



IAC-A5D3x-Kit Linux User Manual

Version:2.0

2014/10/08

QIYANG INTELLIGNET TECHNOLOGY Co., Ltd

Copyright Reserved

Catalogue

| | |
|--|----|
| Preface..... | 4 |
| I .Illustration | 5 |
| II . Burn Linux System Image | 7 |
| III. Function and Test | 8 |
| IV. Install Cross Compiler Tool Chain..... | 8 |
| V . Compile Bootstrap | 12 |
| VI. Compile uboot..... | 19 |
| VII. Compile Kernel..... | 22 |
| VIII. File System Creation..... | 29 |
| IX. Application Program Development..... | 32 |
| X . Conclusion | 33 |

Version Updates

| Version | Hardware Platform | Description | Date | Revisor |
|---------|-------------------|---|------------|---------|
| 1.0 | IAC-A5D3x-Kit | Launched | 2014-02-08 | yao |
| 2.0 | IAC-A5D3x-Kit | Modify uboot, kernel, file system version | 2014-10-08 | WWX |

Preface

Welcome to use the IAC-A5D3x-Kit from Hangzhou Qiyang Intelligent Technology Company. It include four manuals in Linux:

IAC-A5D3x-Kit Linux User Manual. pdf

IAC-A5D3x-Kit Hardware Manual. pdf

IAC-A5D3x-Kit Linux Module Specification and Test Manual. pdf

IAC-A5D3x-Kit Linux System Image Burning Manual. pdf

Hardware Part, you could refer to *IAC-A5D3x-Kit Hardware Manual. pdf*

Mainboard Test, you could refer to *IAC-A5D3x-Kit Linux Module Specification and Test Manual. pdf*

Image Burning, you could refer to *IAC-A5D3x-Kit Linux System Image Burning Manual.pdf*

This manual is used for introducing how to set up the cross compiler, source code and application demo compiling.

Before using this manual, please read *IAC-A5D3x-Kit Hardware Manual.pdf* carefully.

I .Illustration

◆ Build in Linux OS (ubuntu or other Linux release version). Operation Example: ubuntu 12.04. Installation, 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. 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: Connect Debugging UART Interface (J7) on development board to serial port on PC by serial port cable.

- ◆ Network Connect: Connect Ethernet Interface (J16) to Network Interface on PC by network cable.

- ◆ USB Connect: Connect USB Device(J22) on development board to USB on PC by USB cable

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

- ◆ Find out the J1 jumper wire, you will need to use it in the following operation steps.

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

| | | |
|------------------|-----------------|-----|
| Base Board PCB | 2015/4/22 15:28 | 文件夹 |
| Compilation Tool | 2015/4/22 15:30 | 文件夹 |
| Device Manual | 2015/4/22 16:25 | 文件夹 |
| Image | 2015/4/22 16:25 | 文件夹 |
| Schematic | 2015/4/22 15:27 | 文件夹 |
| Source Code | 2015/4/22 16:53 | 文件夹 |
| Structure & Size | 2015/4/22 16:22 | 文件夹 |
| Test Code | 2015/4/22 15:29 | 文件夹 |
| Tool Software | 2015/4/22 16:21 | 文件夹 |
| ubuntu | 2015/4/22 15:19 | 文件夹 |
| User Manual | 2015/4/22 16:50 | 文件夹 |
| Virtual Machine | 2015/4/22 15:30 | 文件夹 |

II . Burn Linux System Image

If you need to re-burn Linux system, the development board provides two boot methods: dataflash and nandflash. Please select the most suitable boot method to burn, the specific method ,please refer to this manual:

IAC-A5D3x-Kit Linux System Image Burning Manual. pdf

This manual introduces the burning method from dataflash and nandflash, and the method to update the image by network.

III. Function and Test

The development board file system has already integrated the test programs. After booting, you could find the corresponding test programs in [/usr/local/board_test]directory. The specific test method ,you could refer to this manual:

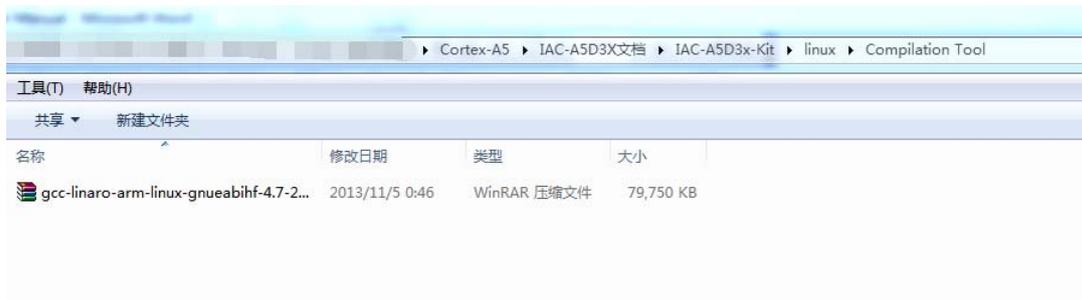
IAC-A5D3x-Kit Linux Module Specification And Test Manual. pdf

IV. Install Cross Compiler Tool Chain

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

All application programs and library files need cross-compiler to compile, they want to run on the mainboard. So, we will install the cross-compiler tool chain at first, there is a prepared cross-compiler tool in CD.

Users could use it directly. The GCC version is 4.7.3 .



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

Copy[gcc-linaro-arm-linux-gnueabi-hf-4.7-2012.12-20121214_linux.tar.bz2] cross-compiler tool chains to [~/work]directory.

```
st@st-virtual-machine:~/work$ ls
gcc-linaro-arm-linux-gnueabi-hf-4.7-2012.12-20121214_linux.tar.bz2
st@st-virtual-machine:~/work$
```

```
$ tar -xjvf gcc-linaro-arm-linux-gnueabi-hf-4.7-2012.12-20121214_linux.tar.bz2
```

Generate[gcc-linaro-arm-linux-gnueabi-hf-4.7-2012.12-20121214_linux] in current directory.

```
gnueabi-hf/4.7.3/liblto_plugin.so.0.0.0
gcc-linaro-arm-linux-gnueabi-hf-4.7-2012.12-20121214_linux/libexec/gcc/arm-linux-gnueabi-hf/4.7.3/collect2
gcc-linaro-arm-linux-gnueabi-hf-4.7-2012.12-20121214_linux/libexec/gcc/arm-linux-gnueabi-hf/4.7.3/liblto_plugin.so.0
gcc-linaro-arm-linux-gnueabi-hf-4.7-2012.12-20121214_linux/libexec/gcc/arm-linux-gnueabi-hf/4.7.3/liblto_plugin.so
st@st-virtual-machine:~/work$ ls
gcc-linaro-arm-linux-gnueabi-hf-4.7-2012.12-20121214_linux
gcc-linaro-arm-linux-gnueabi-hf-4.7-2012.12-20121214_linux.tar.bz2
st@st-virtual-machine:~/work$
```

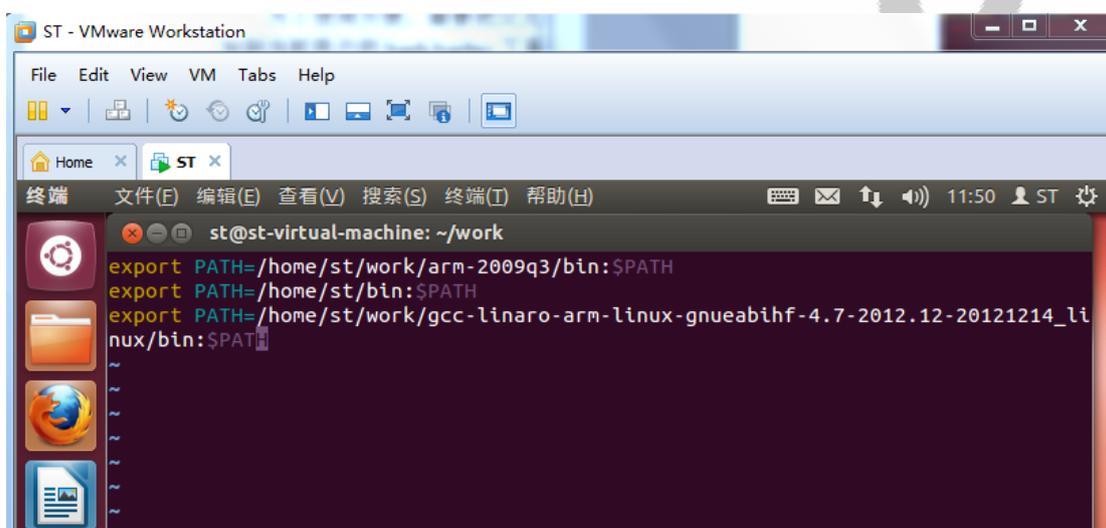
Add this cross-compilers' path to system environment variables [PATH], and add into the current user's [bash.bashrc].

```
$ vi ~/.bashrc
```

Add the following path in the file.

```
export PATH=  
/home/st/work/gcc-linaro-arm-linux-gnueabi-4.7-2012.12-20121214_linux/bin:$PATH
```

Note: The above environment variables do not need a line feed.



Save & Exit!

Make the new environment variables effective.

```
$ source ~/.bashrc
```

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

```

st@st-virtual-machine: ~/work
st@st-virtual-machine:~/work$ arm-linux-gnueabi-gcc -v
使用内建 specs。
COLLECT_GCC=arm-linux-gnueabi-gcc
COLLECT_LTO_WRAPPER=/home/st/work/gcc-linaro-arm-linux-gnueabi-4.7-2012.12-20121214_linux/bin/./libexec/gcc/arm-linux-gnueabi/4.7.3/lto-wrapper
目标: arm-linux-gnueabi
配置为: /cbuild/slaves/oorts/crosstool-ng/builds/arm-linux-gnueabi-linux/.build/src/gcc-linaro-4.7-2012.12/configure --build=i686-build_pc-linux-gnu --host=i686-build_pc-linux-gnu --target=arm-linux-gnueabi --prefix=/cbuild/slaves/oorts/crosstool-ng/builds/arm-linux-gnueabi-linux/install --with-sysroot=/cbuild/slaves/oorts/crosstool-ng/builds/arm-linux-gnueabi-linux/install/arm-linux-gnueabi/libc --enable-languages=c,c++,fortran --enable-multilib --with-arch=armv7-a --with-tune=cortex-a9 --with-fpu=vfpv3-d16 --with-float=hard --with-pkgversion='crosstool-NG linaro-1.13.1-4.7-2012.12-20121214 - Linaro GCC 2012.12' --with-bugurl=https://bugs.launchpad.net/gcc-linaro --enable-__cxa_atexit --enable-libmudflap --enable-libgomp --enable-libssp --with-gmp=/cbuild/slaves/oorts/crosstool-ng/builds/arm-linux-gnueabi-linux/.build/arm-linux-gnueabi/build/static --with-mpfr=/cbuild/slaves/oorts/crosstool-ng/builds/arm-linux-gnueabi-linux/.build/arm-linux-gnueabi/build/static --with-mpc=/cbuild/slaves/oorts/crosstool-ng/builds/arm-linux-gnueabi-linux/.build/arm-linux-gnueabi/build/static --with-ppl=/cbuild/slaves/oorts/crosstool-ng/builds/arm-linux-gnueabi-linux/.build/arm-linux-gnueabi/build/static --with-cloog=/cbuild/slaves/oorts/crosstool-ng/builds/arm-linux-gnueabi-linux/.build/arm-linux-gnueabi/build/static --with-libelf=/cbuild/slaves/oorts/crosstool-ng/builds/arm-linux-gnueabi-linux/.build/arm-linux-gnueabi/build/static --with-host-libstdcxx='-L/cbuild/slaves/oorts/crosstool-ng/builds/arm-linux-gnueabi-linux/.build/arm-linux-gnueabi/build/static/lib -lpwl' --enable-threads=posix --disable-libstdcxx-pch --enable-linker-build-id --enable-gold --with-local-prefix=/cbuild/slaves/oorts/crosstool-ng/builds/arm-linux-gnueabi-linux/install/arm-linux-gnueabi/libc --enable-c99 --enable-long-long --with-mode=thumb
线程模型: posix
gcc 版本 4.7.3 20121205 (prerelease) (crosstool-NG linaro-1.13.1-4.7-2012.12-20121214 - Linaro GCC 2012.12)
st@st-virtual-machine:~/work$
    
```

As shown, GCC version is 4.7.3.

So far, the cross-compiler is totally installed, then we could use it to compile system code and application program.

V. Compile Bootstrap

Bootstrap is the secondary bootstrap program, the primary bootstrap program has been cured into the ROM of the CPU chip. The users could not modify!

The bootstrap is in charge of the initialization of clock, GPIO, memory etc. IAC-A5D3x-Kit development board provides three boot methods base on the different image mediums.

- ◆ dataflash boot method
- ◆ nandflash boot method
- ◆ SD Card boot method

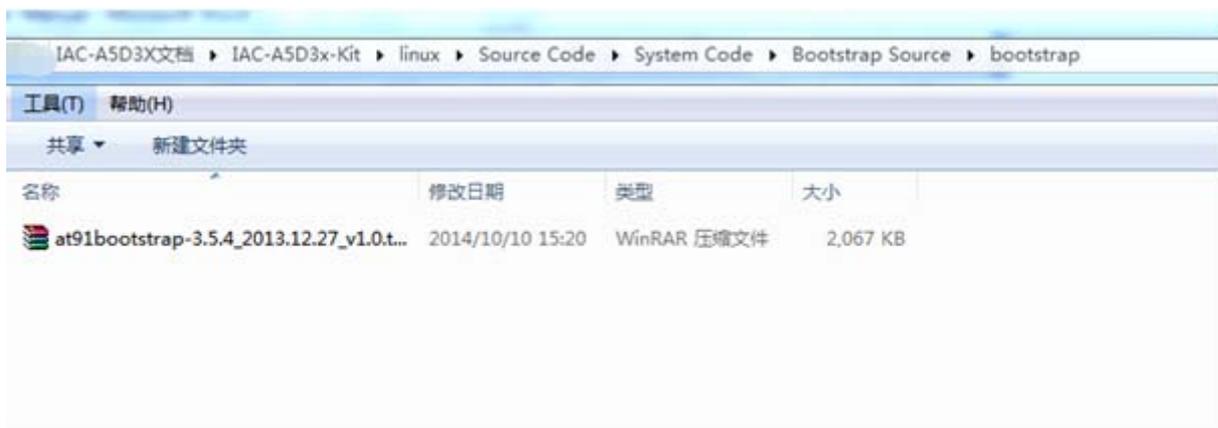
Base on the internal program, CPU will search for boot code base on the storage medium by the following priority :

Dataflash>nandflash>SD Card

Before selecting boot method, please make sure that there is no higher level boot medium or the medium has been interrupted. Or the CPU will use the higher level boot method.

The compiled bootstrap image file is in the CD, you can use it

directly. If you want to recompile, the migrated bootstrap source code is ready too.



Copy the [bootstrap] source code to [~/work] directory in CD, then extract:

```
$ tar -xzf at91bootstrap-3.5.4_2013.12.27_v1.0.tar.gz
```

```
at91bootstrap-3.5.4/fs/include/utsconf.h
at91bootstrap-3.5.4/fs/include/ffconf.h
at91bootstrap-3.5.4/fs/include/ff.h
at91bootstrap-3.5.4/crt0_gnu.S
at91bootstrap-3.5.4/.gitignore
at91bootstrap-3.5.4/main.c
at91bootstrap-3.5.4/README.txt
st@st-virtual-machine:~/work$ ls
at91bootstrap-3.5.4
at91bootstrap-3.5.4_2013.12.27_v1.0.tar.gz
gcc-linaro-arm-linux-gnueabi-4.7-2012.12-20121214_linux
gcc-linaro-arm-linux-gnueabi-4.7-2012.12-20121214_linux.tar.bz2
st@st-virtual-machine:~/work$
```

After extracting, there is a [at91bootstrap-3.5.4] folder, enter into this folder:

```
$ cd at91bootstrap-3.5.4
```

```

st@st-virtual-machine:~/work$ ls
at91bootstrap-3.5.4
at91bootstrap-3.5.4_2013.12.27_v1.0.tar.gz
gcc-linaro-arm-linux-gnueabi-4.7-2012.12-20121214_linux
gcc-linaro-arm-linux-gnueabi-4.7-2012.12-20121214_linux.tar.bz2
st@st-virtual-machine:~/work$ cd at91bootstrap-3.5.4/
st@st-virtual-machine:~/work/at91bootstrap-3.5.4$ ls
board          config          elf32-littlearm.lds  KNOWN_ISSUES  README.txt
build_df.sh    Config.in      fs                  lib            scripts
build_nf.sh    crt0_gnu.S    host-utilities      main.c         topLevel_cpp.mk
build_sd.sh    driver        include             Makefile
st@st-virtual-machine:~/work/at91bootstrap-3.5.4$
    
```

As shown: In order to compile easily, we have already made the corresponding compile steps of the boot method into a script file.

◆ build_df.sh It is the bootstrap script file for dataflash boot method

◆ build_nf.sh It is the bootstrap script file for nandflash.

◆ build_sd.sh It is the bootstrap script file for SD Card.

Users could select the corresponding script file to compile, here is an example : dataflash. The other boot methods' compiling method will not be illustrated again. Execute [build_df.sh]script file.

```
$ ./build_df.sh
```

Execute the above script file, the interface shows the following message :

```

st@st-virtual-machine: ~/work/at91bootstrap-3.5.4
.config - at91bootstrap vBR2_VERSION Configuration

          at91bootstrap Configuration
Arrow keys navigate the menu. <Enter> selects submenus --->.
Highlighted letters are hotkeys. Pressing <Y> selectes a feature,
while <N> will exclude a feature. Press <Esc><Esc> to exit, <?> for
Help, </> for Search. Legend: [*] feature is selected [ ] feature is

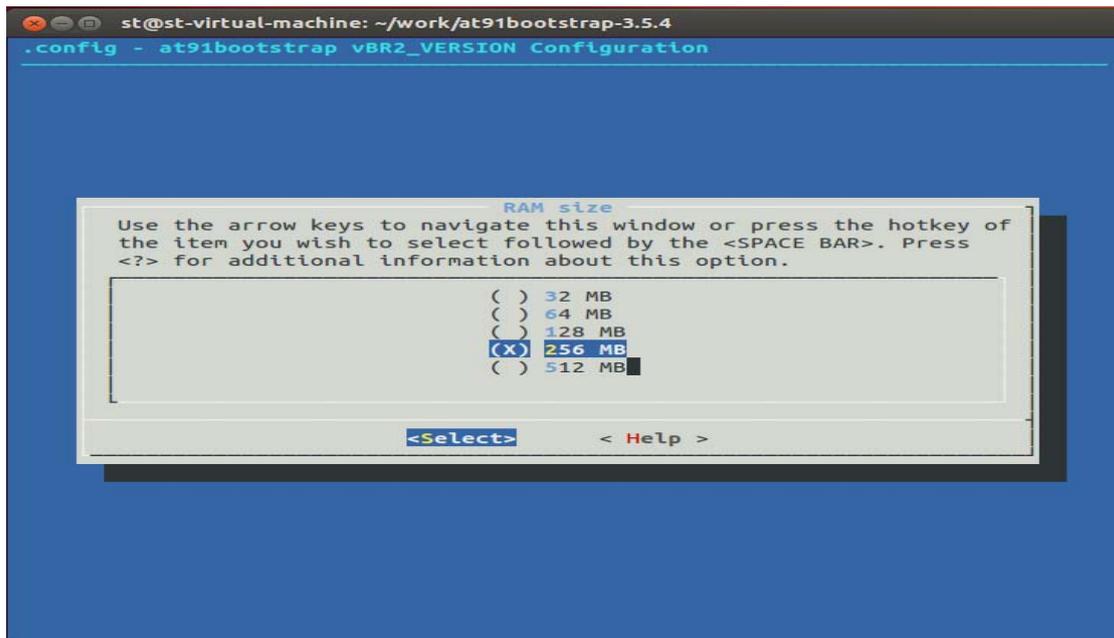
[at91sama5d3xek] Board Name
Board Type (at91sama5d3xek) --->
Crystal Frequency (Build for use of an 12.000 MHz crystal) --
CPU clock (528 MHz) ---->
Bus Speed (133 MHz) ---->
Memory selection --->
Image Loading Strategy (Load U-Boot into last MBYTE of SDRAM)
U-Boot Image Storage Setup ---->
[ ] Perform a memory test at startup
[*] Debug Support
    Debug Level (General debug information) --->
[*] Call Hardware Initialization
[ ] Call User specific Hardware Initialization
[*] Use external 32KHZ oscillator as source of slow clock
[*] Disable Watchdog
---
Load an Alternate Configuration File
Save an Alternate Configuration File

<Select>  < Exit >  < Help >
  
```

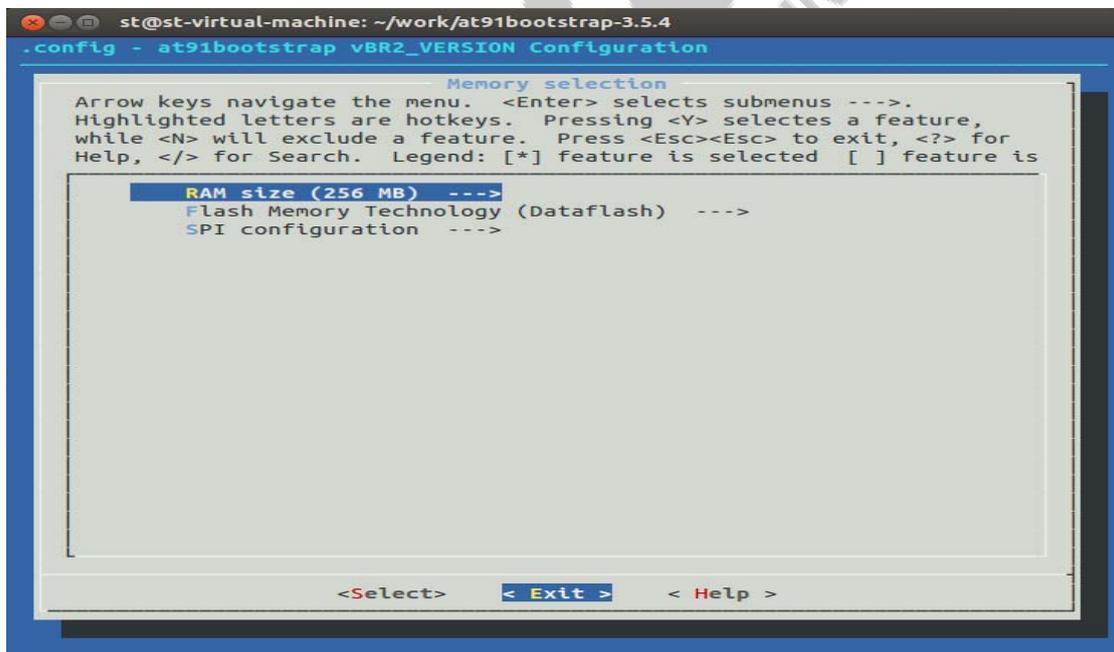
User could configure as requirement, after configuring, select [yes] to save, then exit. If it is no need to modify, then [exit] by default. The IAC-A5D3x-Kit default configuration, you should select the [Memory Selection] to 256M.

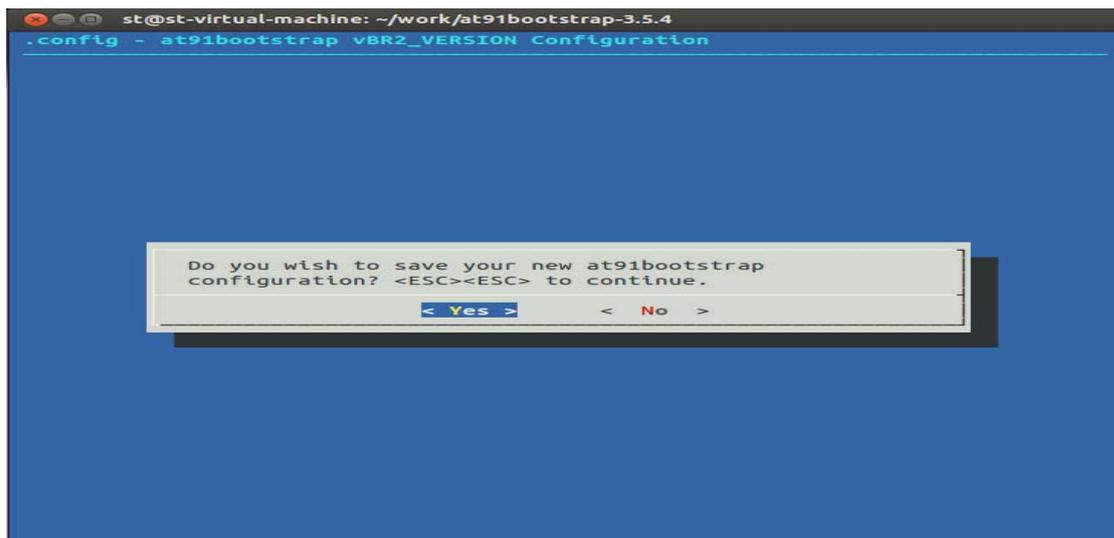
Memory selection --->

RAM size (512 MB) --->



Select, then [exit].





Select [Yes], then save.

Finish configuring, then execute [make] command to compile.

```
$ make
```

Finish compiling , there is no error.

```

st@st-virtual-machine: ~/work/at91bootstrap-3.5.4
G_CPU_CLK_528MHZ -DCONFIG_BUS_SPEED_133MHZ -DCONFIG_HAS_PIO3 -DCONFIG_LOAD_ONE_WI
RE -DCONFIG_MMC_SUPPORT -DCONFIG_DDR2 -DCONFIG_RAM_256MB -DCONFIG_DATAFLASH -DCON
FIG_SMALL_DATAFLASH -DAT91C_SPI_CLK=33000000 -DAT91C_SPI_PCS_DATAFLASH=AT91C_SPI_
PCS0_DATAFLASH -DBOOTSTRAP_DEBUG_LEVEL=DEBUG_INFO -DCONFIG_DISABLE_WATCHDOG

ld FLAGS
=====
-nostartfiles -Map=/home/st/work/at91bootstrap-3.5.4/binaries/at91sama5d3kek-data
flashboot-uboot-3.5.4.map --cref -static -T elf32-littlearm.lds --gc-sections -Tt
ext 0x300000

AS      /home/st/work/at91bootstrap-3.5.4/crt0_gnu.S
CC      /home/st/work/at91bootstrap-3.5.4/main.c
CC      /home/st/work/at91bootstrap-3.5.4/board/at91sama5d3kek/at91sama5d3kek
.c
CC      /home/st/work/at91bootstrap-3.5.4/lib/string.c
CC      /home/st/work/at91bootstrap-3.5.4/lib/eabi_utils.c
CC      /home/st/work/at91bootstrap-3.5.4/lib/div.c
CC      /home/st/work/at91bootstrap-3.5.4/driver/debug.c
CC      /home/st/work/at91bootstrap-3.5.4/driver/at91_slowclk.c
CC      /home/st/work/at91bootstrap-3.5.4/driver/at91_pio.c
CC      /home/st/work/at91bootstrap-3.5.4/driver/pmc.c
CC      /home/st/work/at91bootstrap-3.5.4/driver/at91_pit.c
CC      /home/st/work/at91bootstrap-3.5.4/driver/at91_wdt.c
CC      /home/st/work/at91bootstrap-3.5.4/driver/dbgu.c
CC      /home/st/work/at91bootstrap-3.5.4/driver/ddramc.c
CC      /home/st/work/at91bootstrap-3.5.4/driver/at91_spi.c
CC      /home/st/work/at91bootstrap-3.5.4/driver/dataflash.c
CC      /home/st/work/at91bootstrap-3.5.4/driver/ds24xx.c
mkdir -p /home/st/work/at91bootstrap-3.5.4/binaries
LD      at91sama5d3kek-dataflashboot-uboot-3.5.4.elf
Size of at91sama5d3kek-dataflashboot-uboot-3.5.4.bin is 4280 bytes
[Succeeded] It's OK to fit into SRAM area
st@st-virtual-machine:~/work/at91bootstrap-3.5.4$
    
```

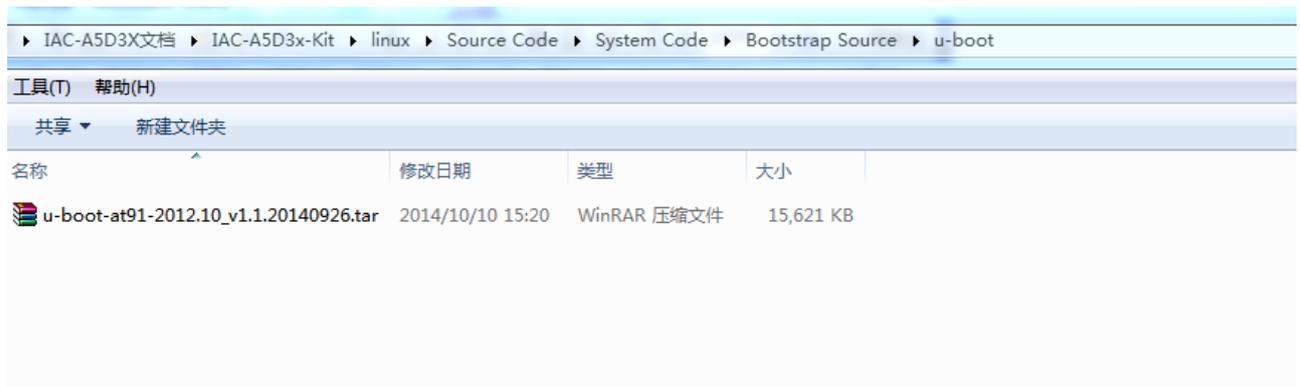
There is an executable file [at91sama5d3kek-dataflashboot-uboot-3.5.4.bin] in [binaries] directory.

```

[Succeeded] It's OK to fit into SRAM area
st@st-virtual-machine:~/work/at91bootstrap-3.5.4$ ls
binaries      config      elf32-littlearm.lds  lib          scripts
board        Config.in   fs                  main.c       topLevel_cpp.mk
build_df.sh  crt0_gnu.o host-utilities      main.o
build_nf.sh  crt0_gnu.S include         Makefile
build_sd.sh  driver     KNOWN_ISSUES      README.txt
st@st-virtual-machine:~/work/at91bootstrap-3.5.4$ ls binaries
at91sama5d3kek-dataflashboot-uboot-3.5.4.bin
at91sama5d3kek-dataflashboot-uboot-3.5.4.elf
at91sama5d3kek-dataflashboot-uboot-3.5.4.map
st@st-virtual-machine:~/work/at91bootstrap-3.5.4$
    
```

VI. Compile uboot

There is a prepared source code [uboot] ,user could compile it directly.



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

```
$ tar -xzf u-boot-at91-2012.10_2013.12.27_v1.0.tar.gz
```

After extracting ,there is a folder [u-boot-at91-2012.10], enter into this folder.

```
$ cd u-boot-at91-2012.10
```

```
$ ls
```

```

u-boot-at91-2012.10/fs/yaffs2/yaffs_nandif.c
u-boot-at91-2012.10/fs/yaffs2/yaffs_ecc.c
u-boot-at91-2012.10/fs/yaffs2/yaffs_attribs.h
u-boot-at91-2012.10/CREDITS
u-boot-at91-2012.10/.gitignore
u-boot-at91-2012.10/config.mk
u-boot-at91-2012.10/MAKEALL
u-boot-at91-2012.10/README
u-boot-at91-2012.10/boards.cfg
st@st-virtual-machine:~/work$ ls
at91bootstrap-3.5.4
at91bootstrap-3.5.4_2013.12.27_v1.0.tar.gz
gcc-linaro-arm-linux-gnueabi-4.7-2012.12-20121214_linux
gcc-linaro-arm-linux-gnueabi-4.7-2012.12-20121214_linux.tar.bz2
u-boot-at91-2012.10
u-boot-at91-2012.10_2013.12.27_v1.0.tar.gz
st@st-virtual-machine:~/work$ ^C
st@st-virtual-machine:~/work$ cd u-boot-at91-2012.10/
st@st-virtual-machine:~/work/u-boot-at91-2012.10$ ls
api          build_nf.sh  CREDITS     env.txt     MAINTAINERS  net          spl
arch         build_sd.sh  disk        examples    MAKEALL      post         test
board        common       doc         fs          Makefile     README      tools
boards.cfg   config.mk    drivers     include     mkconfig     rules.mk
build_df.sh  COPYING     dts         lib         nand_spl     snapshot.commit
st@st-virtual-machine:~/work/u-boot-at91-2012.10$

```

As shown: In order to compile easily, we have already made the corresponding compilation steps of the boot method into a script file.

- ◆ build_sd.sh It is the bootstrap script file for dataflash.
- ◆ build_nf.sh It is the bootstrap script file for nandflash.
- ◆ build_sd.sh It is the bootstrap script file for SD Card.

User could select the corresponding script file to compile, here is the example : dataflash. The other boot method compiling method will be

same. Execute [build_df.sh]script file.

```
$ ./build_df.sh
```

There is no error in compiling, so there will be an image file [uboot.bin] in current directory which could be burnt into the development board.

```
$ ls
```

```
st@st-virtual-machine: ~/work/u-boot-at91-2012.10
d.o drivers/mtd/nand/libnand.o drivers/mtd/onenand/libonenand.o drivers/mtd/spi/libspi_flash.o drivers/mtd/ubi/libubi.o drivers/net/libnet.o drivers/net/phy/libphy.o drivers/pci/libpci.o drivers/pcmcia/libpcmcia.o drivers/power/libpower.o drivers/rtc/librtc.o drivers/serial/libserial.o drivers/spi/libspi.o drivers/twserial/libtwserial.o drivers/usb/eth/libusb_eth.o drivers/usb/gadget/libusb_gadget.o drivers/usb/host/libusb_host.o drivers/usb/musb/libusb_musb.o drivers/usb/phy/libusb_phy.o drivers/usb/ulpi/libusb_ulpi.o drivers/video/libvideo.o drivers/watchdog/libwatchdog.o fs/cramfs/libcramfs.o fs/ext4/libext4fs.o fs/fat/libfat.o fs/fdos/libfdos.o fs/jffs2/libjffs2.o fs/reiserfs/libreiserfs.o fs/ubifs/libubifs.o fs/yaffs2/libyaffs2.o fs/zfs/libzfs.o lib/libfdt/libfdt.o lib/libgeneric.o lib/lzma/liblzma.o lib/lzo/liblzo.o lib/zlib/libz.o net/libnet.o post/libpost.o test/libtest.o board/atmel/sama5d3xek/libsama5d3xek.o --end-group /home/st/work/u-boot-at91-2012.10/arch/arm/lib/eabi_compat.o -L /home/st/work/gcc-linaro-arm-linux-gnueabi-hf-4.7-2012.12-20121214_linux/bin/./lib/gcc/arm-linux-gnueabi-hf/4.7.3 -lgcc -Map u-boot.map -o u-boot
arm-linux-gnueabi-hf-objcopy -O srec u-boot u-boot.srec
arm-linux-gnueabi-hf-objcopy --gap-fill=0xff -O binary u-boot u-boot.bin
make -C examples/standalone all
make[1]: 正在进入目录 `/home/st/work/u-boot-at91-2012.10/examples/standalone'
make[1]: 没有什么可以做的为 `all'。
make[1]: 正在离开目录 `/home/st/work/u-boot-at91-2012.10/examples/standalone'
make -C examples/api all
make[1]: 正在进入目录 `/home/st/work/u-boot-at91-2012.10/examples/api'
make[1]: 没有什么可以做的为 `all'。
make[1]: 正在离开目录 `/home/st/work/u-boot-at91-2012.10/examples/api'
st@st-virtual-machine:~/work/u-boot-at91-2012.10$ ls
api                common            dts                MAKEALL           rules.mk          u-boot.bin
arch               config.mk        env.txt           Makefile          snapshot.commit  u-boot.lds
board              COPYING         examples          mkconfig         spl               u-boot.map
boards.cfg         CREDITS         fs                nand_spl         System.map        u-boot.srec
build_df.sh        disk             include           net               test
build_nf.sh        doc              lib               post              tools
build_sd.sh        drivers          MAINTAINERS      README            u-boot
st@st-virtual-machine:~/work/u-boot-at91-2012.10$
```

*Note: When it is in the burning process, please make sure that the [uboot] and [bootstrap] image are compiled and obtained by the

same boot method.

VII. Compile Kernel

There is a migrated and compiled kernel source file in CD.

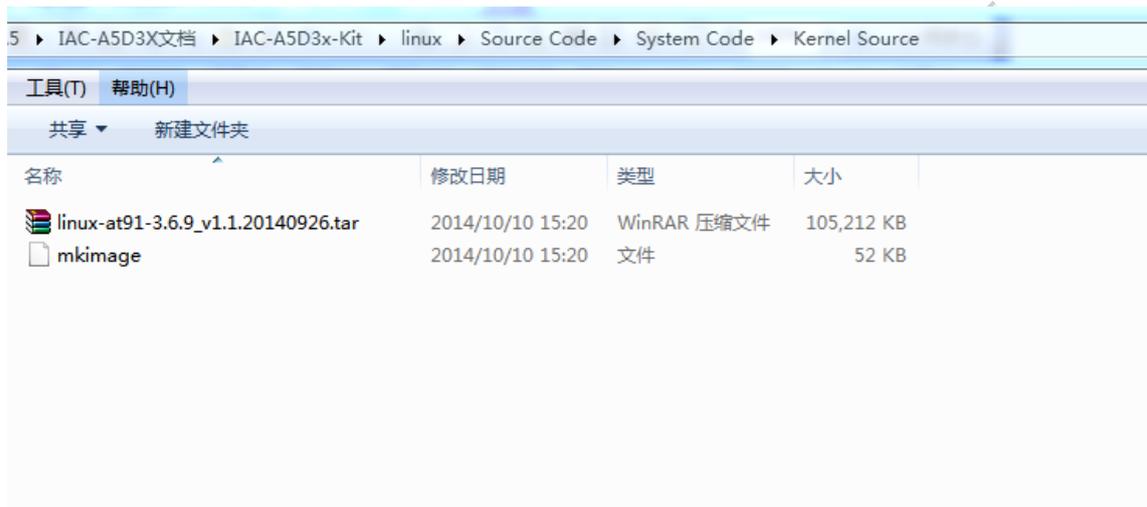


Illustration:[linux-at91-3.6.9_2014.2.8_v2.1.tar.gz] is the kernel source code.

[mkimage] is the tool to generate uImage.

Copy [linux-at91-3.6.9_2014.2.8_v2.1.tar.gz] and [mkimage] in CD into [~/work]directory, then extract kernel source code.

```
$ tar -xvzf linux-at91-3.6.9_2014.2.8_v2.1.tar.gz
```

After extracting , generate [linux-at91-3.6.9]folder in current directory, enter into this folder.

```
$ cd linux-at91-3.6.9
```

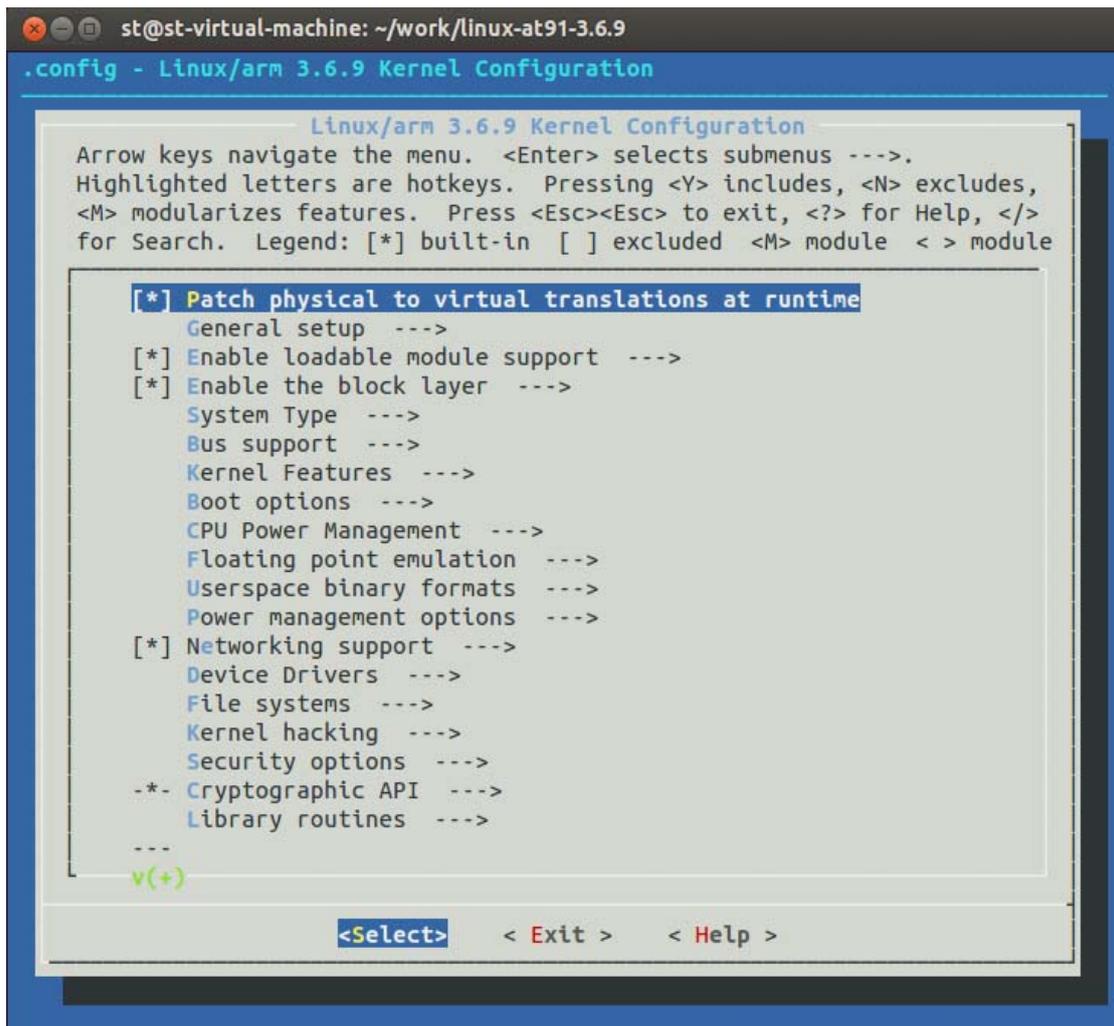
```
$ ls
```

```
st@st-virtual-machine: ~/work/linux-at91-3.6.9
linux-at91-3.6.9/virt/kvm/
linux-at91-3.6.9/virt/kvm/async_pf.h
linux-at91-3.6.9/virt/kvm/Kconfig
linux-at91-3.6.9/virt/kvm/eventfd.c
linux-at91-3.6.9/virt/kvm/kvm_main.c
linux-at91-3.6.9/virt/kvm/iommu.c
linux-at91-3.6.9/virt/kvm/coalesced_mmio.c
linux-at91-3.6.9/virt/kvm/iodev.h
linux-at91-3.6.9/virt/kvm/assigned-dev.c
linux-at91-3.6.9/virt/kvm/coalesced_mmio.h
linux-at91-3.6.9/virt/kvm/ioapic.c
linux-at91-3.6.9/virt/kvm/ioapic.h
linux-at91-3.6.9/virt/kvm/irq_comm.c
linux-at91-3.6.9/virt/kvm/async_pf.c
linux-at91-3.6.9/README
st@st-virtual-machine:~/work$ ls
at91bootstrap-3.5.4
at91bootstrap-3.5.4_2013.12.27_v1.0.tar.gz
gcc-linaro-arm-linux-gnueabi-4.7-2012.12-20121214_linux
gcc-linaro-arm-linux-gnueabi-4.7-2012.12-20121214_linux.tar.bz2
linux-at91-3.6.9
linux-at91-3.6.9_2014.01.03_v1.0.tar.gz
mkimage
u-boot-at91-2012.10
u-boot-at91-2012.10_2013.12.27_v1.0.tar.gz
st@st-virtual-machine:~/work$ cd linux-at91-3.6.9/
st@st-virtual-machine:~/work/linux-at91-3.6.9$ ls
arch      Documentation  ipc           Makefile     samples     virt
block     drivers       Kbuild       mm           scripts
build_dts.sh  firmware     Kconfig     Module.symvers  security
COPYING   fs           kernel       net          sound
CREDITS   include      lib          README       tools
crypto    init         lib          README       tools
          init         MAINTAINERS REPORTING-BUGS  usr
st@st-virtual-machine:~/work/linux-at91-3.6.9$
```

After compiling, it needs the following command to configure kernel.

```
$ make menuconfig
```

After executing, it pop-up the following kernel selection configuration interface.



The users could modify the kernel function options. The other configurations will be illustrated here. According to the specific requirements ,the users could configure it. If there are no special requirements, you could compile kernel by the default kernel option. Here ,we take an sample for modifying kernel resolution.

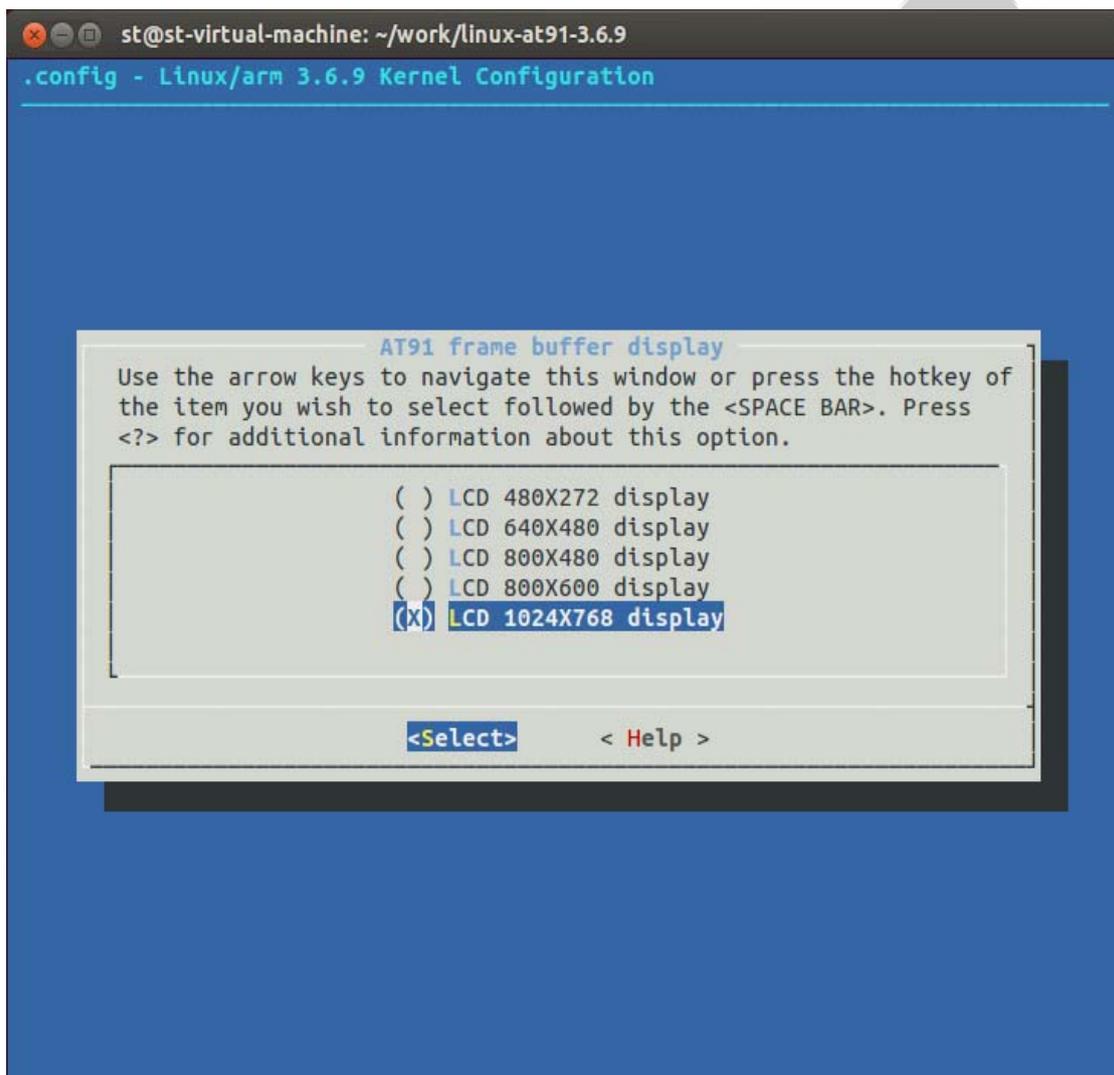
LCD resolution select configuration path in kernel option:

Device Drivers --->

Graphics supports --->

Support for frame buffer devices --->

AT91 frame buffer display (LCD 1024X768 display)



Modify the default configuration from [LCD 800X600 display]

to [LCD 1024X768 display], then save & exit.

Start to compile.

```
$ make
```

The first compilation may need 8-10 minutes, here we take a Computer /dual core 2.4GHz to install virtual machine as a reference. The different computer or server may have some difference in compilation time.

Start to compile kernel image

```
$ make uImage
```

After executing, it hints the following errors:


```
$ make uImage
```

```
st@st-virtual-machine:~/work/linux-at91-3.6.9$ cp ../mkimage ~/work/gcc-linaro-arm-linux-gnueabi-4.7-2012.12-20121214_linux/bin/
st@st-virtual-machine:~/work/linux-at91-3.6.9$ 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
Kernel:  arch/arm/boot/zImage is ready
UIIMAGE  arch/arm/boot/uImage
Image Name:   Linux-3.6.9
Created:     Wed Jan 22 10:53:05 2014
Image Type:  ARM Linux Kernel Image (uncompressed)
Data Size:   2643344 Bytes = 2581.39 kB = 2.52 MB
Load Address: 20008000
Entry Point: 20008000
Image arch/arm/boot/uImage is ready
st@st-virtual-machine:~/work/linux-at91-3.6.9$
```

After compiling, it will generate [uImage] kernel image file in [arch/arm/boot/] directory which could be burnt into the development board.

```
Image arch/arm/boot/uImage is ready
st@st-virtual-machine:~/work/linux-at91-3.6.9$ ls arch/arm/boot/
bootp compressed dts Image install.sh Makefile tftpd32.exe uImage zImage
st@st-virtual-machine:~/work/linux-at91-3.6.9$
```

After compiling, there is a recompiled kernel image [uImage] in [arch/arm/boot/] directory.

After compiling [uImage], we could compile device tree image.

```
$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi- dtbs
```

After compiling, there is sama5d3x series device tree in [arch/arm/boot/] directory. The [sama5d3xek.dtb] is the device tree image. [X] means the corresponding number which use the current using

CPU model.

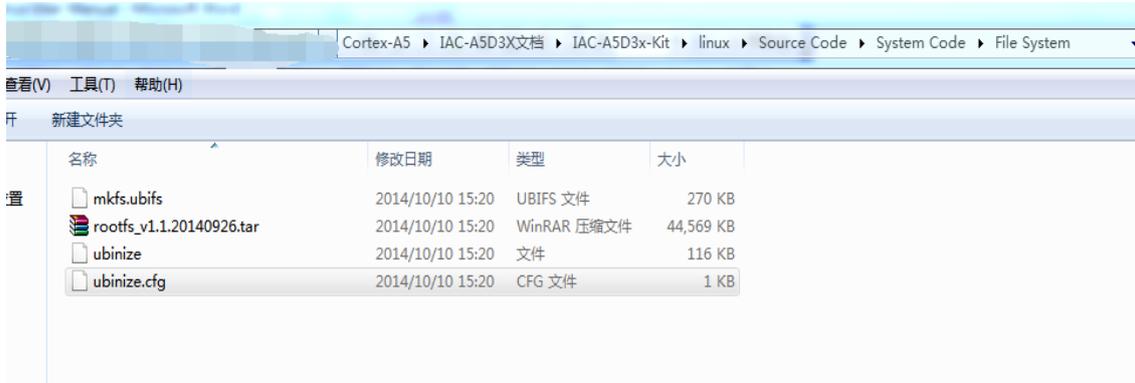
For example, after compiling, there is a device tree image [sama5d34ek.dtb] which sama5d34 compiled.

```

st@st-virtual-machine: ~/work/linux-at91-3.6.9
DTC: dts->dtb on file "arch/arm/boot/dts/at91sam9g15ek.dts"
DTC arch/arm/boot/at91sam9g25ek.dtb
DTC: dts->dtb on file "arch/arm/boot/dts/at91sam9g25ek.dts"
DTC arch/arm/boot/at91sam9g35ek.dtb
DTC: dts->dtb on file "arch/arm/boot/dts/at91sam9g35ek.dts"
DTC arch/arm/boot/at91sam9x25ek.dtb
DTC: dts->dtb on file "arch/arm/boot/dts/at91sam9x25ek.dts"
DTC arch/arm/boot/at91sam9x35ek.dtb
DTC: dts->dtb on file "arch/arm/boot/dts/at91sam9x35ek.dts"
DTC arch/arm/boot/sama5d31ek.dtb
DTC: dts->dtb on file "arch/arm/boot/dts/sama5d31ek.dts"
DTC arch/arm/boot/sama5d31ek_pda.dtb
DTC: dts->dtb on file "arch/arm/boot/dts/sama5d31ek_pda.dts"
DTC arch/arm/boot/sama5d33ek.dtb
DTC: dts->dtb on file "arch/arm/boot/dts/sama5d33ek.dts"
DTC arch/arm/boot/sama5d33ek_pda.dtb
DTC: dts->dtb on file "arch/arm/boot/dts/sama5d33ek_pda.dts"
DTC arch/arm/boot/sama5d34ek.dtb
DTC: dts->dtb on file "arch/arm/boot/dts/sama5d34ek.dts"
DTC arch/arm/boot/sama5d34ek_pda.dtb
DTC: dts->dtb on file "arch/arm/boot/dts/sama5d34ek_pda.dts"
DTC arch/arm/boot/sama5d35ek.dtb
DTC: dts->dtb on file "arch/arm/boot/dts/sama5d35ek.dts"
st@st-virtual-machine:~/work/linux-at91-3.6.9$ ls arch/arm/boot/
aks-cdu.dtb          at91sam9x25ek.dtb  kizbox.dtb          tftpd32.exe
at91sam9263ek.dtb   at91sam9x35ek.dtb  Makefile            tny_a9260.dtb
at91sam9g15ek.dtb   bootp              sama5d31ek.dtb     tny_a9263.dtb
at91sam9g20ek_2mmc.dtb  compressed        sama5d31ek_pda.dtb  tny_a9g20.dtb
at91sam9g20ek.dtb   dts               sama5d33ek.dtb     uImage
at91sam9g25ek.dtb   ethernut5.dtb     sama5d33ek_pda.dtb  usb_a9260.dtb
at91sam9g35ek.dtb   evk-pro3.dtb      sama5d34ek.dtb     usb_a9263.dtb
at91sam9m10g45ek.dtb  Image            sama5d34ek_pda.dtb  usb_a9g20.dtb
at91sam9n12ek.dtb   install.sh        sama5d35ek.dtb     zImage
st@st-virtual-machine:~/work/linux-at91-3.6.9$
    
```

VIII. File System Creation

There is a prepared file system code and creation tool in CD.



Copy file system source code and creation tool to [~/work]directory ,then extract.

File system source code needs [root] permission to do completed extraction.

Add [sudo] before the extracting command.

```
$ sudo tar -xzvf rootfs-2014.2.8_v2.1.tar.gz
```

After extracting , it generate [rootfs]folder.

```
st@st-virtual-machine:~/work$ ls
at91bootstrap-3.5.4
at91bootstrap-3.5.4_2013.12.27_v1.0.tar.gz
gcc-linaro-arm-linux-gnueabi-4.7-2012.12-20121214_linux
gcc-linaro-arm-linux-gnueabi-4.7-2012.12-20121214_linux.tar.bz2
linux-at91-3.6.9
linux-at91-3.6.9_2014.01.03_v1.0.tar.gz
mkfs.ubifs
mkimage
rootfs
rootfs-2014.2.8_v2.1.tar.gz
ubinize
ubinize.cfg
u-boot-at91-2012.10
u-boot-at91-2012.10_2013.12.27_v1.0.tar.gz
st@st-virtual-machine:~/work$
```

Copy creation tool [ubinize]and [mkfs.ubifs] to [bin]directory in cross-compiler.

```
$ cp ubinize ~/work/gcc-linaro-arm-linux-gnueabi-hf-4.7-2012.12-20121214_linux/bin/
```

```
$ cp mkfs.ubifs ~/work/gcc-linaro-arm-linux-gnueabi-hf-4.7-2012.12-20121214_linux/bin/
```

Generate [rootfs.ubifs].

```
$ sudo mkfs.ubifs -r rootfs -m 2048 -e 126976 -c 1816 -o rootfs.ubifs
```

Generate [rootfs.ubi].

```
$ sudo ubinize -o rootfs.ubi -m 2048 -p 128KiB -s 2048 ./ubinize.cfg
```

```
st@st-virtual-machine:~/work$ sudo mkfs.ubifs -r rootfs -m 2048 -e 126976 -c 1816 -o rootfs.ubifs
st@st-virtual-machine:~/work$ sudo ubinize -o rootfs.ubi -m 2048 -p 128KiB -s 2048 ./ubinize.cfg
st@st-virtual-machine:~/work$ ls
at91bootstrap-3.5.4
at91bootstrap-3.5.4_2013.12.27_v1.0.tar.gz
gcc-linaro-arm-linux-gnueabi-hf-4.7-2012.12-20121214_linux
gcc-linaro-arm-linux-gnueabi-hf-4.7-2012.12-20121214_linux.tar.bz2
linux-at91-3.6.9
linux-at91-3.6.9_2014.01.03_v1.0.tar.gz
mkfs.ubifs
mkimage
rootfs
rootfs-2014.2.8_v2.1.tar.gz
rootfs.ubi
rootfs.ubifs
ubinize
ubinize.cfg
u-boot-at91-2012.10
u-boot-at91-2012.10_2013.12.27_v1.0.tar.gz
st@st-virtual-machine:~/work$
```

The generated [rootfs.ubi] is the [ubi]file system image which could be burnt into development board.

IX. Application Program Development

You could develop the application program on Personal Computer , here, we take [Hello World] as an sample. Create [app]folder in [~/work] directory, enter into [app] folder.

```
$ mkdir app
```

```
$ cd app
```

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

```
#include <stdio.h>
```

```
int main(void)
```

```
{
```

```
    printf("Hello World ! \n");
```

```
    return 0;
```

```
}
```

Save it to [hello.c]file.

```
st@st-virtual-machine:~/work/app$ vi hello.c
st@st-virtual-machine:~/work/app$
st@st-virtual-machine:~/work/app$
st@st-virtual-machine:~/work/app$
st@st-virtual-machine:~/work/app$ ls
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$
```

Enable the program to run on the development board. It needs to use

the installed cross-compiler to compile the application programs.

Here is the compiling command:

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

```
$ file hello
```

```
st@st-virtual-machine:~/work/app$ arm-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 (uses shared libs), for GNU/Linux 2.6.31, BuildID[sha1]=0x31f8d5b22d847b154ef6a69a87bd2d3091aadd9, not stripped
st@st-virtual-machine:~/work/app$
```

After compiling, there is an executable binary file [hello] in current directory.

Then, we could copy the executable program [hello] into development board by SD Card, USB flash disk, tftp or nfs mounting method. Then you could execute [hello] program in development board.

X. Conclusion

The above contents may not be detailed enough, if you have any technical problems or suggestion, please contact us: supports@qiyangtech.com. Or you could log in our company forum to contact: <http://www.qiytech.com>. For more information, please contact: sales@qiyangtech.com, or log in <http://www.qiytech.com/index.html>.

Hangzhou 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