AXIOMTEK

rBOX630-FL

Linux

Software User's Manual

RoHS

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.

Aug. 2022, Version A4

Printed in Taiwan

Table of Contents

Disclaimers 2

Chapter 1 Introduction 5

1.1 Specifications 6

Chapter 2 Getting Started 9

2.1 Connecting the rBOX630 9

2.1.1 Serial Console 10

2.1.2 SSH over Ethernet 12

2.2 How to Develop a Sample Program 14

2.2.1 Install Yocto Toolchain 14

2.2.2 Setting Up the Cross-Development Environment 15

2.2.3 Write and Compile Sample Program 15

2.3 How to Put and Run a Sample Program 17

2.3.1 Via FTP (Default disable) 17

2.3.2 Via USB Flash Drive 18

2.4 How to use MFG tool to download image 19

Chapter 3 The Embedded Linux 21

3.1 Embedded Linux Image Managing 21

3.1.1 System Version 21

3.1.2 System Time 21

3.1.3 Internal RTC Time 21

3.1.4 External RTC Time 22

3.1.5 Adjusting System Time 22

3.2 Networking 22

3.2.1 FTP – File Transfer Protocol 22

3.2.2 TFTP – Trivial File Transfer Protocol 22

Chapter 4 Programming Guide 23

4.1 EApi API Functions 23

4.1.1 EApiGPIOGetLevel 24

4.1.2 EApiGPIOSetLevel 25

4.1.3 EApiWDogStart 25

4.1.4 EApiWDogTrigger 26

4.1.5 EApiWDogStop 26

4.1.6 EApiCanGetTermination 26

4.1.7 EApiCanSetTermination 26

4.1.8 EApiComGetType 27

4.1.9 EApiComSetType 27

4.1.10 EApiComGetTermination 28

4.1.11 EApiComSetTermination 28

4.1.12 EApiHWMGetCaps 28
4.2 Compile Demo Program 29
4.2.1 Install Yocto Toolchain ..... 錯誤! 尚未定義書籤。
4.2.2 Run demo program 29
4.3 CAN Bus 30
Chapter 5 Board Support Package (BSP) 32
5.1 Host Development System Installation 32
5.1.1 Install Host System 32
5.1.2 Install Yocto development 33
5.1.3 Build and Install Yocto toolchain 34
5.2 U-Boot for rBOX630 35
5.2.1 Booting the System from eMMC (rBOX630 default) 35
5.2.2 Booting the System from SD CARD 35

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 5.10.72 and our hardware patches to suit rBOX630.

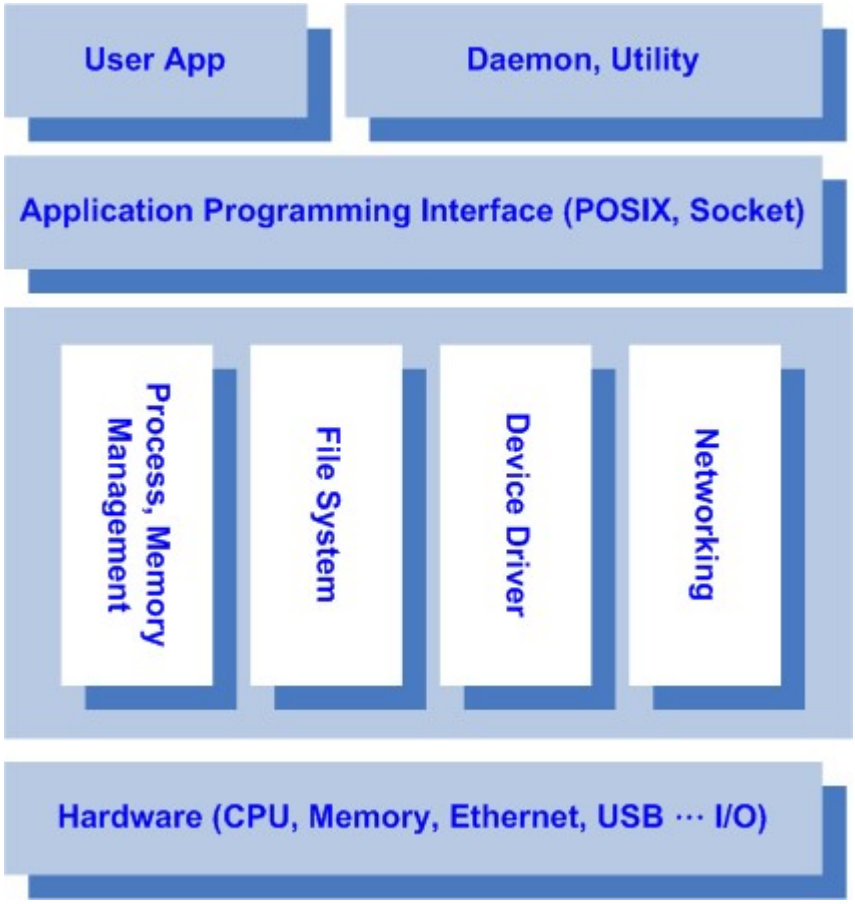
Software structure

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 'libEApi.so' shared library.

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, 4G.

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: 5.10.72 (with Freescale and Axiomtek hardware modified patch)
- **Shell**
  - Bash
- **File system**
  - 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

- **Development Environment**
  - Host OS/ development OS: The recommended minimum Ubuntu version is 18.04 or later
  - Toolchain/ cross compiler: toolchain-x.x.x-hardknott (Yocto project 3.3 Hardknott)

- **HW's Lib (Hardware's Library)**
  - **Digital I/O**
    - Read digital input
    - Write digital output
  - **CAN**
    - Termination setting
  - **Watch Dog Timer**
    - Enable
    - Clean
    - Set timer
  - **COM**
    - RS-232/422/485 mode setting
    - Termination setting

Note

All specifications and images are subject to change without notice.

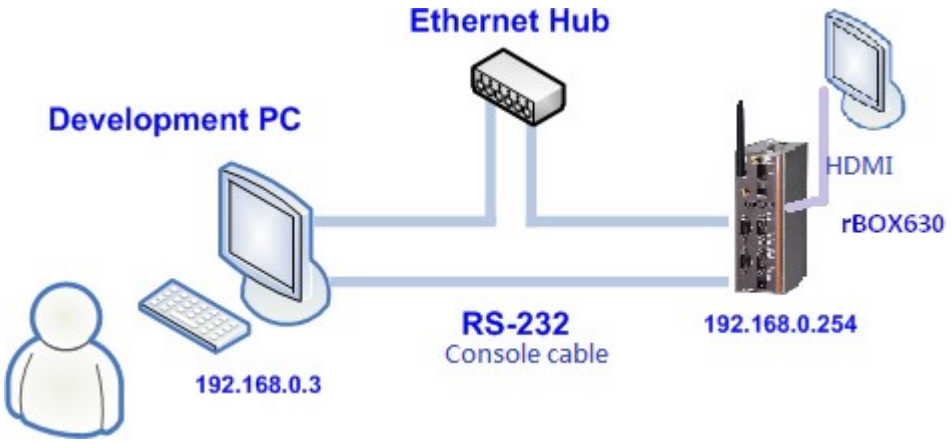
This page is intentionally left blank.

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

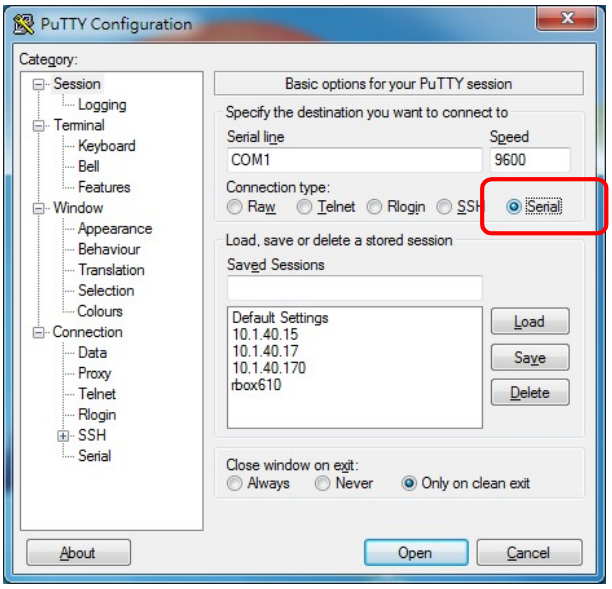
2.1.1 Serial Console

The serial console is a convenient interface for connecting rBOX630 to PC. First of all, it is very important to make sure that your desktop connects to rBOX630 by console cable. Please set the system as follows:

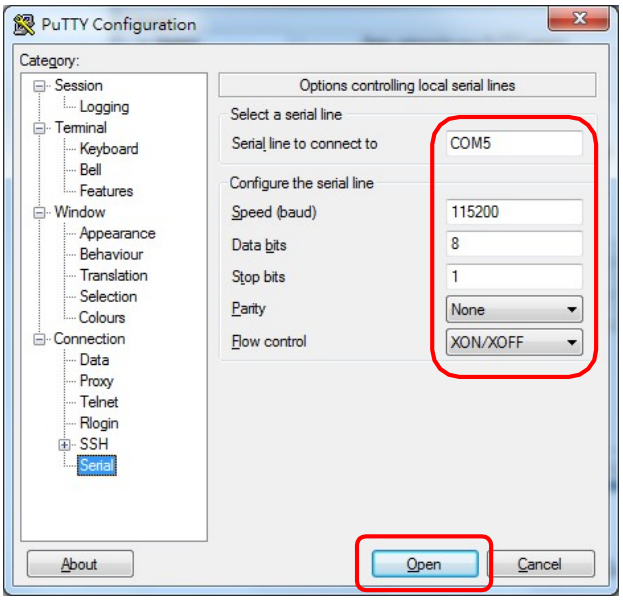
Baudrate: 115200 bps
Parity: None
Data bits: 8
Stop bit: 1
Flow Control: None

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

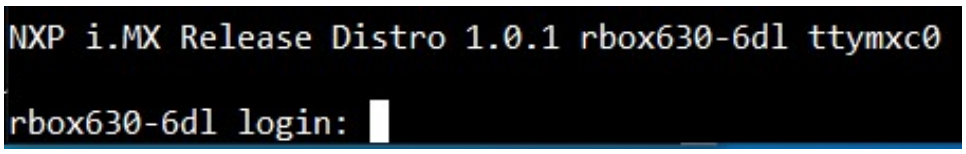
1. Open PuTTY and choose 'Serial' as the connection type.



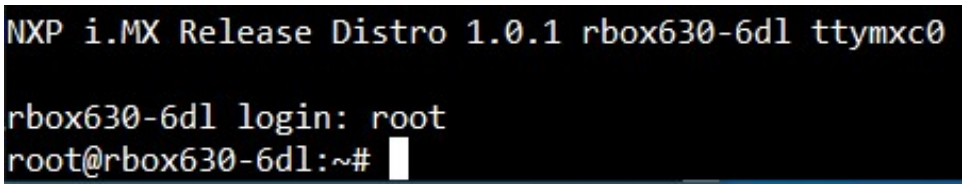
2. Configure the serial port correctly (see image below). Click Open and power on the rBOX630.



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



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

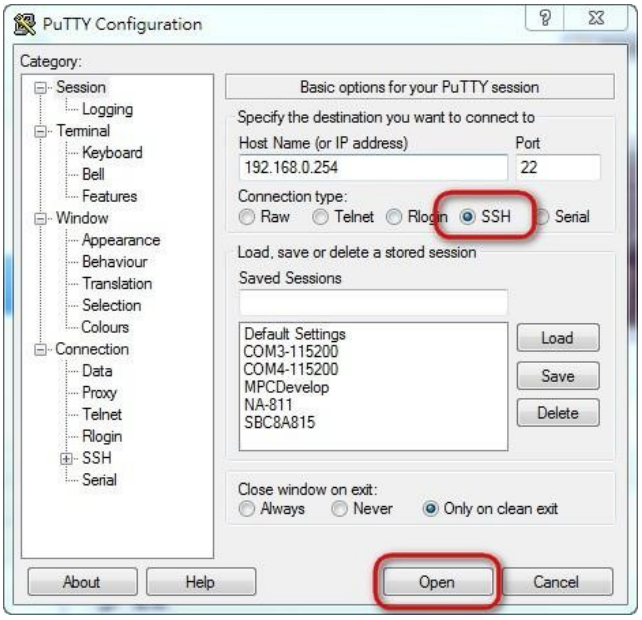


2.1.2 SSH over Ethernet

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.

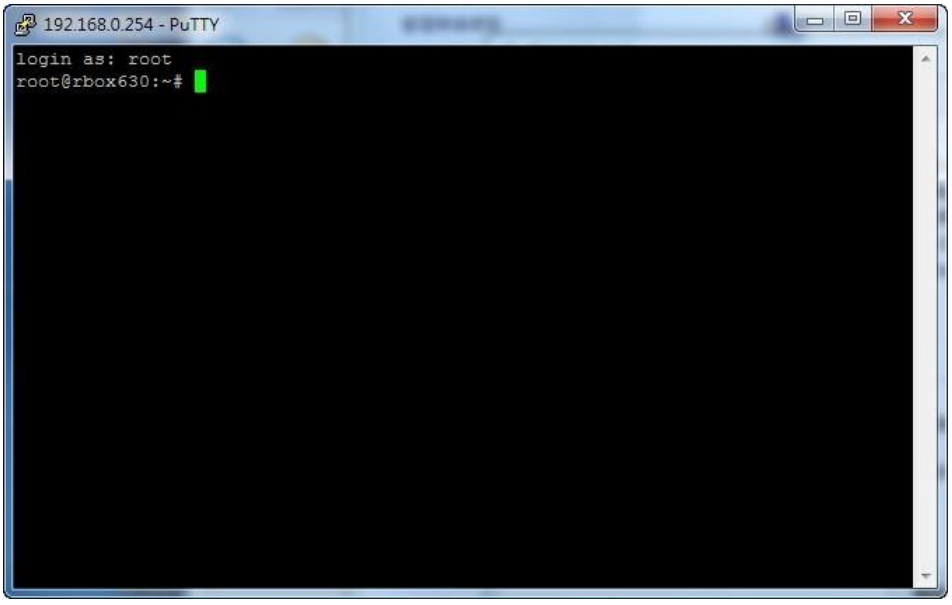
For Windows® users:

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



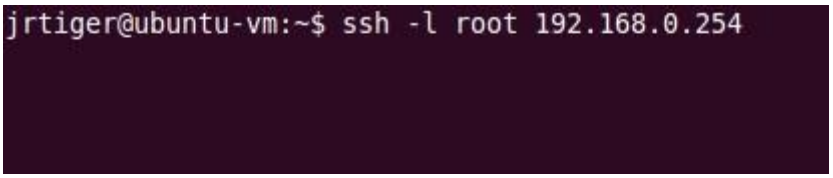
2. If connection is established successfully, you should see the following image.

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



For Linux users:

1. Open terminal and keyin 'ssh' command.



2. After the connection is established successfully.

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. You can follow below step to install Yocto toolchain or refer to Chapter 5 Board Support Package to build the toolchain for rBOX630.

1. Host Ubuntu version.
   Ubuntu 18.04 or later:

```
axiomtek@axiomtek-PC:~$ uname -a
Linux axiomtek-PC 5.4.0-42-generic #46~18.04.1-Ubuntu SMP Fri Jul 10 07
:21:24 UTC 2020 x86_64 x86_64 x86_64 GNU/Linux
```

2. Copy the toolchain script to home directory.

```
axiomtek@axiomtek-PC:~$ ls fsl*
fsl-imx-xwayland-glibc-x86_64-imx-image-full-cortexa9t2hf-neon-rbox630-
6dl-toolchain-1.0.1.sh
```

3 Execute the toolchain script and press Enter to install to default directory.

```
$ ./fsl-imx-xwayland-glibc-x86_64-imx-image-full-cortexa9t2hf-neon-rbox630-6dl-
toolchain-x.x.x.sh
```

```
axiomtek@axiomtek-PC:~$ ./fsl-imx-xwayland-glibc-x86_64-imx-image-full-
cortexa9t2hf-neon-rbox630-6dl-toolchain-1.0.1.sh
NXP i.MX Release Distro SDK installer version 1.0.1
=================================================
Enter target directory for SDK (default: /opt/fsl-imx-xwayland/1.0.1):
```

4 Check the directory.

```
axiomtek@axiomtek-PC:~$ ./fsl-imx-xwayland-glibc-x86_64-imx-image-full-
cortexa9t2hf-neon-rbox630-6dl-toolchain-1.0.1.sh
NXP i.MX Release Distro SDK installer version 1.0.1
=================================================
Enter target directory for SDK (default: /opt/fsl-imx-xwayland/1.0.1):
You are about to install the SDK to "/opt/fsl-imx-xwayland/1.0.1". Proc
eed [Y/n]? Y
```

5 Press your password and wait to installation.

```
axiomtek@axiomtek-PC:~$ ./fsl-imx-xwayland-glibc-x86_64-imx-image-full-
cortexa9t2hf-neon-rbox630-6dl-toolchain-1.0.1.sh
NXP i.MX Release Distro SDK installer version 1.0.1
=================================================
Enter target directory for SDK (default: /opt/fsl-imx-xwayland/1.0.1):
You are about to install the SDK to "/opt/fsl-imx-xwayland/1.0.1". Proc
eed [Y/n]? Y
[sudo] password for axiomtek:
Extracting SDK.....
```

6 Install finish.

```
axiomtek@axiomtek-PC:~$ ./fsl-imx-xwayland-glibc-x86_64-imx-image-full-
cortexa9t2hf-neon-rbox630-6dl-toolchain-1.0.1.sh
NXP i.MX Release Distro SDK installer version 1.0.1
=================================================
Enter target directory for SDK (default: /opt/fsl-imx-xwayland/1.0.1):
You are about to install the SDK to "/opt/fsl-imx-xwayland/1.0.1". Proc
eed [Y/n]? Y
[sudo] password for axiomtek:
Extracting SDK..................................................
................................................................
................................................................
................................................................
................................................................
................................................................
................................................................
................................................................
.................done
Setting it up...done
SDK has been successfully set up and is ready to be used.
Each time you wish to use the SDK in a new shell session, you need to s
ource the environment setup script e.g.
 $ . /opt/fsl-imx-xwayland/1.0.1/environment-setup-cortexa9t2hf-neon-po
ky-linux-gnueabi
```

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.

```
$ . /opt/fsl-imx-xwayland/x.x.x/environment-setup-cortexa9t2hf-neon-poky-linux-gnueabi
axiomtek@axiomtek-PC:~$
axiomtek@axiomtek-PC:~$ . /opt/fsl-imx-xwayland/1.0.1/environment-setup
-cortexa9t2hf-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

```
axiomtek@axiomtek-PC:~$ arm-poky-linux-gnueabi-gcc hello.c -o hello
```

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

~$ ls -l

```
jrtiger@test-H97M-D3H:~/example$ 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:~/example$
```

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 /etc/xinetd.d/ftpd to enable

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

   Restart Internet server
   # systemctl restart xinetd

```
root@rbox630-6dl:/etc/xinetd.d# systemctl restart xinetd
```

2. On your host PC
   $ ftp [ip] (username: root)

```
axiomtek@axiomtek-PC:~$ ftp 10.1.30.91
Connected to 10.1.30.91.
220 Operation successful
Name (10.1.30.91:axiomtek): root
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 /run/media/sda1/hello /home/root

4. # cd /home/root

5. # chmod +x hello

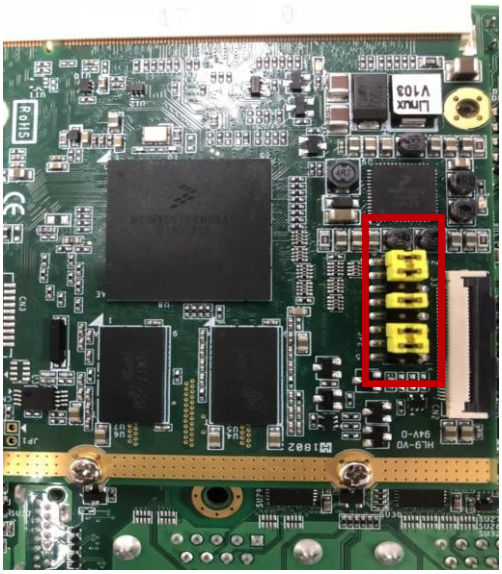
6. # ./hello

```
root@rbox630-6dl:~# cp /run/media/sda1/hello /home/root/
root@rbox630-6dl:~# cd /home/root
root@rbox630-6dl:~# chmod +x hello
root@rbox630-6dl:~# ./hello
hello world
```

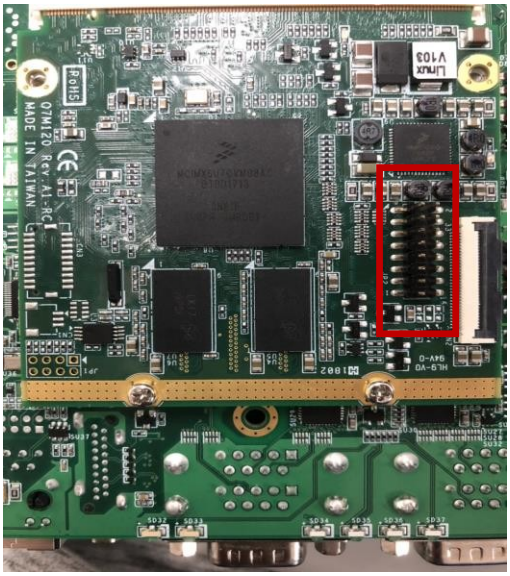
2.4 How to use MFG tool to download image

In this section, we show how to use MFG tool to download image to the rBOX630 System.

1. Before using the mfgtool, you have to change the rBOX630 CPU board JP2 boot mode (default emmc boot) to OTG serial downloader mode. Connect the rBOX630 and PC with a USB cable.

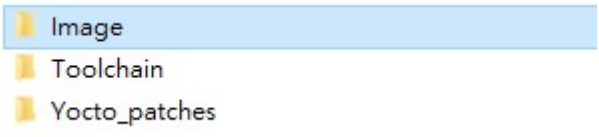


Emmc boot mode



Serial downloader mode

2. Extract Axiomtek's Yocto BSP and you will see Image in the rBOX630_L510_vx.x.x directory

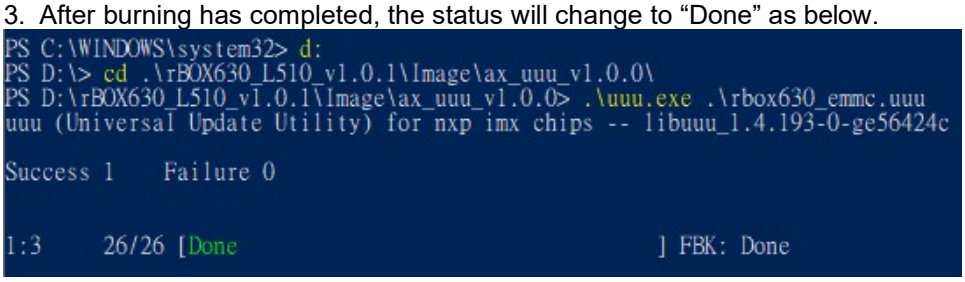


For Windows® users:

1. Open Windows PowerShell or CMD and switch to tool path

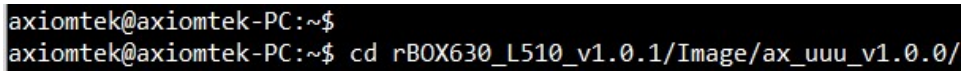
2. Run flash command: > uuu.exe rbox630_emmc.uuu

3. After burning has completed, the status will change to "Done" as below.



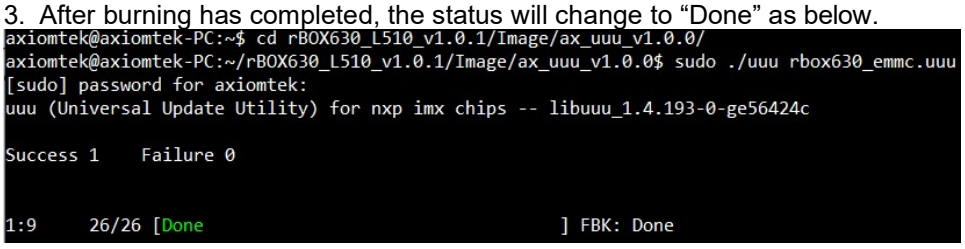
For Linux users:

1. Open Windows PowerShell or CMD and switch to tool path



2. Run flash command: $ sudo ./uuu rbox630_emmc.uuu

3. After burning has completed, the status will change to "Done" as below.



3. Change rBOX630 CPU board JP2 boot mode as emmc mode, success.

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@rbox630-6dl:~# uname -r
5.10.72-lts-5.10.y+gb5139936405c
```

Check root filesystem with the login screen:

```
NXP i.MX Release Distro 1.0.1 rbox630-6dl ttymxc0

rbox630-6dl 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 2.3 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.

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 EApi API Functions

The rBOX630 BSP includes 'libEApi.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 /usr/lib and the API header file is in /usr/include.

Summary table of available API functions

No.	Function	Description
1	EApiGPIOGetLevel()	Read high or low state on digital input/ output channels.
2	EApiGPIOSetLevel()	Write high or low state on digital input/ output channels.
3	EApiWDogStart()	Enable watchdog timer
4	EApiWDogTrigger()	Reset WDT counter.
5	EApiWDogStop()	Disable watchdog timer
6	EApiCanGetTermination()	Get termination of specified CAN port.
7	EApiCanSetTermination()	Set termination of specified CAN port.
8	EApiComGetType()	Get COM port communication mode type.
9	EApiComSetType()	Set COM port communication mode type.
10	EApiComGetTermination()	Get termination of specified COM port.
11	EApiComSetTermination()	Set termination of specified COM port.
12	EApiHWMGetCaps()	Get Hardware infomation

4.1.1 EApiGPIOGetLevel

```
EApiGPIOGetLevel(
    __IN  EApiId_t Id
    __IN  uint32_t Bitmask
    __OUT uint32_t* pLevel
    )
```

Description

Read high or low state on digital input/ output channels

In/Out	Parameter Name	Description
__IN	id	AX_EAPI_DI_0 AX_EAPI_DI_1 AX_EAPI_DI_2 AX_EAPI_DI_3 AX_EAPI_DI_4 AX_EAPI_DI_5 AX_EAPI_DI_6 AX_EAPI_DI_7 AX_EAPI_DO_0 AX_EAPI_DO_1 AX_EAPI_DO_2 AX_EAPI_DO_3 AX_EAPI_DO_4 AX_EAPI_DO_5 AX_EAPI_DO_6 AX_EAPI_DO_7 AX_EAPI_DI AX_EAPI_DO
__IN	Bitmask	currently not supported by EApi, set it as 0
__OUT	pLevel	AX_EAPI_DI_* AX_EAPI_DO_* 0 : low 1: high AX_EAPI_DI AX_EAPI_DO 0~255 all DI/DO

4.1.2 EApiGPIOSetLevel

```
EApiGPIOSetLevel(
    __IN  EApiId_t Id
    __IN  uint32_t Bitmask
    __IN uint32_t Level
    )
```

Description

Write high or low state on digital input/ output channels.

In/Out	Parameter Name	Description
__IN	id	AX_EAPI_DO_0 AX_EAPI_DO_1 AX_EAPI_DO_2 AX_EAPI_DO_3 AX_EAPI_DO_4 AX_EAPI_DO_5 AX_EAPI_DO_6 AX_EAPI_DO_7 AX_EAPI_DO
__IN	Bitmask	currently not supported by EApi, set it as 0
__IN	Level	AX_EAPI_DO_* 0 : low 1: high AX_EAPI_DO 0~255 set all DO high/low

4.1.3 EApiWDogStart

```
EApiWDogStart(
    __IN  uint32 Delay
    __IN  uint32_t EventTimeout
    __IN uint32_t ResetTimeout
    )
```

Description

Start the watchdog timer and set the parameters.

In/Out	Parameter Name	Description
__IN	Delay	currently not supported by EApi, set it as 0
__IN	EventTimeout	currently not supported by EApi, set it as 0

__IN	ResetTimeout	Watchdog timeout interval in milliseconds to trigger a reset.

4.1.4 EApiWDogTrigger

```
EApiWDogTrigger(void)
```

Description

Trigger the watchdog timer

4.1.5 EApiWDogStop

```
EApiWDogStop(void)
```

Description

Stop the operation of the watchdog timer.

4.1.6 EApiCanGetTermination

```
EApiCanGetTermination(
    __IN  EApiId_t Id
    __OUT  uint32_t* CanEnable
    )
```

Description

Get termination of specified CAN port.

In/Out	Parameter Name	Description
__IN	id	AX_EAPI_CAN_0_TERMINATION AX_EAPI_CAN_1_TERMINATION
__OUT	CanEnable	0: disable 1: enable

4.1.7 EApiCanSetTermination

```
EApiCanSetTermination(
    __IN  ax_eapi_arm_id_t Id,
    __IN  uint32_t CanEnable
    )
```

Description

Set termination of specified CAN port.

In/Out	Parameter Name	Description
__IN	id	AX_EAPI_CAN_0_TERMINATION AX_EAPI_CAN_1_TERMINATION
__IN	CanEnable	0: disable 1: enable

4.1.8 EApiComGetType

```
EApiComGetType(
    __IN  EApiId_t Id
    __OUT  uint32_t* type
    )
```

Description

Get COM port communication mode type.

In/Out	Parameter Name	Description
__IN	id	AX_EAPI_COM_1 AX_EAPI_COM_2 AX_EAPI_COM_3 AX_EAPI_COM_4
__OUT	type	0: RSVD 1: RS232 Enable 2: RS422 /RS485 4W Enable 3: RS485 2W Enable

4.1.9 EApiComSetType

```
EApiComSetType(
    __IN  ax_eapi_arm_id_t Id,
    __IN  uint32_t type
    )
```

Description

Set COM port communication mode type.

In/Out	Parameter Name	Description
__IN	id	AX_EAPI_COM_1 AX_EAPI_COM_2 AX_EAPI_COM_3 AX_EAPI_COM_4
__IN	type	0: RSVD 1: RS232 Enable 2: RS422 /RS485 4W Enable 3: RS485 2W Enable

4.1.10 EApiComGetTermination

```
EApiComGetTermination(
    __IN  EApiId_t Id
    __OUT  uint32_t* enable
    )
```

Description

Get termination of specified COM port.

In/Out	Parameter Name	Description
__IN	id	AX_EAPI_COM_1_TERMINATION AX_EAPI_COM_2_TERMINATION AX_EAPI_COM_3_TERMINATION AX_EAPI_COM_4_TERMINATION
__OUT	enable	0: disable 1: enable

4.1.11 EApiComSetTermination

```
EApiComSetTermination(
    __IN  ax_eapi_arm_id_t Id,
    __IN  uint32_t enable
    )
```

Description

Set termination of specified COM port.

In/Out	Parameter Name	Description
__IN	id	AX_EAPI_COM_1_TERMINATION AX_EAPI_COM_2_TERMINATION AX_EAPI_COM_3_TERMINATION AX_EAPI_COM_4_TERMINATION
__IN	enable	0: disable 1: enable

4.1.12 EApiHWMGetCaps

```
EApiHWMGetCaps(
    __IN  EApiId_t Id
    __OUT  uint32_t* pValue
    )
```

Description

Get Hardware information.

In/Out	Parameter Name	Description
__IN	id	AX_EAPI_POWER_DC_1 AX_EAPI_POWER_DC_2
__OUT	pValue	0: disable 1: enable

4.2 Compile Demo Program

4.2.1 Build demo program

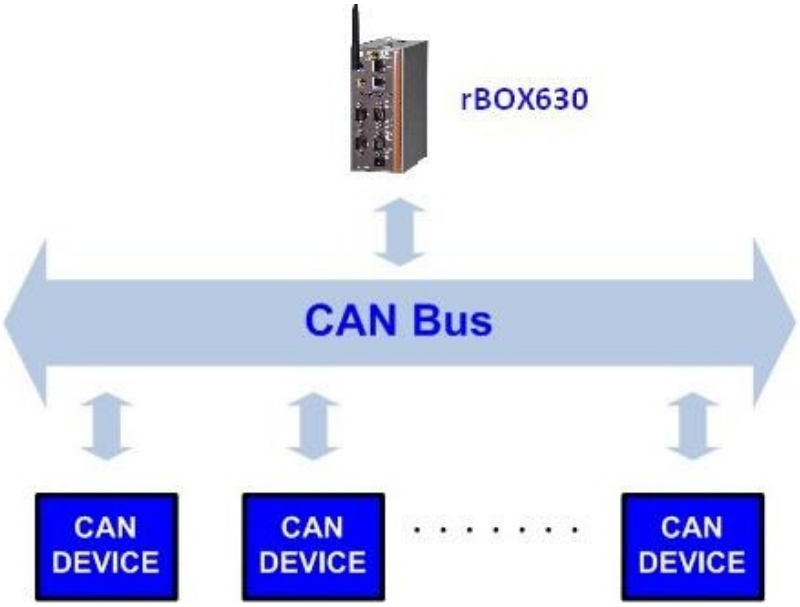
To compile and build demo program for rBOX630, please do:
Change to *demo* directory.
cd /usr/src/eapi_demo
Build the demo program.
$ make

4.2.2 Run demo program

cd /usr/src/eapi_demo
#./testeapi

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_ivalue = xBitRate;
          ioctl(s, SIOCSCANBAUDRATE, &ifr);

          addr1.can_family = AF_CAN;
          addr1.can_ifindex = ifr1.ifr_ifindex;
          ifr.ifr_ifru.ifru_ivalue = 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;
}
```

Chapter 5
Board Support Package (BSP)

5.1 Host Development System Installation

5.1.1 Install Host System

1. Download Ubuntu 18.04 or later LTS iso image.

2. Install Ubuntu 18.04 or later.

3. Install host packages needed by Yocto development as follows:
```
$ sudo apt-get install gawk wget git-core diffstat unzip texinfo gcc-multilib \
build-essential chrpath socat cpio python python3 python3-pip python3-pexpect \
xz-utils debianutils iputils-ping python3-git python3-jinja2 libegl1-mesa libsdl1.2-dev \
pylint3 xterm rsync curl
```

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 https://storage.googleapis.com/git-repo-downloads/repo > ~/bin/repo
   $ chmod a+x ~/bin/repo

   Add the following line to the .bashrcfile to ensure that the ~/binfolder 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 imx-yocto-bsp
   $ cd imx-yocto-bsp
   $ repo init -u https://source.codeaurora.org/external/imx/imx-manifest -b imx-linux-hardknott -m imx-5.10.72-2.2.0.xml
   $ repo sync

4. Extract Axiomtek's Yocto BSP source
   $ unzip rBOX630_L510_v1.0.3.zip
   $ cp rBOX630_L510_v1.0.3/Yocto_patches/rbox630_yocto_v1.0.3/meta-axiomtek imx-yocto-bsp/sources

5. Update bblayers.conf

   $ vi imx-yocto-bsp/sources/base/conf/bblayers.conf

   And add this line after ${BSPDIR}/sources meta-freescale-distro \

   ${BSPDIR}/sources/meta-axiomtek \

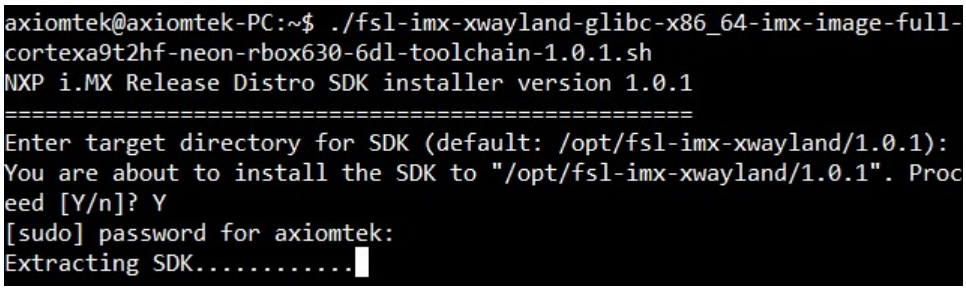
6. First build
   Change to fsl-community-bsp directory
   $ cd imx-yocto-bsp
   Choose your board
   $ DISTRO=imx-xwayland-rbox630-6dl MACHINE=rbox630-6dl source imx-setup-release.sh -b build
   Start to build image
   ~$ bitbake imx-image-full

5.1.3 Build and Install Yocto toolchain

1. Build the toolchain for rBOX630 from Yocto development.
   Change to *Yocto development* directory.
   ~$ cd imx-yocto-bsp
   ~$ source setup-environment build
   ~$ bitbake imx-image-full -c populate_sdk

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

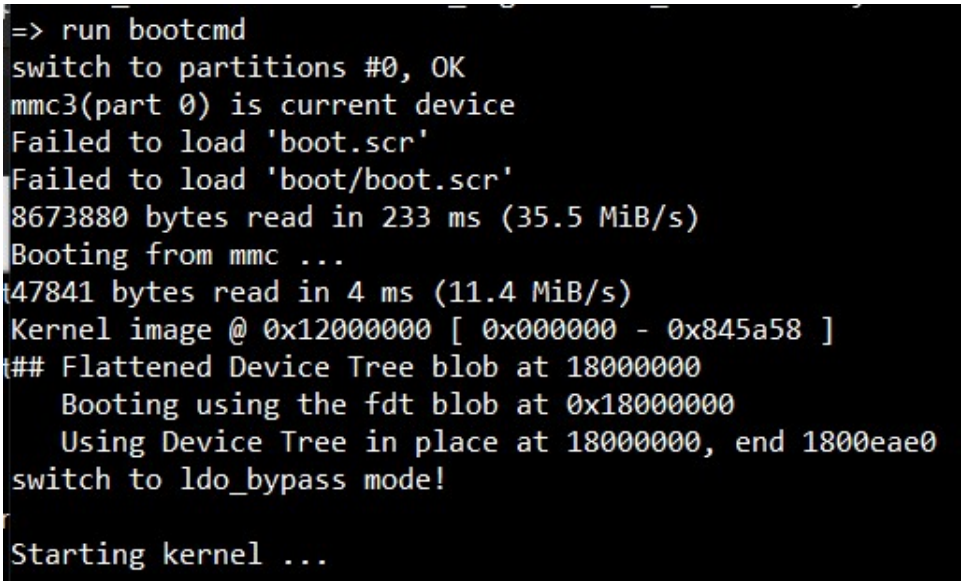
2. Install the toolchain into your host system /opt directory.
   ~$ ./tmp/deploy/sdk/fsl-imx-xwayland-glibc-x86_64-imx-image-full-cortexa9t2hf-neon-rbox630-6dl-toolchain-1.0.1.sh



5.2 U-Boot for rBOX630

5.2.1 Booting the System from eMMC (rBOX630 default)

=> run bootcmd



5.2.2 Booting the System from SD CARD

=> run bootcmd_sd