IFB112

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.

July 2017, 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 IFB112 5

2.1.1 Serial Console 7
2.1.2 SSH over Ethernet 9

2.2 How to Develop a Sample Program 11

2.2.1 Install Yocto Toolchain 11
2.2.2 Setting up the Cross-Development Environment 13
2.2.3 Write and Compile Sample Program 13

2.3 How to Put and Run a Sample Program 14

2.3.1 Via FTP 14
2.3.2 Via USB Flash Drive 16
2.3.3 Via TFTP 17

2.4 How to Recovery System 18

2.4.1 Via run_rescue System Script (under Linux System) 18
2.4.2 Via rescue.scr Script (under u-boot) 18

2.5 How to Update System 19

2.5.1 Via USB Flash Drive 19

2.6 How to use MFG tool to download image 22

Chapter 3 The Embedded Linux 25

3.1 Embedded Linux Image Managing 25

3.1.1 System Version 25
3.1.2 System Time 25
3.1.3 Internal RTC Time 25
3.1.4 External RTC Time 26
3.1.5 Watchdog timer 26
3.1.6 Adjusting System Time 26
3.1.7 LEDs Control 27
3.1.8 CANbus Control 27

3.2 Networking 28

3.2.1 FTP – File Transfer Protocol 28
3.2.2 TFTP – Trivial File Transfer Protocol 28
3.2.3 NFS – Network File System 28
3.2.4 How to use 3G or 4G module (Optional) 29

3.2.5 How to use Wi-Fi module (Optional) 33

Chapter 4 Programming Guide 35

4.1 librsb10x API Functions 35

4.2 Compile Demo Program 47

4.2.1 Install IFB112 I/O Library 47

4.2.2 Run demo program 48

Chapter 5 Board Support Package (BSP) 49

5.1 Host Development System Installation 49

5.1.1 Install Host System 49

5.1.2 Install Yocto Development 50

5.2 U-Boot for IFB112 54

5.2.1 Booting the System with an NFS Filesystem 54

5.2.2 Booting the System from eMMC (IFB112 default) 55

5.2.3 Booting the Rescue System from eMMC 55

Appendix Frequently Asked Questions 57

Chapter 1 Introduction

The extreme compact IFB112 supports the low power RISC-based module (i.MX6UL) 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, Its compact size with Din-rail mounting allows for easy installation into control

This user's manual is for the embedded Linux preinstalled in IFB112. The embedded Linux is derived from Linux Yocto Board Support Package, which is based on Linux Kernel 3.14.52 and our hardware patches to suit IFB112.

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).

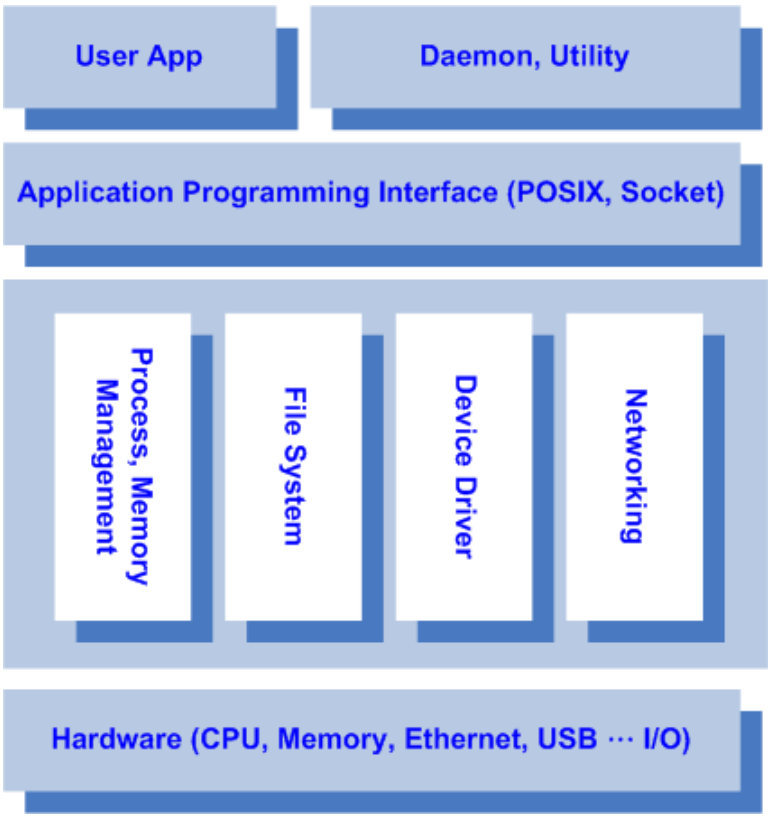
To facilitate user program in monitoring and controlling I/O device such as DIO, Watchdog Timer, CAN, COM, the IFB112 system includes 'librsb10x.so' shared library.

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: 3.14.52 (with NXP and Axiomtek hardware modified patch)
- **Support Protocol Types**
  - ICMP.
  - TCP/IP.
  - UDP, DHCP, Telnet, HTTP, HTTPS, SSL, SMTP, NTP, DNS, PPP, PPPoE, FTP, TFTP, NFS.
- **Shell**
  - Bash
- **Support storage format**
  - FAT32 /FAT/EXT2/EXT3/EXT4
- **BSP: IFB112-LINUX-bsp**
  - AxTools
  - Image
  - Yocto patches
  - Toolchain

- **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(1.23.1)**: 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 32/64bit
    kernel: version: 4.2.0-42
  - machine running Ubuntu, the minimum hard disk space required is about 50 GB for the X11 backend. It is recommended that at least 120 GB is provided, which is enough to compile all backends together.
  - Toolchain/ cross compiler: ARM, gcc-4.9.2 (Yocto project 1.8.1 Fido)

- **HW's Lib (Hardware's Library)**
  - **Digital I/O**
    - Read digital input
    - Write digital output
  - **COM**
    - RS-232/422/485 mode setting(Default RS232)
  - **CAN**
    - Support open/write/read/close functions
  - **Watch Dog Timer**
    - Enable Watch Dog Timer
    - Set Timer
  - **WiFi (Optional)**
    - Use Wi-Fi module WPER-172GN
  - **3G (Optional)**
    - Use 3G module Quectel UC20
  - **4G (Optional**
    - Use 4G module Sierra MC7304

- **Relay**
  Set relay high or low.

Note

1. ***All specifications and images are subject to change without notice..***
   http://www.axiomtek.com/Default.aspx?MenuId=Products&FunctionId=ProductView&ItemId=17865&upcat=134

   .

2. ***Command definition:***

Command	Definition	Example
=>	**U-Boot**	Ex: **=>** setenv ipaddr 192.168.1.103 Meaning: U-Boot setenv ipaddr 192.168.1.103
~$	**Host PC**	Ex: **~$** sudo apt-get install subversion Meaning: To command sudo apt-get install subverhsion on host PC
~#	**Target (IFB112):**	Ex: ~# /etc/run_rescue Meaning: To command /etc/run_rescue on IFB112

Chapter 2
Getting Started

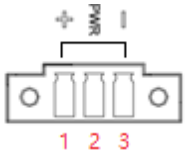
2.1 Connecting the IFB112

The power

Please check you power as below:

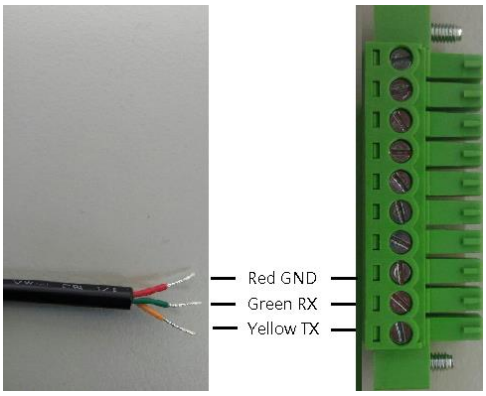
1. DC input range 9~48V
2. DC Terminal Block

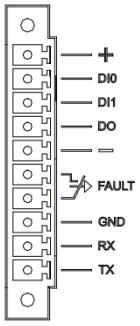
Pin	DC Signal Name
1	Power+
2	N/A
3	Power-



Console Port

- For user setting with debug. You can find TB10 pins for console port as below table.
- Connected to DIO terminal Block



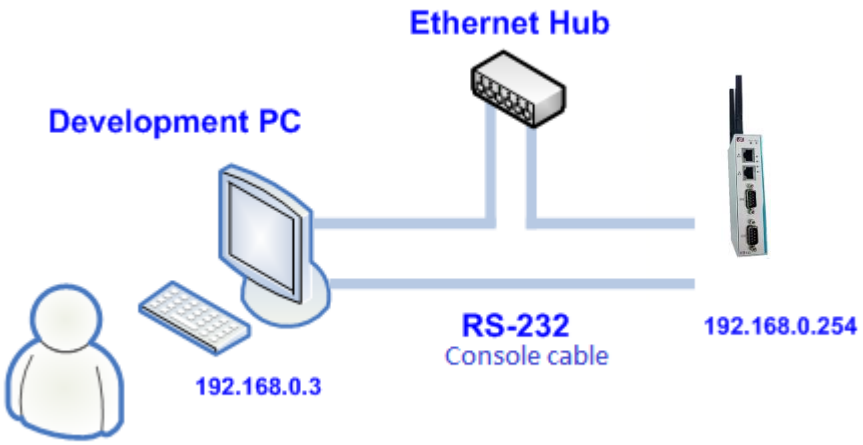


DIO Terminal Block

TB18 Pin No.	Signal name	Meaning
1	COM+	Plus Common for DIO
2	DI0	Digital Input
3	DI1	
4	DO	Digital Output
5	COM-	Minus Common for DIO
6	Relay+	Relay Out
7	Relay-	
8	GND	For Console Port
9	Console RX	
10	Console TX	

You can connect the IFB112 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/Default.aspx?MenuId=Products&FunctionId=gSearch&keyword=IFB112**
-

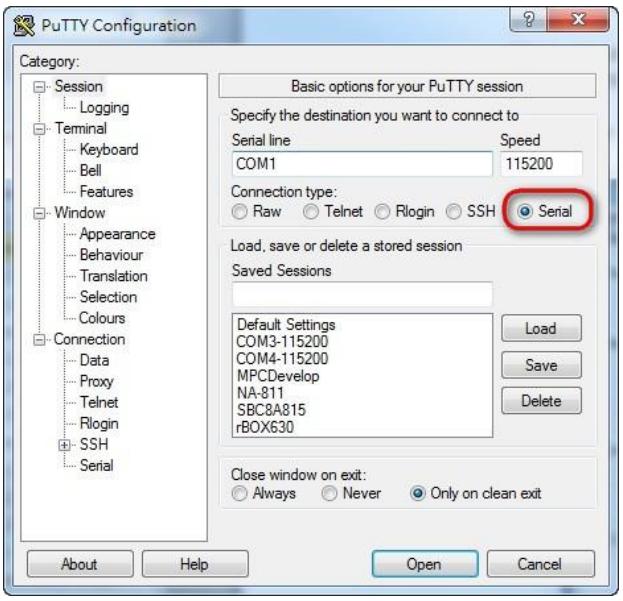
2.1.1 Serial Console

The serial console is a convenient interface for connecting IFB112 to PC. First of all, it is very important to make sure that your desktop connects to IFB112 by console cable. Please set the system as follows:

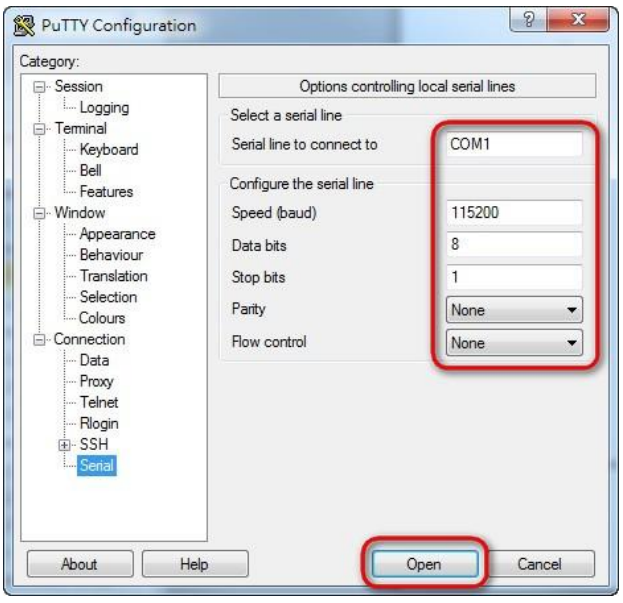
Baudrate: 115200 bps
Parity: None
Data bits: 8
Stop bit: 1
Flow Control: None

Here we use PuTTY to setup and link to the IFB112. Learn how to do it with these step by step instructions:

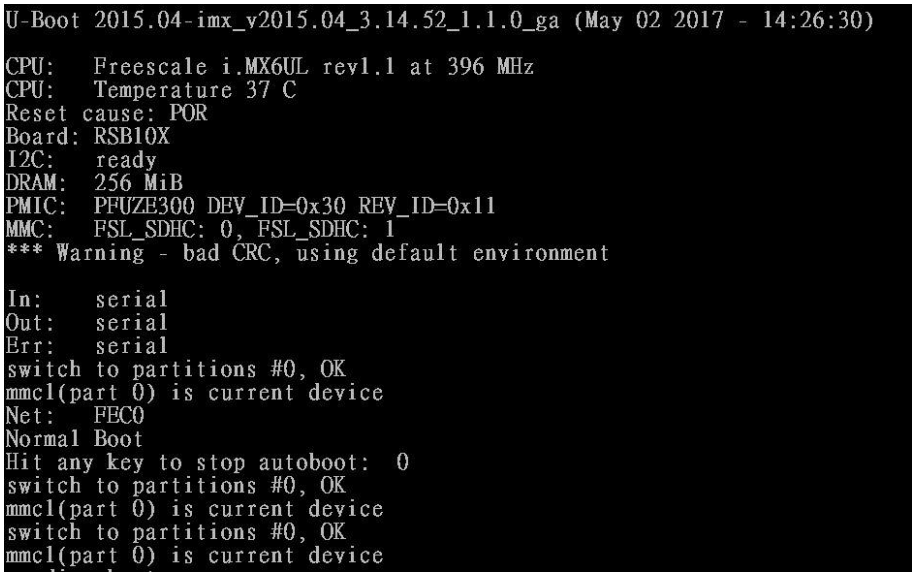
1. Open PuTTY and choose 'Serial' as the connection type.



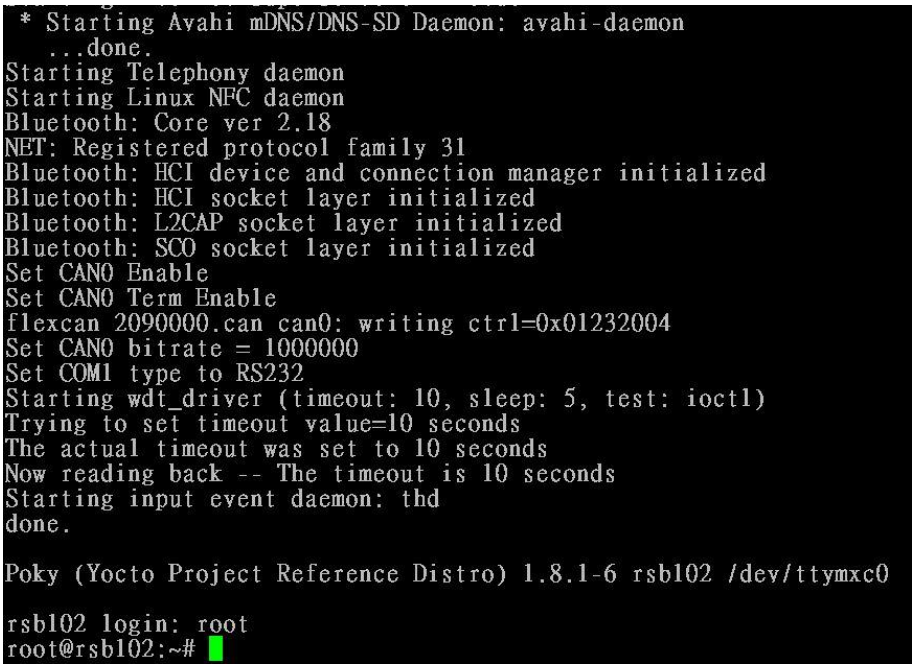
2. Configure the serial port correctly (see image below). Click Open and power on the IFB112.



3. The Bootloader default booting system from eMMC.



4. If connection is established successfully, you should see the image as follows. To login, please enter 'root' (without password).



2.1.2 SSH over Ethernet

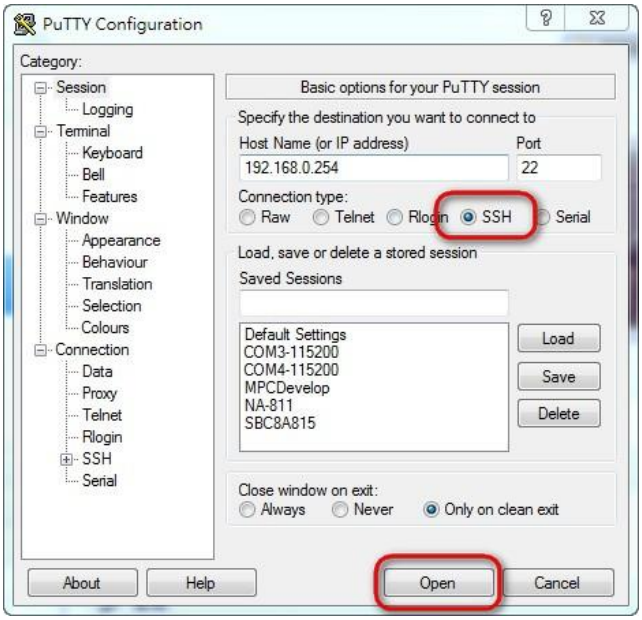
Now, we are going to connect the IFB112 to PC via Ethernet. The following illustrations show you how to do it under Windows® and Linux environment

Note

IFB112 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.

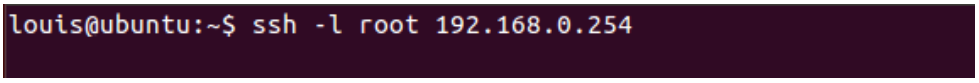


2. If connection is established successfully, you should see the image as follows.

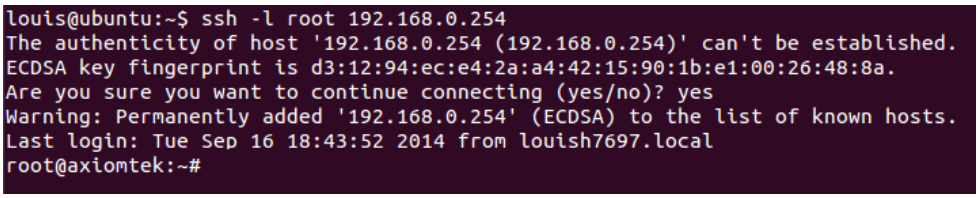
3. To login IFB112, please enter 'root' (with no password).

For Linux users:

1. Open terminal and keyin 'ssh' command.
   ~$ ssh -l root 192.168.0.254



2. After the connection is established successfully.



2.2 How to Develop a Sample Program

In this section, we show you how to develop a sample program for IFB112 with the following step by step instructions. The sample program is named 'hello.c'.

1. Create a directory for IFB112 BSP, and copy IFB112-Linux-bsp-x.x.x.tar.gz into here

~$ mkdir project
~$ cd project

```
ryan@Ubuntu:~$ mkdir project
ryan@Ubuntu:~$ cd project/
ryan@Ubuntu:~/project$ ls
IFB112-Linux-bsp-1.0.0.tar.gz
```

2. After extracted the file, you will find a directory IFB112-Linux-bsp-x.x.x

```
ryan@Ubuntu:~/project$ ls
IFB112-Linux-bsp-1.0.0  IFB112-Linux-bsp-1.0.0.tar.gz
ryan@Ubuntu:~/project$ cd IFB112-Linux-bsp-1.0.0/
ryan@Ubuntu:~/project/IFB112-Linux-bsp-1.0.0$ ls
ChangeLog.txt  IFB112-Linux-bsp-1.0.0
ryan@Ubuntu:~/project/IFB112-Linux-bsp-1.0.0$ cd IFB112-Linux-bsp-1.0.0/
ryan@Ubuntu:~/project/IFB112-Linux-bsp-1.0.0/IFB112-Linux-bsp-1.0.0$ ls
AxTools  Image  README.txt  Toolchain  Yocto patches
```


Note

AxTools : This directory include hardware driver and API library
Image : This directory include kernel, rootfilesystem
Yocto patches : This directory include IFB112 hardware patches for Yocto Project 1.8.1
Toolchain : This directory include cross compiler toolchain build from Yocto Project 1.8.1
README.txt : This BSP's documentation file

2.2.1 Install Yocto Toolchain

Before you develop and compile sample program, you should install Yocto toolchain into development PC. Please follow the steps below to install the Yocto Toolchain.or refer to Chapter 5 Board Support Package to build the toolchain for IFB112.
.

1. To check your Ubuntu version on your host PC.

~$ uname -m

Ubuntu **32-bit** (i686):

```
louis@ubuntu:~$ uname -m
i686
louis@ubuntu:~$
```

Ubuntu **64-bit** (x86_64):

```
louis@ubuntu:~$ uname -m
x86_64
louis@ubuntu:~$
```

2. Copy the toolchain script to home directory.
   i686 for 32-bit machines or x86_64 for 64-bit machines.

```
ryan@Ubuntu:~/project/IFB112-Linux-bsp-1.0.0/IFB112-Linux-bsp-1.0.0/Toolchain$ ls
32-bit  64-bit
```

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

32-bit machines:

~$ bash poky-glibc-i686-meta-toolchain-cortexa7hf-vfp-neon-toolchain-1.8.1.sh

```
ryan@Ubuntu:~/project/IFB112-Linux-bsp-1.0.0/IFB112-Linux-bsp-1.0.0/Toolchain$ cd 32-bit/
ryan@Ubuntu:~/project/IFB112-Linux-bsp-1.0.0/IFB112-Linux-bsp-1.0.0/Toolchain/32-bit$ ls
poky-glibc-i686-meta-toolchain-cortexa7hf-vfp-neon-toolchain-1.8.1.sh
ryan@Ubuntu:~/project/IFB112-Linux-bsp-1.0.0/IFB112-Linux-bsp-1.0.0/Toolchain/32-bit$ bash poky-glibc-i686
-meta-toolchain-cortexa7hf-vfp-neon-toolchain-1.8.1.sh
Enter target directory for SDK (default: /opt/poky/1.8.1):
```

64-bit machines:

~$bash poky-glibc-x86_64-meta-toolchain-cortexa7hf-vfp-neon-toolchain-1.8.1.sh

```
ryan@Ubuntu:~/project/IFB112-Linux-bsp-1.0.0/IFB112-Linux-bsp-1.0.0/Toolchain/64-bit$ ls
poky-glibc-x86_64-meta-toolchain-cortexa7hf-vfp-neon-toolchain-1.8.1.sh
ryan@Ubuntu:~/project/IFB112-Linux-bsp-1.0.0/IFB112-Linux-bsp-1.0.0/Toolchain/64-bit$ bash poky-glibc-x86_
64-meta-toolchain-cortexa7hf-vfp-neon-toolchain-1.8.1.sh
Enter target directory for SDK (default: /opt/poky/1.8.1):
```

4 Check the directory.

```
ryan@Ubuntu:~/project/IFB112-Linux-bsp-1.0.0/IFB112-Linux-bsp-1.0.0/Toolchain/64-bit$ ls
poky-glibc-x86_64-meta-toolchain-cortexa7hf-vfp-neon-toolchain-1.8.1.sh
ryan@Ubuntu:~/project/IFB112-Linux-bsp-1.0.0/IFB112-Linux-bsp-1.0.0/Toolchain/64-bit$ bash poky-glibc-x86_
64-meta-toolchain-cortexa7hf-vfp-neon-toolchain-1.8.1.sh
Enter target directory for SDK (default: /opt/poky/1.8.1):
You are about to install the SDK to "/opt/poky/1.8.1". Proceed[Y/n]?y
```

5 Wait to installation.

```
ryan@Ubuntu:~/project/IFB112-Linux-bsp-1.0.0/IFB112-Linux-bsp-1.0.0/Toolchain/64-bit$ ls
poky-glibc-x86_64-meta-toolchain-cortexa7hf-vfp-neon-toolchain-1.8.1.sh
ryan@Ubuntu:~/project/IFB112-Linux-bsp-1.0.0/IFB112-Linux-bsp-1.0.0/Toolchain/64-bit$ bash poky-glibc-x86_
64-meta-toolchain-cortexa7hf-vfp-neon-toolchain-1.8.1.sh
Enter target directory for SDK (default: /opt/poky/1.8.1):
You are about to install the SDK to "/opt/poky/1.8.1". Proceed[Y/n]?y
Extracting SDK...done
```

6 Install finish.

```
ryan@Ubuntu:~/project/IFB112-Linux-bsp-1.0.0/IFB112-Linux-bsp-1.0.0/Toolchain/64-bit$ ls
poky-glibc-x86_64-meta-toolchain-cortexa7hf-vfp-neon-toolchain-1.8.1.sh
ryan@Ubuntu:~/project/IFB112-Linux-bsp-1.0.0/IFB112-Linux-bsp-1.0.0/Toolchain/64-bit$ bash poky-glibc-x86_
64-meta-toolchain-cortexa7hf-vfp-neon-toolchain-1.8.1.sh
Enter target directory for SDK (default: /opt/poky/1.8.1):
You are about to install the SDK to "/opt/poky/1.8.1". Proceed[Y/n]?y
Extracting SDK...done
Setting it up...done
SDK has been successfully set up and is ready to be used.
```

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.

1. To set up cross-toolchain environment.

~$ source /opt/poky/1.8.1/environment-setup-cortexa7hf-vfp-neon-poky-linux-gnueabi

```
ryan@Ubuntu:~$ source /opt/poky/1.8.1/environment-setup-cortexa7hf-vfp-neon-poky-linux-gnueabi
```

2. To check Cross-Development Environment whether successful or not
   It is successful, if you can find the information as below.

~$ echo $CC

```
ryan@Ubuntu:~$ echo $CC
arm-poky-linux-gnueabi-gcc -march=armv7-a -mfloat-abi=hard -mfpu=neon -mtune=cortex-a7 --sysroot=/opt/poky
/1.8.1/sysroots/cortexa7hf-vfp-neon-poky-linux-gnueabi
```

2.2.3 Write and Compile Sample Program

1. Create a directory on your host PC

~$ mkdir -p example
~$ cd example

```
ryan@Ubuntu:~$ mkdir -p example
ryan@Ubuntu:~$ cd example/
```

2. Use vim to edit hello.c.

~$ vim hello.c

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

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

3. To compile the program as follows:

~$ $CC hello.c -o hello

```
ryan@Ubuntu:~/example$ $CC hello.c -o hello
```

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

~$ ls -l

```
ryan@Ubuntu:~/example$ ls -l
total 16
-rwxrwxr-x 1 ryan ryan 9669  6月  1 15:02 hello
-rw-rw-r-- 1 ryan ryan   71  6月  1 15:02 hello.c
```

5. Check the file ARM executable format.
   If it is successful, you can see the information as below

~$ file hello

```
ryan@Ubuntu:~/example$ file hello
hello: ELF 32-bit LSB executable, ARM, EABI5 version 1 (SYSV), dynamically linked, interpreter /lib/ld-lin
ux-armhf.so.3, for GNU/Linux 2.6.32, BuildID[sha1]=03ffaa4ff511b9c7c92e32c78e22b5d51f1cea7b, not stripped
ryan@Ubuntu:~/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 IFB112 and execute it.

2.3.1 Via FTP

The IFB112 has a built-in FTP server. Users can put 'hello' program to IFB112 via FTP to follow the following steps

1. Enable FTPD daemon on IFB112
   Use "vi" to create " /etc/xinetd.d/ftpd file"

   ~# vi /etc/xinetd.d/ftpd

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

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

2. Restart FTP server on IFB112

~# /etc/init.d/xinetd reload
~# /etc/init.d/xinetd restart

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

3. To connect your host PC to IFB112.
   ~$ ftp 192.168.0.254 (username 'root' without password)

```
louis@ubuntu:~/project/example$ ftp 192.168.0.254
Connected to 192.168.0.254.
220 Operation successful
Name (192.168.0.254:louis): root
331 Please specify password
Password:
230 Operation successful
Remote system type is UNIX.
Using binary mode to transfer files.
```

4. Upload "hello" program to IFB112 from your host PC
   ftp> put hello

```
ftp> put hello
local: hello remote: hello
200 Operation successful
150 Ok to send data
226 Operation successful
9669 bytes sent in 0.00 secs (165655.8 kB/s)
ftp>
```

5. If the operation is successful on IFB112, you can see 'hello' program on IFB112's */home/root* directory.

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

6. To change the file permission for execution on IFB112.
   ~# chmod a+x hello

```
root@axiomtek:~# ls -l
-rw-r--r--    1 root     root          9669 Sep 16 18:40 hello
root@axiomtek:~# chmod a+x hello
root@axiomtek:~# ls -l
-rwxr-xr-x    1 root     root          9669 Sep 16 18:40 hello
root@axiomtek:~#
```

7. Run the 'hello' program on IFB112.
   ~# ./hello

```
root@axiomtek:~# ./hello
hello world
root@axiomtek:~#
```

2.3.2 Via USB Flash Drive

Another method of putting 'hello' program into IFB112 is via USB flash drive. Please follow the instructions below.

IFB112 supports storage format FAT32 /FAT/EXT2/EXT3/EXT4

1. From the host PC, copy 'hello' program to USB flash drive.

2. Attach USB flash drive to IFB112.

3. ~# mkdir /media/sda1

```
root@axiomtek:~# mkdir /media/sda1
root@axiomtek:~#
```

4. ~# mount /dev/sda1 /media/sda1

```
root@axiomtek:~# mount /dev/sda1 /media/sda1/
root@axiomtek:~# ls /media/sda1/
hello
root@axiomtek:~#
```

5. ~# cp /media/sda1/hello /home/root

```
root@axiomtek:~# cp /media/sda1/hello /home/root/
root@axiomtek:~# ls
hello
root@axiomtek:~#
```

6. ~# chmod +x hello

```
root@axiomtek:~# ls -l
-rw-r--r--    1 root     root          9669 Sep 16 18:40 hello
root@axiomtek:~# chmod a+x hello
root@axiomtek:~# ls -l
-rwxr-xr-x    1 root     root          9669 Sep 16 18:40 hello
root@axiomtek:~#
```

7. ~# ./hello

```
root@axiomtek:~# ./hello
hello world
root@axiomtek:~#
```

2.3.3 Via TFTP

Originally the Host Development System Installation already has TFTP server installed. You can put the 'hello' program into IFB112 via TFTP. Please follow the instructions below.

1. Refer to section 5.1.1 step 4. Install and configure TFTP server for install and setup your TFTP:

2. To copy "hello" program to "tftpboot" folder in host PC
   ~$ cp hello /tftpboot

```
louis@ubuntu:~/project/example$ ls
hello  hello.c
louis@ubuntu:~/project/example$ cp hello /tftpboot/
louis@ubuntu:~/project/example$ ls /tftpboot/
hello
louis@ubuntu:~/project/example$
```

3. To enter the following command on IFB112
   ~# tftp -g -r hello 192.168.0.3 (tftp server IP depend on host PC's IP)

```
root@axiomtek:~# tftp -g -r hello 192.168.0.3
root@axiomtek:~# ls
hello
root@axiomtek:~#
```

4. To enter the following command on IFB112
   ~# chmod a+x hello

```
root@axiomtek:~# ls -l
-rw-r--r--    1 root     root          9669 Sep 16 18:40 hello
root@axiomtek:~# chmod a+x hello
root@axiomtek:~# ls -l
-rwxr-xr-x    1 root     root          9669 Sep 16 18:40 hello
root@axiomtek:~#
```

5. Run the 'hello' program on IFB112.
   ~# ./hello

```
root@axiomtek:~# ./hello
hello world
root@axiomtek:~#
```

2.4 How to Recovery System

In this section, we provide 2 methods showing how to recovery IFB112 system to default.

2.4.1 Via run_rescue System Script (under Linux System)

There is a recovery script in /etc folder on IFB112 Embedded Linux system. If you want to recovery your system to factory default settings, you can follow the instructions.

1. Run the run_rescue shell script
   ~# /etc/run_rescue

```
root@axiomtek:~# /etc/run_rescue
Push RESCUE Script to u-boot
Reboot system to RESCUE/UPDATE system

Broadcast message from root@axiomtek (ttymxc0) (Tue Sep 16 20:01:32 2014):
The system is going down for reboot NOW!
INIT: Switching to runlevel: 6
INIT: Sending processes the TERM signal
logout
```

2. When the system reboot, it will auto switch to rescue mode under u-boot, and start to recovery procedure. During this procedure, four custom LEDs will blink like marquee.

3. After recovery procedure completes, the system will reboot again automatically and the system status LED will change from blinking to always on.

2.4.2 Via rescue.scr Script (under u-boot)

Refer to section 5.2.3 for detailed information.

2.5 How to Update System

In this section, we provide a method showing how to update IFB112.

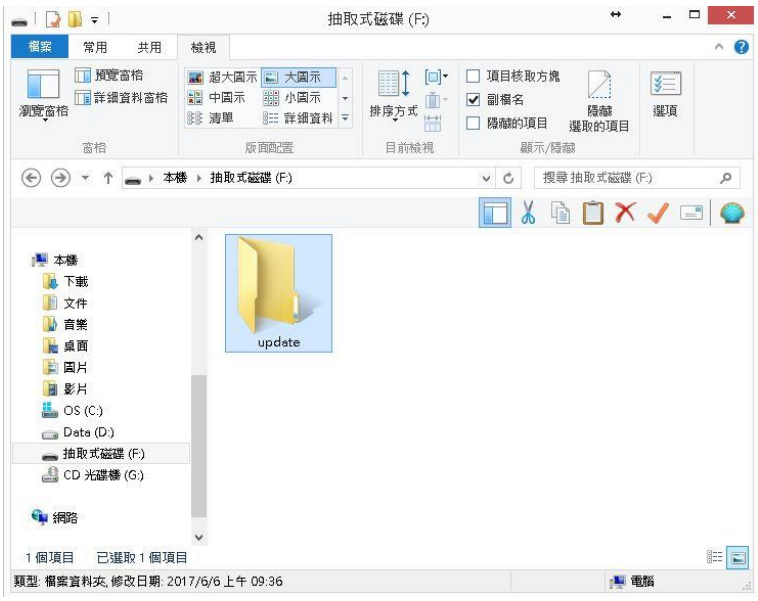
2.5.1 Via USB Flash Drive

The USB flash drive can be DOS FAT32、EXT2、EXT3 or EXT4 format, but update folder must at first partition.

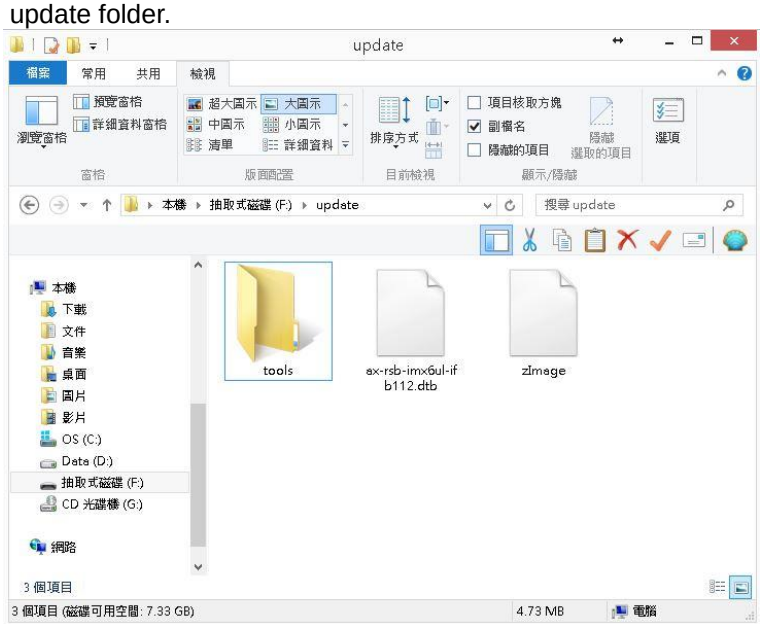
1. From the PC, copy files to USB flash drive.

2. Create a folder name "update"

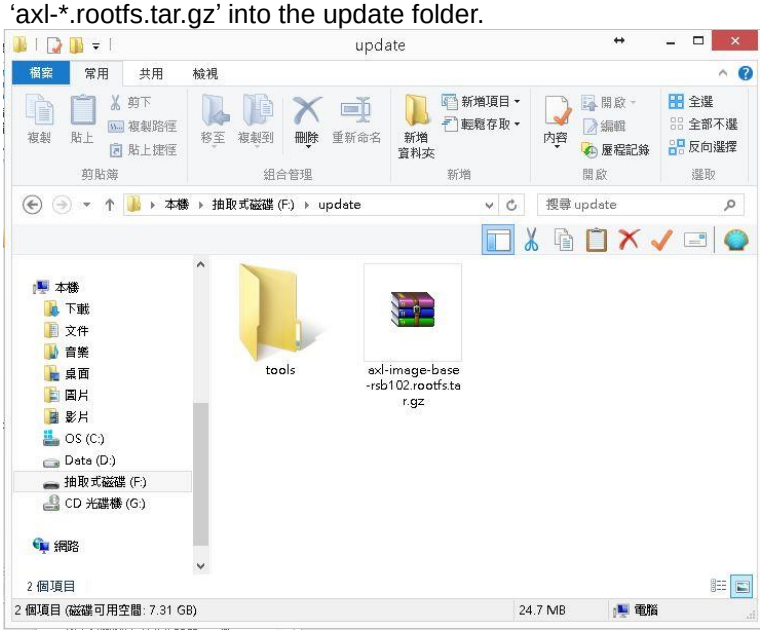
.



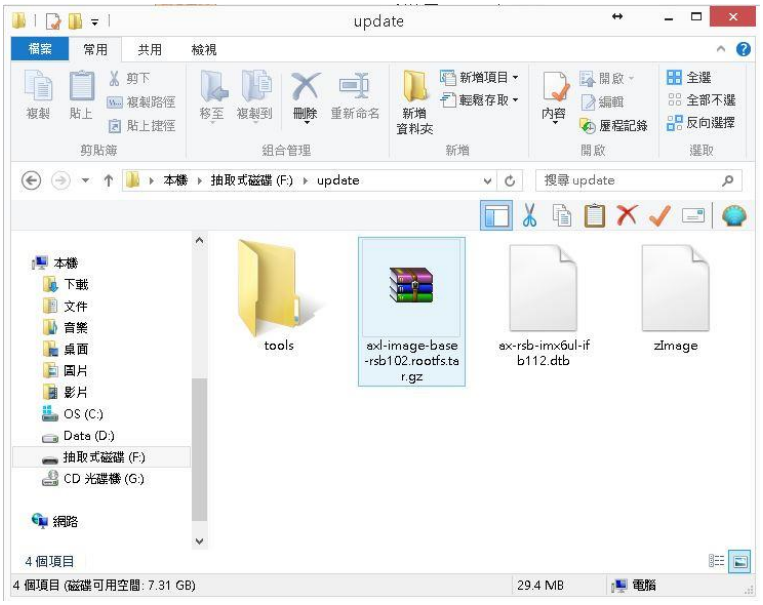
3. If you just want to update kernel without the root filesystem, you just need to rename the new kernel file to 'zImage' and dtb file to 'ax-rsb-imx6ul-ifb112.dtb' and put it into the update folder.



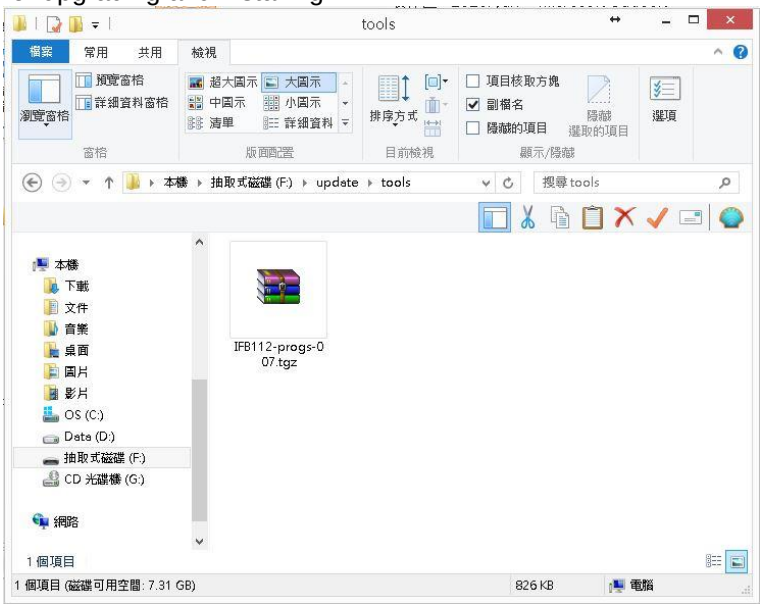
4. If you just want to update the root filesystem without the kernel, you just need to put 'axl-*.rootfs.tar.gz' into the update folder.



5. If you want to update the kernel and the root filesystem, put three files into the update folder.



6. If Axiomtek provide other apps or tools to install, create a tools folder under update folder for upgrading and installing.



7. Attach USB flash drive to IFB112.
8. Run the run_rescue shell script.
   ~# /etc/run_rescue

```
sd 0:0:0:0: [sda] Assuming drive cache: write through
 sda: sda1
sd 0:0:0:0: [sda] No Caching mode page found
sd 0:0:0:0: [sda] Assuming drive cache: write through
sd 0:0:0:0: [sda] Attached SCSI removable disk
Freeing unused kernel memory: 368K (8088d000 - 808e9000)
INIT: version 2.88 booting
Starting udev
udevd[159]: starting version 182
EXT4-fs (mmcblk1p3): re-mounted. Opts: data=ordered
bootlogd: cannot allocate pseudo tty: No such file or directory
random: dd urandom read with 91 bits of entropy available
random: nonblocking pool is initialized
INIT: Entering runlevel: 5
Starting syslogd/klogd: done
FAT-fs (sda1): Volume was not properly unmounted. Some data may be corrupt. Ple.
===== Starting Update Kernel Procedure =====
===== Starting Update RootFilesystem Procedure =====
EXT4-fs (mmcblk1p2): mounted filesystem with ordered data mode. Opts: (null)
EXT4-fs (mmcblk1p2): mounted filesystem with ordered data mode. Opts: (null)
Copy Other tools.....
Extracting /media/sda1/update/tools/IFB122-progs-004.tgz
============== Finished ==============
After 3 seconds will reboot system...
```

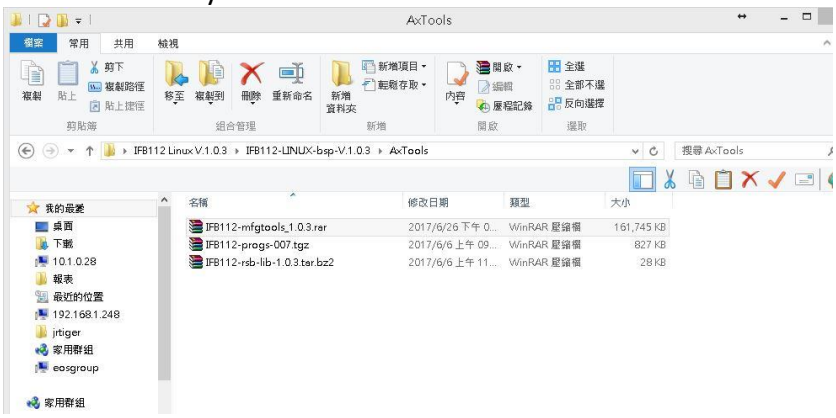
9. During this update procedure, four custom LEDs will blink like marquee. Until procedure completes, the system will reboot again automatically, and the system status LED change from blinking to always on.

2.6 How to use MFG tool to download image

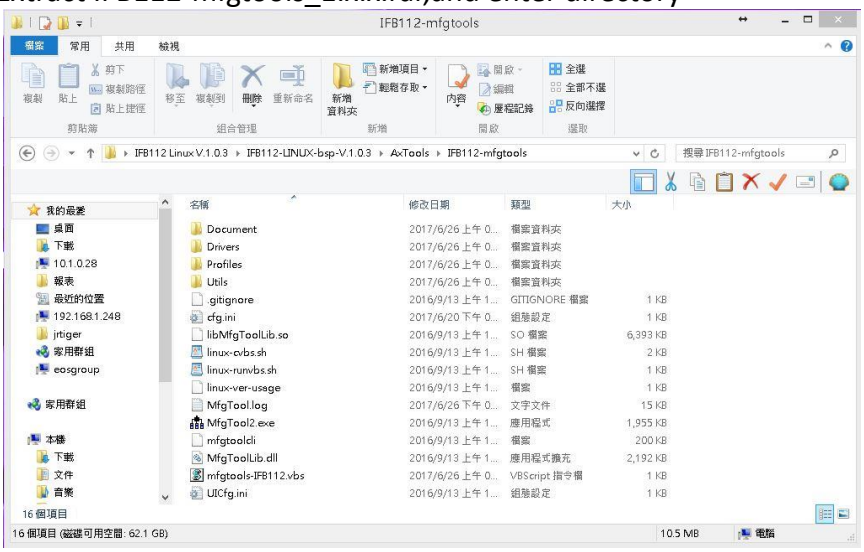
We show you how to use MFG tool to download bin image to the IFB112 system.

1. Before using the mfgtool, you have to change the ifb112 JP1 boot mode (default emmc boot) to OTG serial downloader mode. Then change the JP3 USB mode (default OTG host mode) to OTC client mode. Connect the IFB112 and PC with a USB cable.

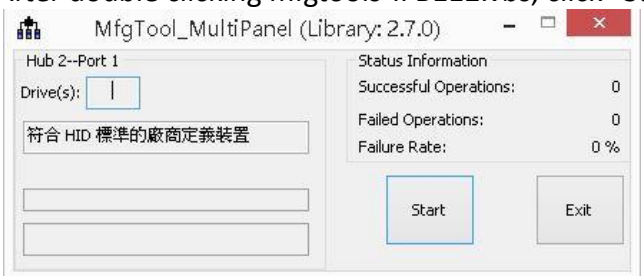
2. Extract Axiomtek's Yocto BSP and you will see IFB112-mfgtools_1.x.x.rar in the AxTools directory



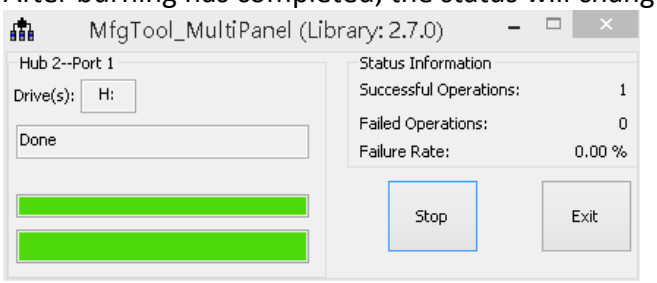
3. Extract IFB112-mfgtools_1.x.x.rar,and enter directory



4. After double clicking mfgtools-IFB112.vbs, click "Start" to start burning.



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



6. For detailed information about Mfgtool, please refer to "Manufacturing Tool V2 Quick Start Guide.docx" in the "Document\V2" directory.

This page is intentionally left blank.

Chapter 3
The Embedded Linux

3.1 Embedded Linux Image Managing

3.1.1 System Version

This section describes how to determine the system version information including kernel and root filesystem version on IFB112.

Check the kernel version with the following command:
~# uname -r

```
root@axiomtek:~# uname -r
3.14.52-RSB10X-011
root@axiomtek:~#
```

Check the root filesystem with the login screen:

```
Poky (Yocto Project Reference Distro) 1.8.1-6 axiomtek /dev/ttymxc0

axiomtek login:
```

3.1.2 System Time

System time is time value loaded from RTC each time the system boots up. Read the system time with the following command on IFB112:

~# date

```
root@axiomtek:~# date
Tue May  2 07:16:36 UTC 2017
```

3.1.3 Internal RTC Time

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

Read internal RTC time with the following command on IFB112:
~# hwclock -r --rtc=/dev/rtc1

```
root@axiomtek:~# hwclock -r --rtc=/dev/rtc1
Thu Jan  1 00:31:56 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@axiomtek:~# hwclock -r
Tue May  2 07:17:40 2017  0.000000 seconds
```

3.1.5 Watchdog timer

Function: wdt_driver_test.out
Description: When <sleep> parameters is more than <timeout> parameters, watchdog timer will be trigger

Note: IFB112 has been enabled for default setting, and the default parameters is ***10 5 0***
Commands example: ~# wdt 10 5 0 &

```
root@axiomtek:~# /usr/sbin/wdt
Usage: wdt_driver_test <timeout> <sleep> <test>
    timeout: value in seconds to cause wdt timeout/reset
    sleep: value in seconds to service the wdt
    test: 0 - Service wdt with ioctl(), 1 - with write()
```

3.1.6 Adjusting System Time

1. Manually set up the system time.

Format: YYYYMMDDHHmm.SS
~# date -s 201706061200.00

```
root@axiomtek:~# date -s 201706061200.00
Tue Jun  6 12:00:00 UTC 2017
```

2. Write sync time to internal RTC

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

```
root@axiomtek:~# hwclock -w --rtc=/dev/rtc1
root@axiomtek:~# hwclock -r --rtc=/dev/rtc1
Tue Jun  6 12:05:42 2017  0.000000 seconds
```

3. Write sync time to external RTC

~# hwclock -w
~# hwclock -r

```
root@axiomtek:~# hwclock -w
root@axiomtek:~# hwclock -r
Tue Jun  6 12:08:03 2017  0.000000 seconds
```

3.1.7 LEDs Control

Four custom LEDs are supported by IFB112: LED1, LED2, LED3 and LED4.

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

1. Turn on LED1

```
~# echo 255 > /sys/class/leds/LED1/brightness
```

```
root@axiomtek:~# echo 255 > /sys/class/leds/LED1/brightness
```

2. Turn on LED2

```
~# echo 255 > /sys/class/leds/LED2/brightness
```

```
root@axiomtek:~# echo 255 > /sys/class/leds/LED2/brightness
```

3.1.8 CANbus Control

Function: canmode
Description : The parameter definition of CANbus's function as below

```
Usage: commode [CAN] [enable] [term] [bitrate]
        CAN: 0/1/2
        enable: 0/1
        term: 0/1
        bitrate >=5000 && bitrate <=1000000
```

The factory default of CANbus is **CAN0**

```
~# canmode 0 1 1 1000000
```

```
root@axiomtek:~# canmode
Usage: commode [CAN] [enable] [term] [bitrate]
        CAN: 0/1/2
        enable: 0/1
        term: 0/1
        bitrate >=5000 && bitrate <=1000000
root@axiomtek:~# canmode 0 1 1 1000000
Set CAN0 Enable
Wrong CAN term type.
Set CAN0 Term Enable
flexcan 2090000.can can0: writing ctrl=0x01232054
Set CAN0 bitrate = 1000000
root@axiomtek:~#
```

1. You can transmit the data through CAN0.
   For example: ID is "123", value is "AABBABCD"

```
~# cansend can0 123#AABBABCD
```

```
root@axiomtek:~# cansend can0 123#AABBABCD
root@axiomtek:~#
```

2. You can receive the data from CAN0

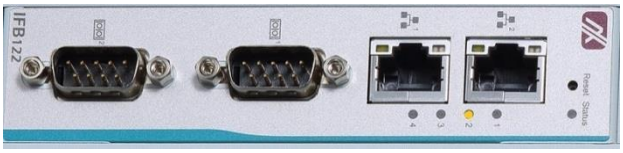
```
~# candump can0
```

```
root@axiomtek:~# candump can0
  can0  456   [8]  11 22 33 44 55 66 00 00
```

3. Turn off LED1

```
~# echo 0 > /sys/class/leds/LED1/brightness
```

```
root@axiomtek:~# echo 0 > /sys/class/leds/LED1/brightness
```



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 IFB112 comes with a built-in FTP server. Section 2.1 shows the steps to put 'hello' program to IFB112 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.

3.2.4 How to use 3G or 4G module (Optional)

1. **3G / 4G module connection to the Internet with PPP**
   This section describes how to use 3G or 4G module connect to the Internet with PPP

   1.1 If your 3G module is Quectel UC20, you can follow the instructions as below.

   Please execute script for internet connection.
   ~# /etp/ppp/ppp-quectel-on

```
root@axiomtek:~# /etc/ppp/ppp-quectel-on
```

   When you execute script, you may find the information as below.

```
PPP generic driver version 2.4.2
pppd options in effect:
dump            # (from command line)
noauth          # (from /etc/ppp/peers/quectel)
user CARD               # (from /etc/ppp/peers/quectel)
password ??????         # (from /etc/ppp/peers/quectel)
/dev/ttyUSB3            # (from /etc/ppp/peers/quectel)
115200          # (from /etc/ppp/peers/quectel)
lock            # (from /etc/ppp/peers/quectel)
connect /usr/sbin/chat -s -v -f /etc/ppp/quectel-chat-connect           # (from)
disconnect /usr/sbin/chat -s -v -f /etc/ppp/quectel-chat-disconnect             )
crtscts         # (from /etc/ppp/peers/quectel)
modem           # (from /etc/ppp/peers/quectel)
hide-password           # (from /etc/ppp/peers/quectel)
ipcp-accept-local               # (from /etc/ppp/peers/quectel)
ipcp-accept-remote              # (from /etc/ppp/peers/quectel)
noipdefault             # (from /etc/ppp/peers/quectel)
defaultroute            # (from /etc/ppp/peers/quectel)
usepeerdns              # (from /etc/ppp/peers/quectel)
nobsdcomp               # (from /etc/ppp/peers/quectel)
root@axiomtek:~#
```

   You can execute command **,ifconfig** to examine PPP0 connection.
   ~# ifconfig

```
root@axiomtek:~# ifconfig
```

   PPP0 will be shown after successful connection.

```
ppp0      Link encap:Point-to-Point Protocol
          inet addr:10.116.2.38  P-t-P:10.64.64.64  Mask:255.255.255.255
          UP POINTOPOINT RUNNING NOARP MULTICAST  MTU:1500  Metric:1
          RX packets:6 errors:0 dropped:0 overruns:0 frame:0
          TX packets:5 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:3
          RX bytes:65 (65.0 B)  TX bytes:86 (86.0 B)

root@axiomtek:~#
```

1.2 If your 4G module is Sierra MC7304, you can follow the instructions as below.

Please execute script for internet connection.
~# /etp/ppp/ppp-sierra-on

```
root@axiomtek:~# /etc/ppp/ppp-sierra-on
```

When you execute script, you may find the information as below.

```
PPP generic driver version 2.4.2
pppd options in effect:
dump            # (from command line)
noauth          # (from /etc/ppp/peers/sierra)
user CARD               # (from /etc/ppp/peers/sierra)
password ??????         # (from /etc/ppp/peers/sierra)
/dev/ttyUSB2            # (from /etc/ppp/peers/sierra)
115200          # (from /etc/ppp/peers/sierra)
lock            # (from /etc/ppp/peers/sierra)
connect /usr/sbin/chat -s -v -f /etc/ppp/sierra-chat-connect            # (from)
disconnect /usr/sbin/chat -s -v -f /etc/ppp/sierra-chat-disconnect              )
crtscts         # (from /etc/ppp/peers/sierra)
modem           # (from /etc/ppp/peers/sierra)
hide-password           # (from /etc/ppp/peers/sierra)
ipcp-accept-local               # (from /etc/ppp/peers/sierra)
ipcp-accept-remote              # (from /etc/ppp/peers/sierra)
noipdefault             # (from /etc/ppp/peers/sierra)
defaultroute            # (from /etc/ppp/peers/sierra)
usepeerdns              # (from /etc/ppp/peers/sierra)
nobsdcomp               # (from /etc/ppp/peers/sierra)
root@axiomtek:~#
```

You can execute command **,ifconfig** to examine PPP0 connection.
~# ifconfig

```
root@axiomtek:~# ifconfig
```

PPP0 will be shown after successful connection.

```
ppp0      Link encap:Point-to-Point Protocol
          inet addr:10.33.122.177  P-t-P:10.64.64.64  Mask:255.255.255.255
          UP POINTOPOINT RUNNING NOARP MULTICAST  MTU:1500  Metric:1
          RX packets:5 errors:0 dropped:0 overruns:0 frame:0
          TX packets:5 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:3
          RX bytes:62 (62.0 B)  TX bytes:86 (86.0 B)

root@axiomtek:~#
```

2 3G / 4G module connection to the Internet with wvdial Tool

2.1 If your 3G module is Quectel UC20, you can follow the instructions as below.

To create a wvdial config
~# vi /etc/wvdial.conf

```
root@axiomtek:~# vi /etc/wvdial.conf
```

Please enter your information as below.

```
[Dialer Defaults]
Modem = /dev/ttyUSB3
Baud = 115200
Init 3 =AT+CGDCONT=1,"IP","INTERNET"
Phone = *99#
Password = any
Username = any
Dial Command = ATD
Modem Type = Analog Modem
NEW PPPD = yes
```

Please execute wvdial for internet connection.
~# wvdial &

```
root@axiomtek:~# wvdial &
```

When you execute wvdia, you may find the information as below.

```
[1] 426
root@axiomtek:~# --> WvDial: Internet dialer version 1.61
--> Initializing modem.
--> Sending: ATZ
ATZ
OK
--> Modem initialized.
--> Sending: ATD*99#
--> Waiting for carrier.
ATD*99#
CONNECT 14400000
--> Carrier detected.  Waiting for prompt.
--> Don't know what to do!  Starting pppd and hoping for the best.
--> Starting pppd at Mon Aug 15 10:51:15 2016
--> Pid of pppd: 429
PPP generic driver version 2.4.2
--> Using interface ppp0
--> local  IP address 10.112.49.117
--> remote IP address 10.64.64.64
--> primary   DNS address 168.95.1.1
--> secondary DNS address 168.95.192.1

root@axiomtek:~#
```

You can execute command **,ifconfig** to examine PPP0 connection
~# ifconfig

```
root@axiomtek:~# ifconfig
```

PPP0 will be shown after successful connection.

```
ppp0      Link encap:Point-to-Point Protocol
          inet addr:10.112.49.117  P-t-P:10.64.64.64  Mask:255.255.255.255
          UP POINTOPOINT RUNNING NOARP MULTICAST  MTU:1500  Metric:1
          RX packets:6 errors:0 dropped:0 overruns:0 frame:0
          TX packets:5 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:3
          RX bytes:65 (65.0 B)  TX bytes:86 (86.0 B)

root@axiomtek:~#
```

2.2 If your 4G module is Sierra MC7304, you can follow the instructions as below.

To create a wvdial config
~# vi /etc/wvdial.conf

```
root@axiomtek:~# vi /etc/wvdial.conf
```

Please enter your information as below.

[Dialer Defaults]
Modem = /dev/ttyUSB2
Baud = 115200
Init 3 =AT+CGDCONT=1,"IP","INTERNET"
Phone = *99#
Password = any
Username = any
Dial Command = ATD
Modem Type = Analog Modem
NEW PPPD = yes

Please execute wvdial for internet connection.
~# wvdial &

```
root@axiomtek:~# wvdial &
```

When you execute **wvdia,** you may find the information as below.

```
[1] 437
root@axiomtek:~# --> WvDial: Internet dialer version 1.61
--> Cannot get information for serial port.
--> Initializing modem.
--> Sending: ATZ
ATZ
OK
--> Modem initialized.
--> Sending: ATD*99#
--> Waiting for carrier.
ATD*99#
CONNECT 100000000
--> Carrier detected.  Waiting for prompt.
--> Don't know what to do!  Starting pppd and hoping for the best.
--> Starting pppd at Mon Aug 15 10:51:09 2016
--> Pid of pppd: 441
PPP generic driver version 2.4.2
--> Using interface ppp0
--> local  IP address 10.33.122.177
--> remote IP address 10.64.64.64
--> primary   DNS address 168.95.1.1
--> secondary DNS address 168.95.192.1

root@axiomtek:~#
```

You can execute command **,ifconfig** to examine PPP0 connection
~# ifconfig

```
root@axiomtek:~# ifconfig
```

PPP0 will be shown after successful connection.

```
ppp0      Link encap:Point-to-Point Protocol
          inet addr:10.33.122.177  P-t-P:10.64.64.64  Mask:255.255.255.255
          UP POINTOPOINT RUNNING NOARP MULTICAST  MTU:1500  Metric:1
          RX packets:5 errors:0 dropped:0 overruns:0 frame:0
          TX packets:5 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:3
          RX bytes:62 (62.0 B)  TX bytes:86 (86.0 B)

root@axiomtek:~#
```

3.2.5 How to use Wi-Fi module (Optional)

If your Wi-Fi module is WPER-172GN, you can follow the instructions as below.

Editor /etc/wpa_supplicant.conf file
~# vi /etc/wpa_supplicant.conf

```
root@axiomtek:~# vi /etc/wpa_supplicant.conf
```

Enter your router,s SSID and Password

```
ctrl_interface=/var/run/wpa_supplicant
ctrl_interface_group=0
update_config=1

network={
        ssid="axiomtwek"
        psk="password"
}
```

If the setting is successful, it will automatically connect after reboot.

You can execute command" **ifconfig"** to check connection.
~# ifconfig

```
wlan0     Link encap:Ethernet  HWaddr B0:1F:81:D0:33:EA
          inet addr:192.168.0.41  Bcast:192.168.0.255  Mask:255.255.255.0
          inet6 addr: fe80::b21f:81ff:fed0:33ea/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:22 errors:0 dropped:24 overruns:0 frame:0
          TX packets:26 errors:0 dropped:5 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:5725 (5.5 KiB)  TX bytes:5679 (5.5 KiB)

root@rsb101:~#
```

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 librsb10x API Functions

The IFB112 BSP includes 'librsb10x.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 RSB10X-rsb_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	Set_COMType()	Set COM port communication mode type.
2	Get_DI0()	Read high or low state on digital input channels.
3	Get_DI1()	Read high or low state on digital input channels.
4	Set_DO()	Set digital output channels to high or low state.
5	Set_RELAY()	Set relay high or low state.
6	Control_LED()	Set LED enable or disable
7	Control_WDT()	Set WDT funciotn
8	Set_CANterm()	Set CANbus termination resistors enable or disable.
9	Set_CANenable()	Set CAN enable
10	Set_CANbitrate()	Set CAN bitrate

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. 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.

COM sample code:

COM receive

```
#include <stdio.h>
#include <termios.h> //header contains the definitions used by the terminal I/O interfaces
#include <unistd.h> //read() write() close()
#include <fcntl.h>
#include <string.h>

//Serial port defines and variables:
#define SERIAL_BAUDRATE B9600
#define SERIAL_DEVICE "/dev/ttyUSB0"
//#define SERIAL_DEVICE "/dev/ttymxc2"
#define SERIAL_BUFFER_SIZE 256

struct termios serial_settings;
int serial_fd, serial_read_ret, serial_write_ret;
//char serial_buffer_send[SERIAL_BUFFER_SIZE]="hello word";
char serial_buffer_recv[SERIAL_BUFFER_SIZE];

int initport(int serial_fd) {
        struct termios options;
        // Get the current options for the port...
        tcgetattr(serial_fd, &options);
        // Set the baud rates to 9600
        cfsetispeed(&options, SERIAL_BAUDRATE);
        cfsetospeed(&options, SERIAL_BAUDRATE);
        // Enable the receiver and set local mode...

        options.c_cflag &= ~PARENB;
        options.c_cflag &= ~CSTOPB;
        options.c_cflag &= ~CSIZE;
        options.c_cflag |= CS8;

        // Set the new options for the port...
        tcsetattr(serial_fd, TCSANOW, &options);
        return 1;
```

```
}

int main() {

	//Try opening serial port
	//serial_fd = open(SERIAL_DEVICE, O_RDWR | O_NOCTTY | O_NDELAY);
	serial_fd = open(SERIAL_DEVICE, O_RDWR|O_NOCTTY);
	if(serial_fd == -1) { //Checks the availability of the Serial Port
		printf("Opening %s Failed.\n",SERIAL_DEVICE);
		fflush(stdout);
		return 0;
	}
	printf("Opening %s Success.\n",SERIAL_DEVICE);
	fflush(stdout);
	initport(serial_fd);

	printf("start read\n");
	while(1){
	serial_read_ret=read(serial_fd,serial_buffer_recv, SERIAL_BUFFER_SIZE);
		serial_buffer_recv[serial_read_ret]=0;
		printf("res=%d buf=%s\n", serial_read_ret, serial_buffer_recv);
		if (serial_buffer_recv[0] == '@') break;
	}
	serial_read_ret = close(serial_fd); //Close the serial port
	printf("Serial port closed.\n");
	return 0;
}
```

COM send:

```
#include <stdio.h>
#include <termios.h> //header contains the definitions used by the terminal I/O interfaces
#include <unistd.h> //read() write() close()
#include <fcntl.h>
#include <string.h>

//Serial port defines and variables:
#define SERIAL_BAUDRATE B9600
//#define SERIAL_DEVICE "/dev/ttyUSB0"
#define SERIAL_DEVICE "/dev/ttymxc1"
#define SERIAL_BUFFER_SIZE 256

struct termios serial_settings;
int serial_fd, serial_read_ret, serial_write_ret;
char serial_buffer_send[SERIAL_BUFFER_SIZE]="hello word";
//char serial_buffer_recv[SERIAL_BUFFER_SIZE];

int initport(int serial_fd) {
	struct termios options;
	// Get the current options for the port...
	tcgetattr(serial_fd, &options);
	// Set the baud rates to 9600
	cfsetispeed(&options, SERIAL_BAUDRATE);
	cfsetospeed(&options, SERIAL_BAUDRATE);
	// Enable the receiver and set local mode...

	options.c_cflag &= ~PARENB;
```

```
        options.c_cflag &= ~CSTOPB;
        options.c_cflag &= ~CSIZE;
        options.c_cflag |= CS8;

        // Set the new options for the port...
        tcsetattr(serial_fd, TCSANOW, &options);
        return 1;
}

int main() {

        //Try opening serial port
        serial_fd = open(SERIAL_DEVICE, O_RDWR | O_NOCTTY | O_NDELAY);
        if(serial_fd == -1) { //Checks the availability of the Serial Port
                printf("Opening %s Failed.\n",SERIAL_DEVICE);
                fflush(stdout);
                return 0;
        }
        printf("Opening %s Success.\n",SERIAL_DEVICE);
        fflush(stdout);
        initport(serial_fd);

        printf("start write\n");
        int len = strlen(serial_buffer_send);
        serial_buffer_send[len] = 0x0d; // stick a <CR> after the command
        serial_buffer_send[len+1] = 0x00; // terminate the string properly
        serial_write_ret = write(serial_fd,serial_buffer_send,strlen(serial_buffer_send));
        if(serial_write_ret < 0)
                printf("Sent to serial port fail\n");
        else
                printf("Sent to serial port = %s\n",serial_buffer_send);

        serial_read_ret = close(serial_fd); //Close the serial port
        printf("Serial port closed.\n");
        return 0;
}
```

Function: Get_DI0()

Function	**int Get_DI0(__u8 *data);**
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_DI1()

Function	**int Get_DO1(__u8 *data);**
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	**int Set_DO(__u8 data);**
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.

DIO sample code:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <linux/types.h>
#include "librsb10x.h"

int main(int argc, char* argv[])
{
	unsigned char xch, xi=1;

	Set_DO(xi);

	Get_DI0(&xch);
	printf("DI0 Input = %d\n", xch);

	Get_DI1(&xch);
	printf("DI1 Input = %d\n", xch);

	return 0;
}
```

Function: Set_RELAY ()

Function	**int Set_RELAY(int hl);**
Description	Set relay high or low state.
Arguments	hl: relay state. 0: LOW. 1: HIGH.
Return	0: No error. 1: Function fails.
Others	None.

Relay sample code:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <linux/types.h>
#include "librsb10x.h"

#define HIGH        1
#define LOW         0

int main(int argc, char* argv[])
{
    printf("Turn relay on\n");
    Set_RELAY(HIGH);

    sleep(2);

    printf("Turn relay off\n");
    Set_RELAY(LOW);

    return 0;
}
```

Function: Control_LED ()

Function	**int Control_LED(int num,int enable);**
Description	Set LED enable or disable
Arguments	Num : LED number,default as 1 ~ 4 Enable : enable or disable led 0: disable 1: enable
Return	0: No error. 1: Function fails.
Others	None.

LED sample code:

```
#include <stdio.h>
#include <stdlib.h>
#include "librsb10x.h"

int Control_LED(int num,int enable);

int main()
{
        printf("Function Name : Control_LED(num,enable)\n");
        printf("num: LED number,default 1~4 \n");
        printf("enable: 1 - enable LED , 2 - Disable LED\n");
        printf("turn on LED 1\n");
        Control_LED(1,1);
        sleep(1);
        printf("turn on LED 2\n");
        Control_LED(2,1);
        sleep(1);
        printf("turn on LED 3\n");
        Control_LED(3,1);
        sleep(1);
        printf("turn on LED 4\n");
        Control_LED(4,1);
        sleep(1);
        printf("turn off LED 1\n");
        Control_LED(1,0);
        sleep(1);
        printf("turn off LED 2\n");
        Control_LED(2,0);
        sleep(1);
        printf("turn off LED 3\n");
        Control_LED(3,0);
        sleep(1);
        printf("turn off LED 4\n");
        Control_LED(4,0);
        return 0;
}
```

Function: Control_WDT ()

Function	**int Control_WDT(int timeout,int sleep_time,int test);**
Description	Set WDT Function
Arguments	timeout : value in seconds to cause wdt timeout/reset sleep_time : value in seconds to service the wdt test : 0 – service wdt with ioctl(), 1 – with write()
Return	0: No error. 1: Function fails.

WDT sample code:

```
#include <stdio.h>
#include <stdlib.h>,
#include "librsb10x.h"


int main()
{
        printf("Function Name : Control_WDT(timeout,sleep_time,test)\n");
        printf("timeout: value in seconds to cause wdt timeout/reset \n");
        printf("sleep_time: value in seconds to service the wdt \n");
        printf("test: 0 - Service wdt with ioctl(), 1 - with write()\n");
        printf("\nRun Contrl_WDT(10,5,0)\n");
        Contrl_WDT(10,5,0);
        return 0;
}
```

Function: Set_CANterm ()

Function	**int Set_CANterm(int number,int type);**
Description	Set CAN term Function
Arguments	Number : CANbus port number(default value as 0) Type : Enable or disable term Type 0: Disable term Type 1: Enable term
Return	0: No error. 1: Function fails.

CANbus sample code:

```
#include <stdio.h>
#include <stdlib.h>,
#include "librsb10x.h"


int main()
{
        Set_CANterm(0, 1);
        printf("Set CAN0 termination enable\n");
        return 0;
}
```

Function: Set_CANenable()

Function	**int Set_CANenable(int number,int type);**
Description	Set CAN enable Function
Arguments	Number : CANbus port number(default value as 0) Type : Enable or disable Type 0: Disable Type 1: Enable
Return	0: No error. 1: Function fails.

CANbus sample code:

```
#include <stdio.h>
#include <stdlib.h>,
#include "librsb10x.h"


int main()
{
        Set_CANenable(0,1);
        printf("Set CAN0 enable\n");
        return 0;
}
```

Function: Set_CANbitrate()

Function	int Set_CANbitrate(int number,int bitrate);
Description	Set CAN bitrate Function
Arguments	Number : CANbus port number(default value as 0) Bitrate : bitrate value as 5000 - 1000000
Return	0: No error. 1: Function fails.

CANbus sample code:

```
#include <stdio.h>
#include <stdlib.h>
#include "librsb10x.h"


int main()
{
        Set_CANbitrate(0, 10000);
        printf("Set CAN0 bitrate = 10000\n");
        return 0;
}
```

CANbus Write sample code:

```
#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;
        int nbytes;
        struct sockaddr_can addr;
        struct can_frame frame;
        struct ifreq ifr;
          int xBitRate=1000000;

        char *ifname = "can0";

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

```
        strcpy(ifr.ifr_name, ifname);
        ioctl(s, SIOCGIFINDEX, &ifr);

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

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

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

        frame.can_id  = 0x123;
        frame.can_dlc = 2;
        frame.data[0] = 0x00;
        frame.data[1] = 0x11;
        frame.data[2] = 0x22;
        frame.data[3] = 0x33;
        frame.data[4] = 0x44;
        frame.data[5] = 0x55;
        frame.data[6] = 0x66;
        frame.data[7] = 0x77;

        write(s, &frame, sizeof(struct can_frame));
        printf("Wrote data[0]:%2x,data[1]:%2x\n",frame.data[0],frame.data[1]);
        return 0;
}
```

CANbus Read sample code:

```
#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,i;
        int nbytes;
        struct sockaddr_can addr;
        struct can_frame frame;
        struct ifreq ifr;
          int xBitRate=1000000;

        char *ifname = "can0";

        if((s = socket(PF_CAN, SOCK_RAW, CAN_RAW)) < 0) {
```

```
                perror("Error while opening socket");
                return -1;
        }

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

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

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

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

          read(s, &frame, sizeof(struct can_frame));
          printf("Read ");
          for (i=0;i<8;i++)
          {
                  printf("[%d]:%2x ",i,frame.data[i]);
          }
          printf("\n");
        return 0;
}
```

4.2 Compile Demo Program

4.2.1 Install IFB112 I/O Library

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.

1. Set up the cross-development environment on your host PC.

~$ source /opt/poky/1.8.1/environment-setup-cortexa7hf-vfp-neon-poky-linux-gnueabi

```
ryan@OMG:~$ source /opt/poky/1.8.1/environment-setup-cortexa7hf-vfp-neon-poky-li
nux-gnueabi
```

2. To compile and build demo program for IFB112, please do:

Change to *your project* directory.

~$ cd project/IFB112-LINUX-bsp-1.0.0/AxTools

```
ryan@OMG:~$ cd project/IFB112-LINUX-bsp-V.1.0.3/AxTools/
```

3. Extract driver source to *your project* directory.

~$ tar -xvf IFB112-rsb-lib-1.0.3.tar.bz2

```
ryan@OMG:~/project/IFB112-LINUX-bsp-V.1.0.3/AxTools$ tar -xvf IFB112-rsb-lib-1.0
.3.tar.bz2
rsb_lib/demo/can_read.c
rsb_lib/demo/ledtest
rsb_lib/demo/commode.c
rsb_lib/demo/ledtest.c
rsb_lib/demo/canmode.c
rsb_lib/demo/Makefile
rsb_lib/librsb10x.h
rsb_lib/librsb10x.so.0
rsb_lib/demo/relay.c
rsb_lib/demo/can_write.c
rsb_lib/demo/librsb10x.h
rsb_lib/demo/serial.h
rsb_lib/demo/diotest.c
rsb_lib/demo/wdttest.c
rsb_lib/demo/
rsb_lib/librsb10x.so.1.0.2
rsb_lib/demo/canmode
rsb_lib/demo/open_comport
rsb_lib/demo/commode
rsb_lib/demo/diotest
rsb_lib/demo/can_read
rsb_lib/demo/can_write
rsb_lib/
rsb_lib/demo/open_comport.c
rsb_lib/demo/wdttest
rsb_lib/librsb10x.so
rsb_lib/demo/relay
```

4. Change to *rsb_lib/demo* directory.

~$ cd ~/project/rsb_lib/demo

```
ryan@OMG:~/project/IFB112-LINUX-bsp-V.1.0.3/AxTools$ cd rsb_lib/demo/
```

5. Build the demo program.

~$ make

```
ryan@OMG:~/project/IFB112-LINUX-bsp-V.1.0.3/AxTools/rsb_lib/demo$ make
arm-poky-linux-gnueabi-gcc  -march=armv7-a -mfloat-abi=hard -mfpu=neon -mtune=co
rtex-a7 --sysroot=/opt/poky/1.8.1/sysroots/cortexa7hf-vfp-neon-poky-linux-gnueab
i  -o commode commode.c -lrsb10x -L../
arm-poky-linux-gnueabi-gcc  -march=armv7-a -mfloat-abi=hard -mfpu=neon -mtune=co
rtex-a7 --sysroot=/opt/poky/1.8.1/sysroots/cortexa7hf-vfp-neon-poky-linux-gnueab
i  -o diotest diotest.c -lrsb10x -L../
arm-poky-linux-gnueabi-gcc  -march=armv7-a -mfloat-abi=hard -mfpu=neon -mtune=co
rtex-a7 --sysroot=/opt/poky/1.8.1/sysroots/cortexa7hf-vfp-neon-poky-linux-gnueab
i  -o open_comport open_comport.c -lrsb10x -L../
arm-poky-linux-gnueabi-gcc  -march=armv7-a -mfloat-abi=hard -mfpu=neon -mtune=co
rtex-a7 --sysroot=/opt/poky/1.8.1/sysroots/cortexa7hf-vfp-neon-poky-linux-gnueab
i  -o relay relay.c -lrsb10x -L../
arm-poky-linux-gnueabi-gcc  -march=armv7-a -mfloat-abi=hard -mfpu=neon -mtune=co
rtex-a7 --sysroot=/opt/poky/1.8.1/sysroots/cortexa7hf-vfp-neon-poky-linux-gnueab
i  -o ledtest ledtest.c -lrsb10x -L../
arm-poky-linux-gnueabi-gcc  -march=armv7-a -mfloat-abi=hard -mfpu=neon -mtune=co
rtex-a7 --sysroot=/opt/poky/1.8.1/sysroots/cortexa7hf-vfp-neon-poky-linux-gnueab
i  -o wdttest wdttest.c -lrsb10x -L../
arm-poky-linux-gnueabi-gcc  -march=armv7-a -mfloat-abi=hard -mfpu=neon -mtune=co
rtex-a7 --sysroot=/opt/poky/1.8.1/sysroots/cortexa7hf-vfp-neon-poky-linux-gnueab
i  -o canmode canmode.c -lrsb10x -L../
arm-poky-linux-gnueabi-gcc  -march=armv7-a -mfloat-abi=hard -mfpu=neon -mtune=co
rtex-a7 --sysroot=/opt/poky/1.8.1/sysroots/cortexa7hf-vfp-neon-poky-linux-gnueab
i  -o can_read can_read.c -lrsb10x -L../
arm-poky-linux-gnueabi-gcc  -march=armv7-a -mfloat-abi=hard -mfpu=neon -mtune=co
rtex-a7 --sysroot=/opt/poky/1.8.1/sysroots/cortexa7hf-vfp-neon-poky-linux-gnueab
i  -o can_write can_write.c -lrsb10x -L../
```

6. Then you should have example programs such as open_comport, diotest, and commode.

```
ryan@OMG:~/project/IFB112-LINUX-bsp-V.1.0.3/AxTools/rsb_lib/demo$ ls
canmode     can_write    diotest    librsb10x.h     relay     wdttest.c
canmode.c   can_write.c  diotest.c  Makefile        relay.c
can_read    commode      ledtest    open_comport    serial.h
can_read.c  commode.c    ledtest.c  open_comport.c  wdttest
```

4.2.2 Run demo program

Refer to section 2.3 for detailed information.

Chapter 5
Board Support Package (BSP)

5.1 Host Development System Installation

5.1.1 Install Host System

1. Download Ubuntu 14.04 LTS iso image.

2. Install Ubuntu 14.04.

3. Install host packages needed by Yocto development as follows:

```
~$sudo apt-get install wget git-core unzip texinfo libsdl1.2-dev gawk diffstat \
        wget git-core unzip texinfo libsdl1.2-dev gawk diffstat \
        texi2html docbook-utils python-pysqlite2 help2man \
        make gcc g++ desktop-file-utils libgl1-mesa-dev \
        libglu1-mesa-dev mercurial autoconf \
        automake groff curl lzop asciidoc xterm chrpath
```

   i.MX layers host packages for a Ubuntu 14.04 host setup only are:

```
~$ sudo apt-get install u-boot-tools
```

4. Install and configure TFTP server:
   After tftpd is installed, configure it by editing /etc/xinetd.d/tftp. Change the default export path (it is either */usr/var/tftpboot* or */var/lib/tftpboot*) to /. Or change the default export path to whatever directory you want to download from. Then reboot the hardware.

   To install tftpd / tftp/ xineted SOFTWARE

```
~$ sudo apt-get install tftpd tftp xinetd
```

   To create tftp directory

```
~$ sudo mkdir /tftpboot
~$ sudo chmod -R 777 /tftpboot
~$ sudo chown -R nobody /tftpboot
```

   To configure tftp sever.

```
~$ sudo vi /etc/xinetd.d/tftp
```

```
service tftp
{
        socket_type     = dgram
        protocol        = udp
        wait            = yes
        user            = root
        server          = /usr/sbin/in.tftpd
        server_args     = -s /tftpboot
        disable         = no
        per_source      = 11
        cps             = 100 2
        flags           = IPv4
}
```

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

```
louis@ubuntu:~$ mkdir ~/bin
louis@ubuntu:~$ curl http://commondatastorage.googleapis.com/git-repo-downloads/
repo > ~/bin/repo
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 26223  100 26223    0     0   3345      0  0:00:07  0:00:07 --:--:--  5815
louis@ubuntu:~$ chmod a+x ~/bin/repo
```

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

~$ export PATH=~/bin:$PATH

```
louis@ubuntu:~$ export PATH=~/bin:$PATH
```

2. Setting up the Git environment
~$ git config --global user.name "Your Name"
~$ git config --global user.email "Your Email"

```
louis@ubuntu:~$ git config --global user.name "axiomtek"
louis@ubuntu:~$ git config --global user.email "axio@axiomtek.com.tw"
```

3. Download the Freescale's Yocto BSP source
~$ mkdir project
~$ mkdir project/fsl-community-bsp
~$ cd project/fsl-community-bsp
~$ repo init -u git://git.freescale.com/imx/fsl-arm-yocto-bsp.git -b imx-3.14.52-1.1.0_ga

```
louis@ubuntu:~$ mkdir project/fsl-community-bsp
louis@ubuntu:~$ cd project/fsl-community-bsp/
louis@ubuntu:~/project/fsl-community-bsp$ repo init -u git://git.freescale.com/i
mx/fsl-arm-yocto-bsp.git -b imx-3.14.52-1.1.0_ga
Get https://gerrit.googlesource.com/git-repo/clone.bundle
Get https://gerrit.googlesource.com/git-repo
```

~$ repo sync

```
louis@ubuntu:~/project/fsl-community-bsp$ repo sync
Fetching project fsl-community-bsp-base
Fetching project meta-openembedded
remote: Counting objects: 215, done.
remote: Total 215 (delta 0), reused 0 (delta 0), pack-reused 215
Receiving objects: 100% (215/215), 46.16 KiB | 0 bytes/s, done.
Resolving deltas: 100% (114/114), done.
From git://github.com/Freescale/fsl-community-bsp-base
 * [new branch]      daisy      -> freescale/daisy
 * [new branch]      danny      -> freescale/danny
```

Clone Finish

```
 * [new tag]         yocto-2.0   -> yocto-2.0
 * [new tag]         yocto-2.0.1 -> yocto-2.0.1
 * [new tag]         yocto-2.0.2 -> yocto-2.0.2
 * [new tag]         yocto-2.1   -> yocto-2.1
 * [new tag]         yocto_1.5_M5.rc8 -> yocto_1.5_M5.rc8
Fetching projects: 100% (9/9), done.
Syncing work tree: 100% (9/9), done.

louis@ubuntu:~/project/fsl-community-bsp$
```

4. Extract Axiomtek's Yocto BSP source

~$ tar -xvf ../IFB112-LINUX-bsp-1.0.0/Yocto\ patches/meta-axiomtek-2.5.3.tar.gz -C sources

```
ryan@OMG:~/project/fsl-community-bsp$ tar -xvf ../IFB112-LINUX-bsp-V.1.0.3/Yocto
\ patches/IFB112-meta-axiomtek-2.5.3.tar.gz -C sources
```

Check meta-axiomtek

```
louis@ubuntu:~/project/fsl-community-bsp/sources$ ls
base          meta-browser  meta-fsl-arm-extra    meta-fsl-demos      meta-qt5
meta-axiomtek meta-fsl-arm  meta-fsl-bsp-release  meta-openembedded   poky
louis@ubuntu:~/project/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 \

```
LCONF_VERSION = "6"

BBPATH = "${TOPDIR}"
BSPDIR := "${@os.path.abspath(os.path.dirname(d.getVar('FILE', True)) + '/../..$

BBFILES ?= ""
BBLAYERS = " \
  ${BSPDIR}/sources/poky/meta \
  ${BSPDIR}/sources/poky/meta-yocto \
  \
  ${BSPDIR}/sources/meta-openembedded/meta-oe \
  ${BSPDIR}/sources/meta-openembedded/meta-multimedia \
  \
  ${BSPDIR}/sources/meta-fsl-arm \
  ${BSPDIR}/sources/meta-fsl-arm-extra \
  ${BSPDIR}/sources/meta-fsl-demos \
  ${BSPDIR}/sources/meta-axiomtek \
"
```

6. First build

Choose your board

~$ DISTRO=poky MACHINE=rsb102 EULA=1 source fsl-setup-release.sh -b build

```
Common targets are:
    core-image-minimal
    meta-toolchain
    meta-toolchain-sdk
    adt-installer
    meta-ide-support

Your build environment has been configured with:

    MACHINE=rsb102
    SDKMACHINE=i686
    DISTRO=poky
    EULA=1
BSPDIR=
BUILD_DIR=.
```

Start to build image

~$ bitbake axl-image-base

```
louis@ubuntu:~/project/fsl-community-bsp/build$ bitbake axl-image-base
Parsing recipes:   7% |###                                      | ETA:  00:02:32
```

7. After build image finish, you can find the file path.

The file path: project/fsl-community-bsp/build/tmp/deploy/images/rsb101

```
ryan@OMG:~/project/fsl-community-bsp/build/tmp/deploy/images/rsb102$ ls
axl-image-base-rsb102-20170606071235.rootfs.manifest
axl-image-base-rsb102-20170606071235.rootfs.tar.gz
axl-image-base-rsb102.manifest
axl-image-base-rsb102.tar.gz
modules--3.14.52-r0-rsb102-20170606071235.tgz
modules-rsb102.tgz
README_-_DO_NOT_DELETE_FILES_IN_THIS_DIRECTORY.txt
zImage
zImage--3.14.52-r0-ax-rsb-imx6ul-ifb112-20170606071235.dtb
zImage--3.14.52-r0-rsb102-20170606071235.bin
zImage-ax-rsb-imx6ul-ifb112.dtb
zImage-rsb102.bin
```

5.1.3 Build and Install user's Yocto Toolchain

We have provided Yocto Toolchain in IFB112 BSP. However, if you want to build your toolchain by Yocto development, you can follow the instructions on host PC:

1. Change to *Yocto development* directory.
   ~$ source setup-environment build

```
ryan@OMG:~/project/fsl-community-bsp$ source setup-environment build

Welcome to Freescale Community BSP

The Yocto Project has extensive documentation about OE including a
reference manual which can be found at:
    http://yoctoproject.org/documentation

For more information about OpenEmbedded see their website:
    http://www.openembedded.org/

You can now run 'bitbake <target>'

Common targets are:
    core-image-minimal
    meta-toolchain
    meta-toolchain-sdk
    adt-installer
    meta-ide-support

Your configuration files at build have not been touched.
```

   ~$ bitbake meta-toolchain

```
louis@ubuntu:~/project/fsl-community-bsp/build$ bitbake meta-toolchain
Parsing recipes:  86% |###################################      | ETA:  00:00:23
```

2. After these steps to generate the toolchain into the Build Directory, you can find the file path: project/fsl-community-bsp/build/tmp/deploy/sdk

   Install the toolchain into your host system /opt directory.
   Note: It needs root authorization
   ~$bash poky-glibc-x86_64-meta-toolchain-cortexa7hf-vfp-neon-toolchain-1.8.1.sh

```
ryan@Ubuntu:~/project/IFB112-Linux-bsp-1.0.0/IFB112-Linux-bsp-1.0.0/Toolchain/64-bit$ ls
poky-glibc-x86_64-meta-toolchain-cortexa7hf-vfp-neon-toolchain-1.8.1.sh
ryan@Ubuntu:~/project/IFB112-Linux-bsp-1.0.0/IFB112-Linux-bsp-1.0.0/Toolchain/64-bit$ bash poky-glibc-x86_
64-meta-toolchain-cortexa7hf-vfp-neon-toolchain-1.8.1.sh
Enter target directory for SDK (default: /opt/poky/1.8.1):
You are about to install the SDK to "/opt/poky/1.8.1". Proceed[Y/n]?y
Extracting SDK...done
Setting it up...done
SDK has been successfully set up and is ready to be used.
```

5.2 U-Boot for IFB112

5.2.1 Booting the System with an NFS Filesystem

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

```
U-Boot 2015.04-imx_v2015.04_3.14.52_1.1.0_ga (Jul 14 2016 - 19:30:14)

CPU:   Freescale i.MX6UL rev1.1 at 396 MHz
CPU:   Temperature 51 C
Reset cause: WDOG
Board: RSB10X
I2C:   ready
DRAM:  256 MiB
PMIC:  PFUZE300 DEV_ID=0x30 REV_ID=0x11
MMC:   FSL_SDHC: 0, FSL_SDHC: 1
In:    serial
Out:   serial
Err:   serial
switch to partitions #0, OK
mmc1(part 0) is current device
Net:   FEC0
Normal Boot
Hit any key to stop autoboot:  0
=> setenv ethaddr 00:60:e0:00:00:0a
=> setenv serverip 192.168.1.101
=> setenv ipaddr 192.168.1.103
=> run netboot
Booting from net ...
```

Setup TFTP server IP:
=> setenv serverip 192.168.1.101

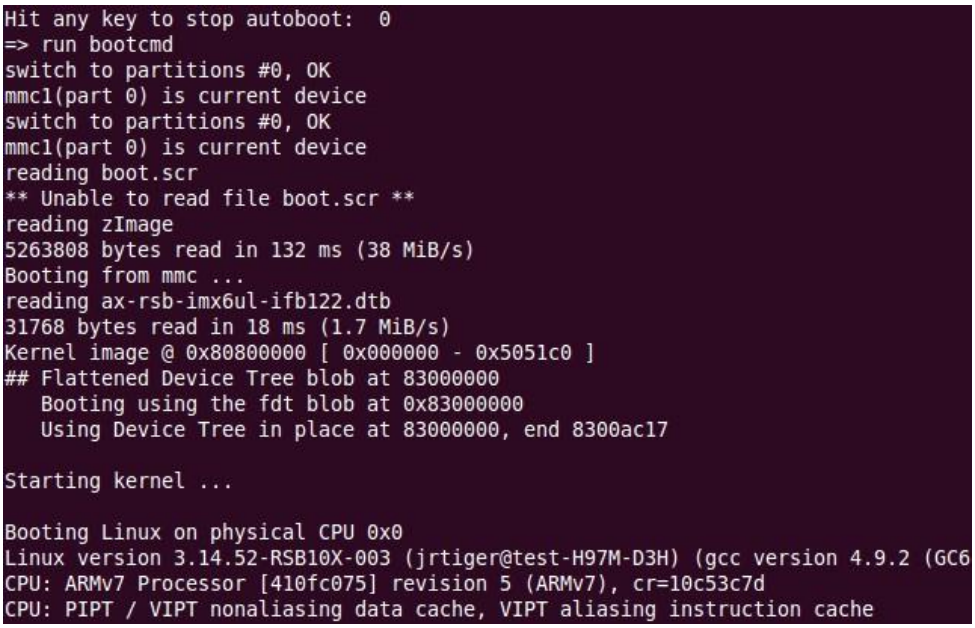
Setup board IP address:
=> setenv ipaddr 192.168.1.103

Run boot from NFS server:
=> run net_boot

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

5.2.2 Booting the System from eMMC (IFB112 default)

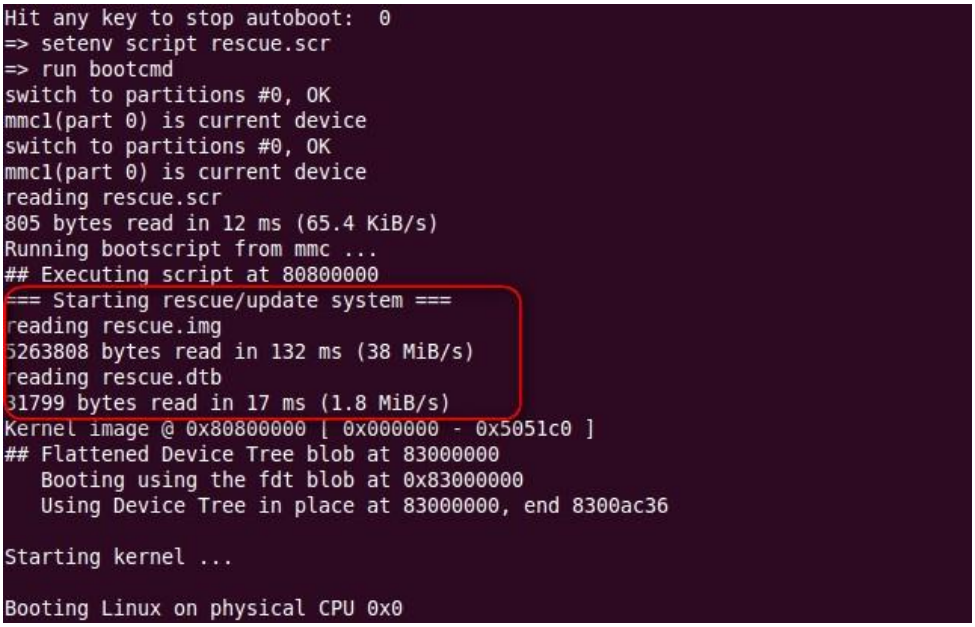
=> run bootcmd



5.2.3 Booting the Rescue System from eMMC

If the Embedded Linux system was broken to boot, you can recovery Linux system on u-boot through rescue mode.

=> setenv script rescue.scr
=> run bootcmd

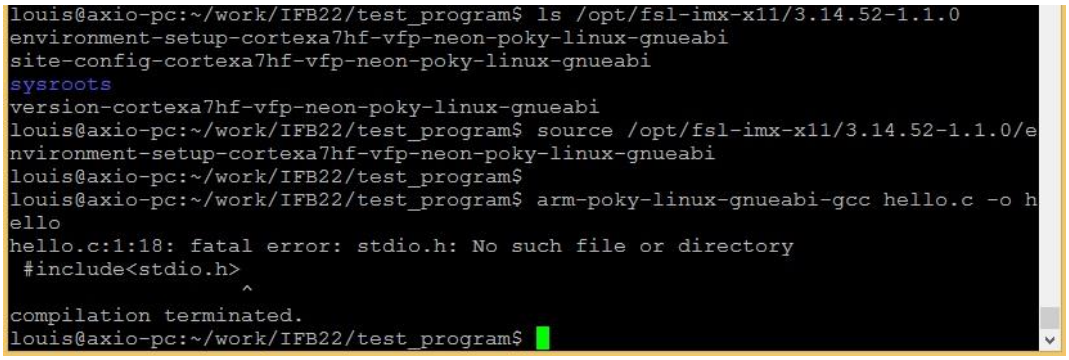


This page is intentionally left blank.

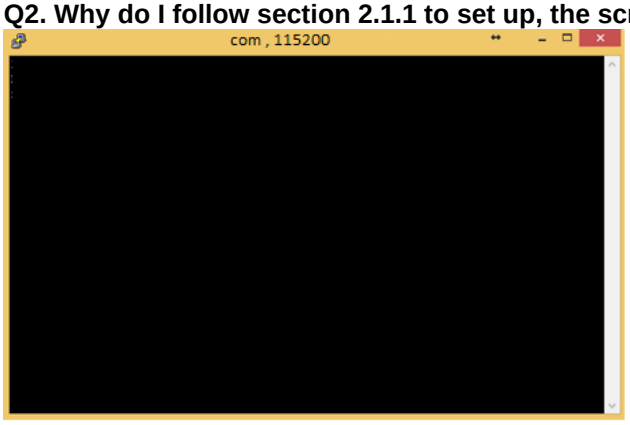
Appendix
Frequently Asked Questions

Q1. When I use toolchain to compile, I can't find " include" file.

A1: Refer to section 2.3 for detailed information 2.2.2 Setting up the Cross-Development Environment
For example: $CC hello.c -o hello



Q2. Why do I follow section 2.1.1 to set up, the screen is shown as below?



A2. Please follow steps as below

1. To check your power.
2. To check serial item "COM port" name and Device Manager "COM port" name are both the same as below.

3. To check if your RS232 port jumper.

Q3. Why can't transfer the file to FTP、TFTP、NFS after following the instructions, or disconnect.

A3: To check your firewall been blocked in your host PC or router.