



AXIOMTEK

rBOX630-FL

Linux

Software User's Manual



Disclaimers

This manual has been carefully checked and believed to contain accurate information. Axiomtek Co., Ltd. assumes no responsibility for any infringements of patents or any third party's rights, and any liability arising from such use.

Axiomtek does not warrant or assume any legal liability or responsibility for the accuracy, completeness or usefulness of any information in this document. Axiomtek does not make any commitment to update the information in this manual.

Axiomtek reserves the right to change or revise this document and/or product at any time without notice.

No part of this document may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of Axiomtek Co., Ltd.

Trademarks Acknowledgments

Axiomtek is a trademark of Axiomtek Co., Ltd.

Windows[®] is a trademark of Microsoft Corporation.

Other brand names and trademarks are the properties and registered brands of their respective owners.

©Copyright 2015 Axiomtek Co., Ltd.

All Rights Reserved

Oct. 2015, Version A1

Printed in Taiwan

Table of Contents

Disclaimers.....	ii
Chapter 1 Introduction.....	1
1.1 Specifications.....	2
Chapter 2 Getting Started	5
2.1 Connecting the rBOX630.....	5
2.1.1 Serial Console	6
2.1.2 SSH over Ethernet	8
2.2 How to Develop a Sample Program.....	10
2.2.1 Install Yocto Toolchain	10
2.2.2 Setting Up the Cross-Development Environment	10
2.2.3 Write and Compile Sample Program.....	10
2.3 How to Put and Run a Sample Program.....	11
2.3.1 Via FTP (Default disable)	11
2.3.2 Via USB Flash Drive.....	12
2.3.3 Via TFTP	12
Chapter 3 The Embedded Linux	13
3.1 Embedded Linux Image Managing	13
3.1.1 System Version	13
3.1.2 System Time.....	13
3.1.3 Internal RTC Time	13
3.1.4 External RTC Time.....	14
3.1.5 Adjusting System Time.....	14
3.2 Networking.....	15
3.2.1 FTP – File Transfer Protocol	15
3.2.2 TFTP – Trivial File Transfer Protocol.....	15
3.2.3 NFS – Network File System	15
Chapter 4 Programming Guide	17
4.1 librb217 API Functions	17
4.2 librb217 API Examples	22
4.2.1 Get Board ID and Power Status	22
4.2.2 COM Port Configuration	22
4.2.3 Watchdog Timer	23
4.2.4 Digital Input and Output.....	24
4.2.5 LEDs Control	24
4.3 CAN Bus.....	25

4.4	Compile Demo Program	27
4.4.1	Install Yocto Toolchain	27
4.4.2	Run demo program	27
Chapter 5	Board Support Package (BSP)	29
5.1	Host Development System Installation	29
5.1.1	Install Host System.....	29
5.1.2	Install Yocto development	30
5.1.3	Build and Install Yocto toolchain.....	31
5.2	U-Boot for rBOX630	32
5.2.1	Booting the System with an NFS Filesystem	32
5.2.2	Booting the System from eMMC	33
5.2.3	Booting the System from SD CARD (rBOX630 default)	33
5.3	Additional Information	34
5.3.1	Create a Bootable SD CARD for rBOX630	34

Chapter 1

Introduction

The extreme compact rBOX630 supports the low power RISC-based module (iMX6) processor with extended temperature range of -40°C to +70°C for using in wide range operating environments. Multiple built-in serial ports, high-speed LANs and USB 2.0 ports enable fast and efficient data computation, communication and acquisition. Its digital I/O feature provides users with the convenience of digital devices connection. Besides, the industrial grade IP40-rated rBOX630 meets Safety Agency requirements and has passed heavy industrial CE and FCC testing.

This user's manual is for the embedded Linux preinstalled in rBOX630. The embedded Linux is derived from Linux Yocto Board Support Package, which is based on Linux Kernel 3.0.35 and our hardware patches to suit rBOX630.

Software structure

The preinstalled embedded Linux image is located in eMMC Flash memory which is partitioned and formatted to accommodate boot loader, kernel and root filesystem. It follows standard Linux architecture to allow user to easily develop and deploy application software that follows Portable Operating System Interface (POSIX).

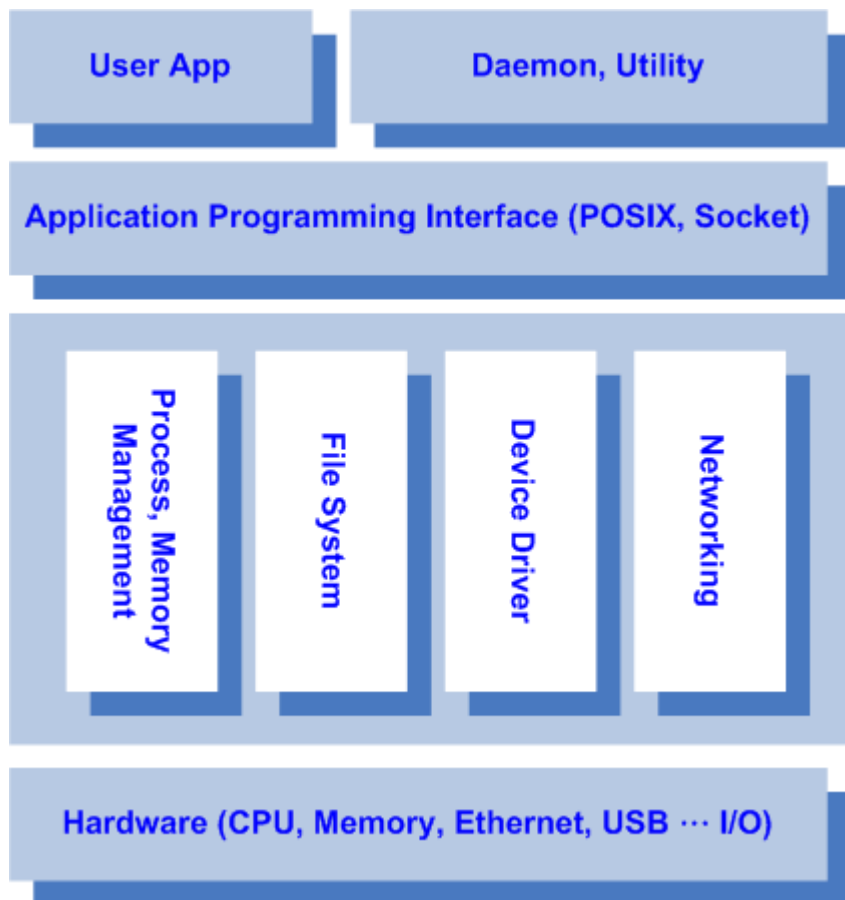
To facilitate user program in monitoring and controlling I/O device such as DIO, CAN, Watchdog Timer, the rBOX630 includes 'librb217.so' shared library.

In addition to ext3 and ext4 file system, this embedded Linux kernel is compiled with support for NFS, including server-side, client-side functionality and 'Root file system on NFS'. Using an NFS root mount we have several advantages such as:

- The root file system is not size-restricted by the device's storage like Flash memory.
- Change made to application files during development is immediately available to the target device.

For connectivity, this image includes most popular internet protocols, some servers and utilities not only making it easy for downloading/uploading files (Linux kernel, application program) or for debugging, but also communicating to outside world via Ethernet, WiFi and 3G.

For the convenience of manipulating embedded Linux, this image includes lots of popular packages such as busybox, udev, etc.



1.1 Specifications

- **OS: Linux**
 - Kernel: 3.0.35 (with Freescale and Axiomtek hardware modified patch)
- **Support Protocol Types**
 - ICMP.
 - TCP/IP.
 - UDP, DHCP, Telnet, HTTP, HTTPS, SSL, SMTP, ARP, NTP, DNS, PPP, PPPoE, FTP, TFTP, NFS.
- **Shell**
 - Bash
- **File system**
 - NFS, ext3, ext4
- **Daemons**
 - Telnetd: Telnet server daemon
 - FTPD: FTP server daemon

- **Utilities**
 - Telnet: Telnet client program
 - FTP: FTP client program
 - TFTP: Trivial File Transfer Protocol client

- **Packages**
 - **busybox**: Small collection of standard Linux command-line utilities
 - **udev**: A device manager for Linux kernel
 - **dosfstools** : Utilities for making and checking MS-DOS FAT file system
 - **e2fsprogs**: A set of utilities for maintaining the ext2, ext3 and ext4 file systems
 - **ethtool**: A Linux command for displaying or modifying the Network Interface Controller (NIC) parameters
 - **i2c-tools** : A heterogeneous set of I2C tools for Linux
 - **procps** : Utilities to report on the state of the system, including the states of running processes, amount of memory
 - **wireless-tools**: A package of Linux commands (simple text-based utilities/tools) intended to support and facilitate the configuration of wireless devices using the Linux Wireless Extension

- **Development Environment**
 - Host OS/ development OS: Ubuntu 14.04 LTS
 - Toolchain/ cross compiler: ARM, gcc-4.8.1 (Yocto project 1.5.4 Dora)

- **HW's Lib (Hardware's Library)**
 - **WiFi (Optional)**
 - Detect signal strength
 - Set AP connection
 - Set web, wpa, wpa2
 - Support search AP

 - **Digital I/O**
 - Read digital input
 - Write digital output

 - **CAN**
 - Support open/write/read/close functions

 - **3G**
 - Set number connection
 - Support user name/password
 - Detect signal strength

 - **GPS**
 - Detect signal strength
 - Support satellite positioning

 - **Watch Dog Timer**
 - Enable
 - Clean
 - Set timer

 - **COM**
 - RS-232/422/485 mode setting



Note

All specifications and images are subject to change without notice.

This page is intentionally left blank.

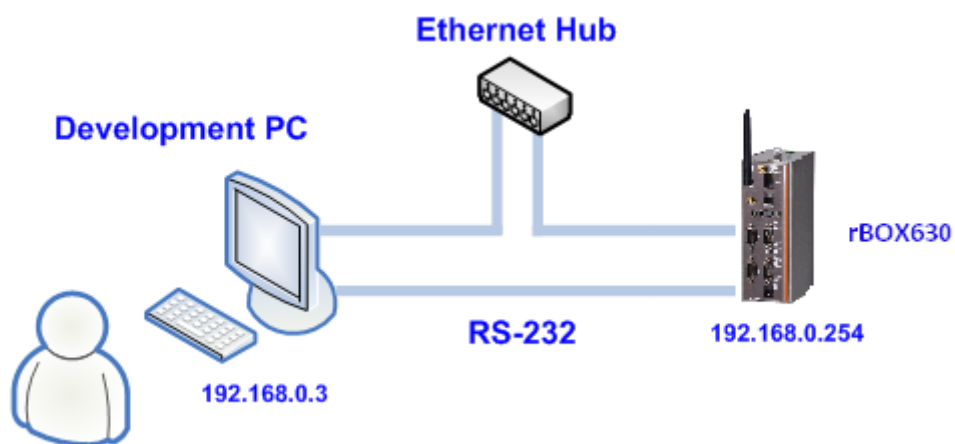
Chapter 2

Getting Started

2.1 Connecting the rBOX630

You can connect the rBOX630 to personal computer (PC) in two ways:

- Serial RS-232 console
- SSH over Ethernet



Note: Please download below data from Axiomtek's website as below list if you have the demand.

- BSP support package.
<http://www.axiomtek.com/products/ViewProduct.asp?view=8947>

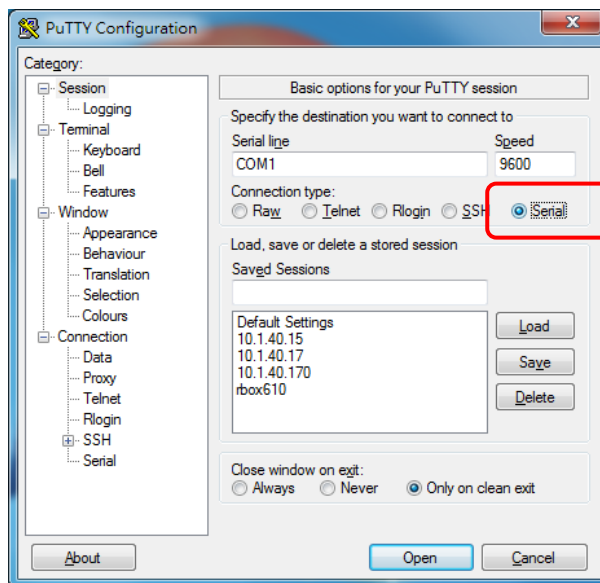
2.1.1 Serial Console

The serial console is a convenient interface for connecting rBOX630 to PC. First of all, it is very important to make sure that the serial port settings are as follows:

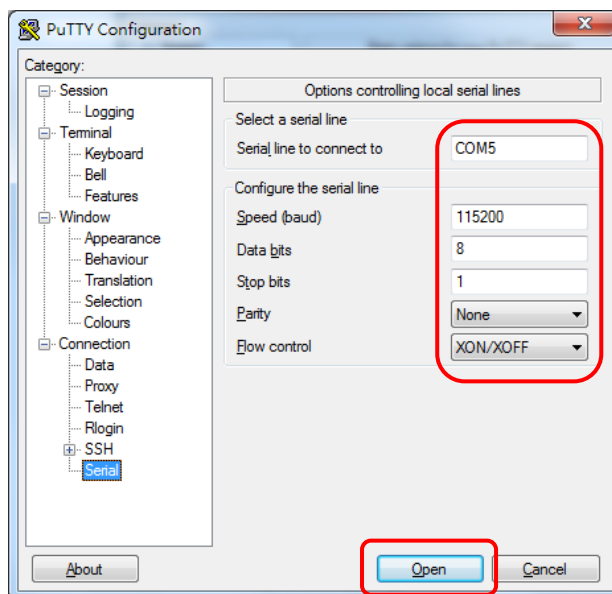
- Baudrate:** 115200 bps
- Parity:** None
- Data bits:** 8
- Stop bit:** 1
- Flow Control:** None

Here we use PuTTY to setup and link to the rBOX630. Learn how to do it with these step by step instructions:

1. Open PuTTY and choose 'Serial' as the connection type.



2. Configure the serial port correctly (see image below). Click Open and power on the rBOX630.



- The Bootloader default booting system setup to SD Card, If you don't have bootable SD Card and want boot from eMMC, Try type this command on u-boot.

```
Hit any key to stop autoboot: 0
Card did not respond to voltage select!
mmc2(part 0) is current device

MMC read: dev # 2, block # 2048, count 12288 ... Card did not respond to voltage!
MMC: block number 0x3800 exceeds max(0x0)
0 blocks read: ERROR
Wrong Image Format for bootm command
ERROR: can't get kernel image!
MX6SDL Q7M120 U-Boot > run bootcmd emmc
```

- If connection is established successfully, you should see the following image.

```
rpcbind: cannot create socket for tcp6
done.
eth1: ax88772b - Link status is: 1
INIT: Entering runlevel: 5
Starting system message bus: dbus.
Starting OpenBSD Secure Shell server: sshd
done.
Starting advanced power management daemon: apmd.
apmd[1703]: apmd 3.2.1 interfacing with apm driver 1.13 and APM BIOS 1.2
creating NFS state directory: done
NFS daemon support not enabled in kernel
Starting syslogd/klogd: done
Starting internet superserver: xinetd.
* Starting Avahi mDNS/DNS-SD Daemon: avahi-daemon
...done.
Starting Telephony daemon
Starting Linux NFC daemon
Starting OProfileUI server
Running local boot scripts (/etc/rc.local).
Stopping Bootlog daemon: bootlogd.

Poky (Yocto Project Reference Distro) 1.5.4 rbox630 /dev/ttymx0

rbox630 login: █
```

- To login, please enter 'root' (with no password).

```
done.
eth1: ax88772b - Link status is: 1
INIT: Entering runlevel: 5
Starting system message bus: dbus.
Starting OpenBSD Secure Shell server: sshd
done.
Starting advanced power management daemon: apmd.
apmd[1703]: apmd 3.2.1 interfacing with apm driver 1.13 and APM BIOS 1.2
creating NFS state directory: done
NFS daemon support not enabled in kernel
Starting syslogd/klogd: done
Starting internet superserver: xinetd.
* Starting Avahi mDNS/DNS-SD Daemon: avahi-daemon
...done.
Starting Telephony daemon
Starting Linux NFC daemon
Starting OProfileUI server
Running local boot scripts (/etc/rc.local).
Stopping Bootlog daemon: bootlogd.

Poky (Yocto Project Reference Distro) 1.5.4 rbox630 /dev/ttymx0

rbox630 login: root
root@rbox630:~# █
```

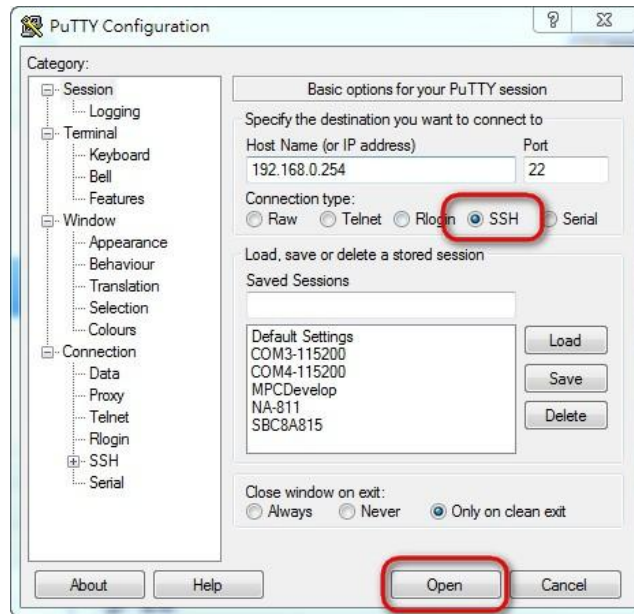
2.1.2 SSH over Ethernet

Now, we are going to connect the rBOX630 to PC over Ethernet. The following illustrations show how to do it under Windows® and Linux environment.

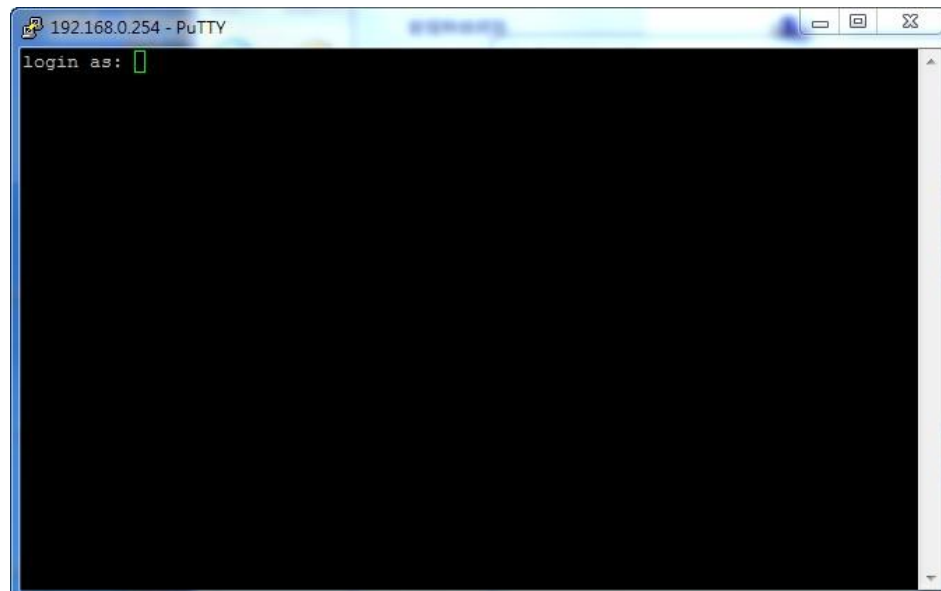
Note: rBOX630 LAN2 default IP address is 192.168.0.254.

For Windows® users:

1. Here we also use PuTTY to setup and link. Open PuTTY and choose 'SSH' as the connection type. Then set the IP address to 192.168.0.254 and click Open.



2. If connection is established successfully, you should see the following image.



3. To login rBOX630, please enter 'root' (with no password).



For Linux users:

1. Open terminal and keyin 'ssh' command.

```
jrtiger@ubuntu-vm:~$ ssh -l root 192.168.0.254
```

2. After the connection is established successfully.

```
jrtiger@ubuntu-vm:~$ ssh -l root 192.168.0.254  
root@rbox630:~#
```

2.2 How to Develop a Sample Program

In this section, learn how to develop a sample program for rBOX630 with the following step by step instructions. The sample program is named 'hello.c'.

2.2.1 Install Yocto Toolchain

Before you develop and compile sample program, you should install Yocto toolchain into development PC. To do so, refer to Chapter 5 Board Support Package.

2.2.2 Setting Up the Cross-Development Environment

Before you can develop using the cross-toolchain, you need to set up the cross-development environment, and then you can find this script in the directory you chose for installation.

```
~$ source /opt/poky/1.5.4/environment-setup-cortexa9hf-vfp-neon-poky-linux-gnueabi
```

2.2.3 Write and Compile Sample Program

```
~$ mkdir -p example
```

```
~$ cd example
```

Use vi to edit hello.c.

```
~$ vi hello.c
```

```
#include<stdio.h>
int main()
{
    printf("hello world\n");
    return 0;
}
```

To compile the program, please do:

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

```
jrtiger@test-H97M-D3H:~/examples$ arm-poky-linux-gnueabi-gcc hello.c -o hello
jrtiger@test-H97M-D3H:~/examples$
```

After compiling, enter the following command and you can see the 'hello' execution file.

```
~$ ls -l
```

```
jrtiger@test-H97M-D3H:~/examples$ ls -al
total 24
drwxrwxr-x  2 jrtiger jrtiger 4096  9月 14 18:08 .
drwxr-xr-x 13 jrtiger jrtiger 4096  9月 14 17:46 ..
-rwxrwxr-x  1 jrtiger jrtiger 9851  9月 14 18:08 hello
-rw-rw-r--  1 jrtiger jrtiger   76  9月 14 17:46 hello.c
jrtiger@test-H97M-D3H:~/examples$
```

2.3 How to Put and Run a Sample Program

In this section, we provide 3 methods showing how to put the 'hello' program into rBOX630 and execute it.

2.3.1 Via FTP (Default disable)

The rBOX630 has a built-in FTP server. Users can put 'hello' program to rBOX630 via FTP by following the steps below.

1. Enable FTPD daemon

Use vi to create /etc/xinetd.d/ftpd file

```
service ftp
{
    port = 21
    disable = no
    socket_type = stream
    protocol = tcp
    wait = no
    user = root
    server = /usr/sbin/ftpd
    server_args = -w /home/root
}
```

Restart Internet server

```
~# /etc/init.d/xinetd reload
~# /etc/init.d/xinetd restart
root@rbox630:~# /etc/init.d/xinetd reload
Reloading internet superserver configuration: xinetd.
root@rbox630:~# /etc/init.d/xinetd restart
Stopping internet superserver: xinetd.
Starting internet superserver: xinetd.
root@rbox630:~#
```

2. ~\$ ftp 192.168.0.254 (without username and password)

```
LTIB> ftp 192.168.0.254
Connected to 192.168.0.254.
220 Operation successful
Name (192.168.0.254:kevin):
230 Operation successful
Remote system type is UNIX.
Using binary mode to transfer files.
ftp>
```

3. ftp> bin
4. ftp> put hello

```
ftp> bin
200 Operation successful
ftp> put hello
local: hello remote: hello
200 Operation successful
150 Ok to send data
226 Operation successful
7800 bytes sent in 0.00 secs (35264.8 kB/s)
ftp>
```

If the operation is successful, you can see 'hello' program at rBOX630's /home/root directory.

```
root@rbox630:~# ls
hello
root@rbox630:~#
```

5. `~# chmod +x hello`
6. Run the 'hello' program.
`~# ./hello`

```
root@rbox630:~# chmod +x hello
root@rbox630:~# ./hello
hello world
root@rbox630:~# █
```

2.3.2 Via USB Flash Drive

Another method of putting 'hello' program into rBOX630 is via USB flash drive. Please follow the instructions below.

1. From the PC, copy 'hello' program to USB flash drive.
2. Attach USB flash drive to rBOX630.
3. `~# cp /media/sda1/hello /home/root`
4. `~# cd /home/root`
5. `~# chmod +x hello`
6. `~# ./hello`

```
root@rbox630:~# cp /media/sda1/hello /home/root/
root@rbox630:~# cd /home/root/
root@rbox630:~# chmod +x hello
root@rbox630:~# ./hello
hello world
root@rbox630:~#
```

2.3.3 Via TFTP

Originally the Host Development System Installation already has TFTP server installed. You can put the 'hello' program into rBOX630 via TFTP. Please follow the instructions below.

1. `~$ cp hello /tftpboot`

```
jrtiger@ubuntu-vm:~$ cp hello /tftpboot/
jrtiger@ubuntu-vm:~$ █
```

2. `~# tftp -g -r hello 192.168.0.3` (tftp server IP)
3. `~# chmod +x hello`
4. Run the 'hello' program.
`~# ./hello`

```
root@rbox630:~# tftp 192.168.0.3 -g -r hello
root@rbox630:~# chmod +x hello
root@rbox630:~# ./hello
hello world
root@rbox630:~# █
```


Chapter 3

The Embedded Linux

3.1 Embedded Linux Image Managing

3.1.1 System Version

This section describes how to determine system version information including kernel and root filesystem version.

Check kernel version with the following command:

```
~# uname -r
```

```
root@rbox630:~# uname -r
3.0.35-rBOX630-002
root@rbox630:~#
```

Check root filesystem with the login screen:

```
Poky (Yocto Project Reference Distro) 1.5.4 rbox630 /dev/ttyxc0
rbox630 login:
```

3.1.2 System Time

System time is the time value loaded from RTC each time the system boots up. Read system time with the following command:

```
~# date
```

```
root@rbox630:~# date
Thu Aug 20 13:00:05 UTC 2015
```

3.1.3 Internal RTC Time

The internal RTC time is read from i.MX processor internal RTC. Note that this time value is not saved, when system power is removed.

Read internal RTC time with the following command:

```
~# hwclock -r --rtc=/dev/rtc1
```

```
root@rbox630:~# hwclock -r --rtc=/dev/rtc1
Fri Jan 2 23:17:21 1970 0.000000 seconds
```

3.1.4 External RTC Time

The external RTC time is read from RS5C372 external RTC. When system power is removed, this time value is kept as RS5C372 is powered by battery.

Read external RTC time with the following command:

```
~# hwclock -r
```

```
root@rbox630:~# hwclock -r
Wed Sep 16 17:09:24 2015  0.000000 seconds
```

3.1.5 Adjusting System Time

Manually setting up the system time.

```
~# date -s YYYYMMDDHHmm.SS
```

```
root@rbox630:~# date -s 201509161714.05
Wed Sep 16 17:14:05 UTC 2015
```

Write sync time to internal RTC

```
$ hwclock -w --rtc=/dev/rtc1
```

Write sync time to external RTC

```
$ hwclock -w
```

```
root@rbox630:~# hwclock -w --rtc=/dev/rtc1
root@rbox630:~# hwclock -r --rtc=/dev/rtc1
Wed Sep 16 17:20:02 2015  0.000000 seconds
root@rbox630:~# hwclock -w
root@rbox630:~# hwclock -r
Wed Sep 16 17:20:11 2015  0.000000 seconds
```

3.2 Networking

3.2.1 FTP – File Transfer Protocol

FTP is a standard network protocol used to transfer files from one host to another host over TCP-based network.

The rBOX630 comes with a built-in FTP server. Section 2.1 shows the steps to put 'hello' program to rBOX630 via FTP.

3.2.2 TFTP – Trivial File Transfer Protocol

TFTP is a lightweight protocol of transfer files between a TFTP server and TFTP client over Ethernet. To support TFTP, this embedded Linux image has built-in TFTP client, so does its accompanying bootloader U-boot.

In Chapter 5, there are descriptions of TFTP server installation and kernel boot up process via TFTP. Section 2.3.3 shows you how to transfer file between server and client.

3.2.3 NFS – Network File System

NFS enables you to export a directory on an NFS server and mount that directory on remote client machine as if it were a local file system. Using NFS on target machine, we can have access to a huge number of files, libraries, and utilities during development and debugging, as well as booting up kernel.

This embedded Linux kernel is compiled with support for NFS, including server-side, client-side functionality and 'Root file system on NFS'. Section 5.1 and 5.2.1 show how to boot up embedded Linux with an NFS support.

This page is intentionally left blank.

Chapter 4

Programming Guide

We release a set of application programming interface (API) functions for users to access/control hardware. With these API functions, users can more easily design their own software. This chapter includes detailed description of each API function and step-by-step code samples showing how it works.

4.1 librb217 API Functions

The rBOX630 BSP includes 'librb217.so' shared library for users to access I/O and read back system information. This shared library is kept in BSP, you can find it in rBOX630-rb_lib-x.x.x.tar.bz2 of AxTools. Extract the compressed file, then besides the shared library you can also see a *demo* folder containing API header file and example programs.

Summary table of available API functions

No.	Function	Description
1	Get_BoardID()	Get board ID.
2	Get_PowerStatus()	Get power status.
3	Set_LEDStatus()	Set ACT/RDY LED status.
4	Set_ALARMLED()	Set ALARM LED status
5	Get_COMType()	Get COM port communication mode type.
6	Set_COMType()	Set COM port communication mode type.
7	Set_COMTermination()	Set termination of specified COM port.
8	CPLD_Get_WDTCCounter()	Get watchdog timer counter value.
9	CPLD_WDT_Enable()	Enable watchdog timer. Also used it to reset WDT counter.
10	CPLD_WDT_Disable()	Disable watchdog timer.
11	CPLD_WDTStatus()	Detect watchdog timer status.
12	Get_DI()	Read high or low state on digital input channels.
13	Get_DO()	Read high or low state on digital output channels.
14	Set_DO()	Set digital output channels to high or low state.
15	Set_CANTermination()	Set termination of specified CAN port.

Function: Get_BoardID()

Function	<code>__u16 Get_BoardID(void);</code>
Description	Get board ID.
Arguments	None.
Return	Board ID (in 2 bytes pattern).
Others	None.

Function: Get_PowerStatus()

Function	<code>int Get_PowerStatus(int number);</code>
Description	Get power status.
Arguments	number: Power number. 0: Power1. 1: Power2.
Return	0: Power fails. 1: Power is OK.
Others	None.

Function: Set_LEDStatus()

Function	<code>int Set_LEDStatus(int onoff);</code>
Description	Set ACT/RDY LED status.
Arguments	onoff: System LED status. 1: On. 0: Blink.
Return	0: No error. 1: Function fails.
Others	None.

Function: Set_ALARMLED ()

Function	<code>int Set_ALARMLED(int onoff);</code>
Description	Set ALARM LED status.
Arguments	onoff: Alarm LED status. 1: On. 0: Off.
Return	0: No error. 1: Function fails.
Others	None.

Function: Get_COMType()

Function	int Get_COMType(int number);
Description	Get COM port communication mode type.
Arguments	number: COM port number. 1: COM1. 2: COM2. 3: COM3. 4: COM4.
Return	0: Reserved. 1: RS232 Enable. 2: RS422/RS485_4W Enable. 3: RS485_2W Enable.
Others	None.

Function: Set_COMType()

Function	int Set_COMType(int number, int type);
Description	Set COM port communication mode type.
Arguments	number: COM port number. 1: COM1. 2: COM2. 3: COM3. 4: COM4. type: COM port mode type 0: Reserved. 1: RS232 Enable. 2: RS422/RS485_4W Enable. 3: RS485_2W Enable.
Return	0: No error. 1: Function fails.
Others	None.

Function: Set_COMTermination()

Function	int Set_COMTermination(int number, int onoff);
Description	Set termination of specified COM port.
Arguments	number: COM port number. 1: COM1. 2: COM2. 3: COM3. 4: COM4. onoff: Enable or disable termination. 1: Enable termination. 0: Disable termination.
Return	0: No error. 1: Function fails.
Others	None.

Function: CPLD_Get_WDTCOUNTER()

Function	__u8 CPLD_Get_WDTCOUNTER(void);
Description	Get watchdog timer counter value.
Arguments	None
Return	0~255 where 1 unit ~= 250ms.
Others	None.

Function: CPLD_WDT_Enable()

Function	int CPLD_WDT_Enable(__u8 timeout);
Description	Enable watchdog timer. Also use it to reset WDT counter.
Arguments	timeout: Timeout value. The range is from 0 to 255 where 1 unit ~= 250ms.
Return	0: No error. 1: Function fails.
Others	None.

Function: CPLD_WDT_Disable()

Function	int CPLD_WDT_Disable(void);
Description	Disable watchdog timer.
Arguments	None.
Return	0: No error. 1: Function fails.
Others	None.

Function: CPLD_WDTStatus()

Function	<code>__u8 CPLD_WDTStatus(void);</code>
Description	Detect watchdog timer status.
Arguments	None.
Return	0: If watchdog timer has not been triggered, return value = 0. 1: If watchdog timer has been triggered, return value = 1.
Others	None.

Function: Get_DI()

Function	<code>int Get_DI(__u8 *data);</code>
Description	Read high or low state on digital input channels.
Arguments	data: This function will store digital input data in this argument.
Return	0: No error. 1: Function fails.
Others	None.

Function: Get_DO()

Function	<code>int Get_DO(__u8 *data);</code>
Description	Read high or low state on digital output channels.
Arguments	data: This function will store digital output data in this argument.
Return	0: No error. 1: Function fails.
Others	None.

Function: Set_DO()

Function	<code>int Set_DO(__u8 data);</code>
Description	Set digital output channels to high or low state.
Arguments	data: Data to be written to digital output channels.
Return	0: No error. 1: Function fails.
Others	None.

Function: Set_CANTermination()

Function	int Set_CANTermination(int number, int onoff);
Description	Set termination of specified CAN port.
Arguments	number: CAN port number. 0: CAN0. 1: CAN1. onoff: Enable or disable termination. 1: Enable termination. 0: Disable termination.
Return	0: No error. 1: Function fails.
Others	None.

4.2 librb217 API Examples**Note**

Before using the API functions, remember to include header file librb217.h.

4.2.1 Get Board ID and Power Status

Use these system API functions to read boardID, check power status, and etc.

```
printf("This Board Device ID: %x\n\n", Get_BoardID());
printf("Check Power 1 status: ");
if(Get_PowerStatus(1) == 1) printf("Good!!\n\n");
else printf("Fail!!\n\n");
```

```
printf("Check Power 2 status: ");
if(Get_PowerStatus(2) == 1) printf("Good!!\n\n");
else printf("Fail!!\n\n");
```

4.2.2 COM Port Configuration

The COM port related API functions enable users to configure specified COM port communication mode to RS-232, 4-wired RS-422/485 and 2-wired RS-485.

```
printf("Read COM1 type: ");
if(Get_COMType(1) == 1) printf(" RS232\n\n");
else if(Get_COMType(1) == 2) printf(" RS422\n\n");
else if(Get_COMType(1) == 3) printf(" RS485\n\n");
else printf(" RSVD\n\n");
```

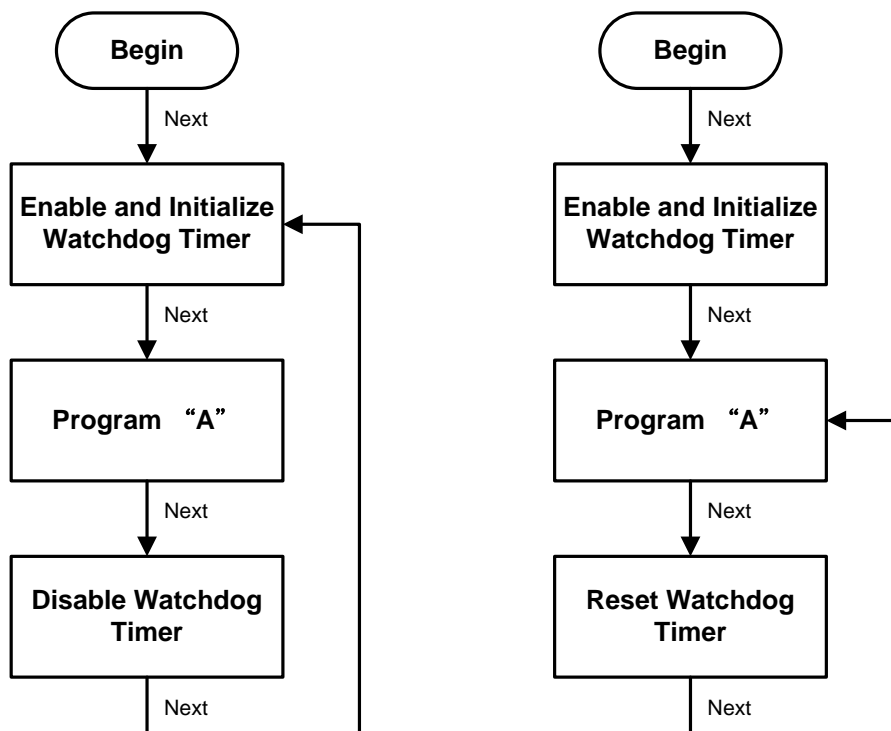
```
printf("Set COM2 to RS485..\n");
```

```
Set_COMType(2, 3);
printf("Read COM2 type: ");
if(Get_COMType(2) == 1) printf(" RS232\n\n");
else if(Get_COMType(2) == 2) printf(" RS422\n\n");
else if(Get_COMType(2) == 3) printf(" RS485\n\n");
else printf(" RSVD\n\n");
```

4.2.3 Watchdog Timer

Software stability is a major issue in most applications. Some embedded systems are not watched by human for 24 hours. It is usually too slow to wait for someone to reboot when the system hangs. The system needs to be able to reset automatically when things go wrong. The watchdog timer gives us a solution.

The watchdog timer is a counter that triggers a system reset when it counts down to zero from a preset value. The software starts the counter with an initial value and must reset it periodically. If the counter ever reaches zero, which means the software has crashed, the system will reboot. With these API functions, you can enable, disable, set watchdog timer value to 0 from 255 where 1 unit \approx 250ms, and etc..



```

printf("Enable CPLD WDT 30 sec..\n");
CPLD_WDT_Enable(120);
printf("CPLD WDT Counter: [%.2f] sec\n\n", (float)
CPLD_Get_WDTCounter()/4);
printf("Delay 5 sec..\n");
sleep(5);

printf("CPLD WDT Counter: [%.2f] sec\n\n", (float)
CPLD_Get_WDTCounter()/4);
printf("Disable CPLD WDT..\n");
CPLD_WDT_Disable();
  
```

4.2.4 Digital Input and Output

The digital related API functions allow users to set digital output signals to high or low and detect signal state from each digital channel.

```
unsigned char xch;
Get_DI(&xch);
printf("Current Digital-Input Data is %2X\n",xch);
Set_DO(0xAA);
Get_DO(&xch);
printf("Current Digital-Output Data is %2X\n",xch);
```

4.2.5 LEDs Control

Four different LEDs are supported by rBOX630: ACT/RDY LED, ALARM LED and Minicard strength, linked LED. Use the LED API functions to control LED on/off state.

```
printf("Turn on ACT/RDY LED ..\n");
Set_LEDStatus(1);
sleep(2);
printf("Turn off(blinking) ACT/RDY LED ..\n");
Set_LEDStatus(0)
printf("Turn on ALARM LED ..\n");
Set_ALARMLED(1);
Set_3GSLEDGreen();
sleep(2);
printf("Turn off ALARM LED ..\n");
Set_ALARMLED(0);
```

Use sysfs filesystem to control Minicard LED on/off state.

Turn on linked LED

```
~# echo 0 > /sys/class/leds/card1-link/brightness
```

Turn off linked LED

```
~# echo 255 > /sys/class/leds/card1-link/brightness
```

Turn on strength LED Green

```
~# echo 0 > /sys/class/leds/card1-good/brightness
```

Turn off strength LED Green

```
~# echo 255 > /sys/class/leds/card1-good/brightness
```

Turn on strength LED Amber

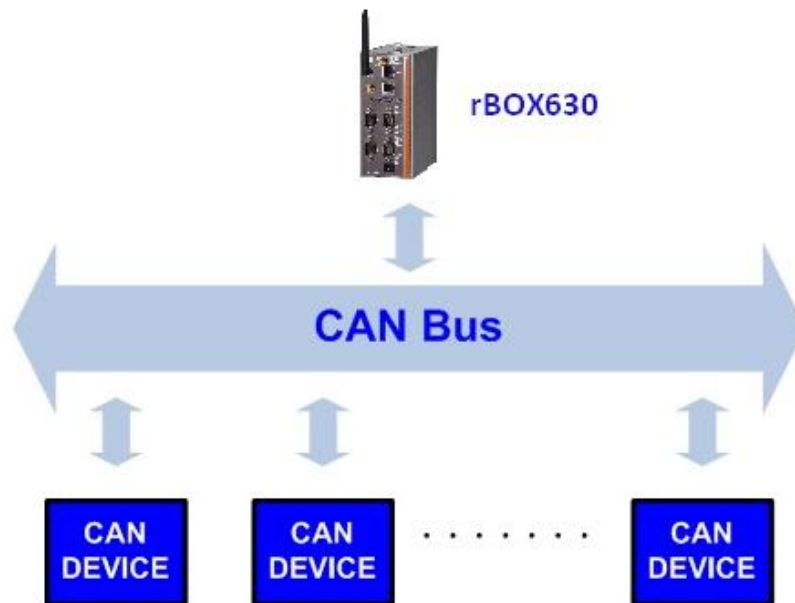
```
~# echo 0 > /sys/class/leds/card1-poor/brightness
```

Turn off strength LED Amber

```
~# echo 255 > /sys/class/leds/card1-poor/brightness
```

4.3 CAN Bus

The Controller Area Network (CAN) bus is a serial bus protocol that usually used in connecting intelligent industrial device networks and building smart automatic control systems. Use the SocketCAN API to read and write to CAN bus on rBOX630. An example program showing how it works is provided below.



```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <net/if.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/ioctl.h>
#include <linux/can.h>
#include <linux/can/raw.h>

#define SIOCSCANBAUDRATE    0x89F0

int main(void)
{
    int s,s1;
    int nbytes;
    struct sockaddr_can addr,addr1;
    struct can_frame frame,frame1;
    struct ifreq ifr,ifr1;
    int xBitRate=500000;
```

```
char *ifname = "can0";
char *ifname1 = "can1";

if((s = socket(PF_CAN, SOCK_RAW, CAN_RAW)) < 0) {
    perror("Error while opening socket");
    return -1;
}
if((s1 = socket(PF_CAN, SOCK_RAW, CAN_RAW)) < 0) {
    perror("Error while opening socket");
    return -1;
}

strcpy(ifr.ifr_name, ifname);
strcpy(ifr1.ifr_name, ifname1);
ioctl(s, SIOCGIFINDEX, &ifr);
ioctl(s1, SIOCGIFINDEX, &ifr1);

addr.can_family = AF_CAN;
addr.can_ifindex = ifr.ifr_ifindex;
ifr.ifr_ifru.ifru_ival = xBitRate;
ioctl(s, SIOCSCANBAUDRATE, &ifr);

addr1.can_family = AF_CAN;
addr1.can_ifindex = ifr1.ifr_ifindex;
ifr1.ifr_ifru.ifru_ival = xBitRate;
ioctl(s1, SIOCSCANBAUDRATE, &ifr1);

printf("%s at index %d\n", ifname, ifr.ifr_ifindex);
printf("%s at index %d\n", ifname1, ifr1.ifr_ifindex);

if(bind(s, (struct sockaddr *)&addr, sizeof(addr)) < 0) {
    perror("Error in socket bind");
    return -2;
}
if(bind(s1, (struct sockaddr *)&addr1, sizeof(addr1)) < 0) {
    perror("Error in socket bind");
    return -2;
}

frame.can_id = 0x123;
frame.can_dlc = 2;
frame.data[0] = 0x11;
frame.data[1] = 0x22;

write(s, &frame, sizeof(struct can_frame));
printf("Wrote data[0]:%2x,data[1]:%2x\n",frame.data[0],frame.data[1]);
read(s1, &frame1, sizeof(struct can_frame));
printf("Read data[0]:%2x,data[1]:%2x\n",frame1.data[0],frame1.data[1]);

return 0;
}
```

4.4 Compile Demo Program

4.4.1 Install Yocto Toolchain

Before you develop and compile sample program, you should install Yocto toolchain into development PC. To do so, refer to Chapter 5 Board Support Package.

To compile and build demo program for rBOX630, please do:

Change to *your project* directory.

```
~$ cd ~/Project
```

Set up the cross-development environment.

```
~$ source /opt/poky/1.5.4/environment-setup-cortexa9hf-vfp-neon-poky-linux-gnueabi
```

Extract driver source to *your project* directory.

```
~$ tar jxf rBOX630-rb_lib-x.x.x.tar.bz2 -C ~/Project
```

Change to *rb_lib/demo* directory.

```
~$ cd ~/Project/rb_lib/demo
```

Build the demo program.

```
~$ make
```

Then you should have example programs such as `open_comport`, `wdt`, `cpld`, `diotest`, and `commode`.

```
jrtiger@test-H97M-D3H:~/rBOX630/tools/rb_lib/demo$ make
arm-poky-linux-gnueabi-gcc -pthread -o cpld cpld.c -lrb217 -L../
arm-poky-linux-gnueabi-gcc -pthread -o commode commode.c -lrb217 -L../
arm-poky-linux-gnueabi-gcc -pthread -o diotest diotest.c -lrb217 -L../
arm-poky-linux-gnueabi-gcc -pthread -o wdt wdt.c -lrb217 -L../
arm-poky-linux-gnueabi-gcc -pthread -o open_comport open_comport.c -lrb217 -L../
jrtiger@test-H97M-D3H:~/rBOX630/tools/rb_lib/demo$ ls
commode  cpld    diotest  librb217.h  open_comport  serial.h  wdt.c
commode.c  cpld.c  diotest.c  Makefile    open_comport.c  wdt
jrtiger@test-H97M-D3H:~/rBOX630/tools/rb_lib/demo$
```

4.4.2 Run demo program

Refer to section 2.3 for detailed information.

This page is intentionally left blank.

Then restart the TFTP server.
\$ sudo /etc/init.d/xinetd restart

5. Install and configure NFS server:
\$ sudo aptitude -y install nfs-common nfs-kernel-server portmap
To configure nfs server, add lines to `/etc/exports` as follows:
`/tools/rootfs *(rw, sync, no_root_squash)`
\$ sudo vi /etc/exports
Create a symbolic link to root filesystem which your build.
\$ sudo mkdir /tools
\$ sudo ln -s ~/Project/rootfs /tools/rootfs

Then restart the NFS server.
\$ sudo /etc/init.d/nfs-kernel-server restart

5.1.2 Install Yocto development

1. Setting up the repo utility.
Create a bin folder in the home directory.
~\$ mkdir ~/bin (this step may not be needed if the bin folder already exists)
~\$ curl http://commondatastorage.googleapis.com/git-repo-downloads/repo >
~/bin/repo
~\$ chmod a+x ~/bin/repo

Add the following line to the `.bashrc` file to ensure that the `~/bin` folder is in your `PATH` variable.

```
export PATH=~/bin:$PATH
```

2. Setting up the Git environment
~\$ git config --global user.name "Your Name"
~\$ git config --global user.email "Your Email"
3. Download the Freescale's Yocto BSP source
~\$ mkdir fsl-community-bsp
~\$ cd fsl-community-bsp
~\$ repo init -u https://github.com/Freescale/fsl-community-bsp-platform -b dora
~\$ repo sync
4. Extract Axiomtek's Yocto BSP source
~\$ tar zxvf meta-axiomtek-x.x.x.tar.gz -C fsl-community-bsp/sources
5. Update `bblayers.conf`
~\$ vi fsl-community-bsp/sources/base/conf/bblayers.conf
And add this line after `_${BSPDIR}/sources/meta-fsl-demos \`
`_${BSPDIR}/sources/meta-axiomtek \`
6. First build
Change to `fsl-community-bsp` directory
~\$ cd fsl-community-bsp
Create your local branch
~\$ repo start <new branch name> --all
Choose your board
~\$ MACHINE=rbox630 source setup-environment build
Start to build image
~\$ bitbake rbox-image-test

5.1.3 Build and Install Yocto toolchain

1. Build the toolchain for rBOX630 from Yocto development.
Change to *Yocto development* directory.

```
~$ cd fsl-community-bsp
~$ source setup-environment build
~$ bitbake meta-toolchain
```

After these steps to generate the toolchain into the Build Directory.

2. Install the toolchain into your host system /opt directory.

```
~$ ./tmp/deploy/sdk/poky-eglibc-x86_64-meta-toolchain-cortexa9hf-vfp-neon-toolchain-1.5.4.sh
```

```
jrtiger@test-H97M-D3H:~/fsl-community-bsp/build$ ./tmp/deploy/sdk/poky-eglibc-x86_64-meta-toolchain-cortexa9hf-vfp-neon-toolchain-1.5.4.sh
Enter target directory for SDK (default: /opt/poky/1.5.4):
You are about to install the SDK to "/opt/poky/1.5.4". Proceed[Y/n]?y
[sudo] password for jrtiger:
Extracting SDK...done
Setting it up...done
SDK has been successfully set up and is ready to be used.
jrtiger@test-H97M-D3H:~/fsl-community-bsp/build$
```

5.2 U-Boot for rBOX630

5.2.1 Booting the System with an NFS Filesystem

By default, U-Boot is configured to boot from SD CARD. To boot from NFS, first you must set some configurations. Press any key to break from the boot progress and set configurations.

```
ahb clock      : 132000000Hz
axi clock      : 198000000Hz
emi_slow clock: 99000000Hz
ddr_clock      : 396000000Hz
usdhc1 clock   : 198000000Hz
usdhc2 clock   : 198000000Hz
usdhc3 clock   : 198000000Hz
usdhc4 clock   : 198000000Hz
nfc clock      : 24000000Hz
Board: Axiomtek Q7M120 with i.MX6DL/Solo [WDOG]
Boot Device: SD
I2C:  ready
DRAM:  1 GB
MMC:   FSL_USDHC: 0,FSL_USDHC: 1,FSL_USDHC: 2,FSL_USDHC: 3
In:    serial
Out:   serial
Err:   serial
Net:   got MAC address from IIM: 00:00:00:00:00:00
FEC0 [PRIME]
Hit any key to stop autoboot: 0
MX6SDL Q7M120 U-Boot > setenv ethaddr 00:60:e0:00:00:0A
MX6SDL Q7M120 U-Boot > setenv serverip 192.168.1.101
MX6SDL Q7M120 U-Boot > setenv ipaddr 192.168.1.103
MX6SDL Q7M120 U-Boot > run bootcmd_net
```

Setup TFTP server IP:

```
MX6SDL Q7M120 U-Boot > setenv serverip 192.168.1.101
```

Setup board IP address:

```
MX6SDL Q7M120 U-Boot > setenv ipaddr 192.168.1.103
```

Run boot from NFS server:

```
MX6SDL Q7M120 U-Boot > run bootcmd_net
```

NOTE: If the MAC address has not burned into fuse, you must set the MAC address to use network in U-Boot.

```
MX6SDL Q7M120 U-Boot > setenv ethaddr xx:xx:xx:xx:xx:xx
```

5.2.2 Booting the System from eMMC

```
MX6SDL Q7M120 U-Boot > run bootcmd_emmc
```

```
Hit any key to stop autoboot: 0
MX6SDL Q7M120 U-Boot > run bootcmd_emmc
mmc3(part 0) is current device

MMC read: dev # 3, block # 2048, count 12288 ... 12288 blocks read: OK
## Booting kernel from Legacy Image at 10800000 ...
Image Name:   Linux-3.0.35-rBOX630-002
Image Type:   ARM Linux Kernel Image (uncompressed)
Data Size:   3496748 Bytes = 3.3 MB
Load Address: 10008000
Entry Point: 10008000
Verifying Checksum ... OK
Loading Kernel Image ... OK
OK

Starting kernel ...

Uncompressing Linux... done, booting the kernel.
Linux version 3.0.35-rBOX630-002 (jrtiger@test-H97M-D3H) (gcc version 4.8.1 (GC5
CPU: ARMv7 Processor [412fc09a] revision 10 (ARMv7), cr=10c53c7d
CPU: VIPT nonaliasing data cache, VIPT aliasing instruction cache
Machine: Axiomtek i.MX 6Quad/DualLite/Solo Q7M120 Board
Ignoring unrecognized tag 0x54410008
Memory policy: ECC disabled, Data cache writealloc
```

5.2.3 Booting the System from SD CARD (rBOX630 default)

```
MX6SDL Q7M120 U-Boot > run bootcmd_sd
```

```
FEC0 [PRIME]
Hit any key to stop autoboot: 0
mmc2 is current device

MMC read: dev # 2, block # 2048, count 12288 ... 12288 blocks read: OK
## Booting kernel from Legacy Image at 10800000 ...
Image Name:   Linux-3.0.35-rBOX630-003
Image Type:   ARM Linux Kernel Image (uncompressed)
Data Size:   3497080 Bytes = 3.3 MB
Load Address: 10008000
Entry Point: 10008000
Verifying Checksum ... OK
Loading Kernel Image ... OK
OK

Starting kernel ...

Uncompressing Linux... done, booting the kernel.
Linux version 3.0.35-rBOX630-003 (jrtiger@test-H97M-D3H) (gcc version 4.8.1 (GC5
CPU: ARMv7 Processor [412fc09a] revision 10 (ARMv7), cr=10c53c7d
CPU: VIPT nonaliasing data cache, VIPT aliasing instruction cache
Machine: Axiomtek i.MX 6Quad/DualLite/Solo Q7M120 Board
Ignoring unrecognized tag 0x54410008
Memory policy: ECC disabled, Data cache writealloc
```

5.3 Additional Information

5.3.1 Create a Bootable SD CARD for rBOX630

1. Copying the Boot Loader image.
`~$ sudo dd if=./u-boot.bin of=/dev/sdb bs=512 seek=2 skip=2 conv=fsync`
 The first 1KB of the SD card, that includes the partition table, will be preserved
2. Copying the Kernel image.
`~$ sudo dd if=./ulmage of=/dev/sdb bs=512 seek=2048 conv=fsync`
 This will copy ulmage to the media at offset 1MB

```
jrtiger@test-H97M-D3H:~$ sudo dd if=./u-boot.bin of=/dev/sde bs=512 seek=2 skip=
2 conv=fsync
839+1 records in
839+1 records out
429840 bytes (430 kB) copied, 0.225398 s, 1.9 MB/s
jrtiger@test-H97M-D3H:~$ sudo dd if=./uImage of=/dev/sde bs=512 seek=2048 conv=f
sync
6830+1 records in
6830+1 records out
3497144 bytes (3.5 MB) copied, 1.38351 s, 2.5 MB/s
jrtiger@test-H97M-D3H:~$ █
```

3. Copying the Root File System (rootfs).
 First, a partition table must be created.

To create a first partition FAT32 format for Windows PC easy access, after offset 16384 (in sectors of 512 bytes) , and Second partition Linux format for Root filesystem enter the following command:

```
~$ sudo fdisk /dev/sdb
```

Type the following parameters (each followed by <Enter>):

```
u      [switch the unit to sectors instead of cylinders]
d      [repeat this until no partition is reported by the 'p' command]
n      [create a new partition]
p      [create a primary partition]
1      [the first partition]
16384  [starting at offset sector #16384, i.e. 8MB, which leaves enough space for
      the kernel, the boot loader and its configuration data]
+1G    [create 1G size for FAT32]
t      [change a partition's system id]
b      [choice W95 FAT32 type]
n      [create a new partition]
p      [create a primary partition]
2      [the second partition]
2113536 [starting after offset sector #2113536]
<enter> [using the default value will create a partition that spans to the last sector
      of the medium]
w      [this writes the partition table to the medium and fdisk exits]
```

```

jrtiger@test-H97M-D3H:~$ sudo fdisk /dev/sde

Command (m for help): n
Partition type:
   p   primary (0 primary, 0 extended, 4 free)
   e   extended
Select (default p): p
Partition number (1-4, default 1): 1
First sector (2048-15523839, default 2048): 16384
Last sector, +sectors or +size{K,M,G} (16384-15523839, default 15523839): +1G

Command (m for help): t
Selected partition 1
Hex code (type L to list codes): 0b
Changed system type of partition 1 to b (W95 FAT32)

Command (m for help): p

Disk /dev/sde: 7948 MB, 7948206080 bytes
245 heads, 62 sectors/track, 1021 cylinders, total 15523840 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x911b6ea2

   Device Boot      Start         End      Blocks   Id  System
/dev/sde1            16384       2113535     1048576    b   W95 FAT32

Command (m for help): █

```

```

Command (m for help): p

Disk /dev/sde: 7948 MB, 7948206080 bytes
245 heads, 62 sectors/track, 1021 cylinders, total 15523840 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x911b6ea2

   Device Boot      Start         End      Blocks   Id  System
/dev/sde1            16384       2113535     1048576    b   W95 FAT32

Command (m for help): n
Partition type:
   p   primary (1 primary, 0 extended, 3 free)
   e   extended
Select (default p): p
Partition number (1-4, default 2): 2
First sector (2048-15523839, default 2048): 2113536
Last sector, +sectors or +size{K,M,G} (2113536-15523839, default 15523839):
Using default value 15523839

Command (m for help): p

Disk /dev/sde: 7948 MB, 7948206080 bytes
245 heads, 62 sectors/track, 1021 cylinders, total 15523840 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x911b6ea2

   Device Boot      Start         End      Blocks   Id  System
/dev/sde1            16384       2113535     1048576    b   W95 FAT32
/dev/sde2            2113536     15523839     6705152    83   Linux

Command (m for help): █

```

Run the following command to format the partition

```
~$ sudo mkfs.vfat /dev/sdb1  
~$ sudo mkfs.ext3 /dev/sdb2
```

```
jrtiger@test-H97M-D3H:~$ sudo mkfs.vfat /dev/sde1  
mkfs.fat 3.0.26 (2014-03-07)  
jrtiger@test-H97M-D3H:~$ sudo mkfs.ext3 /dev/sde2  
mke2fs 1.42.9 (4-Feb-2014)  
Filesystem label=  
OS type: Linux  
Block size=4096 (log=2)  
Fragment size=4096 (log=2)  
Stride=0 blocks, Stripe width=0 blocks  
419328 inodes, 1676288 blocks  
83814 blocks (5.00%) reserved for the super user  
First data block=0  
Maximum filesystem blocks=1719664640  
52 block groups  
32768 blocks per group, 32768 fragments per group  
8064 inodes per group  
Superblock backups stored on blocks:  
    32768, 98304, 163840, 229376, 294912, 819200, 884736, 1605632  
  
Allocating group tables: done  
Writing inode tables: done  
Creating journal (32768 blocks): done  
Writing superblocks and filesystem accounting information: done
```

Copy the target file system to the partition

```
~$ mkdir /home/user/mountpoint  
~$ sudo mount /dev/sdb2 /home/user/mountpoint
```

Extract rootfs package to certain directory

```
~$ sudo tar xzf rootfs.tar.gz -C /home/user/mountpoint
```

```
jrtiger@test-H97M-D3H:~$ sudo mount -v /dev/sde2 /mnt/sdcard  
mount: you didn't specify a filesystem type for /dev/sde2  
    I will try type ext3  
/dev/sde2 on /mnt/sdcard type ext3 (rw)  
jrtiger@test-H97M-D3H:~$ sudo tar xzf rbox-image-test-rbox630.tar.gz -C /mnt/sdcard/  
jrtiger@test-H97M-D3H:~$
```

NOTE: Make sure that the device node is correct for the SD CARD. i.e. sdb or sdc