ORIENT DISPLAY

Your Total LCD Solution Provider

http://www.orientdisplay.com

AIY-A003M

User Manual Ver1.0

Base NXP I.MX6U

ARM@ Cortex@-A9 Processor

Customer's Approval:

	SIGNATURE	DATE
PREPARED BY (RD ENGINEER)		
CHECKED BY		
APPROVED BY		

Chapter 1: Setup Embedded Linux Development Environment.....................................4

1.1 Install cross compiler...5

1.1.1 Unzip the cross compiler 5

1.1.2 Set environment variables 5

1.1.2.1 Modify the global configuration file...5

1.1.2.2 Modify user profile (recommended)..6

1.1.2.3 System dependency package...6

1.1.2.4 Test toolchain..6

Chapter 2: AIY-A003M Basic Operation...7

2.1 Debug the serial port connection.. 8

2.2 Power on and log in.. 8

2.3 Shut down and restart... 8

2.4 Set up automatic startup.. 8

2.4.1 Boot script...8

2.4.2 Add auto-execute command on boot...9

2.5 TF card use...9

2.6 USB Disk Use..11

Chapter 3: Hardware interface programming...13

3.1 GPIO Hardware programming 14

3.1.1 GPIO interface..14

3.1.2 Use GPIO from the command line..................................... 15

3.1.3 Use GPIO by writing C program..16

3.2 Serial programming.. 17

3.2.1 open serial port 17

3.2.2 Set the serial port baud rate..18

3.2.3 Read and write data..19

Chapter 4: Embedded GUI (QT) programming...20

4.1 Qt/Embedded Introduction...21

4.2 Setup Qt/Embedded cross-compilation environment 21

4.2.1 Introduction to Compilation Environment..21

4.2.2 Install tslib...21

4.2.2.1 Before Install...21

4.2.3 Cross compile Qt 4.8.6..22

4.3 QT SDK Build.. 24

4.3.1 QT SDK.. 24

4.3.2 Qt SDK Install..24

4.4. qmake..25

4.5 Qt Creator...25

4.5.1 Qt Creator Configuration..25

4.5.2 Qt Creator use..26

Chapter 1

Setup Embedded Linux Development Environment

This chapter first describes the basic methods of embedded Linux development in the Linux environment, and then introduces the software used in embedded development, including how to install and test. This chapter is indispensable for embedded Linux development and is the basis for embedded Linux development. Please understand it carefully and make correct settings.

1.1 Install cross compiler

The cross compiler is usually released under the name arm-none-linux-gnueabi.tar.bz2 (the tool chain names of different manufacturers and different platforms are mostly different and generally not universal), For AIY-A003M motherboard, the name of the cross-compilation tool chain is gcc-linaro-arm-linux-gnueabihf-4.9-2014.09_linux.tar.bz2.

1.1.1 Unzip the cross compiler

The developer copies the cross-compiler to the development host (Ubuntu is recommended as the development host), and unzip it by referring to the following command:

```
od@Linux-host:~$tar -jxvf gcc-linaro-arm-linux-gnueabihf-4.9-2014.09_linux.tar.bz2 -C /opt
```

1.1.2 Set environment variables

There are many ways to set system environment variables, the two commonly used are described below:

1.1.2.1 Modify the global configuration

/etc/profile is the global configuration file of the system. Set the path of the cross-compiler in this file, so that all users who log in to the machine can use this compiler

Open the terminal, enter the "sudo vi /etc/profile" command to open the /etc/profile file, and add at the end of the file:

```
export PATH=/opt/gcc-linaro-arm-linux-gnueabihf-4.9-2014.09_linux/bin:$PATH
```

Save the file and exit, and then enter ". /etc/profile" (dot + space + file name) in the terminal, and execute the profile file to make the changes just made effective. If there are no input errors, reopen the terminal at this time, enter arm-linux-gnueabihf-, and press the TAB key on the keyboard, you can see many commands starting with arm-linux-gnueabihf-.

1.1.2.2 Modify user profile (recommended)

"/etc/profile" is a global configuration file that will affect all users who log in to this machine. If you don’t want your personal settings to affect other users of the system, you can modify the configuration file that only belongs to the current user, usually "~/.bashrc" or “~/.bash_profile”

The modification method is similar to modifying the "/etc/profile" file. After the file is opened, add at the end:

```
export PATH=/opt/gcc-linaro-arm-linux-gnueabihf-4.9-2014.09_linux/bin:$PATH
```

In the same way as executing "/etc/profile", enter ". .bashrc" or ". .bash_profile" to execute the modified file to make the modification effective. If it is correct, reopen the terminal, enter arm-linux-gnueabihf-, and then press the keyboard TAB key, you can also see many commands beginning with arm-linux-gnueabihf-

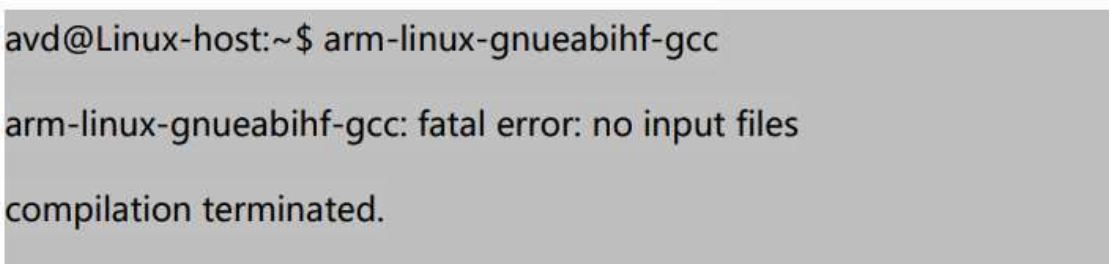
1.1.2.3 System dependency package

The following dependency packages are required in the development environment, which can be installed directly through the apt tool:

```
$ sudo apt-get install lib32z1 lib32z1-dev
$ sudo apt-get install lib32stdc++6
$ sudo apt-get install g++
```

1.1.2.4 Test toolchain

Open the terminal and run the cross-compiler tool. If you can get an output similar to the following, the cross-compiler has been able to work normally



```
avd@Linux-host:~$ arm-linux-gnueabihf-gcc
arm-linux-gnueabihf-gcc: fatal error: no input files
compilation terminated.
```

Further, you can also write a simple c file, and then check whether the cross tool chain can successfully compile it.

Chapter 2

AIY-A003M Basic Operation

This chapter describes the basic operations of the AIY-A003M motherboard, such as hardware connection, booting, and system settings. This chapter is a description of some operations, and they are also common operations in the embedded Linux development process, which need to be mastered.

2.1 Debug the serial port connection

AIY-A003M has a debugging serial port (COM1), which uses RS-232 level. If the debugging computer is equipped with a standard serial port (usually RS-232 level), you can use a serial port extension cable to connect the computer with AIY-A003M. If you are using a portable computer (usually without a standard serial port) or a desktop computer that does not provide a standard serial port, you need a USB to RS-232 serial cable to connect the computer and the motherboard. When using a USB to RS-232 serial cable on a computer running Windows operating system, you need to install the corresponding driver (provided by the manufacturer of the conversion chip).

2.2 Power on and log

After power supply is connected, the AIY-A003M motherboard runs automatically. After the PC is connected to the motherboard through COM1 (the baud rate is 115200), you can log in to the motherboard through the serial port. The login password and account are both root.

2.3 Shut down and restart

When you need to shut down or restart, if there is a data storage operation, in order to ensure that the data is completely written, you can enter the sync command. After completing the data synchronization, turn off the power and press the reset button to restart. You can also enter the reboot command to restart:

```
[root@AIY-A003M~]# reboot
```

This command will automatically complete the data synchronization and restart the system.

2.4 Set up automatic start up

2.4.1 Boot script

The system file: /etc/init.d/S90start_userapp.sh is a script that is automatically executed when booting. The content of the script is shown in code listing 2.1. Commands or applications that need to be automatically executed at boot can be added to this file.

Code list 2.1 start_userapp file content

```
#!/bin/sh

ifconfig eth0 192.168.1.2

# you can add your app start_command here
```

2.4.2 Add auto-execute command on boot

If you want to automatically execute a program when you turn on the machine, such as the mydemo program in the /root directory, add a command to execute the /root/mydemo program in the S90start_userapp.sh file:

```
#!/bin/sh

ifconfig eth0 192.168.1.2

# you can add your app start_command here

/root/mydemo
```

2.5 Use TF card

After inserting the TF card into the TF slot of the motherboard, the Linux system will automatically detect the TF card and print the relevant kernel information:

```
[root@AIY-A003M ~]#
mmc0: new high speed SD card at address 21ed
mmcblk0: mmc0:21ed APPSD 121 M
mmcblk0: p1
```

The system will generate a directory under /media for each partition of the TF card. The name of the directory is mmcblk0pn (n represents a different partition, n=0, 1, 2, 3...). Each partition of the TF card is automatically mounted in these directories. The files saved by the user in these directories will be stored in the corresponding partition of the TF card.

Enter the df command to view the mounting status and usage of each partition:

```
[root@AIY-A003M ~]# df
Filesystem 1K-blocks Used Available Use% Mounted on
/dev/root 6846632 58888 6436624 1% /
devtmpfs 866940 0 866940 0% /dev
tmpfs 1031292 0 1031292 0% /dev/shm
tmpfs 1031292 112 1031180 0% /tmp
 tmpfs 1031292 20 1031272 0% /run
/dev/sda1 30267776 83600 30184176
```

Before the TF card is used up and ejected, you need to uninstall all the partitions first

```
[root@AIY-A003M ~]#umount /dev/mmcblk0p1
```

Note that before unmounting a partition, you must first move the current directory out of the directory where the partition is mounted, that is to say, you cannot unmount /media/mmcblk0p1 under the /media/mmcblk0p1 directory.

2.6 Use USB Disk

AIY-A003M integrates two universal USB ports, which can directly support U disk connection. After inserting the USB flash drive into the USB interface, Linux will automatically detect the connection of the USB flash drive and print out the information:

```
root@AIY-003M ~]# usb 1-1.2: new high-speed USB device number 4 using ci_hdrc
usb-storage 1-1.2:1.0: USB Mass Storage device detected
scsi host1: usb-storage 1-1.2:1.0
scsi 1:0:0:0: Direct-Access SanDisk Cruzer Blade 1.26 PQ: 0 ANSI: 6
sd 1:0:0:0: [sda] 15633408 512-byte logical blocks: (8.00 GB/7.45 GiB)
sd 1:0:0:0: [sda] Write Protect is off
sd 1:0:0:0: [sda] Write cache: disabled, read cache: enabled, doesn't support DPO or FUA
sda: sda4
sd 1:0:0:0: [sda] Attached SCSI removable disk
```

The system will generate a directory for each partition of the U disk under the /media directory, the name of the directory is sdxn (x is used to distinguish different U disks, n is used to distinguish different partitions, x=a, b, c... … N=0, 1, 2, 3…), each partition of the U disk is mounted under these directories. The files saved by the user in these directories will be stored in the corresponding partition of the U disk.

Enter the df command to view the mounting status and usage of each partition of the U disk:

```
[root@AIY-003M ~]# df
Filesystem 1K-blocks Used Available Use% Mounted on
/dev/root 6617864 94472 6180560 2% /
devtmpfs 89312 0 89312 0%
/dev tmpfs 253376 0 253376 0% /dev/shm
tmpfs 253376 104 253272 0% /tmp
tmpfs 253376 172 253204 0% /run
/dev/mmcblk1p1 511720 6592 505128 1% /media/mmcblk1p1
/dev/sda4 7801088 4076864 3724224 52% /media/sda4
```

Similar to the TF card, the U disk should be unloaded before unplugging the relevant partition before using it.

Chapter 3

Hardware interface programming

This chapter mainly talks about hardware interface programming. Since users need secondary development in AIY-A003M, and often operate on the usual serial ports and GPIOs, this chapter focuses on the description of the programming and usage methods of this part of the hardware contact.

3.1 GPIO Hardware programming

This motherboard system provides a very convenient operation interface for GPIO. The GPIO sub-directory of AIY-A003M is /sys/class/gpio. First introduce how to operate GPIO in the onboard Linux system (please refer to the motherboard specification for the specific location and connection of GPIO)

3.1.1 GPIO interface

/sys/class/gpio has following file nodes:

```
root@AIY-A003M ~]# ls /sys/class/gpio/
export   gpiochip128   gpiochip192   gpiochip64   unexport
gpiochip0   gpiochip160   gpiochip32   gpiochip96
```

Export and unexport are the attribute files of the GPIO subsystem. Export is used to export gpio. After gpio is exported, related nodes will appear in the /sys/class/gpio/ directory. Operating related nodes can realize the direction and level of gpio. Read and write status, etc.

Write the sequence number N of the GPIO to the export file to export its device catalog. The calculation formula of the sequence number is as follows:

$$\text{GPIO sequence number} = (\text{BANK} - 1) \times 32 + N$$

In the formula, BANK is the BANK where the GPIO pin is located, and N is the serial number of the pin in that BANK. Take IO4 on the motherboard (the actual GPIO pin is GPIO1_IO04) as an example, its BANK value is 1, N value is 4, so the sequence number is (1-1)*32+4=4.

The operation command to write the sequence number is as follows (note that there is a space on each side of the ">"):

```
[root@AIY-A003M ~]# echo 4> /sys/class/gpio/export
```

After the above command is executed, the gpio4 directory will be generated under the /sys/class/gpio. The purpose of operating this GPIO can be achieved by reading and writing the device files in this directory. Similary, you can also export other GPIO device. After the device directory of GPIO4 is generated, you can see that it contains the following property files:

```
[root@AIY-A003M ~]# ls /sys/class/gpio/gpio4/
active_low   direction   power   uevent
device   edge   subsystem   value
```

Among them, the commonly used ones are the direction and value, the direction is used to configure gpio as input or output, and the value is used for output replacement (when used as output) or read replacement (when used as input)

3.1.2 Use GPIO from the command line

After GPIO is exported, it defaults to input function. You can view the current operating direction of the GPIO by reading the direction file

```
[root@AIY-A003M ~]# cat /sys/class/gpio/gpio4/direction
in
```

Write the "out" string to the direction file to set GPIO as output:

```
[root@AIY-A003M ~]# echo out > /sys/class/gpio/gpio4/direction
```

In the same way, you can write an "in" string to set GPIO back to input.

When GPIO is set as input, the value file records the input level status of the GPIO pin: 1 means input is a high level; 0 means that the input is a low level. You can read the input level of GPIO by viewing the value file

```
[root@AIY-A003M ~]# echo in >
/sys/class/gpio/gpio4/direction
[root@AIY-A003L ~]# cat /sys/class/gpio/gpio4/value
0
```

When the GPIO is set to output, the state of the output level can be controlled by writing 0 or 1 to the value file (0 means output low level, 1 means output high level):

```
[root@AIY-A003M ~]# echo out >/sys/class/gpio/gpio4/direction
[root@AIY-A003M ~]# echo 1 > /sys/class/gpio/gpio4/value
[root@AIY-A003M ~]# echo 0 > /sys/class/gpio/gpio4/value
```

3.1.3 Use GPIO by writing C program

The overall operation is the same as the command line principle, and related operations are also performed by reading and writing file nodes. When using system calls to implement GPIO input and output operations, you also need to export GPIO through the export property file first.

```
#define EXPORT_PATH "/sys/class/gpio/export"                    //export 文件节点路径

#define GPIO "4"                              //GPIO 口序号

int fd_export = open(EXPORT_PATH, O_RDWR); //打开 export 文件

...

write(fd_export,GPIO,strlen(GPIO)); //向 export 文件写入 GPIO 排列序号字符串
```

Then call write function to write an in/out string to the direction device file, and set GPIO as input Or output:

```
#define DIRECT_PATH "/sys/class/gpio/gpio4/direction"        //direction 文件路径

int fd_dir, ret ;

fd_dir = open(DIRECT_PATH,O_RDWR); //打开 direction 文件

...

ret = write(fd_dir, direction_IN, sizeof(direction));      //写入的字符串 direction 为“in”或“out”
```

Finally read or write the value to complete the final operation

```
#define DEV_PATH "/sys/class/gpio/gpio4/value" //value 文件路径

int fd_dev, ret ;

fd_dev = open(DEV_PATH, O_RDWR)

...

ret = read(fd_dev, buf, sizeof(buf));   //读取 GPIO 输入电平值; 若是输出，则为 write
```

After coding, a binary file that can be run on the motherboard is generated through cross-compilation:

```
od@Od-System-Builder:~$ arm-linux-gnueabihf-gcc gpio_test.c -o gpio_test
```

After compiling, copy the binary file to the USB flash drive (assuming to copy to the root directory of the USB flash drive), insert the USB flash drive, and run the binary file.

```
[root@AIY-A003M ~]# /media/sda4/gpio_test
```

If you test the output, you can directly measure the gpio level to confirm whether the program is valid. If you test the input, you can directly connect the GPIO to GND or VCC (3.3v) on the motherboard with a DuPont cable, and then read and print the level through the program.

3.2 Serial programming

Like most other devices, the serial port in Linux appears as a device file. AIY-A003M motherboard serial device file is /dev/ttymxcn (n=0~2, 4~7), there are 7 serial ports in total.

3.2.1 open serial port

Before using a serial port, you must use the open function to open its corresponding device node file. For example, the code to open "/dev/ttymxc0" is shown in code listing 3.1.

Code Listing 3.1 Open the serial port device

```
#define TTY_MXC0_PATH   "/dev/ttymxc0"
int fd;
fd = open(TTY_MXC0_PATH,O_RDWR| O_NOCTTY);
if (fd < 0) {
    printf("open uart device ttymxc0 error\n");
}
```

When the open call succeeds, it will return the file descriptor as a parameter of other operation functions; if it fails, it will return a negative number. When opening a serial port, in addition to the O_RDWR option flag, it is usually necessary to use O_NOCTTY, which means that the opened is a terminal device and the program will not become the controlling terminal of the port. If this flag is not used, an input of the task (such as keyboard termination signal, etc.) may affect the process.

3.2.2 Set the serial port baud rate

After opening the serial port, the serial port uses the default baud rate of 9600. In practical applications, since the baud rate of other communication terminals is not 9600, it is often necessary to set the baud rate.

Before setting the serial port, first read the serial port parameters, and then modify the baud rate: get and set terminal properties:

```
tcgetattr( fd,&options) ；  // options is termios structure
```

The baud rate of the serial port is divided into input baud rate and output baud rate, respectively, through cfsetispeed() and cfsetospeed() function setting (options is the serial port attribute structure obtained through tcgetattr above).

```
cfsetispeed(&options, speed);
cfsetospeed(&options, speed);
```

3.2.3 Read and write data

Use the read/write function to read and write data on the serial port, that is, read the data received by the serial port, or send data from the serial port:

```
#define DEV_NAME "/dev/ttymxc1"
fd = open(DEV_NAME, O_RDWR | O_NOCTTY); ...
len = write(fd, buf, sizeof(buf));          /* Write a string to the serial port */
...
len = read(fd, buf, sizeof(buf));           /* Read string from mouth */
```

3.2.4 Close serial port

After using the serial port, use the close() function to close the serial port (parameter fd is the file descriptor obtained when the serial port is opened):

```
close(fd);
```

Chapter 4

Embedded GUI (QT) programming

QT is a common Linux graphical interface, Qt/Embedded is the embedded version of QT. This chapter introduces the basic programming of embedded Qt, starting from the environment construction, and introduces the qmake tool and Qt Creator.

4.1 Qt/Embedded Introduction

Qt is a cross-platform application and UI development framework. With Qt, you only need to develop applications once, and you can deploy these applications across different desktops and embedded operating systems without rewriting the source code. Qt on the embedded Linux distribution belongs to the Embedded Linux branch platform of Qt (referred to as Qt/E in this article). Based on the original Qt, Qt/E has made many excellent adjustments to suit the embedded environment. AIY-A003M uses the more mature qt4.8.6, and has integrated the relevant operating environment in the motherboard file system. The developer can run the compiled qt program directly on the motherboard.

4.2 Setup Qt/Embedded cross-compilation environment

4.2.1 Introduction to Compilation Environment

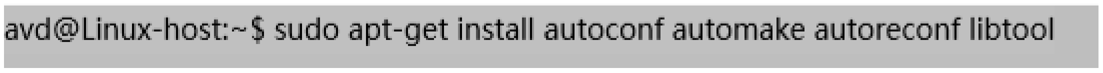
Host system：	Ubuntu 14.04
Cross compilation tool：	arm-linux-gnueabihf
Target board：	AIY-A003M

4.2.2 Install tslib

tslib is an open source program that can provide functions such as filtering, de-jittering, calibration and other functions for the samples obtained by the touch screen driver. It is usually used as the adaptation layer of the touch screen driver and provides a unified interface for the upper application.

4.2.2.1 Before Install

Install autoconf、automake、autoreconf and libtool：



```
avd@Linux-host:~$ sudo apt-get install autoconf automake autoreconf libtool
```

4.2.2.2 Compile and install tslib

Copy tslib-master.zip to ubuntu, unzip the tslib source package:

```
od@Linux-host:~$unzip tslib-master.zip
```

Enter the tslib source directory and configure tslib:

```
od@Linux-host:~$cd   tslib-master
od@Linux-host:~/tslib-master$   ./autogen.sh
od@Linux-host:~/tslib-master$     ./configure   --prefix=/opt/tslib    --host=arm-linux-gnueabihf
ac_cv_func_malloc_0_nonnull=yes
```

Among them, --prefix specifies the installation path of tslib, and users can also specify other directories by themselves; and --host specifies the cross-compiler.

Compile tslib source code:

```
od@Linux-host:~/tslib-master$make
```

Install tslib：

```
od@Linux-host:~/tslib-master$  sudo chmod 777 /opt
od@Linux-host:~/tslib-master$  make install
```

4.2.3 Cross compile Qt 4.8.6

4.2.3.1 Unzip the QT source code package

Copy the source code package of Qt 4.8.6 (qt-everywhere-opensource-src-4.8.6.tar.gz) the user directory of Ubuntu, then execute the decompression command:

```
od@Linux-host:~/$tar zxvf qt-everywhere-opensource-src-4.8.6.tar.gz
```

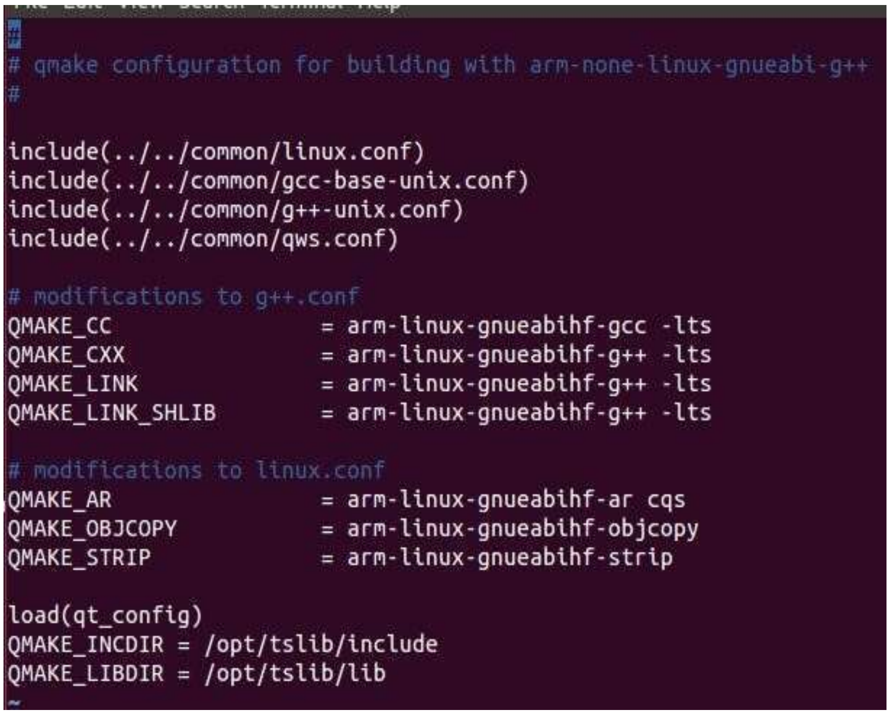
After decompression, you will get the qt-everywhere-opensource-src-4.8.6 directory. Enter the directory and you will see a build-qt script. You can adjust the installation directory by editing the parameters in the script (the directory after the -prefix parameter is the installation directory. For example, it can be modified to /opt/qt4.8.6) and so on.

After adjusting build-qt, you also need to edit mkspec/qws/linux-arm-gnueabi-g++/qmake.conf file, modify the tool chain to arm-linux-gnueabihf, and in
Add the -lts parameter to g++.conf, and finally add the following two lines of parameters at the end of the file:

```
QMAKE_INCDIR = /opt/tslib/include

QMAKE_LIBDIR = /opt/tslib/lib
```

The contents of the modified file are as follows:



```
#
# qmake configuration for building with arm-none-linux-gnueabi-g++
#

include(../../common/linux.conf)
include(../../common/gcc-base-unix.conf)
include(../../common/g++-unix.conf)
include(../../common/qws.conf)

# modifications to g++.conf
QMAKE_CC                = arm-linux-gnueabihf-gcc -lts
QMAKE_CXX               = arm-linux-gnueabihf-g++ -lts
QMAKE_LINK              = arm-linux-gnueabihf-g++ -lts
QMAKE_LINK_SHLIB        = arm-linux-gnueabihf-g++ -lts

# modifications to linux.conf
QMAKE_AR                = arm-linux-gnueabihf-ar cqs
QMAKE_OBJCOPY           = arm-linux-gnueabihf-objcopy
QMAKE_STRIP             = arm-linux-gnueabihf-strip

load(qt_config)
QMAKE_INCDIR = /opt/tslib/include
QMAKE_LIBDIR = /opt/tslib/lib
~
```

After saving the file, execute the build-qt script for configuration.

```
od@Linux-host:~/qt-everywhere-opensource-src-4.8.6$ ./build-qt
```

After the configuration is complete, start to execute the following commands:

```
od@Linux-host:~/qt-everywhere-opensource-src-4.8.6$ make
```

After compiling, execute the installation command:

```
od@Linux-host:~/qt-everywhere-opensource-src-4.8.6$make install
```

4.3 Build QT SDK

4.3.1 QT SDK

Since Qt is a cross-platform graphics framework, users can first develop and debug Qt applications on the PC host. After the applications are basically formed, they can be transplanted to the target board. Therefore, users need to build QT SDK on the PC to improving development efficiency. The QT SDK includes:

QT library suitable for PC environment operation

QT Integrated Development Environment (Qt Creator)

4.3.2 Install Qt SDK

Under Ubuntu environment, the Linux version of the Qt SDK can be obtained through apt-get. When the Ubuntu host can access the Internet normally, you can use the following commands to obtain and install the Qt SDK:

```
od@Linux-host:~$ sudo apt-get install qt-sdk
```

During the installation of the Qt SDK, a Warning: Phonon is not functional warning may appear. Just press Enter and go to the next step.

After the installation is successful, you can see that there are two more executable files qmake and qmake-qt4 in the /usr/bin/ directory, we use qmake-qt4, in order to distinguish between qmake for local compilation and qmake for cross compilation, It is best to set an alias for one of qmake, for example, qmake-arm can be used to indicate that you want to use qmake for cross-compilation. This can be achieved by adding the following command at the end of the ~/.bashrc file.

```
alias qmake-arm=/opt/qt4.8.6/bin/qmake
```

4.4. qmake

Qt provides the qmake tool, which is a tool used to generate Makefiles for different platforms and compilers

Handwriting Makefile is more difficult and error-prone, especially when porting to the target board after PC development, you also need to write multiple Makefiles. When using qmake, developers only need to create a .pro file and run qmake corresponding to the platform to generate the corresponding Makefile.

For some simple projects (such as projects with a small amount of source code), you can directly execute the qmake -project command in the top-level directory of the project to automatically generate Pro files (with the suffix .pro); but for some complex Qt programs, The Pro file which was automatically generated often fails to meet the requirements, so the programmer needs to manually rewrite the Pro file. There are many rules for writing Pro files, and developers can find them online.

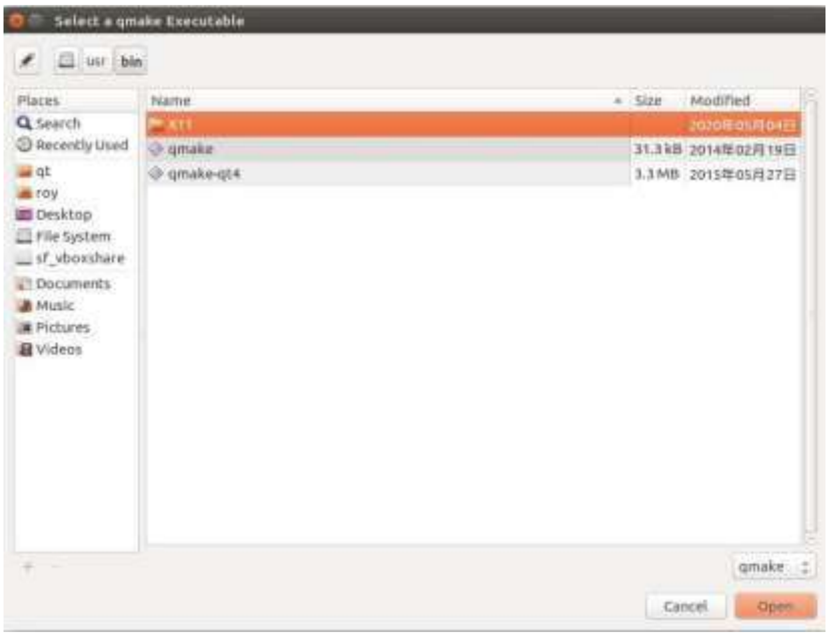
4.5 Qt Creator

4.5.1 Qt Creator Configuration

Qt Creator is a powerful cross-platform IDE that integrates editing, compiling, running, and debugging functions. Qt programming with Qt Creator can greatly improve efficiency and reduce development time. Developers can start directly throuth commands QtCreator:

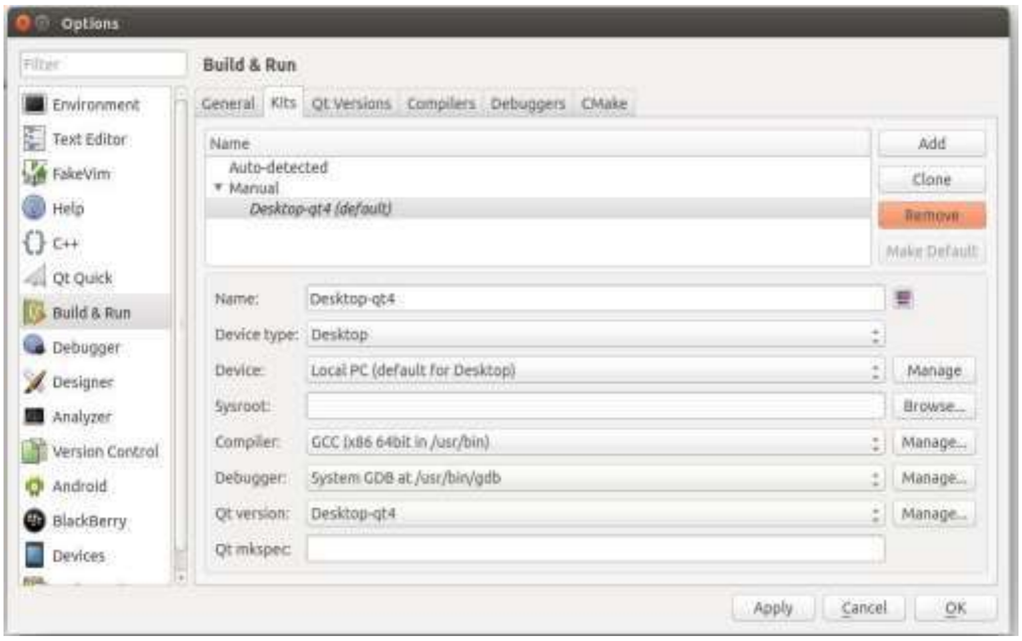
```
od@linux-host: ~qtcreator
```

If you have installed the desktop version of Qt and the embedded version of Qt, you need to set the qmake version used by Qt Creator. Click Tools→Options in the menu, and then click Build & Run on the left. In the Build & Run that pops up on the right, select Qt Version and manually add the Qt version. Take the desktop version of Qt as an example: click Add, and select qmake. Executable file window, and then select the executable file in the path /usr/bin/qmake-qt4, as shown in the figure, click to open



Name	Size	Modified
X11		2020年01月04日
qmake	31.3 kB	2014年02月19日
qmake-qt4	3.3 MB	2015年05月27日

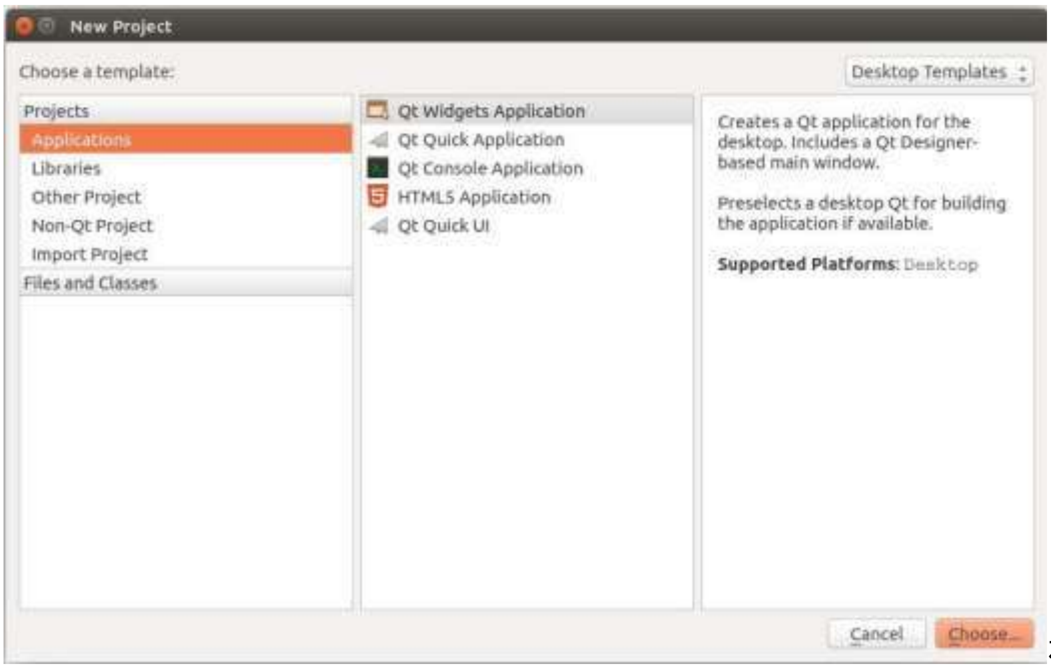
Enter the version name in the Version Name column, such as Destop-qt4, click Apply, then select Kits next to Qt Version, click Add, enter Name, select compiler, Qt-version, etc., click Apply, then click OK.



Name: Desktop-qt4
Device type: Desktop
Device: Local PC (default for Desktop)
Sysroot:
Compiler: GCC (x86 64bit in /usr/bin)
Debugger: System GDB at /usr/bin/gdb
Qt version: Desktop-qt4
Qt mkspec:

4.5.2 Use Qt Creator

The following explains how to use Qt Creator to develop programs. Click New Project on the main interface of Qt Creator, select Qt Widgets Application, and click Choose:



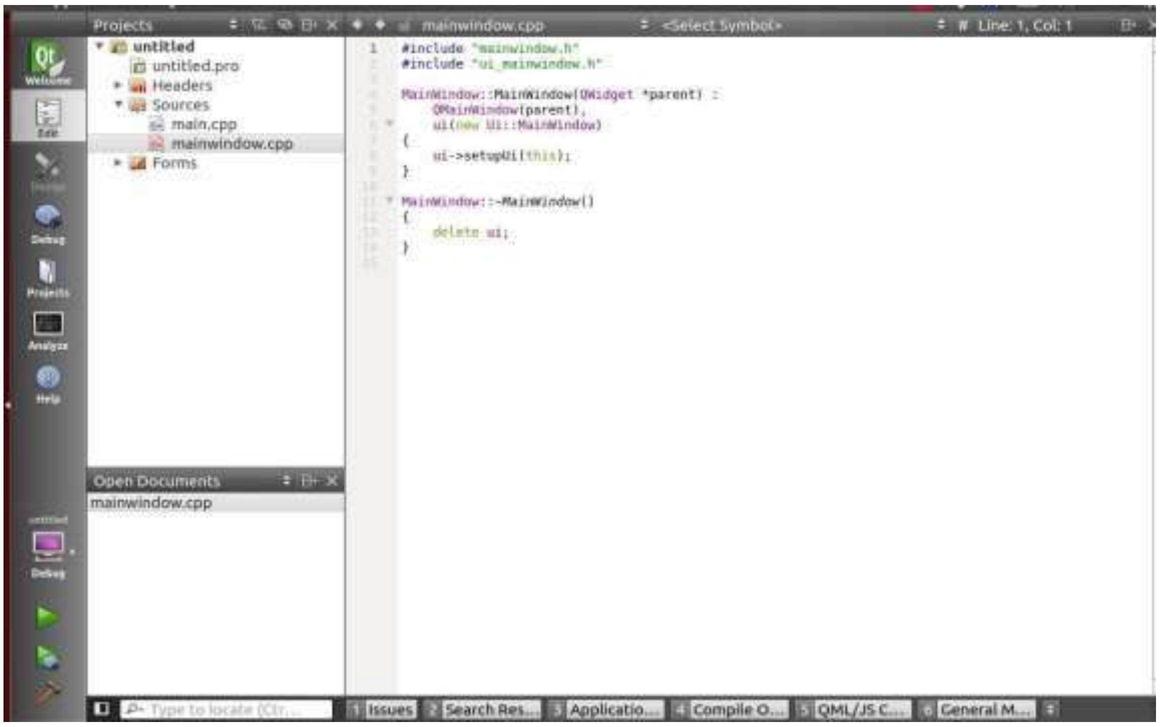
Creates a Qt application for the desktop. Includes a Qt Designer-based main window.

Preselects a desktop Qt for building the application if available.

Supported Platforms: Desktop

在

Set the project name and path in the pop-up window. Then continue to the next step, choose the default method, and click Finish on the last page to complete the project creation and generate default code. Developers can develop their own functional design on basis of the default code.



```
#include "mainwindow.h"
#include "ui_mainwindow.h"

MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);
}

MainWindow::~MainWindow()
{
    delete ui;
}
```