**MYD-JA5D4X Linux**

**Development Manual**

Version 1.0

May.28.2015

**Revision History**

|  |  |  |
| --- | --- | --- |
| **Version Number** | **Instruction** | **Date** |
| V1.0 | Initial Version | 2015.05.28 |

# Contents

[Contents 1](#_Toc420601097)

[Chapter 1 Introduction About Software Resources 3](#_Toc420601098)

[1.1 Overview 3](#_Toc420601099)

[1.1.1 Software Resources 3](#_Toc420601100)

[Chapter 2 Build Linux Develop Environment 5](#_Toc420601101)

[2.1 Create Working Directory 5](#_Toc420601102)

[2.2 Setup Cross-compiler 5](#_Toc420601103)

[2.3 Install Other Necessary Tools 5](#_Toc420601104)

[Chapter 3 Build the Source Code 6](#_Toc420601105)

[3.1 Build Bootloader 6](#_Toc420601106)

[3.2 Build Linux Kernel 6](#_Toc420601107)

[Chapter 4 Program the MYD-JA5D4X 8](#_Toc420601108)

[Chapter 5 Linux Applications Development 9](#_Toc420601109)

[5.1 Embedded Development Environment 9](#_Toc420601110)

[5.1.1 Setup TFTP Service 9](#_Toc420601111)

[5.1.2 Setup NFS Service 10](#_Toc420601112)

[5.1.3 SSH login 11](#_Toc420601113)

[5.2 Demo Applications 12](#_Toc420601114)

[5.2.1 GPIO 13](#_Toc420601115)

[5.2.2 Key&LED 13](#_Toc420601116)

[5.2.3 NET 14](#_Toc420601117)

[5.2.4 RTC 14](#_Toc420601118)

[5.2.5 TWI 15](#_Toc420601119)

[5.2.6 RS485 15](#_Toc420601120)

[5.2.7 RS232 15](#_Toc420601121)

[5.2.8 Audio 16](#_Toc420601122)

[5.2.9 EEPROM 16](#_Toc420601123)

[5.2.10 FrameBuffer 17](#_Toc420601124)

[5.2.11 led\_play 17](#_Toc420601125)

[Chapter 6 Qt Development 19](#_Toc420601126)

[6.1 Usage about Qt SDK Included in the CD-ROM 19](#_Toc420601127)

[6.2 Cross Compile Qt Development Environment 19](#_Toc420601128)

[6.3 Transplant Qt to the Development Board 21](#_Toc420601129)

[6.4 Cross-compile Qt Applications 21](#_Toc420601130)

[Appendix 1 Warranty & Technical Support Services 23](#_Toc420601131)

# Introduction About Software Resources

## Overview

The MYD-JA5D4X provides abundant hardware and software resources. This manual starts from building the environment and introduces the specific steps about how to develop an embedded Linux program. The development commands of this manual are illustrated based-on the Ubuntu system.

### Software Resources

| **Category** | **Name** | **Remark** | **Source** |
| --- | --- | --- | --- |
| **Boot**  **Loader** | MLO(SPL) | 1st Stage Bootloader | Yes |
| U-boot | 2nd Stage Bootloader | Yes |
| **Linux**  **Kernel** | Linux 3.12.10 | Customized Linux Kernel for the MYD-JA5D4X | Yes |
| **Device**  **Driver** | USB Host | USB Host Driver | Yes |
| USB Device | USB Device Driver | Yes |
| I2C(TWI) | i2c-dev Driver | Yes |
| Ethernet | Ethernet Driver | Yes |
| MMC | MMC/SD/TF Card Driver | Yes |
| eMMC | eMMC Driver | Yes |
| LCD | DSS Driver  7-inch LCD Screen Supported | Yes |
| RTC | Real Time Clock Driver | Yes |
| HDMI | SIL9022A Driver | Yes |
| Touch | Capacity Touch Screen Driver | Yes |
| Button | Button Driver | Yes |
| USART | USART Driver | Yes |
| LED | LED Driver | Yes |
| GPIO | GPIO Driver | Yes |
| WDI | WatchDog Driver | Yes |
| Camera | Camera Driver | Yes |
| QSPI | QSPI Flash Driver | Yes |
| EERPOM | EERPOM Driver | Yes |
| **Filesystem** | Ramdisk | Fast RAM Filesystem with Update Tool | Bin |
| Matrix | Matrix Launcher GUI by TI | Bin |
| Buildroot | QT 4.8.5 Library Incorporated | Bin |
| **Applications** | KEY&LED | KEY and LED Demo | Yes |
| NET | TCP/IP Socket C/S Demo | Yes |
| RTC | Real-time Clock Demo | Yes |
| I2C(TWI) | i2c-dev API Demo | Yes |
| RS485 | RS485 API Demo | yes |
| RS232 | RS232 API Demo | yes |
| Audio | Audio Demo | yes |
| EEPROM | EEPROM API Demo | Yes |
| Framebuffer | Display Demo | Yes |
| **Tools** | Cross Compiler | Linaro GCC 4.7 | Bin |

Table ‑ Linux Software Resources

# Build Linux Develop Environment

## Create Working Directory

Copy MYD-JA5D4X Linux SDK source form the product DVDROM to the Host:

$ mkdir -p <WORKDIR>

$ cp -a <DVDROM>/04-Linux\_Source/\* <WORKDIR>

## Setup Cross-compiler

$ cd <WORKDIR>/Toolchain

$ tar -xvjf \

gcc-linaro-arm-linux-gnueabihf-4.7-2013.03-20130313\_linux.tar.bz2

$ export PATH=$PATH:<WORKDIR>/Toolchain/\

gcc-linaro-arm-linux-gnueabihf-4.7-2013.03-20130313\_linux/bin

$ export CROSS\_COMPILE=arm-linux-gnueabihf-

After executing above *export* commands, input *arm* and press the *Tab* key to check whether or not these *export* settings have been applied successfully. But these settings are only valid for the current terminal; if it is required to keep them permanently, please add above *export* commands to script file: *~/.bashrc*

## Install Other Necessary Tools

Install other necessary tools:

$ sudo apt-get install build-essential git-core libncurses5-dev

$ sudo apt-get install flex bison texinfo zip unzip zlib1g-dev gettext

$ sudo apt-get install gperf libsdl-dev libesd0-dev libwxgtk2.6-dev

$ sudo apt-get install uboot-mkimage

$ sudo apt-get install g++ xz-utils

# Build the Source Code

## Build Bootloader

Enter the *Bootloader* directory, Unzip the U-boot source:

$ cd <WORKDIR>/Bootloader

$ tar -xvjf u-boot-2013.10-ti2013.12.01.tar.bz2

$ cd u-boot-2013.10-ti2013.12.01

Start to compile:

$ make ARCH=arm CROSS\_COMPILE=arm-linux-gnueabihf- distclean

$ make ARCH=arm CROSS\_COMPILE=arm-linux-gnueabihf- <config>

$ make ARCH=arm CROSS\_COMPILE=arm-linux-gnueabihf-

When the build is complete, *MLO* (which generates only with the *myd\_ja5d4x\_config* configuration), *u-boot.img*, and *u-boot.bin* files are generated in the working directory. Where <config> refers to the name of configuration option. Different boot mode requires different configuration options. There are two options as follows:

|  |  |  |
| --- | --- | --- |
| **Bootmode** | **Compiler Option** | **Output** |
| Q-SPI Flash | myd\_ja5d4x\_qspiboot\_config | u-boot.bin |
| Micro SD | myd\_ja5d4x\_config | u-boot.img, MLO |

Table ‑

## Build Linux Kernel

Enter *Kernel* directory, Unzip the Linux Kernel source:

$ cd <WORKDIR>/Kernel

$ tar -xvjf linux-3.12.10-ti2013.12.01.tar.bz2

$ cd linux-3.12.10-ti2013.12.01

Start to compile:

$ make ARCH=arm CROSS\_COMPILE=arm-linux-gnueabihf- distclean

$ make ARCH=arm CROSS\_COMPILE=arm-linux-gnueabihf- \

myd\_ja5d4x\_defconfig

$ make ARCH=arm CROSS\_COMPILE=arm-linux-gnueabihf- zImage dtbs

When the build is complete, the Kernel image *zImage* is generated in path:

<WORKDIR>/Kernel/linux-3.12.10-ti2013.12.01/arch/arm/boot/zImage

And the device tree file *myd\_ja5d4x.dtb* is generated in path:

<WORKDIR>/Kernel/linux-3.12.10-ti2013.12.01/arch/arm/boot/dts/myd\_ja5d4x.dtb

# Program the MYD-JA5D4X

# Linux Applications Development

## Embedded Development Environment

An embedded application development usually begins with cross compiling the source code on a PC, and follows by transferring the program to a target board by means of Ethernet or serial. For convenience of debugging, you should first set up a file sharing environment.

### Setup TFTP Service

Trivial File Transfer Protocol(TFTP), is a simple, lock-step, file transfer protocol which allows a client to get from or put a file onto a remote server. It is usefully for transferring files between a Host and a connected development board. In addition, you can use TFTP to load the kernel from the net when the U-boot is in booting.

Install tftp server to ubuntu, for example:

$ sudo apt-get install tftp-hpa tftpd-hpa

Create a working directory for a tftp server

$ mkdir <WORKDIR>/tftpboot

$ chmod 777 <WORKDIR>/tftpboot

Setup tftp server:

$ sudo vi /etc/default/tftpd-hpa

Modify or add the following fields:

TFTP\_DIRECTORY="<WORKDIR>/tftpboot"

TFTP\_OPTIONS="-l -c -s"

Restart the tftp server:

$ sudo service tftpd-hpa restart

Execute the following command to transfer a file from the Host (IP :192.168.1.111) to the development board:

# tftp -l <file-name> -r <file-name> -g 192.168.1.111 69

From the development board controlling terminal, execute the following command to transfer a file form the board to the Host:

# tftp -l <file-name> -r <file-name> -p 192.168.1.111 69

In the U-boot command line, load the Linux Kernel (zImage) from the TFTP server (the *tftpboot* directory of the Host) as follows:

# setenv serverip 192.168.1.111

# setenv ipaddr 192.168.1.222

# setenv ethaddr 00:01:02:03:04:05

# tftp zImage <mem-addr>

# bootz <mem-addr>

### Setup NFS Service

NFS is a network file system, which allows direct file sharing between different Hosts over a network. In addition to being used to mount common directory, the NFS can also be used to mount directory over the NFS as the root filesystem when the Linux is loading.

Install the NFS server to ubuntu on the host, for example:

$ sudo apt-get install nfs-kernel-server

Edit the *exports* file to add the *nfs* folder:

$ sudo vi /etc/exports

Add the following content to the end of the *exports* file to set */home/myir/nfs* as the NFS directory:

/home/myir/nfs \*(rw,subtree\_check,no\_root\_squash,no\_all\_squash,sync)

Set permissions for the NFS directory:

$ chmod 777 –R /home/myir/nfs

Modify configuration for the NFS, bind the port:

$ sudo vi /etc/default/nfs-kernel-server

Modify RPCMOUNTDOPTS as:

#RPCMOUNTDOPTS=--manage-gids

RPCMOUNTDOPTS="-p 13100"

Restart the NFS server:

$ sudo service nfs-kernel-server restart

Test the NFS on the local Host:

$ sudo mount -t nfs 127.0.0.1:/home/myir/nfs /mnt

Mount the NFS on the development Board, and set Host IP, such as:

$ sudo ifconfig eth0 192.168.1.111

Set the Eval board IP, such as:

# ifconfig eth0 192.168.1.222

Ping the Host to test the connectivity between the development board and the PC:

# ping 192.168.1.111

Create a new directory on the development board:

# mkdir -p /mnt/nfs

Mount the NFS directory:

# mount -t nfs -o nolock,rw 192.168.1.111:/home/myir/nfs /mnt/nfs

So far, we have successfully mounted a shared directory over the NFS. It is convenient to share files between the Host and the development board through this directory. A target program can also be started directly in the shared directory */mnt/nf*.

In addition to being used to mount common directory, the NFS can also be used to mount directory over the NFS as the root filesystem when the Linux is loading. Please refer to the follow website for more detailed information:

<http://wiki.emacinc.com/wiki/Booting_with_an_NFS_Root_Filesystem>

### SSH login

You can remotely manipulate the Linux command line through the network by SSH connection in the same way as through the serial terminal.

The *dropbear ssh* service is activated by default. You can open the SSH login by starting a SSH protocol based software and setting the IP for connection, which is shown as follows:

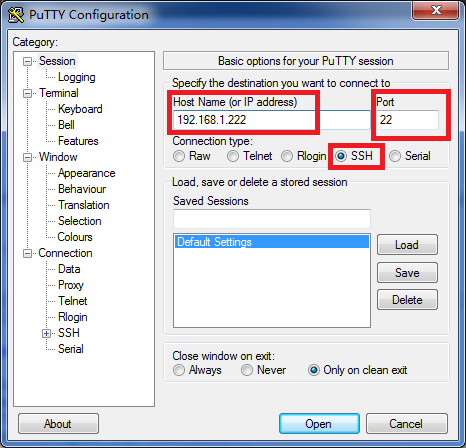


Figure 5-1

Enter the target IP and port number in relevant textbox, click *open* to connect the development board, and then click *YES* to ignore the pop-up dialog about security warnings. In the command line interface, enter the username and password. By default, the username is *root* and the password is *NULL*. But for some filesystem, the password cannot be *NULL*, you are required to set this password before using the SSH login function.

## Demo Applications

MYD-JA5D4X provides some demo programs for common peripherals. The programs and source are located in the *<WORKDIR>/Examples/* directory, Please compile them according to the *Makefile* or *README* file:

$ cd <WORKDIR>/Examples/<app dir>

$ export CROSS\_COMPILE=arm-linux-gnueabihf-

$ make

You must have the permission to start a program over the development board, if not, execute the following command to add it.

# chmod +x <program-to-be-execute>

### GPIO

The GPIO5\_5, taken as an example here, describes how to use the Linux command line to operate the GPIO.

The GPIO5\_5 corresponds to Pin-6 of MYD-JA5D4X expansion interface J4

The number of GPIO5\_5 in Linux is GPIO165, which is 5\*32+5=165.

Export GPIO 165:

# echo 165 > /sys/class/gpio/export

Examine the newly exported GPIO port:

# ls /sys/class/gpio/gpio165

Set the GPIO output direction:

# echo "out" > /sys/class/gpio/gpio165/direction

Set GPIO to output high level:

# echo 1 > /sys/class/gpio/gpio165/value

Set GPIO to output low level:

# echo 0 > /sys/class/gpio/gpio165/value

Unexport the GPIO port:

# echo 165 > /sys/class/gpio/unexport

### Key&LED

This example describes how to operate the on-board USER key and LED by using Linux API. Please refer to the source for more detailed instructions.

Copy the executable program *key\_led* in directory *<WORKDIR>/Examples/key\_led* to the development board and execute it. The program will light the LED0~LED3 automatically, it will also print relevant key information when the SW3 and SW4 button are pushed down.

# ./key\_led

status led0 on

status led0 off

status led1 on

status led1 off

status led2 on

status led2 off

status led3 on

status led3 off

Hit any key on board ......

key 102 Released

key 102 Pressed

key 158 Released

key 158 Pressed

### NET

This example shows how to create a simple C/S program using the TCP/IP socket API. Please refer to the source for more detailed information. Copy the executable program *arm\_client* in directory *<WORKDIR>/Examples/network* to the development board and *pc\_server* to the Host PC. Connect the development board and the Host to the network and set their IP Addresses:

$ sudo ifconfig eth0 192.168.1.111

# ifconfig eth0 192.168.1.222

Run the service program on the PC:

$ ./pc\_server

Run the client program on the board, it will receive the following text:

# ./arm\_client 192.168.1.111

form server: Make Your idea Real!

At the same time, the PC receives the client IP with which the PC is connected:

$ ./pc\_server

REC FROM： 192.168.1.222

### RTC

This example shows how to set and inquire RTC using the Linux API. Please refer to the source for more detailed information.

Copy the executable program *rtc\_test* in directory *<WORKDIR>/Examples/rtc* to the development board. Execute the program to update the time as follows:

# ./rtc\_test -s 11 03 40 2014 08 14

date/time is updated to: 14-8-2014, 11:03:40.

Then inquire the RTC and check the time changes:

# ./rtc\_test

Current RTC date/time is 14-8-2014, 11:03:50.

# ./rtc\_test

Current RTC date/time is 14-8-2014, 11:03:51.

### TWI

### RS485

Copy the executable file rs485 to the board, it can be find in the *<WORKDIR>/Examples/rs485*. Connect U16-A and U16-B to another board’s U16-A and U16-B. And execute the following commands, respectively, PC serial terminal to print the following information:

# ./rs485 -d /dev/ttyO1 -b 9600

SEND:0123456789

RECV:0123456789, total:10

SEND:0123456789

RECV:0123456789, total:10

SEND:0123456789

RECV:0123456789, total:10

SEND:0123456789

RECV:0123456789, total:10

SEND:0123456789

RECV:0123456789, total:10

SEND:0123456789

RECV:0123456789, total:10

### RS232

Copy the executable file client and server to board, it can be find in the *<WORKDIR>/Examples/rs232.* Connect J13 to another board J13. Enter the following command:

# ./rs232 -d /dev/ttyO2 -b 115200

SEND:0123456789

RECV:0123456789, total:10

SEND:0123456789

RECV:0123456789, total:10

SEND:0123456789

RECV:0123456789, total:10

SEND:0123456789

RECV:0123456789, total:10

SEND:0123456789

RECV:0123456789, total:10

SEND:0123456789

RECV:0123456789, total:10

### Audio

This example show how to use the Linux API to control audio input and output.

Copy the app to Development board from *<WORKDIR>/Examples/audio*. The headset plugged into the audio output port J10, the audio input to the audio input port J11, The debug serial ports J12 connected to the PC and the PC serial port baud rate is set to 115200, the data bits of 8, stop bit is 1,and parity bits is none. Execute the audio program beginning to record. press ctrl-c to end recording and begin to play.

# ./audio

recording... press 'Ctrl-c' to stop

record stopped, playing...

play done!

### EEPROM

This example shows how to read and write the EEPROM of the development board using the EEPROM API. Please refer to the source for more detailed information.

Disable EEPROM WP:

# echo 103 > /sys/class/gpio/export"

# echo "out" > /sys/class/gpio/gpio103/direction"

# echo 0 > /sys/class/gpio/gpio103/value"

Note: If the gpio103 has been exported, the first command will fail, but this does not affect the execution of the program, and the remaining operations can still be continued.

Start the program:

# ./eeprom\_test

eeprom size: 32 KB

write 'eeprom write/read test!' to eeprom

get the following string from eeprom:

eeprom write/read test!

### FrameBuffer

This example demonstrates how to operate the FrameBuffer of Linux to achieve color grid testing.

Copy the executable program *framebuffer\_test* in directory *<WORKDIR>/Examples/framebuffer* to the development board. If the matrix GUI is activated, use the following command to stop it.

# /etc/init.d/matrix-gui-2.0 stop

Run the program:

# ./framebuffer\_test

The terminal will print the information of displayer with 8 background colors varying in sequence and color bars in the end. The HDMI output is as follows:

The framebuffer device was opened successfully.

vinfo.xres=1024

vinfo.yres=768

vinfo.bits\_per\_bits=32

vinfo.xoffset=0

vinfo.yoffset=0

finfo.line\_length=4096

The framebuffer device was mapped to memory successfully.

......

Then execute the following command to restart the matrix GUI.

# /etc/init.d/matrix-gui-2.0 start

### led\_play

This example demonstrates how to read the keys in the user space and how to control user led lights.

Copy the executable program *led\_play* in directory *<WORKDIR>\Examples\led\_play* to the development board and start the program:

# ./led\_play

Keeping the SW3 button down for 3 seconds will trigger the blink of three LEDs: D24, D25 and D26. Please refer to the source for more detailed information.

NOTE: This APP has been embedded into the Matrix GUI launcher and will startup automatically. It can be triggered directly by keeping the SW3 button down for 3 seconds without the process of copy and exection steps. Users can disable the automatic exection of *led\_play* by deleting the */etc/rc5.d/S99led\_play* file.

# Qt Development

This section describes the methods and procedures to develop GUI applications with Qt on the MYD-JA5D4X. It consists of two parts, the usage of Qt SDK is the first part; for general Qt application development, we use Qt SDK in the CD-ROM only. The second part describes how to customize Qt development environment from Qt source when the Qt library provided in the CD-ROM cannot meet the development process.

## Usage about Qt SDK Included in the CD-ROM

⑴ Unzip *tslib* to the PC:

$ sudo tar xvjf \

/media/cdrom/05-Linux\_Source/Qt\_Arm/tslib-prebuild.tar.bz2 -C /opt

⑵ Unzip Qt SDK to the PC:

$ sudo tar xvjf \

/media/cdrom/05-Linux\_Source/Qt\_Arm/qt-4.8.5-sdk.tar.bz2 -C /opt

## Cross Compile Qt Development Environment

⑴ Please refer to section [1.4.2] for installing cross compiler tool and setting the environment variables.

⑵ Install *automake*, *libtool* and *autoconf* packages, compile and install *tslib*:

$ sudo apt-get install automake libtool autoconf

$ cd <WORKDIR>

$ cp /media/cdrom/05-Linux\_Souce/Qt\_Arm/tslib-1.4.tar.bz2 ./

$ tar –jxf tslib-1.4.tar.bz2

$ cd tslib

$ ./ts-build

$ sudo make install

**NOTE:**  If error occurs on executing *ts-build*, modify the cross compiler tool path in *ts-build* file.

After compilation, *tslib* will be installed under directory */opt/tslib*. Replace the text "# module\_raw input" in the second line of *tslib/etc/ts.conf* under current directory with "module\_raw input". Be sure that "module\_raw input" must be at the top as shown in Figure 6‑1:

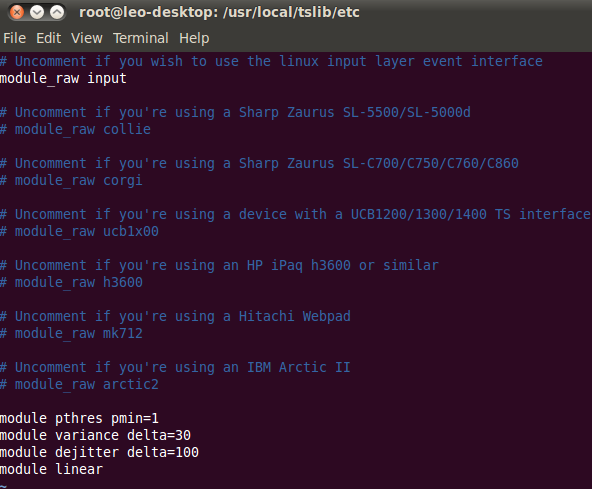


Figure ‑

⑶ Build Qt

Copy Qt source to the working directory:

$ cd <WORKDIR>

$ cp \

/media/cdrom/05-Linux\_Source/Qt\_Arm/qt-everywhere-opensource-src-4.8.5.tar.bz2\ ./

Install necessary toolkits:

$ sudo apt-get install xorg-dev libfontconfig1-dev

libfreetype6-dev libx11-dev libxcursor-dev libxext-dev

libxfixes-dev libxft-dev libxi-dev libxrandr-dev libxrender-dev

Start to Compile:

$ tar -jxf qt-everywhere-opensource-src-4.8.5.tar.bz2

$ cd qt-everywhere-opensource-src-4.8.5

$ ./qt-build

$ sudo make install

After compilation, qt-4.8.5 will be installed to directory */opt/qt-4.8.5*.

## Transplant Qt to the Development Board

⑴ Copy the *tslib* installation files in directory */usr/local/tslib* to the *rootfs /usr/local/ tslib* directory which is used to compile the file system.

$ cp –ra /opt/tslib <WORKDIR>/rootfs/opt/

⑵ Copy qt-4.8.5 installation files in directory */opt/* qt-4.8.5 to *rootfs /opt*:

$ cp –r /opt/qt-4.8.5 <WORKDIR>/rootfs/opt/

⑶ Enter directory *<WORKDIR>/Filesystem/rootfs/etc/init.d/*, edit the script *qt.sh* as follows:

export TSLIB\_CONSOLEDEVICE=none

export TSLIB\_FBDEVICE=/dev/fb0

export TSLIB\_TSDEVICE=/dev/input/touchscreen0

export LD\_LIBRARY\_PATH=/lib:/usr/lib:/opt/tslib/lib:/opt/qt-4.8.5/lib

export QT\_QWS\_FONTDIR=/opt/qt-4.8.5/lib/fonts

export QWS\_USB\_KEYBOARD=/dev/input/event2

export PATH=/bin:/sbin:/usr/bin/:/usr/sbin:/opt/tslib/bin

if grep "display1" /proc/cmdline > /dev/null ; then

export QWS\_MOUSE\_PROTO="Tslib:/dev/input/touchscreen0 MouseMan:/dev/input/mice"

else

export QWS\_MOUSE\_PROTO="Tslib:/dev/input/touchscreen0"

fi

export QWS\_DISPLAY=:1

Export QWS\_DISPLAY=:1Modify to the *rcS* file in directory <WORKDIR>/Filesystem/rootfs/etc/init.d/, add the following contents at the end of the document:

if [ -e /etc/init.d/qt.sh ];then

/etc/init.d/qt.sh

fi

⑷ Re-transplant a filesystem with Qt library and *tslib* file.

## Cross-compile Qt Applications

⑴ Configure the Qt compile environment on the PC:

$ export QT\_PREFIX=/opt/qt-4.8.5

$ export PATH=${QT\_PREFIX}/bin:$PATH

$ export QMAKESPEC=${QT\_PREFIX}/mkspecs/qws/linux-arm-g++

⑵ Create the source file:

$ mkdir hellomyir

$ cd hellomyir

$ gedit hellomyir.cpp

Input the following code:

#include <QApplication>

#include <QLabel>

int main(int argc, char \*\*argv)

{

QApplication app(argc,argv);

QLabel label("Make Your idea Real!");

label.show();

return app.exec();

}

⑶ Build:

$ qmake -project

$ qmake

$ make

⑷ Copy the *hellomyir* file to the development board, and execute :

# ./hellomyir -qws

Copy the executable file *hellomyir* to the development board, and execute it. A Qt window of "Make Your idea Real!" will display on the LCD screen.

# Appendix 1 Warranty & Technical Support Services

**MYIR Tech Limited** is a global provider of ARM hardware and software tools, design solutions for embedded applications. We support our customers in a wide range of services to accelerate your time to market.

MYIR is an ARM Connected Community Member and work closely with ARM and many semiconductor vendors. We sell products ranging from board level products such as development boards, single board computers and CPU modules to help with your evaluation, prototype, and system integration or creating your own applications. Our products are used widely in industrial control, medical devices, consumer electronic, telecommunication systems, Human Machine Interface (HMI) and more other embedded applications. MYIR has an experienced team and provides custom design services based on ARM processors to help customers make your idea a reality.

The contents below introduce to customers the warranty and technical support services provided by MYIR as well as the matters needing attention in using MYIR’s products.

**Service Guarantee**

MYIR regards the product quality as the life of an enterprise. We strictly check and control the core board design, the procurement of components, production control, product testing, packaging, shipping and other aspects and strive to provide products with best quality to customers. We believe that only quality products and excellent services can ensure the long-term cooperation and mutual benefit.

**Price**

MYIR insists on providing customers with the most valuable products. We do not pursue excess profits which we think only for short-time cooperation. Instead, we hope to establish long-term cooperation and win-win business with customers. So we will offer reasonable prices in the hope of making the business greater with the customers together hand in hand.

**Delivery Time**

**MYIR will always keep a certain stock for its regular products. If your order quantity is less than the amount of inventory, the delivery time would be within three days; if your order quantity is greater than the number of inventory, the delivery time would be always four to six weeks. If for any urgent delivery, we can negotiate with customer and try to supply the goods in advance.**

**Technical Support**

**MYIR has a professional technical support team. Customer can contact us by email (**[support@myirtech.com](mailto:support@myirtech.com)**), we will try to reply you within 48 hours. For mass production and customized products, we will specify person to follow the case and ensure the smooth production.**

**After-sale Service**

**MYIR offers one year free technical support and after-sales maintenance service from the purchase date. The service covers:   
1. Technical support service**

1. MYIR offers technical support for the hardware and software materials which have provided to customers;
2. To help customers compile and run the source code we offer;
3. To help customers solve problems occurred during operations if users follow the user manual documents;
4. To judge whether the failure exists;
5. To provide free software upgrading service.

However, the following situations are not included in the scope of our free technical support service:

1. Hardware or software problems occurred during customers’ own development;
2. Problems occurred when customers compile or run the OS which is tailored by themselves;
3. Problems occurred during customers’ own applications development;
4. Problems occurred during the modification of MYIR’s software source code.

2. After-sales maintenance service

The products except LCD, which are not used properly, will take the twelve months free maintenance service since the purchase date. But following situations are not included in the scope of our free maintenance service:

1. The warranty period is expired;
2. The customer cannot provide proof-of-purchase or the product has no serial number;
3. The customer has not followed the instruction of the manual which has caused the damage the product;
4. Due to the natural disasters (unexpected matters), or natural attrition of the components, or unexpected matters leads the defects of appearance/function;
5. Due to the power supply, bump, leaking of the roof, pets, moist, impurities into the boards, all those reasons which have caused the damage of the products or defects of appearance;
6. Due to unauthorized weld or dismantle parts or repair the products which has caused the damage of the products or defects of appearance;
7. Due to unauthorized installation of the software, system or incorrect configuration or computer virus which has caused the damage of products.

Warm tips:

1. MYIR does not supply maintenance service to LCD. We suggest the customer first check the LCD when receiving the goods. In case the LCD cannot run or no display, customer should contact MYIR within 7 business days from the moment get the goods.
2. Please do not use finger nails or hard sharp object to touch the surface of the LCD.
3. MYIR suggests user purchasing a piece of special wiper to wipe the LCD after long time use, please avoid clean the surface with fingers or hands to leave fingerprint.
4. Do not clean the surface of the screen with chemicals.
5. Please read through the product user manual before you using MYIR’s products.
6. For any maintenance service, customers should communicate with MYIR to confirm the issue first. MYIR’s support team will judge the failure to see if the goods need to be returned for repair service, we will issue you RMA number for return maintenance service after confirmation.

3. Maintenance period and charges

a) MYIR will test the products within three days after receipt of the returned goods and inform customer the testing result. Then we will arrange shipment within one week for the repaired goods to the customer. For any special failure, we will negotiate with customers to confirm the maintenance period.

b) For products within warranty period and caused by quality problem, MYIR offers free maintenance service; for products within warranty period but out of free maintenance service scope, MYIR provides maintenance service but shall charge some basic material cost; for products out of warranty period, MYIR provides maintenance service but shall charge some basic material cost and handling fee.

4. Shipping cost

During the warranty period, the shipping cost which delivered to MYIR should be responsible by user; MYIR will pay for the return shipping cost to users when the product is repaired. If the warranty period is expired, all the shipping cost will be responsible by users.

5. Products Life Cycle

MYIR will always select mainstream chips for our design, thus to ensure at least ten years continuous supply; if meeting some main chip stopping production, we will inform customers in time and assist customers with products updating and upgrading.

**Value-added Services**

1. MYIR provides services of driver development base on MYIR’s products, like serial port, USB, Ethernet, LCD, etc.
2. MYIR provides the services of OS porting, BSP drivers’ development, API software development, etc.
3. MYIR provides other products supporting services like power adapter, LCD panel, etc.
4. ODM/OEM services.



**MYIR Tech Limited**

Room 1306, Wensheng Center, Wenjin Plaza,

North Wenjin Road, Luohu District, Shenzhen, China 518020

Support Email: [support@myirtech.com](mailto:support@myirtech.com)

Sales Email: [sales@myirtech.com](mailto:sales@myirtech.com)

Phone: +86-755-22984836

Fax: +86-755-25532724

Website: www.myirtech.com