IAC-IM6-Kit Linux User Manual

Version No:V2.0

2014.11

QIYANG INTELLIGENT TECHNOLOGY CO., LTD

Sales email: trade@qiyangtech.com; sales@qiyangtech.com

Website：http://www.qiytech.com

Catalogue

Version Update

Version	Hardware	Description	Date	Revisor
1.0	IAC-IM6X-KIT	Launched	2014-02-21	wujj
2.0	IAC-IM6X-KIT	Update hardware version	2014-11-11	wangwx

Preface

Welcome to use IAC-IMX6-KIT from Zhejiang Qiyang Intelligent Technology Co., Ltd.

Here are 4 Linux manual for reference:

IAC-IMX6-KIT User manual.pdf

IAC-IMX6-KIT Hardware Manual.pdf.

IAC-IMX6-KIT Functions and test manual.pdf

IAC-IMX6-KIT Image burning manual.pdf

• This manual mainly introduce cross-compilation environment construction, source code and compilation of application routine.

•Before using, please read ***IAC-IMX6-KIT Hardware Manual.pdf.***

•Please read this manual carefully before using.

Company Profile:

Zhejiang Qiyang Technology Co., Ltd. is located at the bank of the beautiful West Lake. It is a high and new technology enterprise which is specializing in R&D, manufacture and sell embedded computer main board with high performance, low power consumption, low cost, small volume, and provides embedded hardware solutions.

We Offer:

◆ Research & develop, manufacture and sell embedded module products which have independent intellectual property rights, and cooperate with TI, ATMEL, Cirrus Logic, Freescale, and other famous processor manufacturers. It has launched a series of hardware products, such as ARM development board, ARM core module, ARM industrial board, sound/video decoding transmission platform, supporting tools and software resources which support user for their next embedded design.

◆ We give full play to the technical accumulation in ARM platform and Windows CE, Linux, Android operating system for many users providing custom service (OEM/ODM), to realize embedded products into the market stably, reliably and quickly.

Tel: +86 571 87858811, +86 571 87858822

Fax: +86 571 87858822

Technology Support E-mail: support@qiyangtech.com

Website: http://www.qiytech.com

Address: 5F, Building 3A, No.8 Xiyuanyi Road, West Lake Science Park, Hangzhou, China

Post code: 310030

Ⅰ.Illustration

◆ Build in Linux OS (ubuntu or other Linux release version). Operation Example: ubuntu 12.04. Installation steps, please refer to *Ubuntu Installation for Virtual Machine Manual.PDF*

◆ Copy file to virtual machine [ubuntu] while it is in compiling process, create a directory[mkdir~/work /*], [~]means user catalogue; Absolute Path is[/home/st*/].

All documentations are copied to this directory, users could create directory by themselves. Here just the Example:[~/work]

◆ Please refer to relevant materials about the common commands and vi operation in Linux.

◆ All of the copies of PC and virtual machine adopt samba shared access mode.

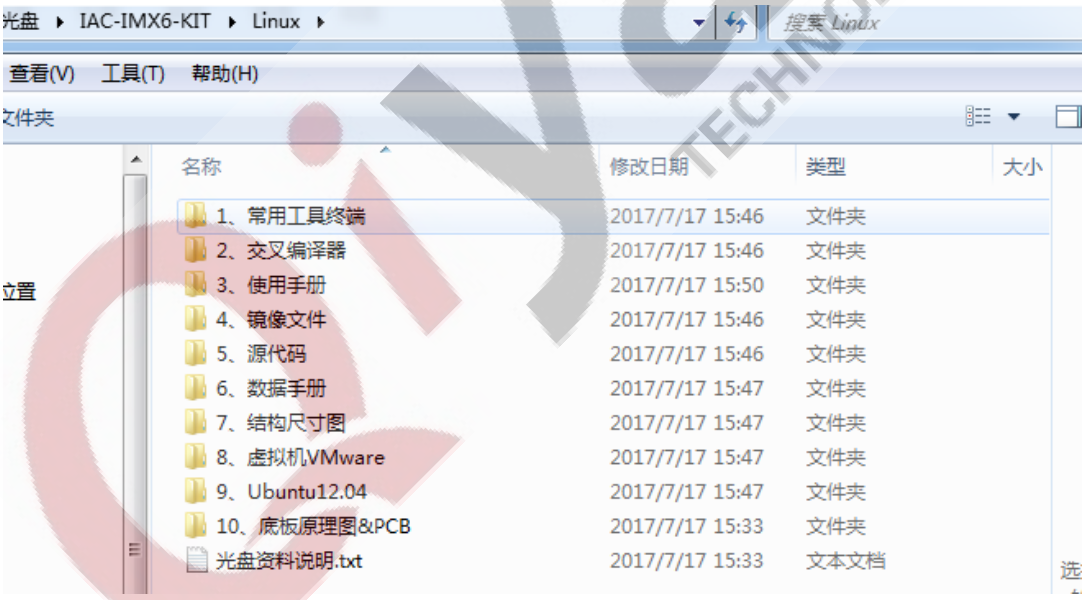
◆ Serial Connect: Use the provided 3 PIN debug port line to connect to PC mainframe's serial port, then the debug port line connect to mainboard's debug port (J7).

◆Network Connect: Connect Ethernet Interface (J8) to Network Interface on PC by network cable.

◆ USB Connect: Connect USB Device(J13) to USB on PC by USB cable

◆ Set Serial Port: Open terminal communication software (minicom or hyper terminal in Windows), select baud rate [115200], stop bit [1], data bit [8], parity bit [none] and data flow control[none]. Then test every serial ports.

◆ Mainboard has CD catalogue, the tools software and code file are in corresponding catalogue in CD. Please ensure that the materials are all in readiness.



Ⅱ.Program Linux System Image

IMX6 has its special programming tool [Mfgtools], please choose the most suitable boot method to burn.

Specific boot method, please refer to ***IAC-IMX6-KIT Linux System Image Burning Manual .pdf.***

Ⅲ.Function and Test

File system has integrated test program, after booting, you will find the corresponding test program under the [/user/test] directory.

Specific test method, please refer to ***IAC-IMX6-KIT Linux Function and Test Manual.pdf.***

Ⅳ .Install Cross-Compiler Tool Chains

[Bootloader], [kernel] and [fs] need to use the cross-compiler.

All application programs and library files need cross-compiler to compile if running on the mainboard. So we will install the cross-compiler tool chain at first, there is a finished cross-compiler tool in CD. User could use it directly. The GCC version is 4.6.2.

Next, we will introduce "How to install Cross-Compiler Tool Chains?"

Copy [fsl-linaro-toolchain.tar.gz] cross-compiler tool chains to [~/work]directory.

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

Use the following command to extract:

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

[fsl-linaro-toolchain] will be generated in current directory

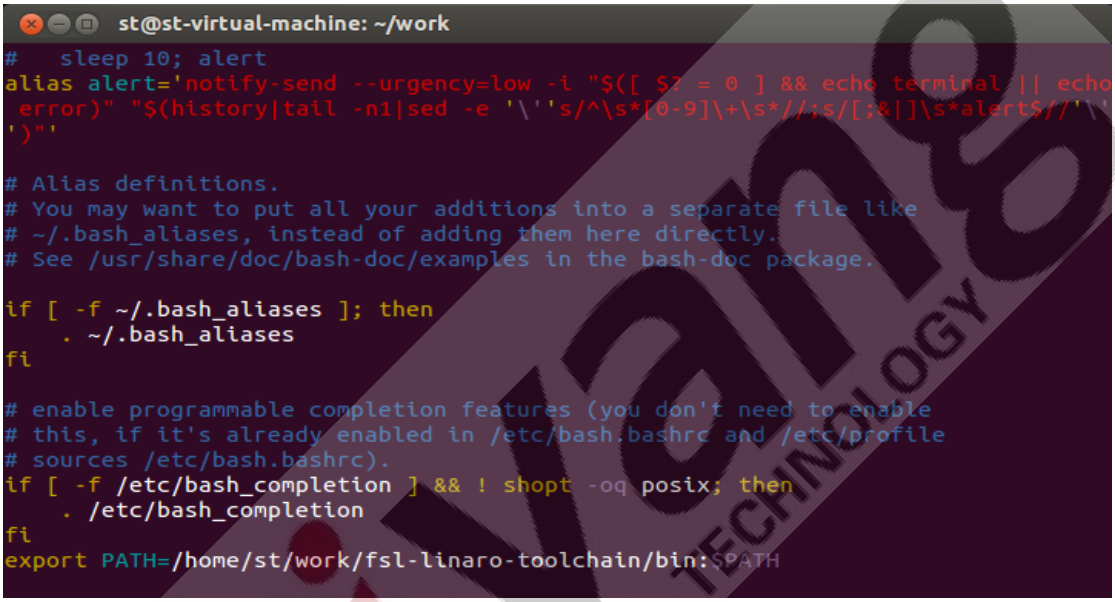
```
fsl-linaro-toolchain/native/usr/libexec/gcc/arm-fsl-linux-gnueabi/4.6.2/cc1plus
fsl-linaro-toolchain/native/usr/libexec/gcc/arm-fsl-linux-gnueabi/4.6.2/cc1
fsl-linaro-toolchain/native/usr/libexec/gcc/arm-fsl-linux-gnueabi/4.6.2/collect2
fsl-linaro-toolchain/native/usr/include/
fsl-linaro-toolchain/native/usr/include/bfdlink.h
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$
```

Add this cross-compilers' path to system environment variable [PATH], and add to current user's [bash.bashrc].

$ vi ~/.bashrc

Add the following path in file:

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



Save & Exit!

Make the new environment variable effective.

$ source ~/.bashrc

After the environment variables taking effect, we confirm whether the cross-compiler is installed successfully：

$ 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$
```

As shown, GCC version is 4.6.2.

So far, our cross-compiler are totally installed, then we could use it to compile our source code and application program.

Ⅴ.Compile Test Code

Provide all test codes in \Linux\5 and source code\app directory. You can modify and compile according to your own need.

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

Here we take an example of Buzzer test program [buzzer_test] to introduce.

Create [app] file in [~/work], then enter into [app] file.

$ mkdir app

$ cd app

Copy CD/Test code [/buzzer_test] file and [include] file to [app] directory, then enter into [app] directory.

$ ls

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

Enter into [buzzer_test] file.

$ 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$
```

Note: [buzzer_test] is compiled executable application

[Buzzer_test.c] is test code

Including our [Makefile].

Please clear previous compiled content before starting compiling.

$ 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$
```

Compile test program

$ 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$
```

The [buzzer_test] is the executable test application program in our mainboard.

Ⅵ .Compile u-boot

The migrated [uboot] source code is in CD\Linux\5 and source code\uboot directory, users can compile it directly.

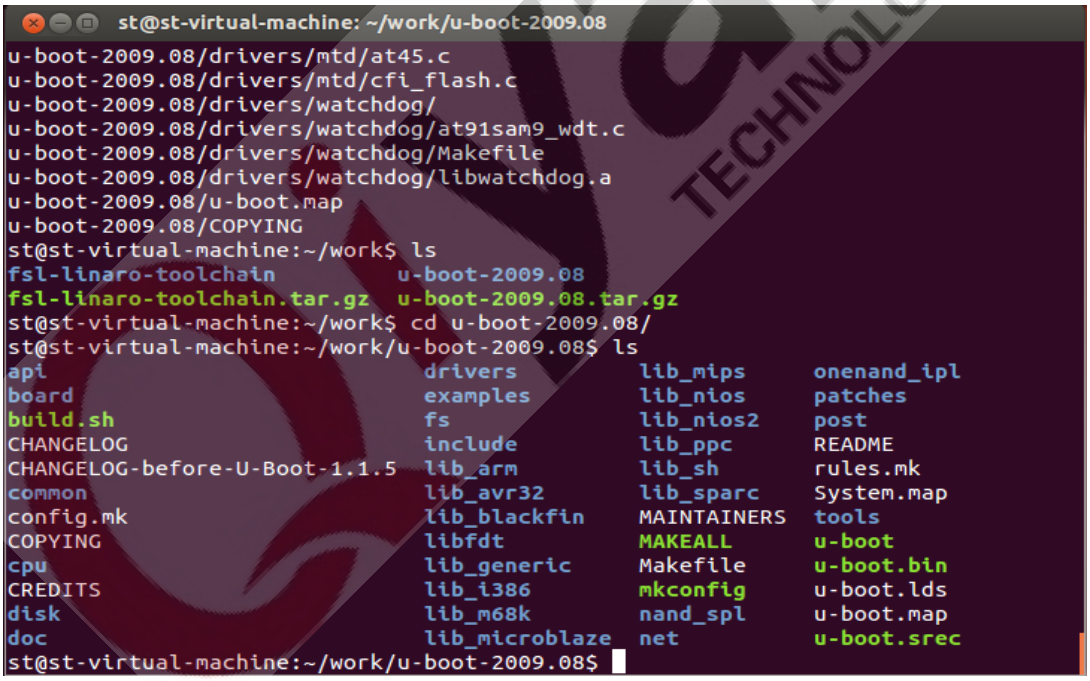
Copy source code of [u-boot] in CD to [~/work]directory, then extract by the following command:

$ tar -xzvf u-boot-2009.08.tar.gz

After Unzip, get the u-boot-at91-2012.10 folder, Enter this folder.

$ cd u-boot-at91-2012.10

$ ls



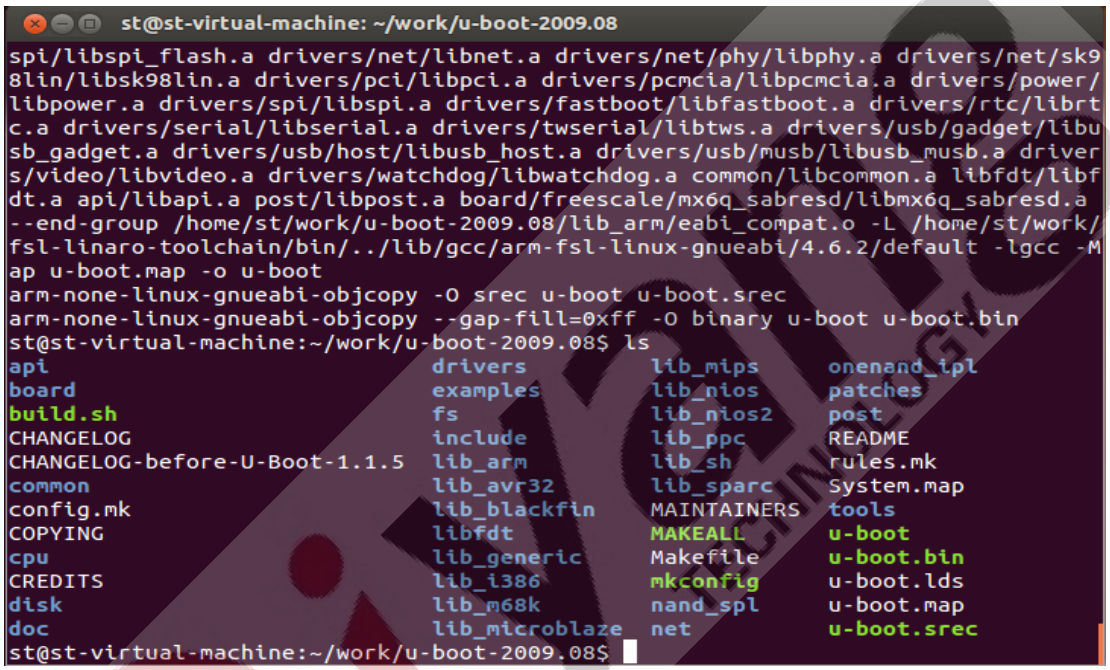
Perform compilation command:

$ make distclean

$make mx6q_qiyang_config

$make

After executing, then compile ,the compiling process will keep about 1-3 minutes. After compiling, the directory will generate image file[u-boot.bin] that can be burnt into mainboard.



Ⅶ .Compile Kernel

There are configured kernel source files in CD.

Copy kernel source code under \Linux\5, source code\kernel directory to ~/work directory, unzip the kernel source code:

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

After unzip, generated the linux-3.0.35 folder, enter this folder

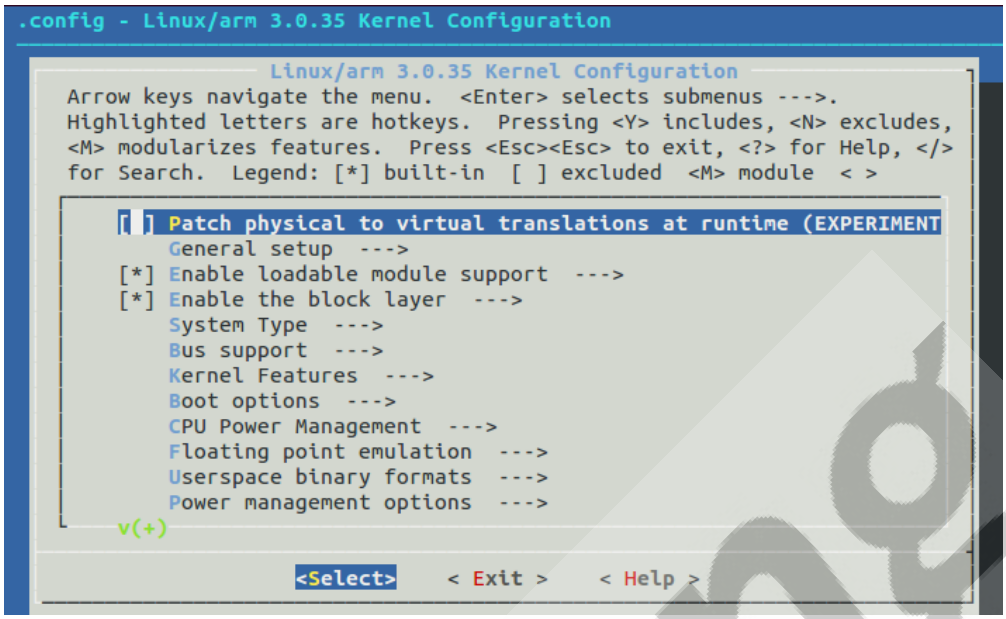
$ cd qiyang_kernel

$ 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$
```

Before compiling, you need to configure kernel with the following command: $ make menuconfig

After executing, it will popup the following kernel option configuration interface.



Users can make adjustment in kernel function option, about the other configuration and cutting, users can configure them according to your own needs. If you do not have any other special needs, you can use the defaulted kernel option configuration to compile kernel.

Save and exit

Before exiting, please choose "YES" to save configuration. If not, it will hints error as shown:

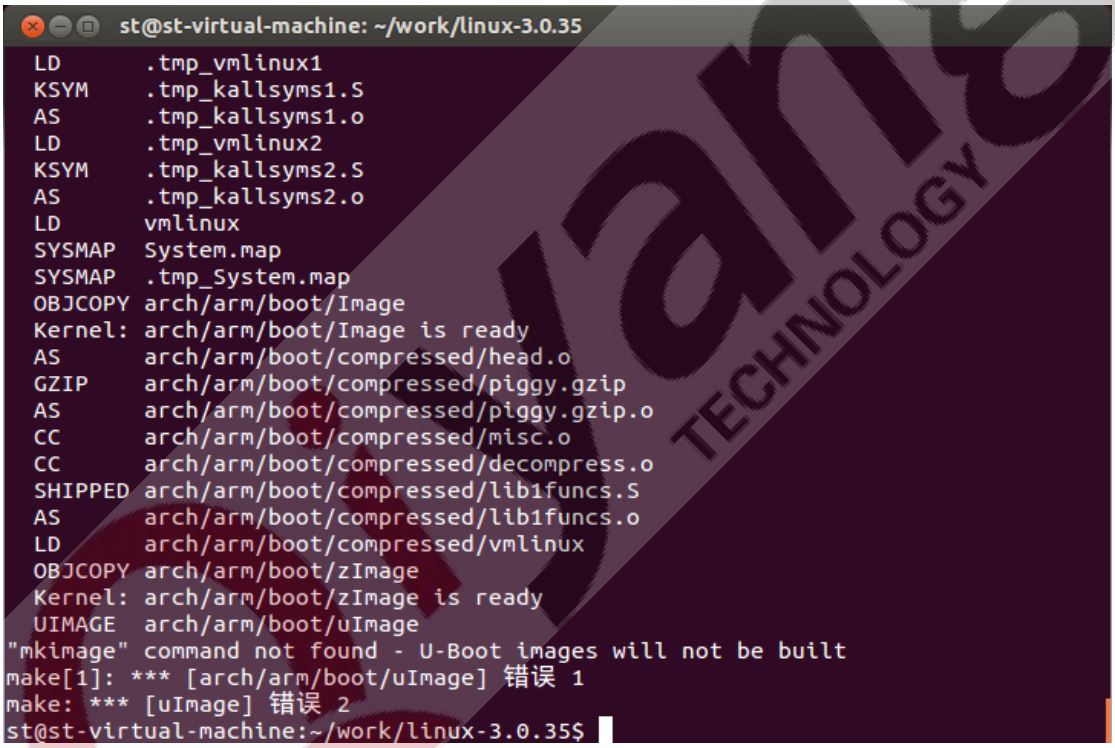
```
st@st-virtual-machine:~/work/linux-3.0.35$ make uImage
  HOSTLD  scripts/kconfig/conf
scripts/kconfig/conf --silentoldconfig Kconfig
***
*** Configuration file ".config" not found!
***
*** Please run some configurator (e.g. "make oldconfig" or
*** "make menuconfig" or "make xconfig").
***
make[2]: *** [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。
```

Start to compile kernel image

$ make uImage

Start to compile after executing. The initial compiling may need a certain time, Please be patient!

After finishing compiling, generate [uImage], if it hints the following errors:



Above picture shows lacking [mkimage] command, it needs [mkimage] tool to generate kernel image. Just now we have copied [mkimage] tool to work directory, we should add it to system environment variables, so that the system can use automatically. To be

brief, copy [mkimage] to [bin] directory of cross compiler.

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

Now we can execute compile command to compile kernel image smoothly.

$ 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$
```

After finishing compiling, generate kernel image file [uImage] in [arch/arm/boot/] directory which could be burnt into mainboard.

```
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$
```

Ⅷ .Develop Application Program

You can develop application program in PC. Here is the sample [Hello World]. At first, create [app] folder in [~/work] directory, then enter into the [app] folder:

```
$ mkdir app
$ cd app
```

At first, compile [Hello World] program code as follows:

```
#include <stdio.h>
int main(void)
{
    printf("Hello World ! \n");
    return 0;
}
```

Save to [hello.c] file.

```
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$
```

Use the installed cross-compiler to compile the application program. Use the following command to compile:

$ 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$
```

It will generate executable binary file in current directory.

Next, copy the executable program [hello] to mainboard through SD, USB Hardware Disk, tftp, or nfs. Then we could execute the [hello] program in mainboard.

Ⅸ .Add Application Program to File System

As usual, the application programs, libraries and configuration files are placed in file system. Then we just need to burn the file system, do not need to add the application programs, libraries and configuration files manually.

Then we will introduce" How to add the application program to file system?"

The finished file system source code and authoring tool are in CD disk. Copy file system source code of source code\filesystem directory and authoring tool to [~/work] directory.

Create [fs] folder in [~/work] directory.

```
$ mkdir fs
```

Move file system source code[rootfs.tar.bz2] to [fs]folder.

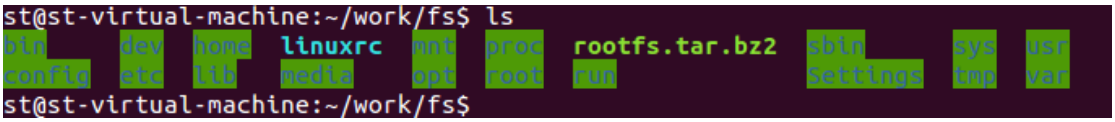
```
$ mv qiyang_filesystem_IAC_IMX6_CM_V2.03_XXXX.tar.bz2 fs/
```

Enter into [fs]folder ,and extract [rootfs.tar.bz2]

The file system needs [root] limitation, then it could do the complete extraction , add [sudo] before the extracting command.

```
$ cd fs
$ sudo tar -xjvf qiyang_filesystem_IAC_IMX6_CM_V2.03_XXXX.tar.bz2
```

After extracting, it appears as follows:



Add application programs, libraries, and configuration files to directory respectively in [fs] directory.

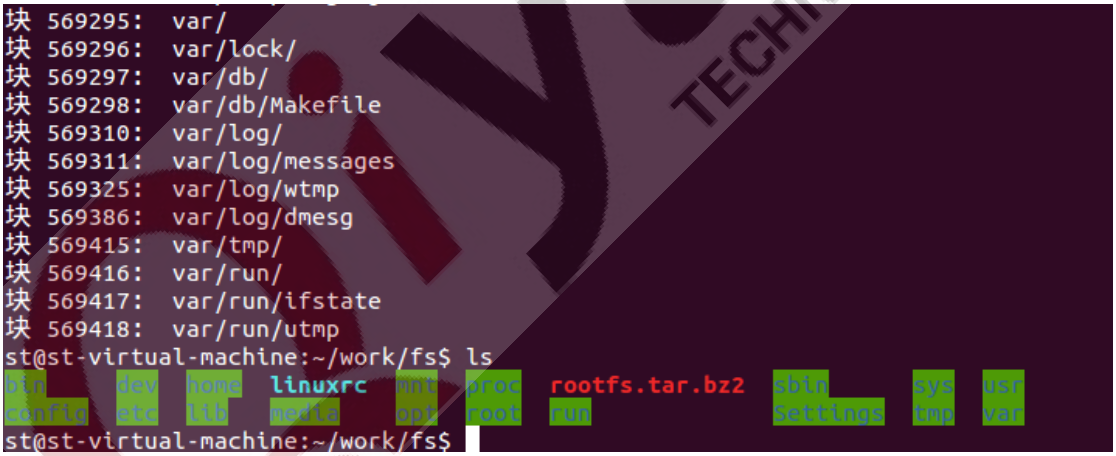
Delete the original file system[rootfs.tar.bz2].

$ rm qiyang_filesystem_IAC_IMX6_CM_V2.03_XXXX.tar.bz2

Compress file system again.

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

After compressing, regenerate [rootfs.tar.bz2] file in [fs] directory.



Burn the files into the mainboard, after booting, the mainboard, the application programs, libraries and configuration files are in the corresponding directory in file system.

Ⅹ.Conclusion

If you have any technical problems, please contact us: supports@qiyangtech.com, trade@qiyangtech.com.

Zhejiang Qiyang Intelligent Technology Co., Ltd

Tel: 86 -571-87858811 / 87858822

Fax: 86-571-89935912

Technology Support: 86-571-89935913

E-MAIL: supports@qiyangtech.com

Website: http://www.qiytech.com

Address: 5F, Building 3A, NO.8 Xiyuanyi Road, West Lake Science Park, Hangzhou, China

Post Code: 310030