.pdf》

本手册主要介绍交叉编译环境的搭建、源代码以及应用例程的编译。

使用之前请仔细阅读《QY-IMX6S 硬件说明手册.pdf》!

公司简介

浙江启扬智能科技有限公司位于美丽的西子湖畔,是一家集研发、生产、销售为一体的高新技术产业。公司致力于成为嵌入式解决方案的专业提供商，为嵌入式应用领域客户提供软硬件开发工具和嵌入式系统完整解决方案。产品范围主要包括：Cirrus Logic EP93xx 系列 ARM9 主板、ATMEL AT91SAM926x 系列主板，FreeScale iMX 系列主板，TI Davinci 系列音/视频通用开发平台等等。可运行 Linux2.4/2.6、WinCE5.0/6.0 操作系统，并可根据客户需求开发各种功能组合的嵌入式硬件系统。应用领域涉及：工业控制、数据采集、信息通讯、医疗设备、视频监控、车载娱乐等等。

客户的需求是公司发展的动力，公司将不断完善自身，与客户互助互惠，共同发展。

电话：0571-87858811，87858822
传真：0571-87858822
技术支持 E-MAIL：support@qiyangtech.com
网址： http://www.qiyangtech.com
地址：杭州市西湖区西湖科技园西园一路 8 号 3A 幢 5 层
邮编：310012

一、准备工作

◆ 装有 Linux 系统（ubuntu 或其它 Linux 发行版）,本手册以 ubuntu 12.04 操作为例,具体搭建请参照《虚拟机安装 ubuntu 指导手册.pdf》!

◆ 由于编译过程中需要请文件拷贝到虚拟机 ubuntu，在 ubuntu 用户目录下创建一个工作目录：mkdir ~/work　　/* 这里的~/表示用户目录，实际对应的绝对路径为/home/st */

为了统一和陈述方便，所有文件都拷贝到该目录进行操作，具体目录用户可以自行创建，这里只是以~/work 目录为例!

◆ 关于 linux 下的常用命令以及 vi 的操作等在这里都不做详细说明，请用户自行查阅相关资料!

◆ 所有 PC 机和虚拟机的拷贝都采用 samba 共享访问的方式来拷贝!

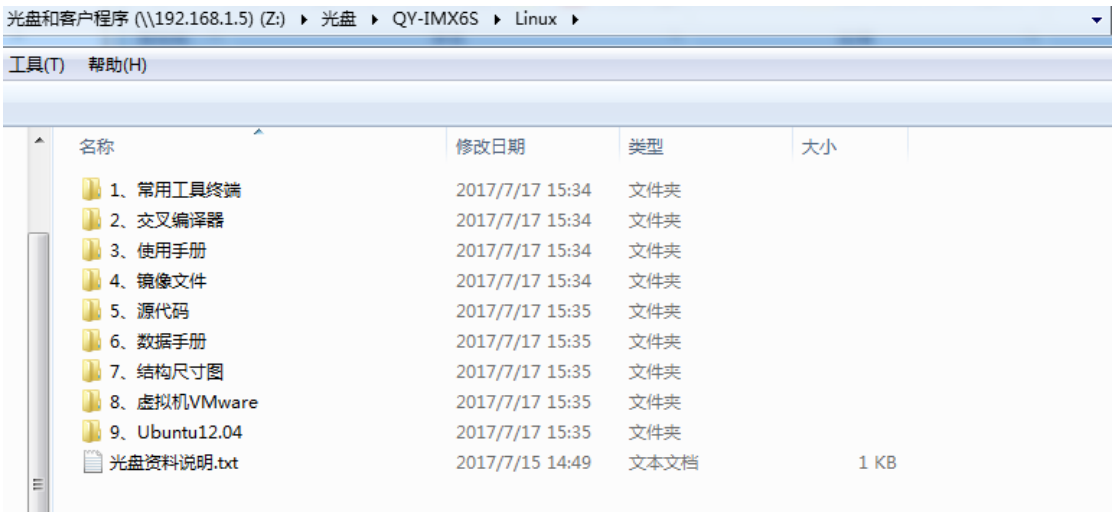
◆ 串口连接：通过提供的 3 PIN 调试串口线与 PC 机的串口连接，将调试串口线接开发板的 J6

◆ 网络连接：通过网线将开发板的以太网接口(J13)与 PC 机的网络接口连接

◆ USB 连接：通过 USB 连接线将开发板的 USB Device(J10)与 PC 机的 USB 连接

◆ 串口设置：打开终端通讯软件（迷你终端、Windows 下的超级终端或者 SecureCRT 工具），选择所用到的串口并设置如下参数：波特率（115200）、数据位（8 位）、停止位（1 位）、校验位（无）、数据流控制（无）

◆ 开发板标配光盘目录,文档说明中用到的所有工具软件以及代码文件全部在光盘的对应目录下，使用前请确保光盘资料齐全!



二、烧写 Linux 系统镜像

IMX6 有专门的烧写工具 Mfgtools，请根据需要选择最合适的启动方式进行烧写，具体烧写方法请参照：

《QY-IMX6S Linux 系统镜像烧写手册.pdf》

三、功能说明与测试

板子文件系统中已经集成测试程序，启动板子后可在/usr/test 目录下找到相应的测试程序，具体测试方法请参照：

《QY-IMX6S Linux 功能说明与测试手册.pdf》

四、安装交叉编译器

bootloader、kernel、fs 的重新编译都需要用到交叉编译器，所有的应用程序以及库文件需要在开发板上运行的话也需要交叉编译器进行编译。所以，在这里需要先安装一下交叉编译工具链，在光盘的编译工具的目录下已经有制作好的交叉编译工具，用户可以直接安装使用，该交叉编译器的 GCC 版本为 4.6.2。

下面就来介绍一下如何安装交叉编译工具链：

将fsl-linaro-toolchain.tar.gz这个交叉编译工具链拷贝到~/work目录下。

```
st@st-virtual-machine:~/work$ ls
fsl-linaro-toolchain.tar.gz
st@st-virtual-machine:~/work$
```

用如下命令进行解压：

$ tar -xzvf fsl-linaro-toolchain.tar.gz

在当前目录下生成fsl-linaro-toolchain文件夹。

```
fsl-linaro-toolchain/native/usr/libexec/gcc/arm-fsl-linux-gnueabi/4.6.2/cc1plus
fsl-linaro-toolchain/native/usr/libexec/gcc/arm-fsl-linux-gnueabi/4.6.2/cc1
fsl-linaro-toolchain/native/usr/libexec/gcc/arm-fsl-linux-gnueabi/4.6.2/collect2
fsl-linaro-toolchain/native/usr/include/
fsl-linaro-toolchain/native/usr/include/bfdlink.h
fsl-linaro-toolchain/native/usr/include/symcat.h
fsl-linaro-toolchain/native/usr/include/dis-asm.h
fsl-linaro-toolchain/native/usr/include/ansidecl.h
fsl-linaro-toolchain/native/usr/include/bfd.h
fsl-linaro-toolchain/native/usr/include/plugin-api.h
fsl-linaro-toolchain/include/
st@st-virtual-machine:~/work$ ls
fsl-linaro-toolchain  fsl-linaro-toolchain.tar.gz
st@st-virtual-machine:~/work$
```

为了使用方便，需要把交叉编译器的路径添加到系统环境变量 PATH 里面，在这里添加到当前用户的 bash.bashrc 下面就可以了。

$ vi ~/.bashrc

在文件后面添加以下路径：

export PATH=/home/st/work/fsl-linaro-toolchain/bin:$PATH

```
st@st-virtual-machine: ~/work
#    sleep 10; alert
alias alert='notify-send --urgency=low -i "$([ $? = 0 ] && echo terminal || echo
 error)" "$(history|tail -n1|sed -e '\''s/^\s*[0-9]\+\s*//;s/[;&|]\s*alert$//'\''
')"'

# Alias definitions.
# You may want to put all your additions into a separate file like
# ~/.bash_aliases, instead of adding them here directly.
# See /usr/share/doc/bash-doc/examples in the bash-doc package.

if [ -f ~/.bash_aliases ]; then
    . ~/.bash_aliases
fi

# enable programmable completion features (you don't need to enable
# this, if it's already enabled in /etc/bash.bashrc and /etc/profile
# sources /etc/bash.bashrc).
if [ -f /etc/bash_completion ] && ! shopt -oq posix; then
    . /etc/bash_completion
fi
export PATH=/home/st/work/fsl-linaro-toolchain/bin:$PATH
```

保存，退出！使新的环境变量生效

$ source ~/.bashrc

环境变量生效之后，下面我们来确认一下交叉编译器是否安装成功：

$ arm-fsl-linux-gnueabi-gcc -v

```
st@st-virtual-machine: ~/work
st@st-virtual-machine:~/work$ arm-fsl-linux-gnueabi-gcc -v
Using built-in specs.
COLLECT_GCC=arm-fsl-linux-gnueabi-gcc
COLLECT_LTO_WRAPPER=/home/st/work/fsl-linaro-toolchain/bin/../libexec/gcc/arm-fs
l-linux-gnueabi/4.6.2/lto-wrapper
Target: arm-fsl-linux-gnueabi
Configured with: /work/build/.build/src/gcc-linaro-4.6-2011.06-0/configure --bui
ld=i686-build_pc-linux-gnu --host=i686-build_pc-linux-gnu --target=arm-fsl-linux
-gnueabi --prefix=/work/fsl-linaro-toolchain-2.13 --with-sysroot=/work/fsl-linar
o-toolchain-2.13/arm-fsl-linux-gnueabi/multi-libs --enable-languages=c,c++ --wit
h-pkgversion='Freescale MAD -- Linaro 2011.07 -- Built at 2011/08/10 09:20' --en
able-__cxa_atexit --disable-libmudflap --disable-libgomp --disable-libssp --with
-gmp=/work/build/.build/arm-fsl-linux-gnueabi/build/static --with-mpfr=/work/bui
ld/.build/arm-fsl-linux-gnueabi/build/static --with-mpc=/work/build/.build/arm-f
sl-linux-gnueabi/build/static --with-ppl=/work/build/.build/arm-fsl-linux-gnueab
i/build/static --with-cloog=/work/build/.build/arm-fsl-linux-gnueabi/build/stati
c --with-libelf=/work/build/.build/arm-fsl-linux-gnueabi/build/static --with-hos
t-libstdcxx='-static-libgcc -Wl,-Bstatic,-lstdc++,-Bdynamic -lm -L/work/build/.b
uild/arm-fsl-linux-gnueabi/build/static/lib -lpwl' --enable-threads=posix --enab
le-target-optspace --enable-plugin --enable-multilib --with-local-prefix=/work/f
sl-linaro-toolchain-2.13/arm-fsl-linux-gnueabi/multi-libs --disable-nls --enable
-c99 --enable-long-long --with-system-zlib
Thread model: posix
gcc version 4.6.2 20110630 (prerelease) (Freescale MAD -- Linaro 2011.07 -- Buil
t at 2011/08/10 09:20)
st@st-virtual-machine:~/work$
```

从这里可以看出安装好的交叉编译器的gcc版本为 4.6.2。

交叉编译器安装好之后，就可以用来编译系统源码和应用程序了。

五、编译测试源码

在光盘\Linux\5、源代码\app 目录下，提供了各测试源码，可根据需要进行修改编译。

buzzer_test	文件夹	2017-04-14 15:39
can_test	文件夹	2017-04-17 16:59
ds18b20_test	文件夹	2017-04-14 11:10
gpio_test	文件夹	2017-04-26 10:41
i2c_test	文件夹	2017-04-14 11:10
include	文件夹	2017-04-17 14:42
iperf_test	文件夹	2017-04-14 11:10
keyboard_test	文件夹	2017-04-14 11:10
keybutton_test	文件夹	2017-04-14 11:10
rs232_test	文件夹	2017-04-26 11:36
rs485_test	文件夹	2017-04-28 11:21
rtc_test	文件夹	2017-04-14 15:36
spi_test	文件夹	2017-04-26 18:25
watchdog_test	文件夹	2017-04-17 11:37

此处以蜂鸣测试程序 buzzer_test 为例进行介绍。

在~/work 目录下创建 app 文件夹，并进入 app 文件夹

$ mkdir app

$ cd app

将测试源码/buzzer_test 文件夹以及共用 include 头文件夹拷贝到 app 目录下，并且进入 app 目录。

$ ls

```
st@st-virtual-machine:~/work/app$ ls
buzzer_test  include
st@st-virtual-machine:~/work/app$
```

进入 buzzer_test 文件夹

$ cd buzzer_test

$ ls

```
st@st-virtual-machine:~/work/app/buzzer_test$ ls
buzzer_test  buzzer_test.c  Makefile
st@st-virtual-machine:~/work/app/buzzer_test$
```

说明： buzzer_test 为编译好的可执行应用程序

buzzer_test.c 为测试源代码

其中有我们做好的 Makefile

编译之前先清理之前编译的内容

$ make clean

```
st@st-virtual-machine:~/work/app/buzzer_test$ ls
buzzer_test  buzzer_test.c  Makefile
st@st-virtual-machine:~/work/app/buzzer_test$ make clean
已删除"buzzer_test"
st@st-virtual-machine:~/work/app/buzzer_test$ ls
buzzer_test.c  Makefile
st@st-virtual-machine:~/work/app/buzzer_test$
```

编译测试程序

$ make

```
st@st-virtual-machine:~/work/app/buzzer_test$ ls
buzzer_test.c  Makefile
st@st-virtual-machine:~/work/app/buzzer_test$ make
arm-none-linux-gnueabi-gcc -o buzzer_test buzzer_test.c
st@st-virtual-machine:~/work/app/buzzer_test$ ls
buzzer_test  buzzer_test.c  Makefile
st@st-virtual-machine:~/work/app/buzzer_test$ file buzzer_test
buzzer_test: ELF 32-bit LSB executable, ARM, version 1 (SYSV), dynamically linke
d (uses shared libs), for GNU/Linux 2.6.31, not stripped
st@st-virtual-machine:~/work/app/buzzer_test$
```

生成的 buzzer_test 就是编译得到可在开发板上执行的测试应用程序。

六、编译 U-Boot

光盘\ Linux\5、源代码\uboot 目录下的 uboot 源码，用户可直接编译使用。

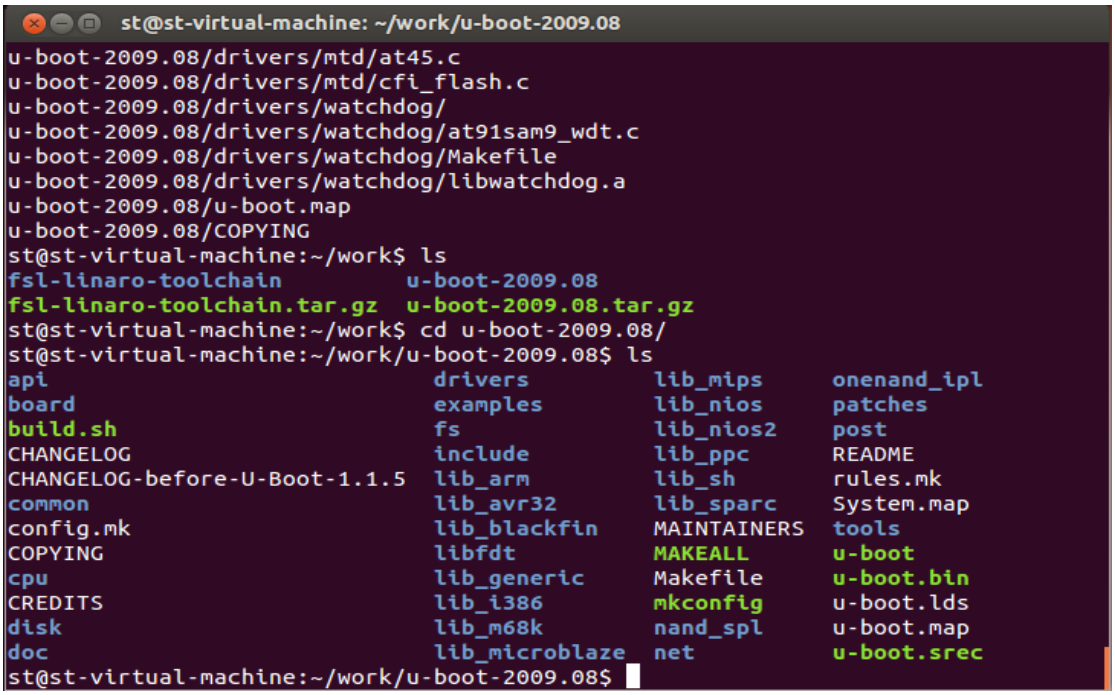
将光盘中的 u-boot 源码拷贝到~/work 工作目录中，并用以下命令解压：

$ tar -xjvf qiyang_uboot_QY_IMX6S_V1.2_XXXX.tar.bz2

解压之后得到 uboot 源码文件夹，进入该文件夹

$ cd qiyang_uboot

$ ls



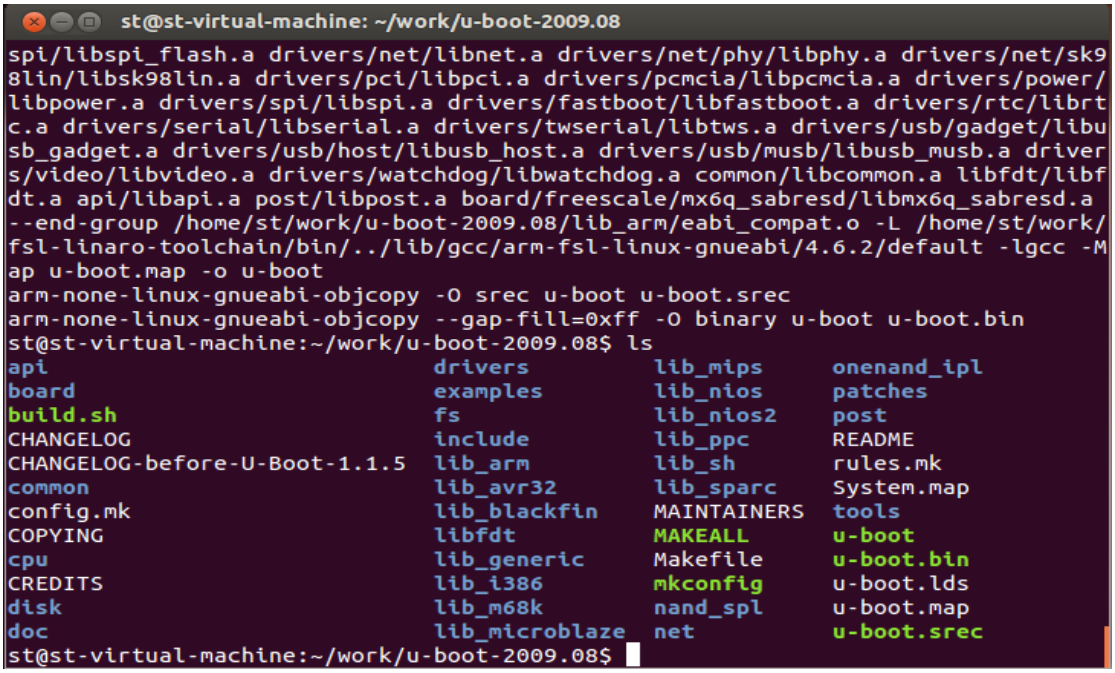
执行编译命令

$ make distclean

$make mx6q_qiyang_config

$make

执行之后开始编译，编译过程可能持续 1~3 分钟，编译完成之后在当前目录下生成可烧写到开发板的镜像文件 u-boot.bin。



七、编译内核

光盘中有移植配置好的内核源码文件。

将光盘\Linux\5、源代码\kernel 目录下的 kernel 源码文件拷贝到~/work 工作目录中，解压内核源码

$ tar -xjvf qiyang_kernel_IMX6S_V1.2_XXXX.tar.bz2

解压之后，在当前目录下生成 linux-3.0.35 文件夹，进入该文件夹

$ cd qying_kernel

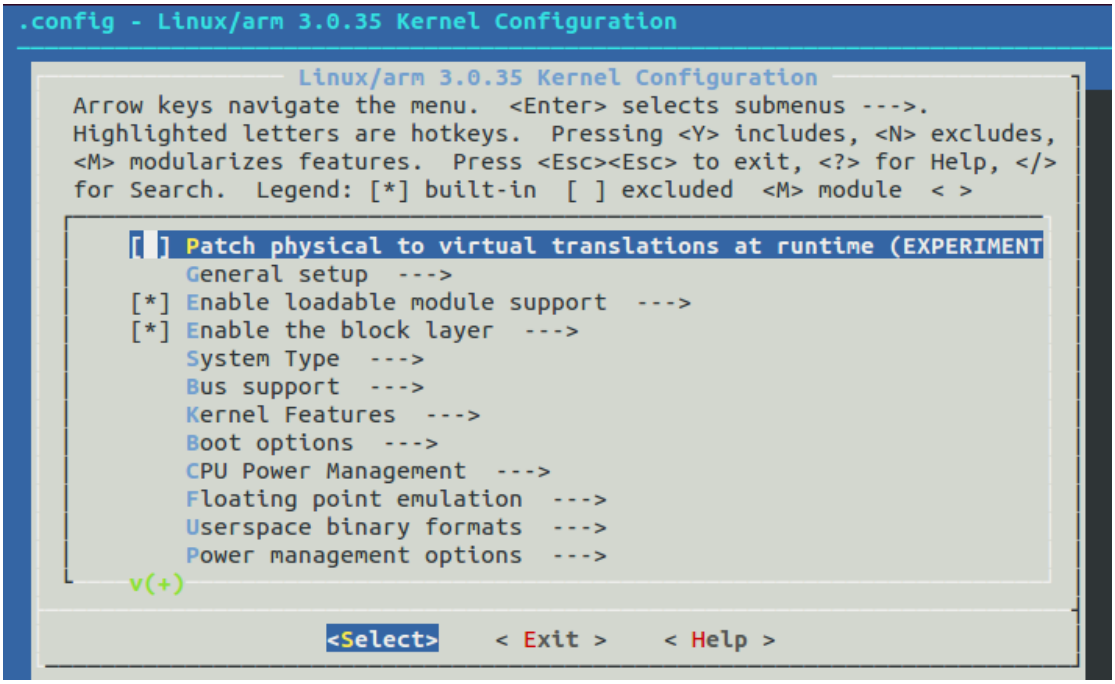
$ ls

```
st@st-virtual-machine:~/work$ ls
fsl-linaro-toolchain          linux-3.0.35.tar.gz  u-boot-2009.08.tar.gz
fsl-linaro-toolchain.tar.gz   mkimage
linux-3.0.35                  u-boot-2009.08
st@st-virtual-machine:~/work$ cd linux-3.0.35/
st@st-virtual-machine:~/work/linux-3.0.35$ ls
arch            Kbuild              linux-3.0.35.PS            samples
block           Kconfig             linux-3.0.35.SearchResults scripts
COPYING         kernel              linux-3.0.35.WK3           security
CREDITS         lib                 localversion               sound
crypto          linux-3.0.35.IAB    MAINTAINERS                System.map
Documentation   linux-3.0.35.IAD    Makefile                   tools
drivers         linux-3.0.35.IMB    mm                         usr
firmware        linux-3.0.35.IMD    Module.symvers             virt
fs              linux-3.0.35.PFI    net                        vmlinux
include         linux-3.0.35.PO     patches                    vmlinux.o
init            linux-3.0.35.PR     README
ipc             linux-3.0.35.PRI    REPORTING-BUGS
st@st-virtual-machine:~/work/linux-3.0.35$
```

编译之前，需要先用如下命令配置内核

$ make menuconfig

执行之后会弹出如下内核选项配置界面



用户可以对内核功能选项做必要的调整，其他的配置以及裁剪在这里就不予与详细说明，请用户根据自己实际需求来配置。如无特殊需求，直接使用默认内核选项配置来编译内核就可以了。

配置好之后保存退出。

退出的时候，请选择“YES”来保存配置，否则会出现如下错误：

```
st@st-virtual-machine:~/work/linux-3.0.35$ make uImage
  HOSTLD  scripts/kconfig/conf
scripts/kconfig/conf --silentoldconfig Kconfig
***
*** Configuration file ".config" not found!
***
*** Please run some configurator (e.g. "make oldconfig" or
*** "make menuconfig" or "make xconfig").
***
make[2]: *** [silentoldconfig] 错误 1
make[1]: *** [silentoldconfig] 错误 2
make: *** 没有规则可以创建“include/config/kernel.release”需要的目标“include/conf
ig/auto.conf”。  停止。
```

开始编译内核镜像

$ make uImage

执行之后开始编译，第 1 次编译可能持续比较长的时间，请耐心等待！

当编译完成开始生成 uImage 的时候如果提示如下错误：

```
st@st-virtual-machine: ~/work/linux-3.0.35
  LD      .tmp_vmlinux1
  KSYM    .tmp_kallsyms1.S
  AS      .tmp_kallsyms1.o
  LD      .tmp_vmlinux2
  KSYM    .tmp_kallsyms2.S
  AS      .tmp_kallsyms2.o
  LD      vmlinux
  SYSMAP  System.map
  SYSMAP  .tmp_System.map
  OBJCOPY arch/arm/boot/Image
  Kernel: arch/arm/boot/Image is ready
  AS      arch/arm/boot/compressed/head.o
  GZIP    arch/arm/boot/compressed/piggy.gzip
  AS      arch/arm/boot/compressed/piggy.gzip.o
  CC      arch/arm/boot/compressed/misc.o
  CC      arch/arm/boot/compressed/decompress.o
  SHIPPED arch/arm/boot/compressed/lib1funcs.S
  AS      arch/arm/boot/compressed/lib1funcs.o
  LD      arch/arm/boot/compressed/vmlinux
  OBJCOPY arch/arm/boot/zImage
  Kernel: arch/arm/boot/zImage is ready
  UIMAGE  arch/arm/boot/uImage
"mkimage" command not found - U-Boot images will not be built
make[1]: *** [arch/arm/boot/uImage] 错误 1
make: *** [uImage] 错误 2
st@st-virtual-machine:~/work/linux-3.0.35$
```

上图提示缺少 mkimage 命令，生成内核镜像需要用到 mkimage 工具，刚才已经把 mkimage 工具拷贝到工作目录中，需要把它添加到系统环境变量中，以便系统能够自动调用。在这里简单一点，把 mkimage 拷贝到交叉编译器的 bin 目录下。

$ cp ../mkimage ~/work/fsl-linaro-toolchain/bin/

现在我们就可以执行编译命令，顺利进行编译内核镜像

$ make uImage

```
st@st-virtual-machine:~/work/linux-3.0.35$ make uImage
  CHK     include/linux/version.h
  CHK     include/generated/utsrelease.h
make[1]: “include/generated/mach-types.h”是最新的。
  CALL    scripts/checksyscalls.sh
  CHK     include/generated/compile.h
  Kernel: arch/arm/boot/Image is ready
  SHIPPED arch/arm/boot/compressed/lib1funcs.S
  AS      arch/arm/boot/compressed/lib1funcs.o
  LD      arch/arm/boot/compressed/vmlinux
  OBJCOPY arch/arm/boot/zImage
  Kernel: arch/arm/boot/zImage is ready
  UIMAGE  arch/arm/boot/uImage
Image Name:   Linux-3.0.35-2508-g54750ff
Created:      Fri Feb 21 15:19:28 2014
Image Type:   ARM Linux Kernel Image (uncompressed)
Data Size:    3853144 Bytes = 3762.84 kB = 3.67 MB
Load Address: 10008000
Entry Point:  10008000
  Image arch/arm/boot/uImage is ready
st@st-virtual-machine:~/work/linux-3.0.35$
```

编译完成后在 arch/arm/boot/目录下生成可烧写到开发板的内核镜像文件 uImage。

```
st@st-virtual-machine:~/work/linux-3.0.35$ ls arch/arm/boot/
bootp  compressed  Image  install.sh  Makefile  tftpd32.exe  uImage  zImage
st@st-virtual-machine:~/work/linux-3.0.35$
```

八、应用程序的开发

您可以在 PC 机上自由地开发应用程序，这里以最简单的 Hello Wordl 为例。在~/work 目录下创建 app 文件夹，并进入 app 文件夹

$ mkdir app

$ cd app

首先编写一个 Hello World 程序代码如下：

```
#include <stdio.h>

int main(void)

{

    printf("Hello World ! \n");

    return 0;
```

```
}
```

保存在 hello.c 文件

```
st@st-virtual-machine:~/work$ mkdir app
st@st-virtual-machine:~/work$ cd app/
st@st-virtual-machine:~/work/app$ vi hello.c
st@st-virtual-machine:~/work/app$ cat hello.c
#include <stdio.h>
int main(void)
{
        printf("Hello World ! \n");
        return 0;
}
st@st-virtual-machine:~/work/app$
```

要使得程序能够运行在开发板上，需要使用之前安装好的交叉编译器对应用程序进行编译，编译命令如下：

```
$ arm-fsl-linux-gnueabi-gcc -o hello hello.c
$ file hello
```

```
st@st-virtual-machine:~/work/app$ arm-none-linux-gnueabi-gcc -o hello hello.c
st@st-virtual-machine:~/work/app$ ls
hello  hello.c
st@st-virtual-machine:~/work/app$ file hello
hello: ELF 32-bit LSB executable, ARM, version 1 (SYSV), dynamically linked (use
s shared libs), for GNU/Linux 2.6.31, not stripped
st@st-virtual-machine:~/work/app$
```

编译后可以看到在当前目前目录下生成了用于 ARM 平台的可执行的二进制文件 hello。

接下来把 hello 这个可执行程序通过 SD、U 盘、tftp 下载或者 nfs 挂载的方式拷贝到开发板，就可以在开发板上执行 hello 程序了，拷贝过程在这里不予以详细介绍！

九、添加应用程序到文件系统

为了生产方便，一般我们都把编译测试好的应用程序、库以及配置文件文件打包到文件系统当中，这样生产的时候只要烧写文件系统就可以了，不需要手动去添加应用程序、库以及配置文件。

下面我们就来介绍一下如何将应用程序添加到文件系统中。

光盘中有制作好的的文件系统源码和制作工具。

将光盘\Linux\5、源代码\filesystem 目录下的文件系统源码和制作工具拷贝到~/work 工作目录下

在~/work 目录下创建一个 fs 文件夹

$ mkdir fs

将文件系统源码 rootfs.tar.bz2 移动到 fs 文件夹中

$ mv qiyang_filesystem_QY_IMX6S_V1.2_XXXX.tar.bz2 fs/

进入 fs 文件夹并解压 rootfs.tar.bz2

文件系统源码需要 root 权限才能进行完整解压，在解压命令之前加上 sudo

$ cd fs

$ sudo tar -xjvf qiyang_filesystem_QY_IMX6S_V1.2_XXXX.tar.bz2

解压后如下图

```
st@st-virtual-machine:~/work/fs$ ls
bin      dev   home   linuxrc  mnt   proc  rootfs.tar.bz2  sbin      sys  usr
config   etc   lib    media    opt   root  run             Settings  tmp  var
st@st-virtual-machine:~/work/fs$
```

在 fs 目录下各目录添加自己的应用程序、库以及配置文件。

删除原先的文件系统 rootfs.tar.bz2

$ rm qiyang_filesystem_QY_IMX6S_V1.2_XXXX.tar.bz2

重新压缩文件系统

$ sudo tar -jcvf rootfs.tar.bz2 -R *

压缩好之后在当前 fs 目录下重新生成一个 rootfs.tar.bz2 文件

```
块 569295:  var/
块 569296:  var/lock/
块 569297:  var/db/
块 569298:  var/db/Makefile
块 569310:  var/log/
块 569311:  var/log/messages
块 569325:  var/log/wtmp
块 569386:  var/log/dmesg
块 569415:  var/tmp/
块 569416:  var/run/
块 569417:  var/run/ifstate
块 569418:  var/run/utmp
st@st-virtual-machine:~/work/fs$ ls
bin      dev   home   linuxrc  mnt   proc  rootfs.tar.bz2  sbin      sys  usr
config   etc   lib    media    opt   root  run             Settings  tmp  var
st@st-virtual-machine:~/work/fs$
```

把该文件烧写到开发板中，开发板启动之后，文件系统对应的目录下就有之前添加的应用程序、库以及配置文件了。

九、结束语

以上内容可能不够翔实，有任何技术问题或建议，欢迎联系我们：supports@qiyangtech.com，也可登录我司论坛进行交流：http://www.qiytech.com/bbs/，关于更多产品的信息，欢迎联系销售 sales@qiyangtech.com ，或登录 http://www.qiytech.com/index.html。

浙江启扬智能科技有限公司

电话：0571-87858811 / 87858822

传真：0571-89935912

技术支持：0571-87858811 转 805

E-MAIL：supports@qiyangtech.com

网址： http://www.qiyangtech.com

地址：杭州市西湖科技园西园一路 8 号 3A 幢 5 层

邮编：310012