IA3341 Linux User's Manual

First Edition, May 2010

www.moxa.com/product

IA3341 Linux User's Manual

Copyright Notice

Trademarks

Disclaimer

Information in this document is subject to change without notice and does not represent a commitment on the part of Moxa.

Moxa provides this document "as is," without warranty of any kind, either expressed or implied, including, but not limited to, its particular purpose. Moxa reserves the right to make improvements and/or changes to this manual, or to the products and/or the programs described in this manual, at any time.

Information provided in this manual is intended to be accurate and reliable. However, Moxa assumes no responsibility for its use, or for any infringements on the rights of third parties that may result from its use.

This product might include unintentional technical or typographical errors. Changes are periodically made to the information herein to correct such errors, and these changes are incorporated into new editions of the publication.

Technical Support Contact Information
www.moxa.com/support

Moxa Americas:
Toll-free: 1-888-669-2872
Tel: +1-714-528-6777
Fax: +1-714-528-6778

Moxa China (Shanghai office):
Toll-free: 800-820-5036
Tel: +86-21-5258-9955
Fax: +86-10-6872-3958

Moxa Europe:
Tel: +49-89-3 70 03 99-0
Fax: +49-89-3 70 03 99-99

Moxa Asia-Pacific:
Tel: +886-2-8919-1230
Fax: +886-2-8919-1231

Table of Contents

1

Introduction

The IA3341 is based on the MOXA ART ARM9 industrial processor, and features 2 RS-232/422/485 serial ports, dual LANs, 4 digital input channels, and 4 digital output channels. In addition, the IA3341 computer has 2 analog input channels and 2 thermocouple channels, making it the ideal solution for a variety of industrial applications, such as solar power and environmental monitoring.

The industrial-grade design of the IA3341 provides a robust, reliable computer that can fit any industrial environment, and the open source Linux platform gives programmers a convenient tool for developing sophisticated, bug-free application software at a lower cost. In addition, the built-in Modbus TCP library and web server help users easily monitor and retrieve the AI data, which is particularly useful for solar power and environmental monitoring applications.

The following topics are covered in this chapter:

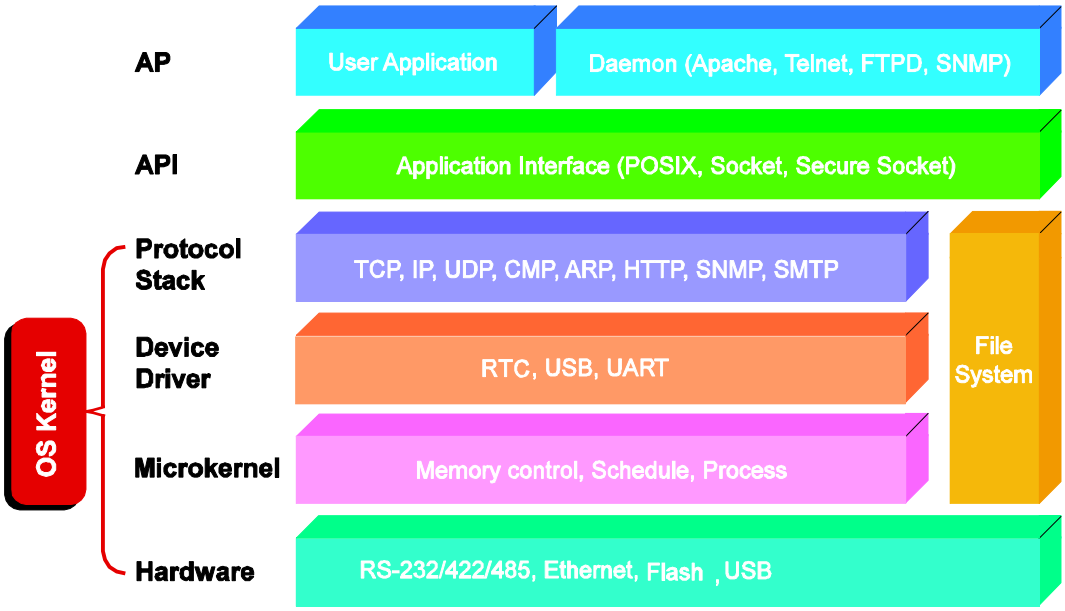
- **Overview**
- **Software Architecture**
  - Journaling Flash File System (JFFS2)
  - Software

Overview

The pre-installed Linux operating system (OS) provides an open software operating system for your software program development. Software written for desktop PCs can be easily ported to the computer with a GNU cross compiler, without needing to modify the source code. The OS, device drivers (e.g., serial and buzzer control), and your own applications, can all be stored in the NOR Flash.

Software Architecture

The Linux operating system that is pre-installed in the IA3341 follows the standard Linux architecture, making it easy to accept programs that follow the POSIX standard. Program porting is done with the GNU Tool Chain provided by Moxa. In addition to Standard POSIX APIs, device drivers for the USB storage, buzzer and Network controls, and UART are also included with the Linux OS.



The IA3341's built-in Flash ROM is partitioned into **Boot Loader**, **Linux Kernel**, **Root File System**, and **User directory** partitions.

In order to prevent user applications from crashing the Root File System, the IA3341 comptuers use a specially designed **Root File System with Protected Configuration** for emergency use. This **Root File System** comes with serial and Ethernet communication capability for users to load the **Factory Default Image** file. The user directory saves the user's settings and application.

To improve system reliability, the IA3341 has a built-in mechanism that prevents the system from crashing. When the Linux kernel boots up, the kernel will mount the root file system for read only, and then enable services and daemons. During this time, the kernel will start searching for system configuration parameters with ***rc*** or ***inittab***.

Normally, the kernel uses the Root File System to boot up the system. The Root File System is protected, and cannot be changed by the user. This type of setup creates a "safe" zone.

For more information about the memory map and programming, refer to Chapter 6, *Programmer's Guide*.

Journaling Flash File System (JFFS2)

The Root File System and User directory in the flash memory is formatted with the **Journaling Flash File System version 2 (JFFS2)**. The formatting process places a compressed file system in the flash memory. This operation is transparent to the user.

The Journaling Flash File System (JFFS2), which was developed by Axis Communications in Sweden, puts a file system directly on the flash, instead of emulating a block device. It is designed for use on flash-ROM chips and recognizes the special write requirements of a flash-ROM chip. JFFS2 implements wear-leveling to extend the life of the flash disk, and stores the flash directory structure in the RAM. A log-structured file system is maintained at all times. The system is always consistent, even if it encounters crashes or improper power-downs, and does not require fsck (file system check) on boot-up.

JFFS2 is the newest version of JFFS. It provides improved wear-leveling and garbage-collection performance, improved RAM footprint and response to system-memory pressure, improved concurrency and support for suspending flash erases, marking of bad sectors with continued use of the remaining good sectors (enhancing the write-life of the devices), native data compression inside the file system design, and support for hard links.

The key features of JFFS2 are:

- Targets the Flash ROM Directly
- Robustness
- Consistency across power failures
- No integrity scan (fsck) is required at boot time after normal or abnormal shutdown
- Explicit wear leveling
- Transparent compression

Although JFFS2 is a journaling file system, this does not preclude the loss of data. The file system will remain in a consistent state across power failures and will always be mountable. However, if the board is powered down during a write then the incomplete write will be rolled back on the next boot, but writes that have already been completed will not be affected.

Additional information about JFFS2 is available at:

http://sources.redhat.com/jffs2/jffs2.pdf
http://developer.axis.com/software/jffs/
http://www.linux-mtd.infradead.org/

Software Specifications

Boot Loader	Moxa private (V1.2)
Kernel	Linux 2.6.9
Protocol Stack	ARP, PPP, CHAP, PAP, IPv4, ICMP, TCP, UDP, DHCP, FTP, SNMP V1/V3, HTTP, NTP, NFS, SMTP, SSH 1.0/2.0, SSL, Telnet, PPPoE, OpenVPN
File System	JFFS2, NFS, Ext2, Ext3, VFAT/FAT
OS shell command	Bash
Busybox	Linux normal command utility collection
Utilities	
tinylogin	login and user manager utility
telnet	telnet client program
ftp	FTP client program
smtpclient	email utility
scp	Secure file transfer Client Program
Daemons	
pppd	dial in/out over serial port daemon
snmpd	snmpd agent daemon
telnetd	telnet server daemon
inetd	TCP server manager program
ftpd	ftp server daemon
apache	web server daemon
sshd	secure shell server
openvpn	virtual private network
openssl	open SSL
Linux Tool Chain	
Gcc (V3.3.2)	C/C++ PC Cross Compiler
GDB (V5.3)	Source Level Debug Server
Glibc (V2.2.5)	POSIX standard C library

2

Getting Started

In this chapter, we explain how to connect the IA3341, how to turn on the power, how to get started programming, and how to use the IA3341's other functions.

The following topics are covered in this chapter:

- **Powering on the IA3341**
- **Connecting the IA3341 to a PC**
  - Serial Console
  - SSH Console
- **Configuring the Ethernet Interface**
  - Modifying Network Settings with the Serial Console
  - Modifying Network Settings over the Network
- **SD Socket and USB for Storage Expansion**
- **Test Program—Developing Hello.c**
  - Installing the Tool Chain (Linux)
  - Checking the Flash Memory Space
  - Compiling Hello.c
  - Uploading and Running the "Hello" Program
- **Developing Your First Application**
  - Testing Environment
  - Compiling tcps2.c
  - Uploading and Running the "tcps2-release" Program
  - Summary of the Testing Procedure

Powering on the IA3341

Connect the SG wire to the shielded contact located in the upper left corner of the IA3341, and then power on the computer by connecting it to the power adaptor. It takes about 30 to 60 seconds for the system to boot up. Once the system is ready, the Ready LED will light up.

NOTE After connecting the IA3341 to the power supply, it will take about 30 to 60 seconds for the operating system to boot up. The green Ready LED will not turn on until the operating system is ready.

ATTENTION

This product is intended to be supplied by a Listed Power Unit and output marked with "LPS" and rated 12-48 VDC, 580 mA (minimum requirements).

Connecting the IA3341 to a PC

There are two ways to connect the IA3341 to a PC: through the serial console port or by Telnet over the network.

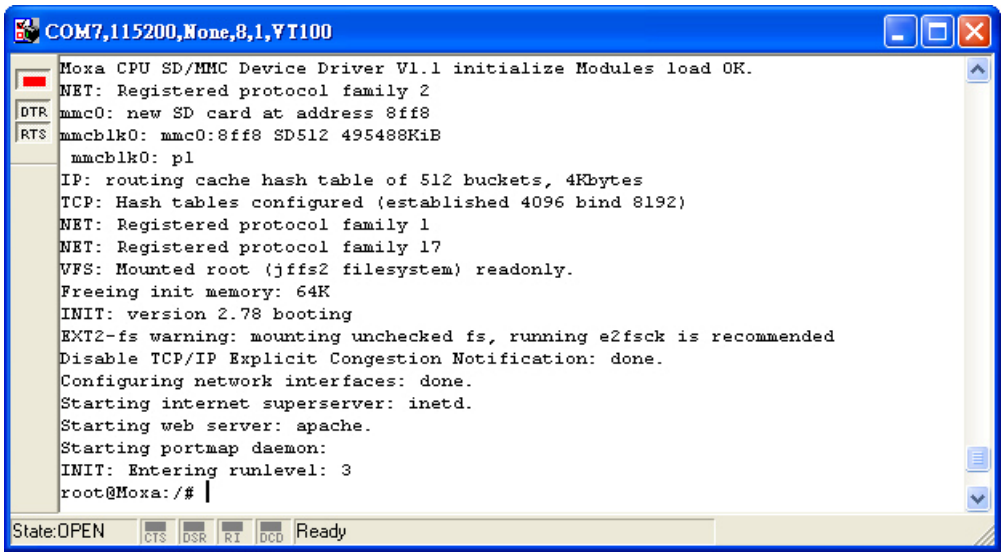
Serial Console

The serial console gives users a convenient way of connecting to the IA3341. This method is particularly useful when using the computer for the first time. The serial console is useful for connecting the IA3341 when you do not know either of the two IP addresses.

Use the serial console port settings shown below.

Baudrate	115200 bps
Parity	None
Data bits	8
Stop bit	1
Flow Control	None
Terminal	VT100

Once the connection is established, the following window will open.



ATTENTION

Serial Console Reminder

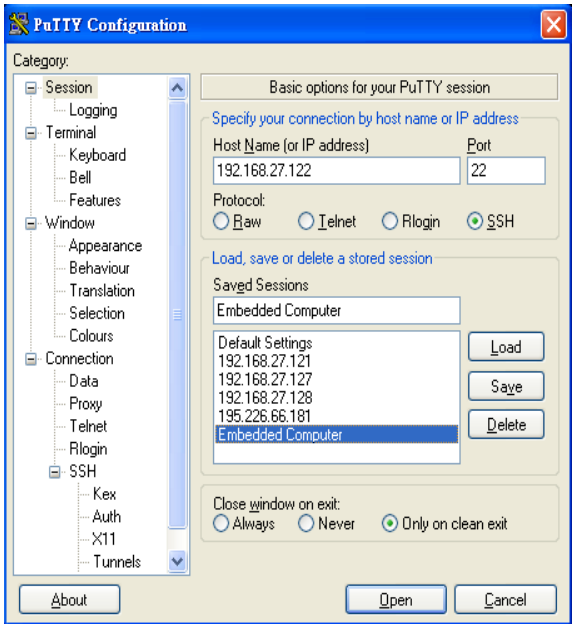
Remember to choose VT100 as the terminal type. Use the cable CBL-4PINDB9F-100, which comes with the IA3341, to connect to the serial console port.

SSH Console

The IA3341 supports an SSH Console to provide users with better security options.

Windows Users

Click on the link http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html to download PuTTY (free software) to set up an SSH console for the IA3341 in a Windows environment. The following figure shows a simple example of the configuration that is required.



Linux Users

From a Linux machine, use the "ssh" command to access the IA3341's console utility via SSH.

#ssh 192.168.3.127

Select **yes** to complete the connection.

```
 [root@localhost root]# ssh 192.168.4.127
The authenticity of host '192.168.4.127 (192.168.3.127)' can't be established.
RSA key fingerprint is 8b:ee:ff:84:41:25:fc:cd:2a:f2:92:8f:cb:1f:6b:2f.
Are you sure you want to continue connection (yes/no)? yes
```

NOTE SSH provides better security compared to Telnet for accessing the IA3341's console utility over the network.

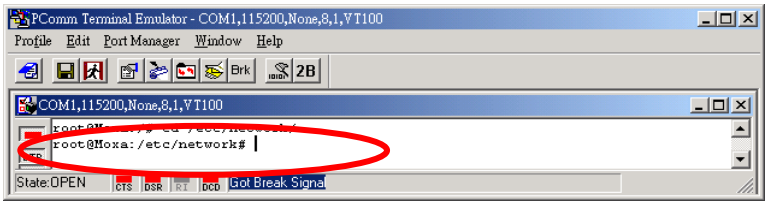
Configuring the Ethernet Interface

The network settings of the IA3341 can be modified with the serial console, or online over the network.

Modifying Network Settings with the Serial Console

In this section, we use the serial console to configure the network settings of the target computer.

1. Follow the instructions given in a previous section to access the Console Utility of the target computer through the serial console port, and then type **`#cd /etc/network`** to change directories.



2. Type **`#vi interfaces`** to use vi editor to edit the network configuration file. You can configure the Ethernet ports of the IA3341 for **static** or **dynamic** (DHCP) IP addresses.

   Static IP addresses:

   As shown below, 4 network addresses must be modified: **address**, **network**, **netmask**, and **broadcast**. The default IP address for LAN1 is DHCP, and for LAN 2 is 192.168.4.127, both with default netmask of 255.255.255.0.

   Dynamic IP addresses:

   The default setting for LAN1 is DHCP. To request an IP address dynamcially, replace dhcp with static and then delete the address, network, netmask, and broadcast items.

Dynamic Setting Using DHCP	Default Settings for LAN 2
iface eth0 inet **dhcp**	iface eth0 inet **static** address 192.168.4.127 network: 192.168.4.0 netmask 255.255.255.0 broadcast 192.168.4.255

3. After the boot settings of the LAN interface have been modified, issue the following command to activate the LAN settings immediately:

   `#/etc/init.d/networking restart`

NOTE After changing the IP settings, use the **networking restart** command to activate the new IP address.

Modifying Network Settings over the Network

IP settings can be activated over the network, but the new settings will not be saved to the flash ROM without modifying the **file /etc/network/interfaces**.

For example, type the command **#ifconfig eth0 192.168. 27.125** to change the IP address of LAN to 192.168. 27.125.

```
root@Moxa:~# ifconfig eth0 192.168.27.125
root@Moxa:~#
```

SD Socket and USB for Storage Expansion

The IA3341 has an SD socket for storage expansion. The SD slot allows users to plug in a Secure Digital (SD) memory card compliant with the SD 1.0 standard for up to 1 GB of additional memory space, or a Secure Digital High Capacity (SDHC) memory card compliant with the SD 2.0 standard for up to 16 GB of additional memory space. Please refer to the IA3341 Hardware User's Manual to see how to install the SD card.

After installing an SD card, the SD card will be mounted at **/mnt/sd**.

In addition to the SD socket, a USB 2.0 host is located on the front panel. The USB host is also designed for storage expansion. To expand the amount of storage with a USB flash disk, you just need to plug the USB flash disk into this USB port. The flash disk will be detected automatically, and its file partition will be mounted into the OS. The USB storage device will be mounted in one of the following four directories: **/mnt/usbstorage1**, **/mnt/usbstorage2**, **/mnt/usbstorage3**, or **/mnt/usbstorage4**.

Test Program—Developing Hello.c

In this section, we use the standard "Hello" programming example to illustrate how to develop a program for the IA3341. In general, program development involves the following seven steps.

Step 1:
Connect the IA3341 to a Linux PC.
Step 2:
Install Tool Chain (GNU ross Compiler & glibc).
Step 3:
Set the cross compiler and glibc environment variables.
Step 4:
Code and compile the program.
Step 5:
Download the program to the IA3341 via FTP or NFS.
Step 6:
Debug the program
→ If bugs are found, return to Step 4.
→ If no bugs are found, continue with Step 7.
Step 7:
Back up the user directory (distribute the program to additional IA3341 units if needed).

Installing the Tool Chain (Linux)

The Linux Operating System must be pre-installed in the PC before installing the IA3341 GNU Tool Chain. Fedora core or compatible versions are recommended. The Tool Chain requires approximately200 MB of hard disk space on your PC. The IA3341 Tool Chain software is located on the IA3341 CD. To install the Tool Chain, insert the CD in your PC and then issue the following commands:

#mount /dev/cdrom /mnt/cdrom
#sh /mnt/cdrom/tool-chain/linux/install.sh

The Tool Chain will be installed automatically on your Linux PC within a few minutes. Before compiling the program, be sure to set the following path first, since the Tool Chain files, including the compiler, link, library, and include files are located in this directory.

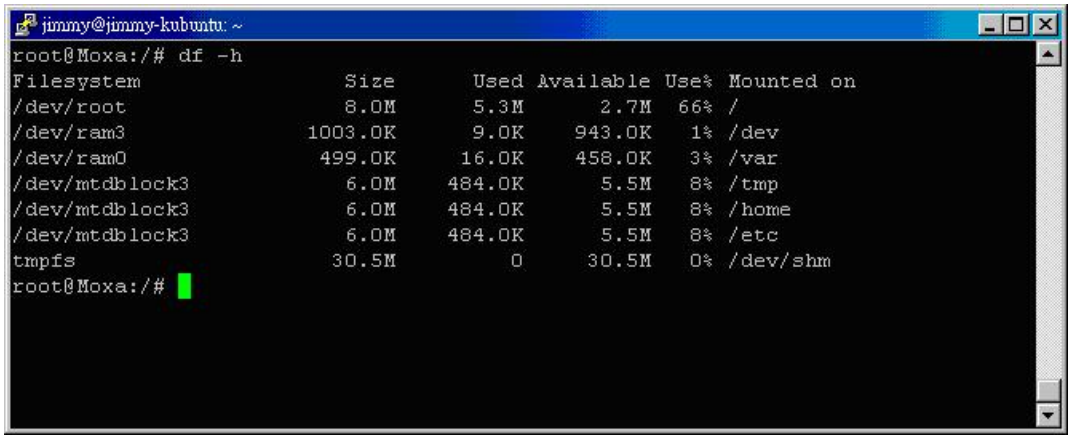
PATH=/usr/local/arm-linux/bin:$PATH

Setting the path allows you to run the compiler from any directory.

Checking the Flash Memory Space

If the flash memory is full, you will not be able to save data to the Flash ROM. Use the following command to calculate the amount of "Available" flash memory:

/>df –h



If there isn't enough "Available" space for your application, you will need to delete some existing files. To do this, connect your PC to the IA3341 with the console cable, and then use the console utility to delete the files from the IA3341's flash memory. To check the amount of free space available, look at the directories in the read/write directory **/dev/mtdblock3**. Note that the directories /**home** and /**etc** are both mounted in the directory **/dev/mtdblock3**.

NOTE If the flash memory is full, you will need to free up some memory space before saving files to the Flash ROM.

You can ONLY write files in **/home, /tmp, /etc, /var** directories. Files in **/var** directory will not be kept after reboot. Users are not allowed to have write privilege in other directories, including the directory **/root**.

Compiling Hello.c

The package CD contains several example programs. Here we use **Hello.c** as an example to show you how to compile and run your applications. Type the following commands from your PC to copy the files used for this example from the CD to your computer's hard drive:

 # cd /tmp/
 # mkdir example
 # cp –r /mnt/cdrom/example/* /tmp/example

To compile the program, go to the Hello subdirectory and issue the following commands:

#cd example/hello
#make

You should receive the following response:

```
[root@localhost hello]# make
/usr/local/arm-linux/bin/arm-linux-gcc -o hello-release hello.c
/usr/local/arm-linux/bin/arm-linux-strip -s hello-release
/usr/local/arm-linux/bin/arm-linux-gcc -ggdb -o hello-debug hello.c
[root@localhost hello]#
```

Next, execute hello.exe to generate **hello-release** and **hello-debug**, which are described below:

hello-release—an ARM platform execution file (created specifically to run on the IA3341)

hello-debug—an ARM platform GDB debug server execution file (see Chapter 5 for details about the GDB debug tool).

NOTE Since Moxa's tool chain places a specially designed **Makefile** in the directory **/tmp/example/hello**, be sure to type the **#make** command from within that directory. This special Makefile uses the arm-linux-gcc compiler to compile the hello.c source code for the Xscale environment. If you type the **#make** command from within any other directory, Linux will use the x86 compiler (for example, cc or gcc).

Refer to Chapter 5 to see a Makefile example.

Uploading and Running the "Hello" Program

Use the following commands to upload **hello-release** to the IA3341 via FTP.

1. From the PC, type:

   ```
   #ftp 192.168.3.127
   ```

2. Use the bin command to set the transfer mode to Binary mode, and then use the put command to initiate the file transfer:

   ```
   ftp> bin
   ftp> put hello-release
   ```

3. From the IA3341, type:

   ```
   # chmod +x hello-release
   # ./hello-release
   ```

The word **Hello** will be printed on the screen.

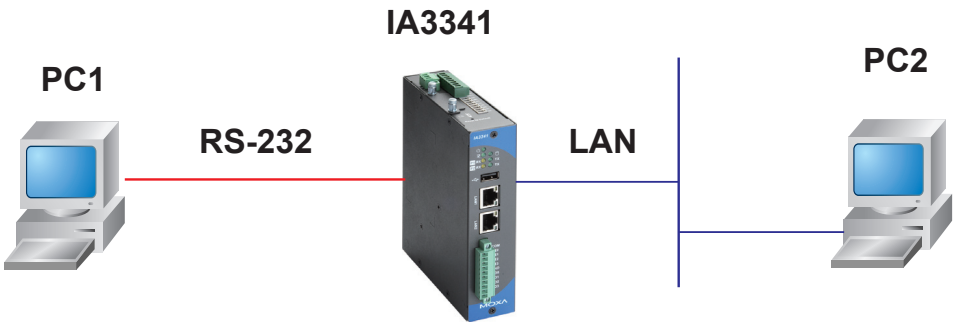
```
root@Moxa:~# ./hello-release
Hello
```

Developing Your First Application

We use the tcps2 example to illustrate how to build an application. The procedure outlined in the following subsections will show you how to build a TCP server program plus serial port communication that runs on the IA3341.

Testing Environment

The tcps2 example demonstrates a simple application program that delivers transparent, bi-directional data transmission between the IA3341's serial and Ethernet ports. As illustrated in the following figure, the purpose of this application is to transfer data between PC 1 and the IA3341 through an RS-232 connection. At the remote site, data can be transferred between the IA3341's Ethernet port and PC 2 over an Ethernet connection.



Compiling tcps2.c

The source code for the tcps2 example is located on the CD-ROM at **CD-ROM://example/TCPServer2/tcps2.c.** Use the following commands to copy the file to a specific directory on your PC. We use the direrctory **/home/ ia3341/1st_application/**. Note that you need to copy 3 files—Makefile, tcps2.c, tcpsp.c—from the CD-ROM to the target directory.

#mount –t iso9660 /dev/cdrom /mnt/cdrom
#cp /mnt/cdrom/example/TCPServer2/tcps2.c /home/IA3341/1st_application/tcps2.c
#cp /mnt/cdrom/example/TCPServer2/tcpsp.c /home/ IA3341/1st_application/tcpsp.c
#cp /mnt/cdrom/example/TCPServer2/Makefile /home/ IA3341/1st_application/Makefile

Type **#make** to compile the example code:

You will get the following response, indicating that the example program was compiled successfully.

```
root@server11:/home/IA3341-LX/1st_application
[root@server11 1st application]# pwd
/home/IA3341/1st application
[root@server11 1st application]# 11
total 20
-rw-r--r--  1 root root  514 Nov 27 11:52 Makefile
-rw-r--r--  1 root root 4554 Nov 27 11:52 tcps2.c
-rw-r--r--  1 root root 6164 Nov 27 11:55 tcps2.c
[root@server11 1st application]# make
/usr/local/arm-linux/bin/arm-linux-gcc -o tcps2-release tcps2.c
/usr/local/arm-linux/bin/arm-linux-strip -s tcps2-release
/usr/local/arm-linux/bin/arm-linux-gcc -o tcpsp-release tcpsp.c
/usr/local/arm-linux/bin/arm-linux-strip -s tcpsp-release
/usr/local/arm-linux/bin/arm-linux-gcc -ggdb -o tcps2-debug tcps2.c
/usr/local/arm-linux/bin/arm-linux-gcc -ggdb -o tcpsp-debug tcpsp.c
[root@server11 1st application]# 11
total 92
-rw-r--r--  1 root root   514  Nov 27 11:52 Makefile
-rwxr-xr-x  1 root root 25843 Nov 27 12:03 tcps2-debug
-rwxr-xr-x  1 root root  4996  Nov 27 12:03 tcps2-release
-rw-r--r--  1 root root  4554  Nov 27 11:52 tcps2.c
-rwxr-xr-x  1 root root 26823 Nov 27 12:03 tcpsp-debug
-rwxr-xr-x  1 root root  5396  Nov 27 12:03 tcpsp-release
-rw-r--r--  1 root root  6164  Nov 27 11:55 tcpsp.c
[root@server11 1st application]#
```

Two executable files, tcps2-release and tcps2-debug, are created.

tcps2-release—an ARM platform execution file (created specifically to run on the IA3341)

tcps2-debug—an ARM platform GDB debug server execution file (see Chapter 5 for details about the GDB debug tool).

NOTE If you get an error message at this point, it could be because you neglected to put tcps2.c and tcpsp.c in the same directory. The example Makefile we provide is set up to compile both tcps2 and tcpsp into the same project Makefile. Alternatively, you could modify the Makefile to suit your particular requirements.

Uploading and Running the "tcps2-release" Program

Use the following commands to use FTP to upload **tcps2-release** to the IA3341.

1. From the PC, type:

```
#ftp 192.168.3.127
```

2. Next, use the **bin** command to set the transfer mode to **Binary**, and the **put** command to initiate the file transfer:

```
ftp> bin
ftp> cd home
ftp> put tcps2-release
```

```
root@server11:/home/IA3341-LX/1st_application
[root@server11 1st application]# ftp 192.168.3.127
Connected to 192.168.3.127
220 Moxa FTP server (Version wu-2.6.1(2) Mon Nov 24 12:17:04 CST 2003) ready.
530 Please login with USER and PASS.
530 Please login with USER and PASS.
KERBEROS V4 rejected as an authentication type
Name (192.168.3.127:root): root
331 Password required for root.
Password:
230 User root logged in.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> bin
200 Type set to I.
ftp> put tcps2-release
local: tcps2-release remote: tcps2-release
277 Entering Passive Mode (192.168.3.127.82.253)
150 Opening BINARY mode data connection for tcps2-release.
226 Transfer complete
4996 bytes sent in 0.00013 seconds (3.9e+04 Kbytes/s)
ftp> ls
227 Entering Passive Mode (192.168.3.127.106.196)
150 Opening ASCII mode data connection for /bin/ls.
-rw-------    1 root     root          899 Jun 10 08:11  bash_history
-rw-r--r--    1 root     root         4996 Jun 12 02:15 tcps2-release
226 Transfer complete
ftp>
```

3. From the IA3341, type:

```
# chmod +x tcps2-release
# ./tcps2-release &
```

```
192.168.3.127 - PuTTY
root@Moxa:~# ls -al
drwxr-xr-x  2 root  root     0 Jun 12 02:14
drwxr-xr-x 15 root  root     0 Jan  1   1970
-rw-------  1 root  root   899 Jun 10 08:11 .bash_history
-rw-r--r--  1 root  root  4996 Jun 12 02:15 tcps2-release
root@Moxa:~# chmod +x tcps2-release
root@Moxa:~# ls -al
drwxr-xr-x  2 root  root     0 Jun 12 02:14
drwxr-xr-x 15 root  root     0 Jan  1   1970
-rw-------  1 root  root   899 Jun 10 08:11 .bash_history
-rwxr-xr-x  1 root  root  4996 Jun 12 02:15 tcps2-release
root@Moxa:~#
```

4. The program should start running in the background. Use the **#ps -ef** command to check if the tcps2 program is actually running in the background.

```
#ps  // use this command to check if the program is running
```

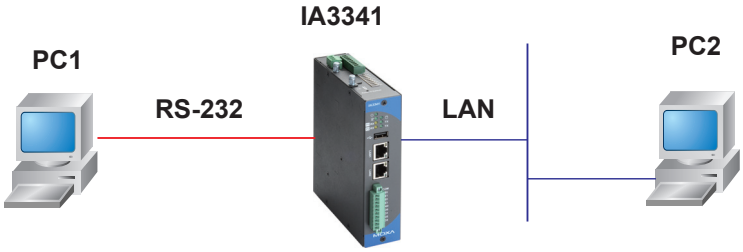
```
192.168.3.127 - PuTTY
root@Moxa:~# ls -al
drwxr-xr-x  2 root  root     0 Jun 12 02:14
drwxr-xr-x 15 root  root     0 Jan  1   1970
-rw-------  1 root  root   899 Jun 10 08:11 .bash history
-rw-r--r--  1 root  root  4996 Jun 12 02:15 tcps2-release
root@Moxa:~# chmod +x tcps2-release
root@Moxa:~# ls -al
drwxr-xr-x  2 root  root     0 Jun 12 02:14
drwxr-xr-x 15 root  root     0 Jan  1   1970
-rw-------  1 root  root   899 Jun 10 08:11 .bash history
-rwxr-xr-x  1 root  root  4996 Jun 12 02:15 tcps2-release
root@Moxa:~# ./tcps2-release &
[1] 187
start
root@Moxa:~# ps
[1]+  Running    ./tcps2-release &
root@Moxa:~#
```

NOTE Use the **`kill -9`** command for PID 187 to terminate this program: **`#kill -9 %141`**

Summary of the Testing Procedure

1. Compile **tcps2.c (`#make`)**.
2. Upload and run **tcps2-release** in the background (**`#./tcps2-release &`**).
3. Check that the process is running (**`#jobs`** or **`#ps -ef`**).
4. Use a serial cable to connect PC1 to the IA3341's serial port 1.
5. Use an Ethernet cable to connect PC2 to the IA3341.
6. On PC1: If running Windows, use HyperTerminal (**38400, n, 8, 1**) to open COMn.
7. On PC2: Type **`#telnet 192.168.4.127 4001`**.
8. On PC1: Type some text on the keyboard and then press **Enter**.
9. On PC2: The text you typed on PC1 will appear on PC2's screen.

The testing environment is illustrated in the following figure. However, note that there are limitations to the example program **tcps2.c**.



NOTE The **tcps2.c** application is a simple example designed to give users a basic understanding of the concepts involved in combining Ethernet communication and serial port communication. However, the example program has some limitations that make it unsuitable for real-life applications.

1. The serial port is in canonical mode and block mode, making it impossible to send data from the Ethernet side to the serial side (i.e., from PC 2 to PC 1 in the above example).
2. The Ethernet side will not accept multiple connections.

3

Managing Embedded Linux

This chapter includes information about version control, deployment, updates, and peripherals. The information in this chapter will be particularly useful when you need to run the same application on several IA3341 units.

The following topics are covered in this chapter:

- **System Version Information**
- **System Image Backup**
  - Upgrading the Firmware
  - Loading Factory Defaults
  - Backing Up the User Directory
  - Deploying the User Directory to Additional IA3341 Units
- **Enabling and Disabling Daemons**
- **Starting a Program Automatically at Run-Level**
- **Setting the Run-Level**
- **Adjusting the System Time**
  - Setting the Time Manually
  - NTP Client
  - Updating the Time Automatically
- **Cron—Daemon to Execute Scheduled Commands**

System Version Information

To determine the hardware capability of your IA3341, and what kind of software functions are supported, check the version numbers of your IA3341's hardware, kernel, and user file system. Contact Moxa to determine the hardware version. You will need the **Production S/N** (Serial number), which is located on the IA3341's bottom label.

To check the kernel version, type:
#kversion

```
192.168.3.127 - PuTTY
root@Moxa:~# kversion
IA3341-LX version 1.0
root@Moxa:~#
```

NOTE The kernel version number is for the factory default configuration, and if you download the latest firmware version from Moxa's website and then upgrade the IA3341's hardware.

System Image Backup

Upgrading the Firmware

The IA3341's boot loader, kernel, and root file system are combined into one firmware file, which can be downloaded from Moxa's website (www.moxa.com). The name of the file has the form **IA3341-x.x.x.**, in which "x.x.x." indicates the firmware version. To upgrade the firmware, download the firmware file to a PC, and then transfer the file to the IA3341 through a console port or Telnet console connection.

ATTENTION

Upgrading the firmware will erase all data on the Flash ROM

If you are using the **ramdisk** to store code for your applications, beware that updating the firmware will erase all of the data on the Flash ROM. You should back up your application files and data before updating the firmware.

Since different Flash disks have different sizes, it's a good idea to check the size of your Flash disk before upgrading the firmware, or before using the disk to store your application and data files. Use the **#df -h** command to list the size of each memory block and how much free space is available in each block.

```
192.168.3.127 - Putty
root@Moxa:~# df -h
Filesystem              Size      Used Available Use% Mounted on
 /dev/root              8.0M      5.7M      2.3M  71%  /
/dev/ram3            1003.0K      9.0K    943.0K   1%  /dev
/dev/ram0             499.0K     18.0K    456.0K   4%  /var
/dev/mtdblock3          6.0M    492.0K      5.5M   8%  /tmp
/dev/mtdblock3          6.0M    492.0K      5.5M   8%  /home
/dev/mtdblock3          6.0M    492.0K      5.5M   8%  /etc
 tmpfs                 30.5M         0     30.5M   0%  /dev/shm
root@Moxa:~#
```

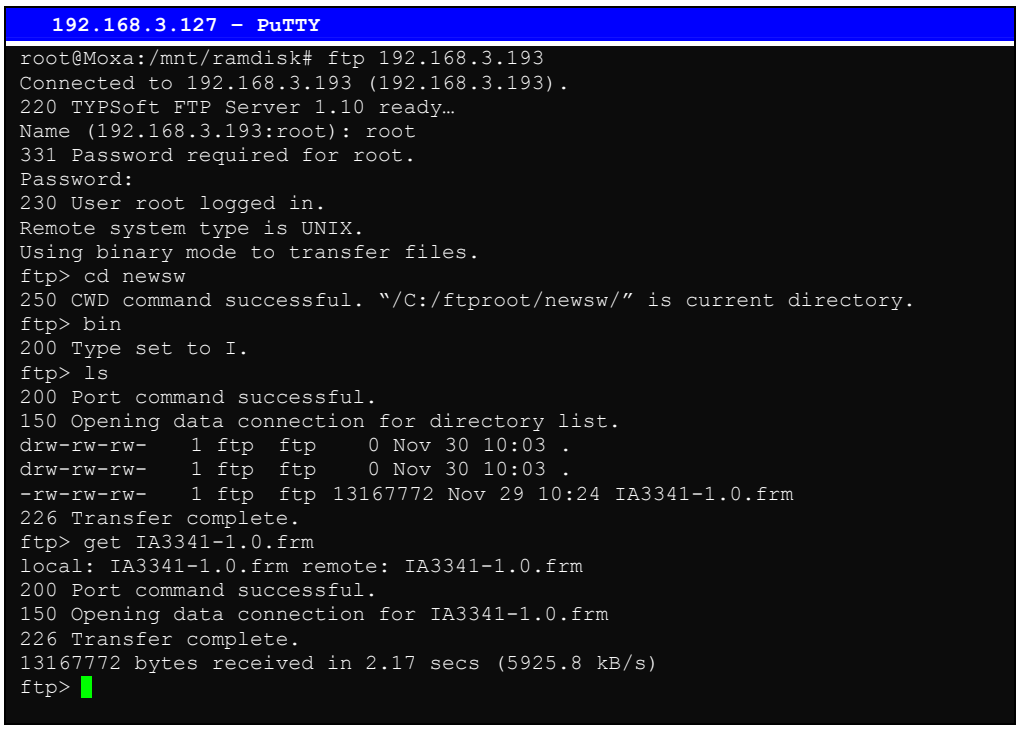
The following instructions give the steps required to save the firmware file to the IA3341's RAM disk and how to upgrade the firmware.

1. Type the following commands to enable the RAM disk:

```
#upramdisk
#cd /mnt/ramdisk
```

2. Type the following commands to use the IA3341's built-in FTP client to transfer the firmware file (**IA3341-x.x.frm**) from the PC to the IA3341:

```
/mnt/ramdisk> ftp <destination PC's IP>
Login Name: xxxx
Login Password: xxxx
ftp> bin
ftp> ftp> get IA3341-x.x.frm
```



3. Next, use the **upfirm** command to upgrade the kernel and root file system:

```
#upfirm IA3341-x.x.frm
```

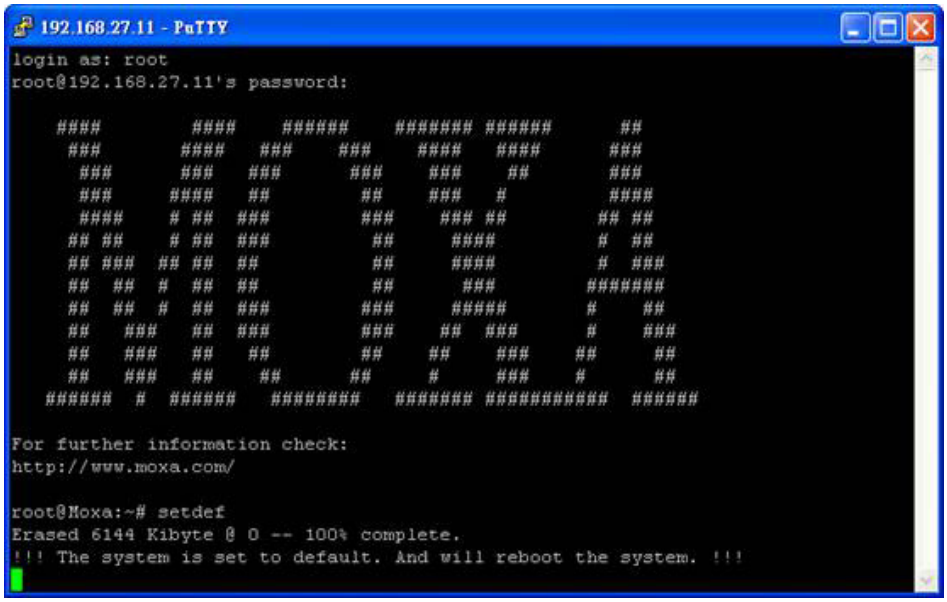
```
192.168.3.127 - PuTTY
root@Moxa:/mnt/ramdisk# upfirm IA3341-1.0.frm
Moxa  IA3341 upgrade firmware utility version 1.0.
To check source firmware file context.
The source firmware file conext is OK.
This step will destroy all your firmware.
Continue ? (Y/N) : Y
Now upgrade the file [kernel].
Format MTD device [/dev/mtd1] . . .
MTD device [/dev/mtd1] erase 128 Kibyte @ 1C0000 - 100% complete.
Wait to write file . . .
Compleleted 100%
Now upgrade the file [usrdisk].
Format MTD device [/dev/mtd2] . . .
MTD device [/dev/mtd2] erase 128 Kibyte @ 800000 - 100% complete.
Wait to write file . . .
Compleleted 100%
Upgrade the firmware is OK.
```

ATTENTION

The upfirm utility will reboot your target after the upgrade is OK.

Loading Factory Defaults

To load the the factory default settings, you must press the reset-to-default button for more than 5 seconds. All files in the **/home** and **/etc** directories will be destroyed. Note that while pressing the reset-to-default button, the Ready LED will blink once every second for the first 5 seconds. The Ready LED will turn off after 5 seconds, and the factory defaults will be loaded. In addition, you can also use the command **setdef** for loading factory defaults. When finished, the system will reboot and the factory defaults will be successfully loaded.



Backing Up the User Directory

1. Create a backup file. First type the following command to enable the RAM disk:
   #upramdisk

   Next, use the file system backup utility provided by Moxa:
   #backupuf /mnt/ramdisk/usrfs-backup

2. Once the file system is backed up, use FTP to transfer the file **usrfs-backup** to your PC.

```
192.168.3.127 - Putty
root@Moxa:~# upramdisk
root@Moxa:~# cd /mnt/ramdisk
root@Moxa: /mnt/ramdisk# df -h
Filesystem              Size      Used Available  Use% Mounted on
/dev/root               8.0M      5.7M      2.3M   71%  /
/dev/ram3            1003.0K      9.0K    943.0K   1%   /dev
/dev/ram0             499.0K     18.0K    456.0K   4%   /var
/dev/mtdblock3          6.0M    492.0K      5.5M   8%   /tmp
/dev/mtdblock3          6.0M    492.0K      5.5M   8%   /home
/dev/mtdblock3          6.0M    492.0K      5.5M   8%   /etc
tmpfs                  30.5M         0     30.5M   0%   /dev/shm
/dev/ram1              16.0M      1.0K     15.1M   0%   /var/ramdisk
root@Moxa:/mnt/ramdisk# backupuf /mnt/ramdisk/usrfs-backup
Sync the file system...
Now backup the user root file system. Please wait...
.....................................................................................
Backup user root file system OK.
root@Moxa:/mnt/ramdisk#
```

Deploying the User Directory to Additional IA3341 Units

For some applications, you may need to ghost one IA3341 user file system to other IA3341 units. Back up the user file system to a PC (refer to the previous subsection, "Backing Up the User File System," for instructions), and then type the following commands to copy the backup to additional IA3341 units.

#upramdisk
#cd /mnt/ramdisk
#upfirm usrfs-backup

```
192.168.3.127 – PuTTY
root@Moxa:/mnt/ramdisk# ls -al
drwxr-xr-x  3 root  root    1024 Jun 15 02:47
drwxr-xr-x 15 root  root       0 Sep 29  2004
-rw-------  1 root  root   12288 Jun 15 02:45  lost+found
-rw-r--r--  1 root  root    27263140 Jun 15 02:48 usrfs-backup
root@Moxa:/mnt/ramdisk# upfirm usrfs-backup
Moxa ThinkCore IA3341 upgrade firmware utility version 1.0.
To check source firmware file context.
The source firmware file conext is OK.
This step will destroy all your firmware.
Continue ? (Y/N) : Y
Now upgrade the file [userdisk]:
Format MTD device [/dev/mtd3] . . .
MTD device [/dev/mtd3] erase 128 Kibyte @ 600000 - 100% complete.
Wait to write file . . .
Completed 100%
Upgrade the firmware is OK.
```

Enabling and Disabling Daemons

The following daemons are enabled when the IA3341 boots up for the first time.

Inetd……...Internet Daemons
ftpd……FTP Server / Client daemon
sshd……...Secure Shell Server daemon
mbat……..Modbus daemon

Type the command "ps –ef" to list all processes currently running.

```
192.168.3.127 - PuTTY
root@Moxa:~# cd /etc
root@Moxa:/etc# ps -ef
  PID USER      VSZ STAT COMMAND
    1 root     1248 S    init [3]
    2 root        0 SWN  [ksoftirqd/0]
    3 root        0 SW<  [events/0]
    4 root        0 SW<  [khelper]
    5 root        0 SW<  [kblockd/0]
    7 root        0 SW   [pdflush]
    6 root        0 SW   [khubd]
    8 root        0 SW   [pdflush]
   10 root        0 SW<  [aio/0]
    9 root        0 SW   [kswapd0]
   11 root        0 SW   [mxcrypto_dispat]
   12 root        0 SW   [mtdblockd]
   13 root        0 SW<  [kmmcd]
   23 root        0 SWN  [jffs2 gcd mtd3]
   66 root     1252 S    dhcpcd eth0
   78 root     1280 S    /bin/inetd
   81 bin      1220 S    /bin/portmap
   86 root     2096 S    /bin/sh --login
   91 root     1396 S    /bin/mbat
  163 root     2300 R    ps -ef
root@Moxa:/ect#
```

Starting a Program Automatically at Run-Level

To set a private daemon to run at run-level, you can edit the file **rc.local**, as follows:

#cd /etc/rc.d
#vi rc.local

```
192.168.3.127 - PuTTY
root@Moxa:~# cd /etc/rc.d
root@Moxa:/etc/rc.d# vi rc.local
```

Next, use vi to open your application program. We use the example program **tcps2-release**, and set it to run in the background.

```
192.168.3.127 - PuTTY
# !/bin/sh
# Add you want to run daemon
/root/tcps2-release &~
```

You will find that the following daemons are enabled after you reboot the system.

```
192.168.3.127 – PuTTY
root@Moxa:~# ps -ef
  PID USER      VSZ STAT COMMAND
    1 root     1248 S    init [3]
    2 root        0 SWN  [ksoftirqd/0]
    3 root        0 SW<  [events/0]
    4 root        0 SW<  [khelper]
    5 root        0 SW<  [kblockd/0]
    7 root        0 SW   [pdflush]
    6 root        0 SW   [khubd]
    8 root        0 SW   [pdflush]
   10 root        0 SW<  [aio/0]
    9 root        0 SW   [kswapd0]
   11 root        0 SW   [mxcrypto dispat]
   12 root        0 SW   [mtdblockd]
   13 root        0 SW<  [kmmcd]
   23 root        0 SWN  [jffs2_gcd_mtd3]
   60 root     1252 S    dhcpcd eth0
   72 root     1280 S    /bin/inetd
   75 bin      1220 S    /bin/portmap
   77 root     1312 S    /root/tcps2-release
   81 root     2084 S    /bin/sh --login
   86 root     1396 S    /bin/mbat
   90 root     1596 S    /bin/ATSAGENT
   91 root     2300 R    ps -ef
root@Moxa:~#
```

Setting the Run-Level

In this section, we outline the steps you should take to set the Linux run-level and execute requests. Use the following command to enable or disable settings:

```
192.168.3.127 – PuTTY
root@Moxa:/ect/rc.d/rc3.d# ls
S99mbat         S99rmnologin    S99showreadyled
root@Moxa:/etc/rc.d/rc3.d#
```

#cd /etc/rc.d/init.d

Edit a shell script to execute **/root/tcps2-release** and save to **tcps2** as an example.

#cd /etc/rc.d/rc3.d
#ln –s /etc/rc.d/init.d/tcps2 S60tcps2

SxxRUNFILE stands for

S: start the run file while linux boots up.
xx: a number between 00-99. The smaller number has a higher priority.
RUNFILE: the file name.

```
192.168.3.127 – PuTTY
root@Moxa:/ect/rc.d/rc3.d# ls
S99mbat         S99rmnologin    S99showreadyled
root@Moxa:/ect/rc.d/rc3.d# ln -s /root/tcps2-release S60tcps2
root@Moxa:/ect/rc.d/rc3.d# ls
S99mbat         S99rmnologin    S99showreadyled    S60tcps2
root@Moxa:/etc/rc.d/rc3.d#
```

KxxRUNFILE stands for
K: start the run file while linux shuts down or halts.
xx: a number between 00-99. Smaller numbers have a higher priority.
RUNFILE: is the file name.
To remove the daemon, use the following command to remove the run file from /**etc/rc.d/rc3.d**:
#rm –f /etc/rc.d/rc3.d/S60tcps2

Adjusting the System Time

Setting the Time Manually

The IA3341 has two time settings. One is the system time, and the other is the RTC (Real Time Clock) time kept by the IA3341's hardware. Use the **#date** command to query the current system time or set a new system time. Use **#hwclock** to query the current RTC time or set a new RTC time.

Use the following command to query the system time:
#date

Use the following command to query the RTC time:
#hwclock

Use the following command to set the system time:
#date MMDDhhmmYYYY

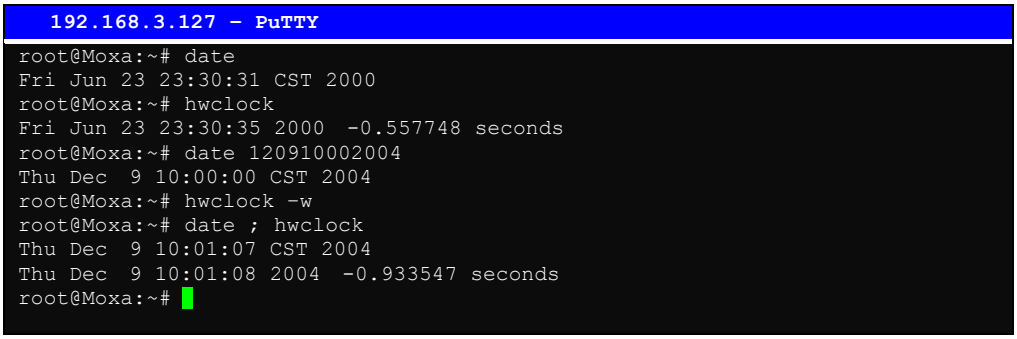
MM = Month
DD = Date
hhmm = hour and minute
YYYY = Year

Use the following command to set the RTC time:

#hwclock –w

Write current system time to RTC

The following figure illustrates how to update the system time and set the RTC time.



NTP Client

The IA3341 has a built-in NTP (Network Time Protocol) client that is used to initialize a time request to a remote NTP server. Use **#ntpdate <this client utility>** to update the system time.

#ntpdate time.stdtime.gov.tw
#hwclock –w

Visit http://www.ntp.org for more information about NTP and NTP server addresses.

```
10.120.53.100 - PuTTY
root@Moxa:~# date ; hwclock
Sat Jan  1 00:00:36 CST 2000
Sat Jan  1 00:00:37 2000  -0.772941 seconds
root@Moxa:~# ntpdate time. stdtime.gov.tw
 9 Dec 10:58:53 ntpdate[207]: step time server 220.130.158.52 offset 155905087.9
84256 sec
root@Moxa:~# hwclock -w
root@Moxa:~# date ; hwclock
Thu Dec  9 10:59:11 CST 2004
Thu Dec  9 10:59:12 2004  -0.844076 seconds
root@Moxa:~#
```

NOTE Before using the NTP client utility, check your IP and DNS settings to make sure that an Internet connection is available. Refer to Chapter 2 for instructions on how to configure the Ethernet interface, and see Chapter 4 for DNS setting information.

Updating the Time Automatically

In this subsection, we show how to use a shell script to update the time automatically.

Example shell script to update the system time periodically

```
#!/bin/sh
ntpdate time.nist.gov # You can use the time server's ip address or domain
        # name directly. If you use domain name, you must
        # enable the domain client on the system by updating
        # /etc/resolv.conf file.
hwclock --systohc
sleep 100  # Updates every 100 seconds. The sleeping time is 100 seconds. Change
    # 100 to a larger number to update RTC less often.
```

Save the shell script using any file name. E.g., **fixtime**

How to run the shell script automatically when the kernel boots up

Copy the example shell script **fixtime** to directory **/etc/init.d**, and then use **chmod 755 fixtime** to change the shell script mode. Next, use vi editor to edit the file **/etc/inittab**. Add the following line to the bottom of the file:

```
ntp : 2345 : respawn : /etc/init.d/fixtime
```

Use the command **#init q** to re-init the kernel.

Cron—Daemon to Execute Scheduled Commands

Cron is a scheduling service in Linux. Cron wakes up every minute, and checks the configuration file named crontab to see if any scheduled command should be run in the current minute.

Crontab is located in the **/etc/cron.d** directory. Modify the file **/etc/cron.d/crontab** to set up your scheduled applications. Crontab has the following format:

mm	h	dom	mon	dow	user	command
min	hour	date	month	week	user	command
0-59	0-23	1-31	1-12	0-6 (0 is Sunday)		

The following example demonstrates how to use Cron.

How to use cron to update the system time and RTC time every day at 8:00.

STEP1: Write a shell script named fixtime.sh and save it to /home/.

```
#!/bin/sh
ntpdate time.nist.gov
hwclock --systohc
exit 0
```

STEP2: Change mode of fixtime.sh

```
#chmod 755 fixtime.sh
```

STEP3: Modify /etc/cron.d/crontab file to run fixtime.sh at 8:00 every day.

Add the following line to the end of crontab:

```
* 8 * * * root /home/fixtime.sh
```

STEP4: Enable the cron daemon manually.

```
#/etc/init.d/cron start
```

STEP5: Enable cron when the system boots up.

By default, cron service is disabled on boot. To enable cron service, please refer to the section "Enabling and Disabling Daemons" in this chapter

4

Managing Communications

In this chapter, we explain how to configure the IA3341's various communication functions. The following topics are covered in this chapter:

- **FTP**
- **DNS**
- **Web Service—Apache**
- **Install PHP for Apache Web Server**
- **IPTABLES**
  - Observe and erase chain rules
  - Define policy for chain rules
  - Append or delete rules:
- **NAT**
  - NAT Example
  - Enabling NAT at Bootup
- **Dial-up Service—PPP**
  - Example 1: Connecting to a PPP server over a simple dial-up connection
  - Example 2: Connecting to a PPP server over a hard-wired link
  - How to check the connection
  - Setting up a Machine for Incoming PPP Connections
- **PPPoE**
- **NFS (Network File System)**
  - Setting up the IA3341 as an NFS Client
- **Mail**
- **Installing Net-SNMP**

FTP

In addition to supporting FTP client/server, the IA3341 also supports SSH and sftp client/server. To enable or disable the ftp server, you first need to edit the file **/etc/inetd.conf**.

Enabling the ftp server

The following example shows the default content of the file /**etc/inetd.conf**. The default is to enable the ftp server:

```
discard dgram udp wait root /bin/discard
discard stream tcp nowait root /bin/discard
ftp stream tcp nowait root /bin/ftpd –l
ssh stream tcp nowait root /bin/sshd -i
```

Disabling the ftp server

Disable the daemon by typing '#' in front of the first character of the row to comment out the line.

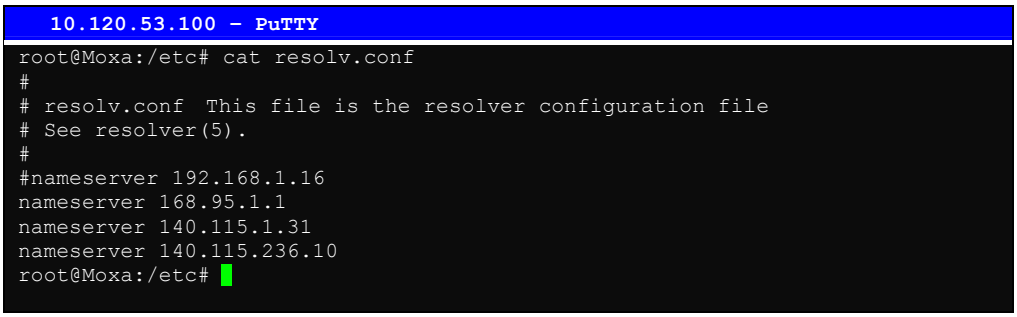
DNS

The IA3341 supports DNS client (but not DNS server). To set up DNS client, you need to edit three configuration files: **/etc/hosts**, **/etc/resolv.conf**, and **/etc/nsswitch.conf.**

/etc/hosts
This is the first file that the Linux system reads to resolve the host name and IP address.

/etc/resolv.conf
This is the most important file that you need to edit when using DNS for the other programs. For example, before you use **#ntpdate time.nist.goc** to update the system time, you will need to add the DNS server address to the file. Ask your network administrator which DNS server address you should use. The DNS server's IP address is specified with the "nameserver" command. For example, add the following line to /etc/resolv.conf if the DNS server's IP address is 168.95.1.1:

```
nameserver 168.95.1.1
```



/etc/nsswitch.conf
This file defines the sequence to resolve the IP address by using **/etc/hosts** or **/etc/resolv.conf**.

Web Service—Apache

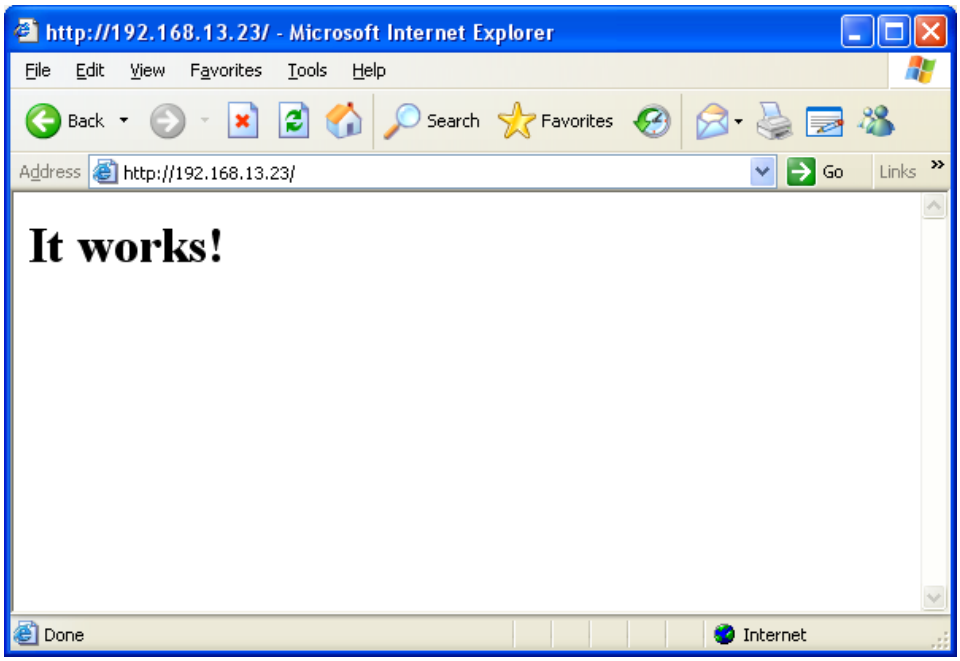
The Apache web server's main configuration file is **/etc/apache/conf/httpd.conf**, with the default homepage located at **/home/httpd/htdocs/index.html**. Save your own homepage to the following directory:

/home/httpd/htdocs/

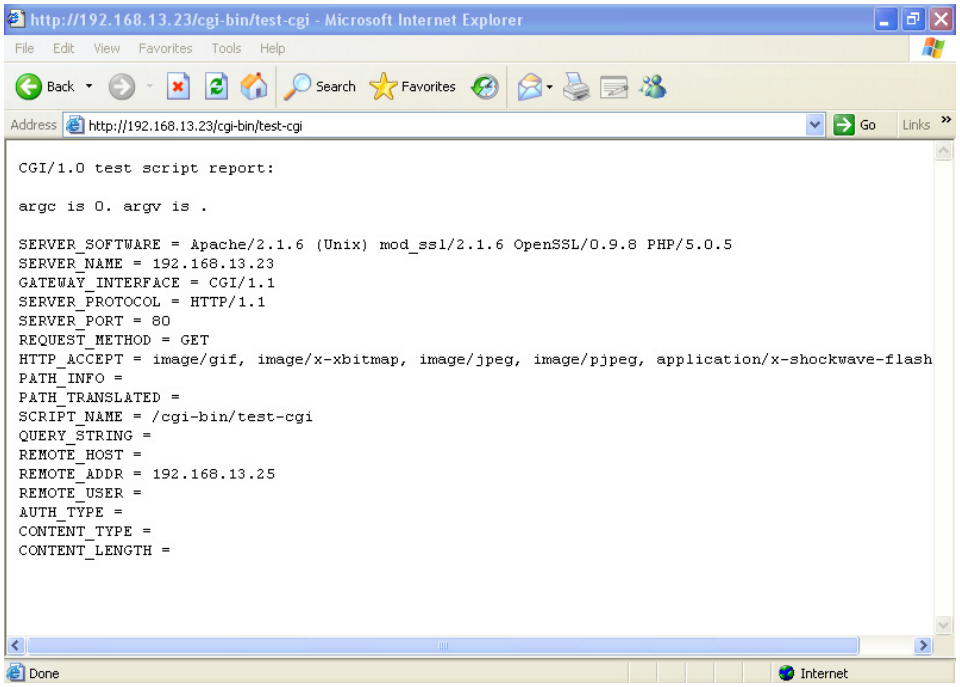
Save your CGI page to the following directory:

/home/httpd/cgi-bin/

Before you modify the homepage, use a browser (such as Microsoft Internet Explore or Mozilla Firefox) from your PC to test if the Apache Web Server is working. Type the LAN IP address in the browser's address box to open the homepage. E.g., if the default IP address is still active, type **http://192.168.13.23** in the address box.



To open the default CGI page, type **http://192.168.13.23/cgi-bin/test-cgi** in your browser's address box.



NOTE The CGI function is enabled by default. If you want to disable the function, modify the file **/etc/apache/conf/httpd.conf**. When you develop your own CGI application, make sure your CGI file is executable.

```
192.168.3.127 - PuTTY
root@Moxa:/home/httpd/cgi-bin# ls -al
drwxr-xr-x  2 root  root       0 Aug 24 1999
drwxr-xr-x  5 root  root       0 Nov  5 16:16
-rwxr-xr-x  1 root  root     757 Aug 24 1999 test-cgi
root@Moxa:/home/httpd/cgi-bin#
```

Install PHP for Apache Web Server

This embedded computer supports the PHP option. However, since the PHP file is 3 MB, it is not installed by default. To install it yourself, first make sure there is enough free space (at least 3 MB) on your embedded flash ROM).

Step 1: Check that you have enough free space The following figure illustrates how to check that the **/dev/mtdblock3** has more than 3 MB of free space.

```
192.168.3.127 - Putty
root@Moxa:~# df -h
Filesystem              Size      Used  Available  Use%  Mounted on
/dev/root               8.0M      5.7M       2.3M   71%  /
/dev/ram3            1003.0K      9.0K     943.0K    1%  /dev
/dev/ram0             499.0K     18.0K     456.0K    4%  /var
/dev/mtdblock3          6.0M    492.0K       5.5M    8%  /tmp
/dev/mtdblock3          6.0M    492.0K       5.5M    8%  /home
/dev/mtdblock3          6.0M    492.0K       5.5M    8%  /etc
tmpfs                  30.5M         0      30.5M    0%  /dev/shm
root@Moxa:~#
```

Step 2: Type **upramdisk** to get the free space ram disk to save the package.

```
192.168.3.127 - Putty
root@Moxa:~# upramdisk
root@Moxa:~# df -h
Filesystem       Size      Used  Available  Use%  Mounted on
/dev/root        8.0M      5.7M       2.3M   71%  /
/dev/ram3     1003.0K      9.0K     943.0K    1%  /dev
/dev/ram0      499.0K     18.0K     456.0K    4%  /var
/dev/mtdblock  3 6.0M    492.0K       5.5M    8%  /tmp
/dev/mtdblock  3 6.0M    492.0K       5.5M    8%  /home
/dev/mtdblock  3 6.0M    492.0K       5.5M    8%  /etc
tmpfs           30.5M         0      30.5M    0%  /dev/shm
/dev/ram1       16.0M      1.0K      15.1M    0%  /var/ramdisk
root@Moxa:~#
```

Step 3: Download the PHP package from the CD-ROM. You can find the package in **CD-ROM/target/php/php.tgz**.

```
192.168.3.127 - PuTTY
root@Moxa:/bin# cd /mnt/ramdisk
root@Moxa:/mnt/ramdisk# ftp 192.168.27.130
Connected to 192.168.27.130.
220 (vsFTPd 2.0.1)
Name (192.168.27.130:root): root
331 Please specify the password.
Password:
230 Login successful.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> cd /tmp
250 Directory successfully changed.
ftp> bin
200 Switching to Binary mode.
ftp> get php.tgz
local: php.tgz remote: php.tgz
200 PORT command successful. Consider using PASV.
150 Opening BINARY mode data connection for php.tgz (1789032 bytes).
226 File send OK.
1789032 bytes received in 0.66 secs (2.6e+03 Kbytes/sec)
ftp>
```

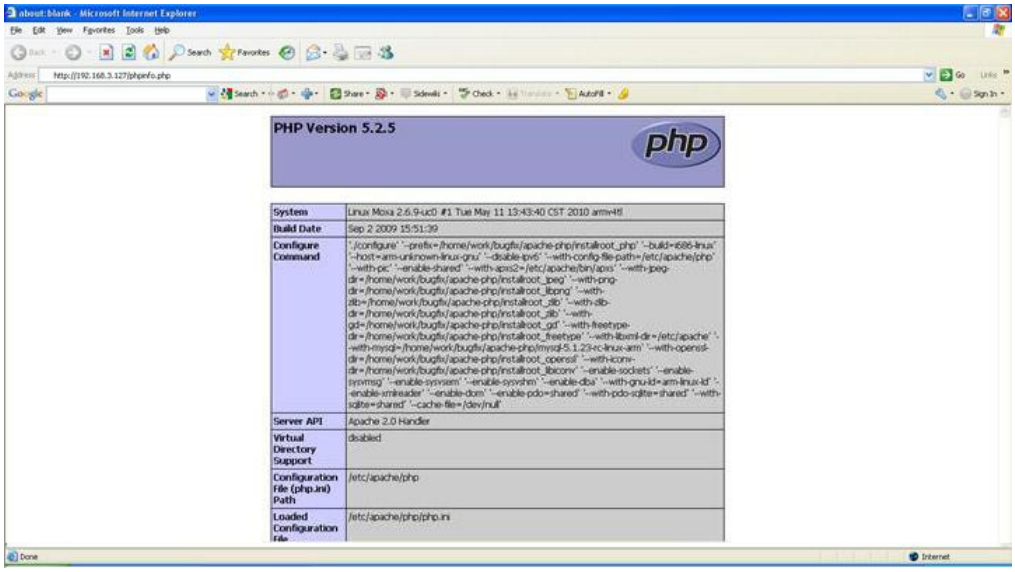
Step 4: Untar the package. To do this, type the command **tar xvzf php.tgz**.

```
192.168.3.127 - PuTTY
root@Moxa:/mnt/ramdisk# tar xvzf php.tgz
envvars
envvars.old
httpd.conf
httpd.conf.old
install.sh
lib
lib/libmysqlclient.so.15
lib/libpng.so.2
lib/libphp5.so
lib/libmysqlclient.so.15.0.0
lib/libgd.so
lib/libxml2.so.2.6.22
lib/libgd.so.2.0.0
lib/libjpeg.so
lib/libxml2.so.2
lib/libgd.so.2
php
php/php.ini
phpinfo.php
root@Moxa:/mnt/ramdisk#
```

Step 5: Run **install.sh** and select to install php.

```
192.168.3.127 - PuTTY
root@Moxa:/mnt/ramdisk/apache-sdlinux@ ./install.sh
Choose:
1. Install Apache.
2. Install Apache + PHP.
3. Install Apache + SSL + PHP.
4. Uninstall Apache.
5. Exit
2
Installing, Please wait……
Install successful..
```

Step 6: Test it. Use the browser to access http://192.168.3.127/phpinfo.php.



If you want to uninstall PHP, follow steps 2 to 5 but select the uninstall option.

IPTABLES

IPTABLES is an administrative tool for setting up, maintaining, and inspecting the Linux kernel's IP packet filter rule tables. Several different tables are defined, with each table containing built-in chains and user-defined chains.

Each chain is a list of rules that apply to a certain type of packet. Each rule specifies what to do with a matching packet. A rule (such as a jump to a user-defined chain in the same table) is called a "target."

The IA3341 supports 3 types of IPTABLES table: **Filter** tables, **NAT** tables, and **Mangle** tables:

A. **Filter Table**—includes three chains:

INPUT chain
OUTPUT chain
FORWARD chain

B. **NAT Table**—includes three chains:

PREROUTING chain—transfers the destination IP address (DNAT)

POSTROUTING chain—works after the routing process and before the Ethernet device process to transfer the source IP address (SNAT)

OUTPUT chain—produces local packets

sub-tables

Source NAT (SNAT)—changes the first source packet IP address

Destination NAT (DNAT)—changes the first destination packet IP address

MASQUERADE—a special form for SNAT. If one host can connect to Internet, then other computers that connect to this host can connect to the Internet when the computer does not have an actual IP address.

REDIRECT—a special form of DNAT that re-sends packets to a local host independent of the destination IP address.

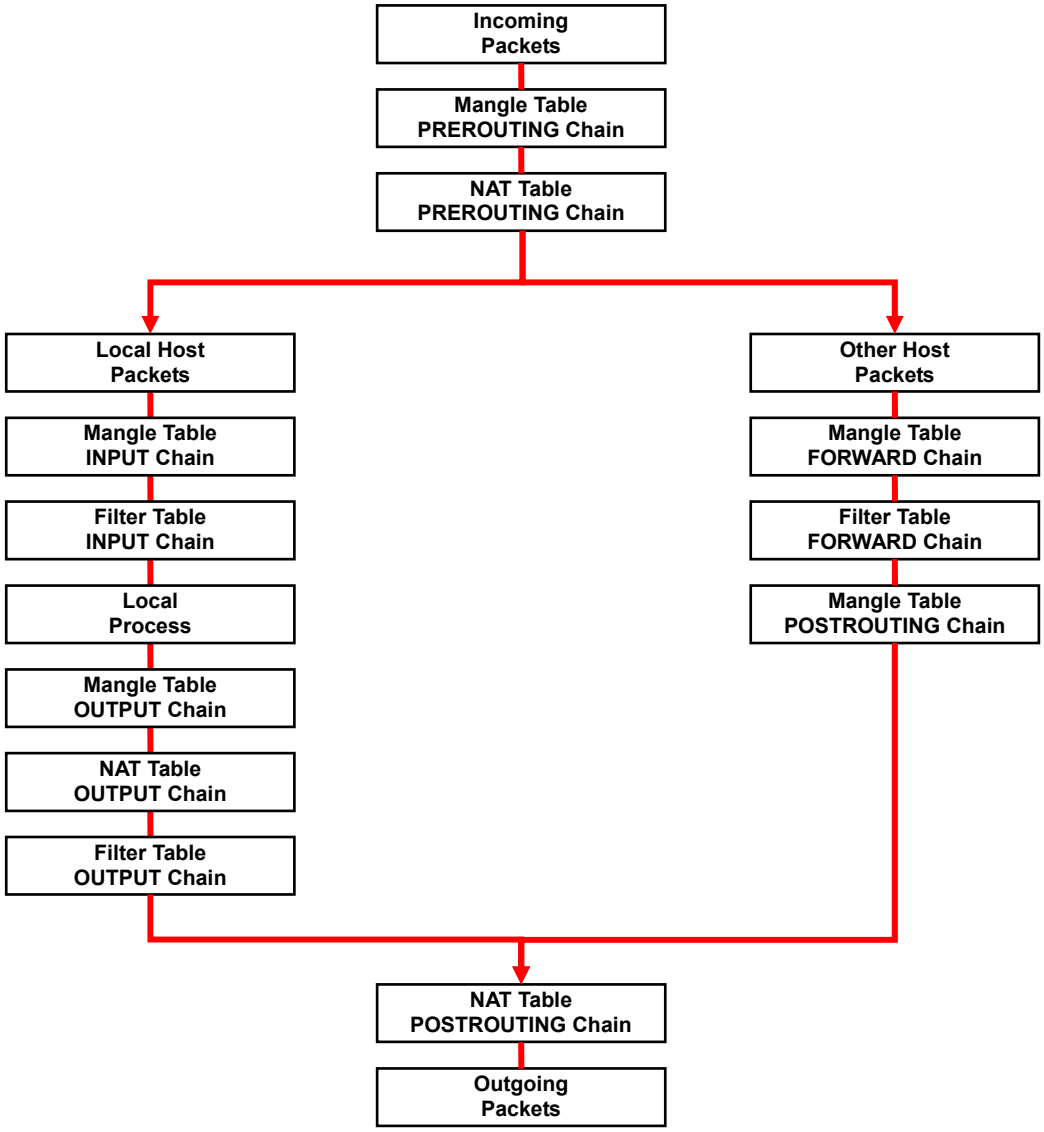
C. **Mangle Table**—includes two chains

PREROUTING chain—pre-processes packets before the routing process.

OUTPUT chain—processes packets after the routing process.

It has three extensions—TTL, MARK, TOS.

The following figure shows the IPTABLES hierarchy.



The IA3341 supports the following sub-modules. Be sure to use the module that matches your application.

ip_conntrack	ipt_MARK	ipt_ah	ipt_state
ip_conntrack_ftp	ipt_MASQUERADE	ipt_esp	ipt_tcpmss
ipt_conntrack_irc	ipt_MIRROT	ipt_length	ipt_tos
ip_nat_ftp	ipt_REDIRECT	ipt_limit	ipt_ttl
ip_nat_irc	ipt_REJECT	ipt_mac	ipt_unclean
ip_nat_snmp_basic	ipt_TCPMSS	ipt_mark	
ip_queue	ipt_TOS	ipt_multiport	
ipt_LOG	ipt_ULOG	ipt_owner	

NOTE The IA3341 does NOT support IPV6 and ipchains.

The basic syntax to enable and load an IPTABLES module is as follows:

```
#lsmod
#modprobe ip_tables
#modprobe iptable_filter
```

Use **lsmod** to check if the ip_tables module has already been loaded in the IA3341. Use **modprobe** to insert and enable the module.

Use the following command to load the modules (iptable_filter, iptable_mangle, iptable_nat):

```
#modprobe iptable_filter
```

NOTE IPTABLES plays the role of packet filtering or NAT. Take care when setting up the IPTABLES rules. If the rules are not correct, remote hosts that connect via a LAN or PPP may be denied access. We recommend using the Serial Console to set up the IPTABLES.

Click on the following links for more information about iptables.

http://www.linuxguruz.com/iptables/
http://www.netfilter.org/documentation/HOWTO//packet-filtering-HOWTO.html

Since the IPTABLES command is very complex, to illustrate the IPTABLES syntax we have divided our discussion of the various rules into three categories: **Observe and erase chain rules**, **Define policy rules,** and **Append or delete rules**.

Observe and erase chain rules

Usage:

```
# iptables [-t tables] [-L] [-n]
```

-t tables: Table to manipulate (default: 'filter'); example: nat or filter.
-L [chain]: ListList all rules in selected chains. If no chain is selected, all chains are listed.
-n: Numeric output of addresses and ports.

```
# iptables [-t tables] [-FXZ]
```

-F: Flush the selected chain (all the chains in the table if none is listed).
-X: Delete the specified user-defined chain.
-Z: Set the packet and byte counters in all chains to zero.

Examples:

```
# iptables -L -n
```

In this example, since we do not use the -t parameter, the system uses the default 'filter' table. Three chains are included: INPUT, OUTPUT, and FORWARD. INPUT chains are accepted automatically, and all connections are accepted without being filtered.

```
#iptables –F
#iptables –X
#iptables -Z
```

Define policy for chain rules

Usage:

```
# iptables [-t tables] [-P] [INPUT, OUTPUT, FORWARD, PREROUTING, OUTPUT, POSTROUTING]
[ACCEPT, DROP]
```

-P: Set the policy for the chain to the given target.
INPUT: For packets coming into the IA3341.

OUTPUT: For locally-generated packets.
FORWARD: For packets routed out through the IA3341.
PREROUTING: To alter packets as soon as they come in.
POSTROUTING: To alter packets as they are about to be sent out.

Examples:

```
#iptables –P INPUT DROP
#iptables –P OUTPUT ACCEPT
#iptables –P FORWARD ACCEPT
#iptables –t nat –P PREROUTING ACCEPT
#iptables –t nat –P OUTPUT ACCEPT
#iptables -t nat –P POSTROUTING ACCEPT
```

In the above example, the policy accepts outgoing packets and denies incoming packets.

Append or delete rules:

Usage:

```
# iptables [-t table] [-AI] [INPUT, OUTPUT, FORWARD] [-io interface] [-p tcp, udp, icmp,
all] [-s IP/network] [--sport ports] [-d IP/network] [--dport ports] –j [ACCEPT. DROP]
```

-A: Append one or more rules to the end of the selected chain.
-I: Insert one or more rules in the selected chain as the given rule number.
-i: Name of an interface via which a packet is going to be received.
-o: Name of an interface via which a packet is going to be sent.
-p: The protocol of the rule or of the packet to check.
-s: Source address (network name, host name, network IP address, or plain IP address).
--sport: Source port number.
-d: Destination address.
--dport: Destination port number.
-j: Jump target. Specifies the target of the rules; i.e., how to handle matched packets. For example, ACCEPT the packet, DROP the packet, or LOG the packet.

Examples:

Example 1: Accept all packets from lo interface.

```
# iptables –A INPUT –i lo –j ACCEPT
```

Example 2: Accept TCP packets from 192.168.0.1.

```
# iptables –A INPUT –i eth0 –p tcp –s 192.168.0.1 –j ACCEPT
```

Example 3: Accept TCP packets from Class C network 192.168.1.0/24.

```
# iptables –A INPUT –i eth0 –p tcp –s 192.168.1.0/24 –j ACCEPT
```

Example 4: Drop TCP packets from 192.168.1.25.

```
# iptables –A INPUT –i eth0 –p tcp –s 192.168.1.25 –j DROP
```

Example 5: Drop TCP packets addressed for port 21.

```
# iptables –A INPUT –i eth0 –p tcp --dport 21 –j DROP
```

Example 6: Accept TCP packets from 192.168.0.24 to IA3341's port 137, 138, 139

```
# iptables –A INPUT –i eth0 –p tcp –s 192.168.0.24 --dport 137:139 –j ACCEPT
```

Example 7: Drop all packets from MAC address 01:02:03:04:05:06.

```
# iptables –A INPUT –i eth0 –p all –m mac --mac-source 01:02:03:04:05:06 –j DROP
```

NOTE: In Example 7, remember to issue the command **`#modprobe ipt_mac`** first to load module **ipt_mac**.

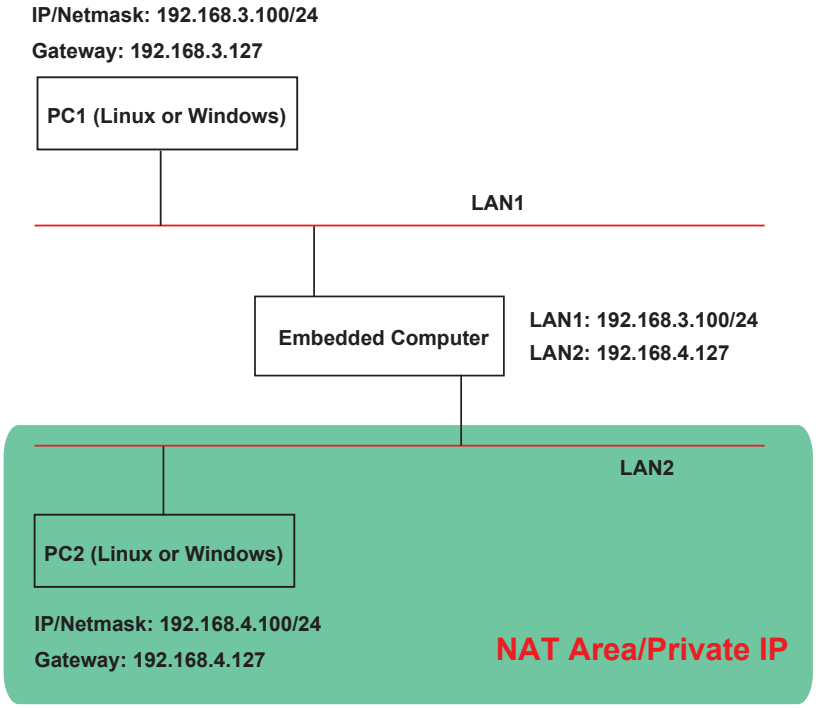
NAT

NAT (Network Address Translation) protocol translates IP addresses used on one network to different IP addresses used on another network. One network is designated the inside network and the other is the outside network. Typically, the IA3341 connects several devices on a network and maps local inside network addresses to one or more global outside IP addresses, and un-maps the global IP addresses on incoming packets back into local IP addresses.

NOTE Click on the following link for more information about iptables and NAT:
http://www.netfilter.org/documentation/HOWTO/NAT-HOWTO.html

NAT Example

The IP address of the LAN1 is changed to 192.168.3.127 (you will need to load the module ipt_MASQUERADE):




```
1.  #echo 1 > /proc/sys/net/ipv4/ip_forward
2.  #modprobe ip_tables
3.  #modprobe iptable_filter
4.  #modprobe ip_conntrack
5.  #modprobe iptable_nat
6.  #modprobe ipt_MASQUERADE
7.  #iptables -t nat -A POSTROUTING -o eth0 -j SNAT --to-source 192.168.3.127
8.  #iptables -t nat -A POSTROUTING -o eth0 -s 192.168.3.0/24 -j MASQUERADE
```

Enabling NAT at Bootup

In most real world situations, you will want to use a simple shell script to enable NAT when the IA3341 boots up. The following script is an example.

```
#!/bin/bash
# If you put this shell script in the /home/nat.sh
# Remember to chmod 744 /home/nat.sh
# Edit the rc.local file to make this shell startup automatically.
# vi /etc/rc.d/rc.local
# Add a line in the end of rc.local /home/nat.sh
EXIF='eth0'  #This is an external interface for setting up a valid IP address.
EXNET='192.168.4.0/24'  #This is an internal network address.
# Step 1. Insert modules.
# Here 2> /dev/null means the standard error messages will be dump to null device.
modprobe ip_tables  2> /dev/null
modprobe ip_conntrack  2> /dev/null
modprobe ip_conntrack_ftp  2> /dev/null
modprobe ip_conntrack_irc  2> /dev/null
modprobe iptable_nat 2> /dev/null
modprobe ip_nat_ftp 2> /dev/null
modprobe ip_nat_irc 2> /dev/null

# Step 2. Define variables, enable routing and erase default rules.

PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin
export PATH
echo "1" > /proc/sys/net/ipv4/ip_forward
/bin/iptables -F
/bin/iptables -X
/bin/iptables -Z
/bin/iptables -F -t nat
/bin/iptables -X -t nat
/bin/iptables -Z -t nat
/bin/iptables -P INPUT   ACCEPT
/bin/iptables -P OUTPUT  ACCEPT
/bin/iptables -P FORWARD ACCEPT
/bin/iptables -t nat -P PREROUTING  ACCEPT
/bin/iptables -t nat -P POSTROUTING ACCEPT
/bin/iptables -t nat -P OUTPUT     ACCEPT

# Step 3. Enable IP masquerade.
```

Dial-up Service—PPP

PPP (Point to Point Protocol) is used to run IP (Internet Protocol) and other network protocols over a serial link. PPP can be used for direct serial connections (using a null-modem cable) over a Telnet link, and links established using a modem over a telephone line.

Modem / PPP access is almost identical to connecting directly to a network through the IA3341's Ethernet port. Since PPP is a peer-to-peer system, the IA3341 can also use PPP to link two networks (or a local network to the Internet) to create a Wide Area Network (WAN).

NOTE Click on the following links for more information about ppp:
http://tldp.org/HOWTO/PPP-HOWTO/index.html
http://axion.physics.ubc.ca/ppp-linux.html

The pppd daemon is used to connect to a PPP server from a Linux system. For detailed information about pppd see the man page.

Example 1: Connecting to a PPP server over a simple dial-up connection

The following command is used to connect to a PPP server by modem. Use this command for old ppp servers that prompt for a login name (replace username with the correct name) and password (replace password with the correct password). Note that debug and defaultroute 192.1.1.17 are optional.

#pppd connect 'chat -v " " ATDT5551212 CONNECT" " ogin: *username* word: *password*' /dev/ttyM0 115200 debug crtscts modem defaultroute

If the PPP server does not prompt for the username and password, the command should be entered as follows. Replace *username* with the correct username and replace *password* with the correct password.

#pppd connect 'chat -v " " ATDT5551212 CONNECT" " ' user *username* password *password* /dev/ttyM0 115200 crtscts modem

The pppd options are described below:

connect 'chat etc...'
This option gives the command to contact the PPP server. The 'chat' program is used to dial a remote computer. The entire command is enclosed in single quotes because pppd expects a one-word argument for the 'connect' option. The options for 'chat' are given below:

-v
verbose mode; log what we do to syslog

" "
Double quotes—don't wait for a prompt, but instead do ... (note that you must include a space after the second quotation mark)

ATDT5551212
Dial the modem, and then ...

CONNECT
Wait for an answer.

" "
Send a return (null text followed by the usual return)

ogin: *username* word: *password*
Log in with *username* and *password*.

Refer to the chat man page, chat.8, for more information about the chat utility.

/dev/
Specify the callout serial port.

115200
The baudrate.

debug
Log status in syslog.

crtscts
Use hardware flow control between computer and modem (at 115200 this is a must).

modem
Indicates that this is a modem device; pppd will hang up the phone before and after making the call.

defaultroute
Once the PPP link is established, make it the default route; if you have a PPP link to the Internet, this is probably what you want.

192.1.1.17
This is a degenerate case of a general option of the form x.x.x.x:y.y.y.y. Here x.x.x.x is the local IP

address and y.y.y.y is the IP address of the remote end of the PPP connection. If this option is not specified, or if just one side is specified, then x.x.x.x defaults to the IP address associated with the local machine's hostname (located in **/etc/hosts**), and y.y.y.y is determined by the remote machine.

Example 2: Connecting to a PPP server over a hard-wired link

If a username and password are not required, use the following command (note that *noipdefault* is optional):

```
#pppd connect 'chat –v" " " " ' noipdefault /dev/ttyM0 19200 crtscts
```

If a username and password is required, use the following command (note that *noipdefault* is optional, and *root* is both the username and password):

```
#pppd connect 'chat –v" " " " ' user root password root noipdefault
/dev/ttyM0 19200 crtscts
```

How to check the connection

Once you've set up a PPP connection, there are some steps you can take to test the connection. First, type:

```
#ifconfig
```

You should be able to see all the network interfaces that are UP. ppp0 should be one of them, and you should recognize the first IP address as your own, and the "P-t-P address" (or point-to-point address) the address of your server. Here's what it looks like on one machine:

lo	Link encap Local Loopback
	inet addr 127.0.0.1 Bcast 127.255.255.255 Mask 255.0.0.0
	UP LOOPBACK RUNNING MTU 2000 Metric 1
	RX packets 0 errors 0 dropped 0 overrun 0
ppp0	Link encap Point-to-Point Protocol
	inet addr 192.76.32.3 P-t-P 129.67.1.165 Mask 255.255.255.0
	UP POINTOPOINT RUNNING MTU 1500 Metric 1
	RX packets 33 errors 0 dropped 0 overrun 0
	TX packets 42 errors 0 dropped 0 overrun 0

Now, type:

```
ping z.z.z.z
```

where z.z.z.z is the address of your name server. This should work. Here's what the response could look like:

 # ping 129.67.1.165
PING 129.67.1.165 (129.67.1.165): 56 data bytes
64 bytes from 129.67.1.165: icmp_seq=0 ttl=225 time=268 ms
64 bytes from 129.67.1.165: icmp_seq=1 ttl=225 time=247 ms
64 bytes from 129.67.1.165: icmp_seq=2 ttl=225 time=266 ms
^C
--- 129.67.1.165 ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max = 247/260/268 ms
waddington:~$

Try typing:

```
netstat -nr
```

This should show three routes, something like this:

Kernel routing table						
Destination	Gateway	Genmask	Flags	Metric	Ref	Use
iface						
129.67.1.165	0.0.0.0	255.255.255.255	UH	0	0	6
ppp0						
127.0.0.0	0.0.0.0	255.0.0.0	U	0	0	0 lo
0.0.0.0	129.67.1.165	0.0.0.0	UG	0	0	6298
ppp0						

If your output looks similar but doesn't have the destination 0.0.0.0 line (which refers to the default route used for connections), you may have run pppd without the 'defaultroute' option. At this point you can try using Telnet, ftp, or finger, bearing in mind that you'll have to use numeric IP addresses unless you've set up /etc/resolv.conf correctly.

Setting up a Machine for Incoming PPP Connections

This first example applies to using a modem, and requiring authorization with a username and password.

```
pppd/dev/ttyM0 115200 crtscts modem 192.168.16.1:192.168.16.2 login auth
```

You should also add the following line to the file **/etc/ppp/pap-secrets**:

```
*    *    ""    *
```

The first star (*) lets everyone login. The second star (*) lets every host connect. The pair of double quotation marks ("") is to use the file **/etc/passwd** to check the password. The last star (*) is to let any IP connect.

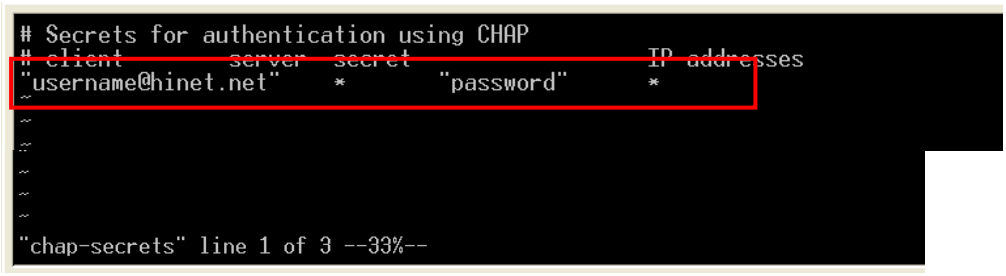
The following example does not check the username and password:

```
pppd/dev/ttyM0 115200 crtscts modem 192.168.16.1:192.168.16.2
```

PPPoE

1. Connect the IA3341's LAN port to an ADSL modem with a cross-over cable, HUB, or switch.
2. Log on to the IA3341 as the root user.
3. Edit the file **/etc/ppp/chap-secrets** and add the following text:

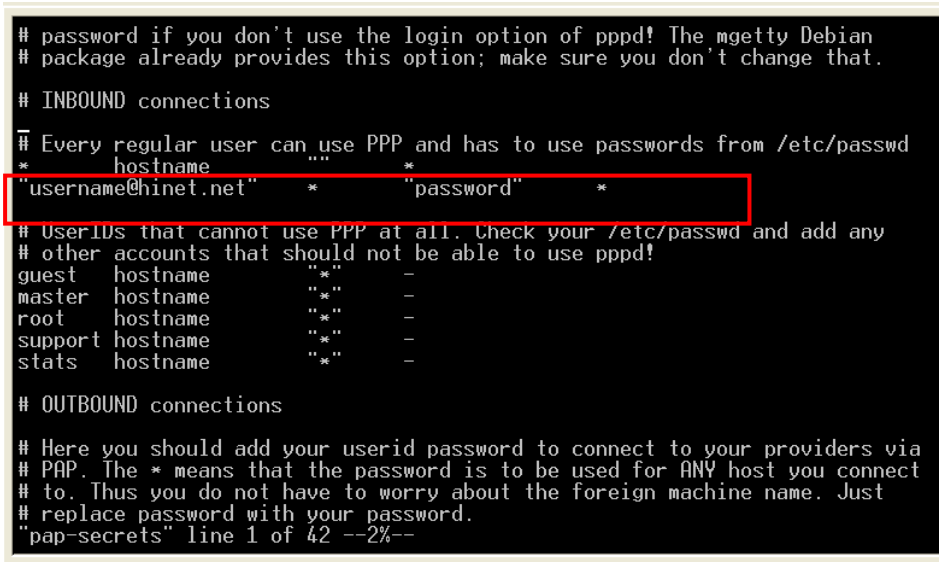
   "username@hinet.net"*"password"*



"username@hinet.net" is the username obtained from the ISP to log in to the ISP account.
"password" is the corresponding password for the account.

4. Edit the file **/etc/ppp/pap-secrets** and add the following text:

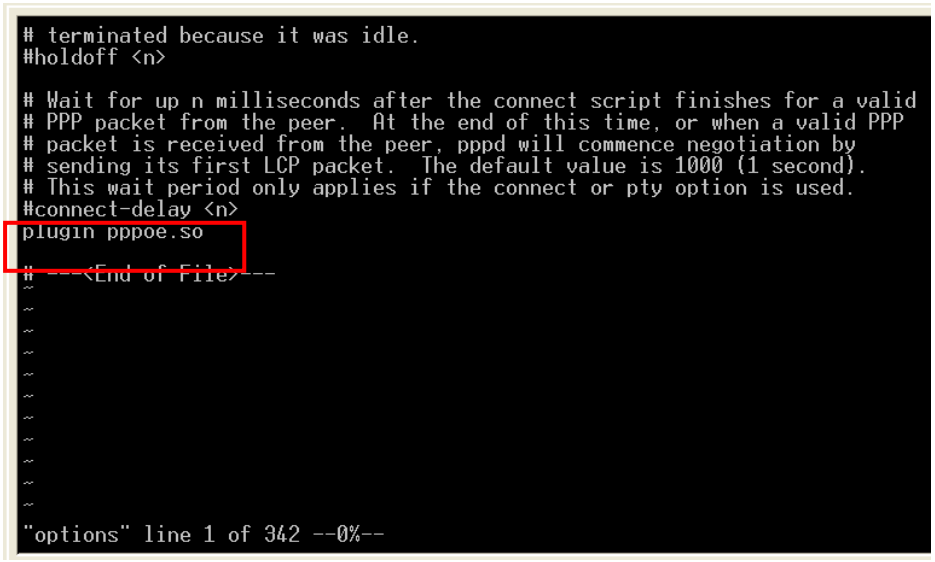
   "username@hinet.net"*"password"*



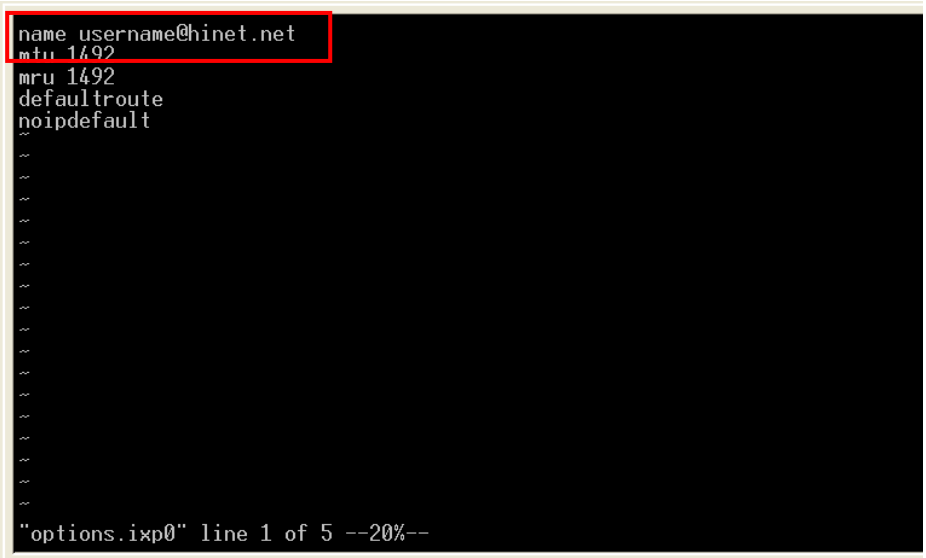
   "username@hinet.net" is the username obtained from the ISP to log in to the ISP account.
   "password" is the corresponding password for the account.

5. Edit the file **/etc/ppp/options** and add the following line:

   plugin pppoe



6. Edit the file **/etc/ppp/options.eth0**.



Type your username (the one you set in the **/etc/ppp/pap-secrets** and **/etc/ppp/chap-secrets** files) after the "name" option. You may add other options as desired.

7. Set up DNS

   If you are using DNS servers supplied by your ISP, edit the file **/etc/resolv.conf** by adding the following lines of code:

   nameserver ip_addr_of_first_dns_server

   nameserver ip_addr_of_second_dns_server

   For example:

   nameserver 168..95.1.1

   nameserver 139.175.10.20

8. Use the following command to create a pppoe connection:

   pppd eth0

   The eth0 is what is connected to the ADSL modem LAN port.

9. Type **ifconfig ppp0** to check if the connection is OK or has failed. If the connection is OK, you will see information about the ppp0 setting for the IP address. Use ping to test the IP.

10. If you want to disconnect it, use the kill command to kill the pppd process.

NFS (Network File System)

The Network File System (NFS) is used to mount a disk partition on a remote machine, as if it were on a local hard drive, allowing fast, seamless sharing of files across a network. NFS allows users to develop applications for the IA3341, without worrying about the amount of disk space that will be available. The IA3341 supports NFS protocol for client.

NOTE Click on the following links for more information about NFS:
http://www.tldp.org/HOWTO/NFS-HOWTO/index.html
http://nfs.sourceforge.net/nfs-howto/client.html
http://nfs.sourceforge.net/nfs-howto/server.html

Setting up the IA3341 as an NFS Client

The following procedure is used to mount a remote NFS Server.

1. To know the NFS Server's shared directory.
2. Establish a mount point on the NFS Client site.
3. Mount the remote directory to a local directory.

```
#mkdir -p /home/nfs/public
#mount -t nfs NFS_Server(IP):/directory /mount/point
```

Example

```
#mount -t nfs 192.168.3.100:/home/public /home/nfs/public
```

Mail

smtpclient is a minimal SMTP client that takes an email message body and passes it on to an SMTP server. It is suitable for applications that use email to send alert messages or important logs to a specific user.

NOTE Click on the following link for more information about smtpclient:
http://www.engelschall.com/sw/smtpclient/

To send an email message, use the 'smtpclient' utility, which uses SMTP protocol. Type **`#smtpclient -help`** to see the help message.

Example:

```
smtpclient -s test -f sender@company.com -S IP_address receiver@company.com
< mail-body-message
```

- **`-s:`** The mail subject.
- **`-f:`** Sender's mail address
- **`-S:`** SMTP server IP address

The last mail address **receiver@company.com** is the receiver's e-mail address.
mail-body-message is the mail content. The last line of the body of the message should contain ONLY the period '.' character.

You will need to add your hostname to the file **/etc/hosts**.

Installing Net-SNMP

The IA3341 supports the Net-SNMP daemon. However, it has not been included in the default value, so you need to install it manually if you need to use this function. Make sure you have enough memory space available to install Net-SNMp, which will occupy about 3 MB on your embedded flash ROM. The IA3341 has the SNMP V1 (Simple Network Management Protocol) agent software built in. SNMP V1 supports RFC1317 RS-232 like groups and RFC 1213 MIB-II.

Step 1: Check to make sure you have enough memory space.

```
192.168.3.127   - Putty
root@Moxa:~#
root@Moxa:~# df –h
Filesystem                Size      Used Available Use% Mounted on
/dev/root                 8.0M      5.7M      2.3M  71% /
/dev/ram3              1003.0K      9.0K    943.0K   1% /dev
/dev/ram0               499.0K     18.0K    456.0K   4% /var
 /dev/mtdblock3           6.0M    492.0K      5.5M   8% /tmp
/dev/mtdblock3            6.0M    492.0K      5.5M   8% /home
/dev/mtdblock3            6.0M    492.0K      5.5M   8% /etc
tmpfs                    30.5M         0     30.5M   0% /dev/shm
 root@Moxa:~#
```

Check that /dev/mtdblock3 has more than 3.5 MB of free space.

Step 2: Type **upramdisk** to get the free space ram disk to save the package.

```
192.168.3.127 - Putty
root@Moxa:~# upramdisk
root@Moxa:~# df –h
Filesystem          Size      Used Available Use% Mounted on
/dev/root           8.0M      5.7M      2.3M  71% /
/dev/ram3        1003.0K      9.0K    943.0K   1% /dev
/dev/ram0         499.0K     18.0K    456.0K   4% /var
/dev/mtdblock   3 6.0M      492.0K      5.5M   8% /tmp
/dev/mtdblock   3 6.0M      492.0K      5.5M   8% /home
/dev/mtdblock   3 6.0M      492.0K      5.5M   8% /etc
tmpfs              30.5M         0     30.5M   0% /dev/shm
/dev/ram1          16.0M      1.0K     15.1M   0% /var/ramdisk
root@Moxa:~#
```

Step 3: Download the Net-SNMP package from the CD-ROM. You can find the package in CD-ROM/target/net-snmp/Net-SNMP.tar.gz

```
192.168.3.127 – PuTTY
root@Moxa:/bin# cd /mnt/ramdisk
root@Moxa:/mnt/ramdisk# ftp 192.168.27.130
Connected to 192.168.27.130.
220 (vsFTPd 2.0.1)
Name (192.168.27.130:root): root
331 Please specify the password.
Password:
230 Login successful.
```

```
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> cd /tmp
250 Directory successfully changed.
ftp> bin
200 Switching to Binary mode.
ftp> get Net-SNMP.tgz
local: Net-SNMP.tgz remote: Net-SNMP.tgz
200 PORT command successful. Consider using PASV.
150 Opening BINARY mode data connection for Net-SNMP.tgz (3019282 bytes).
226 File send OK.
3019282 bytes received in 2.35 secs (1.3e+03 Kbytes/sec)
```

Step 4: Type command **tar xvzf Net-Snmp.tgz** to untar the package.

```
192.168.3.127 - PuTTY
root@Moxa:/mnt/ramdisk# tar xvzf Net-SNMP.tgz
Net-SNMP/
Net-SNMP/bin/
Net-SNMP/bin/net-snmp-config
Net-SNMP/bin/snmpgetnext
Net-SNMP/bin/snmpvacm
Net-SNMP/bin/snmpbulkwalk
Net-SNMP/bin/snmpcheck
Net-SNMP/bin/snmpusm
Net-SNMP/bin/snmpget
Net-SNMP/bin/snmpbulkget
Net-SNMP/bin/snmpset
Net-SNMP/bin/mib2c
Net-SNMP/bin/snmptranslate
Net-SNMP/bin/traptoemail
Net-SNMP/bin/ipf-mod.pl
Net-SNMP/bin/snmptable
Net-SNMP/bin/snmpstatus
Net-SNMP/bin/snmpnetstat
Net-SNMP/bin/snmpinform
Net-SNMP/bin/snmpdf
Net-SNMP/bin/snmpwalk
Net-SNMP/bin/tkmib
Net-SNMP/bin/snmpconf
Net-SNMP/bin/snmpdelta
Net-SNMP/bin/snmptrap
Net-SNMP/bin/snmptest
Net-SNMP/bin/fixproc
Net-SNMP/bin/encode keychange
Net-SNMP/install.sh
Net-SNMP/EXAMPLE.conf
Net-SNMP/sbin/
Net-SNMP/sbin/snmptrapd
Net-SNMP/sbin/snmpd
```

Step 5: Run **install.sh** and select to install the snmp daemon.

```
192.168.3.127 - PuTTY
root@Moxa:/mnt/ramdisk/Net-SNMP# ./install.sh

Press the number:
1. Install Net-Snmp package
2. Uninstall Net-Snmp package
3. Exit.1
root@Moxa:/mnt/ramdisk#
```

Step 6: Run the command "snmpd -c /etc/snmpd/snmpd.conf" to wake up snmp daemon.

```
192.168.3.127 - PuTTY
root@Moxa:/mnt/ramdisk/Net-SNMP# ./install.sh

Press the number:
1. Install Net-Snmp package
2. Uninstall Net-Snmp package
3. Exit.1
root@Moxa:/mnt/ramdisk#
```

Step 7: Use snmp-client to query from target for testing.

The following simple example allows you to use an snmpwalk program on the host site to query the IA3341, which is the SNMP agent. The IA3341 will respond.

Uage like: snmpwalk –v 1 –c public TARGET IP .

***** SNMP QUERY STARTED *****

1: sysDescr.0 (octet string) Version 1.0
2: sysObjectID.0 (object identifier) enterprises.8691.12.240
3: sysUpTime.0 (timeticks) 0 days 03h:50m:11s.00th (1381100)
4: sysContact.0 (octet string) Moxa Systems Co., LDT.
5: sysName.0 (octet string) Moxa
6: sysLocation.0 (octet string) Unknown
7: sysServices.0 (integer) 6
8: ifNumber.0 (integer) 6
9: ifIndex.1 (integer) 1
10: ifIndex.2 (integer) 2
11: ifIndex.3 (integer) 3
12: ifIndex.4 (integer) 4
13: ifIndex.5 (integer) 5
14: ifIndex.6 (integer) 6
15: ifDescr.1 (octet string) eth0
16: ifDescr.2 (octet string) eth1
17: ifDescr.3 (octet string) Serial port 0
18: ifDescr.4 (octet string) Serial port 1
19: ifDescr.5 (octet string) Serial port 2
20: ifDescr.6 (octet string) Serial port 3

...
...
...
...
...
...
...
...
...

502: snmpInGenErrs.0 (counter) 0
503: snmpInTotalReqVars.0 (counter) 503
504: snmpInTotalSetVars.0 (counter) 0
505: snmpInGetRequests.0 (counter) 0
506: snmpInGetNexts.0 (counter) 506
507: snmpInSetRequests.0 (counter) 0
508: snmpInGetResponses.0 (counter) 0
509: snmpInTraps.0 (counter) 0
510: snmpOutTooBigs.0 (counter) 0
511: snmpOutNoSuchNames.0 (counter) 0
512: snmpOutBadValues.0 (counter) 0
513: snmpOutGenErrs.0 (counter) 0

```
514: snmpOutGetRequests.0 (counter) 0
515: snmpOutGetNexts.0 (counter) 0
516: snmpOutSetRequests.0 (counter) 0
517: snmpOutGetResponses.0 (counter) 517
518: snmpOutTraps.0 (counter) 0
519: snmpEnableAuthenTraps.0 (integer) disabled(2)
***** SNMP QUERY FINISHED *****
```

NOTE Click on the following links for more information about MIB II and RS-232 like groups:
http://www.faqs.org/rfcs/rfc1213.html
http://www.faqs.org/rfcs/rfc1317.html

The IA3341 does NOT support SNMP trap.

5

Development Tool Chains

This chapter describes how to install a tool chain in the host computer that you use to develop your applications. In addition, the process of performing cross-platform development and debugging are also introduced. For clarity, the IA3341 embedded computer is called a target computer.

The following functions are covered in this chapter:

- **Linux Tool Chain**
  - Steps for Installing the Linux Tool Chain
  - Compilation for Applications
  - On-Line Debugging with GDB

Linux Tool Chain

The Linux tool chain contains a suite of cross compilers and other tools, as well as the libraries and header files that are necessary to compile your applications. These tool chain components must be installed in your host computer (PC) running Linux. We have confirmed that the following Linux distributions can be used to install the tool chain.

Fefora core 1 & 2.

Steps for Installing the Linux Tool Chain

The tool chain needs about 485 MB of hard disk space. To install it, follow the steps.

1. Insert the package CD into your PC and then issue the following commands:
   ```
   #mount /dev/cdrom /mnt/cdrom
   #sh /mnt/cdrom/tool-chain/linux/install.sh
   ```
2. Wait for the installation process to complete. This should take a few minutes.
3. Add the directory **/usr/local/arm-linux/bin** to your path. You can do this for the current login by issuing the following commands:
   ```
   #export PATH="/usr/local/arm-linux/bin:$PATH"
   ```
   Alternatively, you can add the same commands to **$HOME/.bash_profile** to make it effective for all login sessions.

Compilation for Applications

To compile a simple C application, use the cross compiler instead of the regular compiler:

```
#arm-linux-gcc  -o example -Wall -g -O2 example.c
#arm-linux-strip  -s example
#arm-linux-gcc  -ggdb -o example-debug example.c
```

Most of the cross compiler tools are the same as their native compiler counterparts, but with an additional prefix that specifies the target system. In the case of x86 environments, the prefix is **`i386-linux-`** and in the case of IA3341 ARM boards, it is **`arm-linux-`**.

For example, the native C compiler is **`gcc`** and the cross C compiler for ARM in the IA3341 is **`arm-linux-gcc.`**

The following cross compiler tools are provided:

ar	Manages archives (static libraries)
as	Assembler
c++, g++	C++ compiler
cpp	C preprocessor
gcc	C compiler
gdb	Debugger
ld	Linker
nm	Lists symbols from object files
objcopy	Copies and translates object files
objdump	Displays information about object files
ranlib	Generates indexes to archives (static libraries)
readelf	Displays information about ELF files

size	Lists object file section sizes
strings	Prints strings of printable characters from files (usually object files)
strip	Removes symbols and sections from object files (usually debugging information)

On-Line Debugging with GDB

The tool chain also provides an on-line debugging mechanism to help you develop your program. Before performing a debugging session, add the option **-ggdb** to compile the program. A debugging session runs on a client-server architecture on which the server **`gdbserver`** is installed int the targe computer and the client **ddd** is installed in the host computer. We'll asuumne that you have uploaded a program named **hello-debug** to the target computer and strat to debug the program.

1. Log on to the target computer and run the debugging server program.

   ```
   #gdbserver 192.168.4.142:2000 hello-debug
   Process hello-debug created; pid=38
   ```

   The debugging server listens for connections at network port 2000 from the network interface 192.168.4.142. The name of the program to be debugged follows these parameters. For a program requiring arguments, add the arguments behind the program name.

2. In the host computer, change the directory to where the program source resides**.**

   ```
   cd /my_work_directory/myfilesystem/testprograms
   ```

3. Execute the client program.

   ```
   #ddd --debugger arm-linux-gdb hello-debug &
   ```

4. Enter the following command at the GDB, DDD command prompt.

   ```
   Target remote 192.168.4.99:2000
   ```

   The command produces a line of output on the target console, similar to the following.

   ```
   Remote debugging using 192.168.4.99:2000
   ```

   192.168.4.99 is the machine's IP address, and 2000 is the port number. You can now begin debugging in the host environment using the interface provided by DDD.

5. Set a break point on main by double clicking, or by entering **`b main`** on the command line.
6. Click the **cont** button.

6

Programmer's Guide

This chapter includes important information for programmers.

The following functions are covered in this chapter:

- **Before Programming Your Embedded System**
  - Caution Required when Using File Systems
  - Using a RAM File System instead of a Flash File System
- **Flash Memory Map**
- **Device API**
- **RTC (Real Time Clock)**
- **Buzzer**
- **WDT (Watch Dog Timer)**
- **UART**
  - Example to set the baudrate
  - Example to get the baudrate
  - Baudrate error
  - Special Note
- **Digital I/O**
  - Application Programming Interface
  - Special Note
  - Example
- **Modbus**

Before Programming Your Embedded System

Caution Required when Using File Systems

We recommend that you only store your programs on the on-board NOR Flash. The log data generated by your programs should be stored on an external storage device, such as an SD card or Network File System. Note that a Network File System will generally provide the largest amount of storage space. In addition, it is easier to replace a full or damaged SD card than an on-board NOR Flash.

A NOR Flash has a life cycle of 100,000 write operations in the block (128 KB) level, but does not support BBM (Bad Block Management). An SD card also has a life cycle, but most SD cards are made from a NAND Flash, for which the hardware controllers implement BBM. This feature allows FAT to skip bad blocks if they exist. Furthermore, the memory space of an SD card is much larger than that of the NOR Flash. Cautiously utilizing this space will ensure that its life cycle will not be exceeded. When creating a file for storing log data, we suggest setting up your program to create a large empty file (e.g., 30 MB), and then write data evenly over the space. When reaching the end of the space, the program rewinds the write operations. As a result, the number of write operations on each block will be reduced.

Using a RAM File System instead of a Flash File System

Although data in the RAM file system will be wiped out after a power off, this file system has several advantages over a Flash file system. The RAM file system includes faster read/write access, and has no life cycle issues.

For timely and/or important applications that relay data directly back to the host, you should write the necessary log data to the RAM file system. After the host accesses the data, the application will erase the data to free up the space for further uses.

The embedded computer has limited resources, and for this reason, designers should determine if storing data in a file system is really necessary. If it is necessary, then be sure to choose the most appropriate file system for your application.

Flash Memory Map

Partition sizes are hard coded into the kernel binary. To change the partition sizes, you will need to rebuild the kernel. The flash memory map is shown in the following table.

Address	Size	Contents
0x00000000 – 0x0003FFFF	256 KB	Boot Loader—Read ONLY
0x00040000 – 0x001FFFFF	1.8 MB	Kernel object code—Read ONLY
0x00200000 – 0x009FFFFF	8 MB	Root file system (JFFS2) —Read ONLY
0x00A00000 – 0x00FFFFFF	6 MB	User directory (JFFS2) —Read/Write

Device API

The IA3341 supports control devices with the **ioctl** system API. You will need to **`include <moxadevice.h>`**, and use the following **ioctl** function.

```
int ioctl(int d, int request,…);
 Input: int d - open device node return file handle
 int request – argument in or out
```

Use the desktop Linux's man page for detailed documentation:

```
#man ioctl
```

RTC (Real Time Clock)

The device node is located at **/dev/rtc**. The IA3341 supports Linux standard simple RTC control. You must **include <linux/rtc.h>**.

1. Function: RTC_RD_TIME

   ```
   int ioctl(fd, RTC_RD_TIME, struct rtc_time *time);
   ```

   Description: read time information from RTC. It will return the value on argument 3.

2. Function: RTC_SET_TIME

   ```
   int ioctl(fd, RTC_SET_TIME, struct rtc_time *time);
   ```

   Description: set RTC time. Argument 3 will be passed to RTC.

Buzzer

The device node is located at **/dev/console**. The IA3341 supports Linux standard buzzer control, with the IA3341's buzzer running at a fixed frequency of 100 Hz. You must **include <sys/kd.h>**.

Function: KDMKTONE

```
ioctl(fd, KDMKTONE, unsigned int arg);
```

Description: The buzzer's behavior is determined by the argument **arg**. The "high word" part of arg gives the length of time the buzzer will sound, and the "low word" part gives the frequency.

The buzzer's on / off behavior is controlled by software. If you call the "ioctl" function, you MUST set the frequency at 100 Hz. If you use a different frequency, the system could crash.

WDT (Watch Dog Timer)

1. Introduction

The WDT works like a watch dog function. You can enable it or disable it. When the user enables WDT but the application does not acknowledge it, the system will reboot. You can set the acknowledgement time from a minimum of 50 msec to a maximum of 60 seconds.

2. How the WDT works

The WDT function is disabled when the system boots up. The user application can also enable acknowledgement. When the user does not acknowledge, it will let the system reboot.

Kernel boot
…..
….
User application running and enable user acknowledgement
….
….

3. The user API

The user application must **include <moxadevic.h>**, and **link moxalib.a**. A makefile example is shown below:

```
all:
  arm-linux-gcc –o xxxx  xxxx.c -lmoxalib
  int swtd_open(void)
```

Description:	Open the file handle to control the WDT. If you want to do something you must first to this. And keep the file handle to do other.
Input:	None
Output:	The return value is file handle. If has some error, it will return < 0 value. You can get error from errno().

int swtd_enable(int fd, unsigned long time)

Description:	Enable application WDT And you must do ack after this process.
Input:	int fd - the file handle, from the swtd_open() return value. unsigned long time - The time you wish to ack sWatchDog periodically. You must ack the WDT before timeout. If you do not ack, the system will reboot automatically. The minimal time is 50 msec, the maximum time is 60 seconds. The time unit is msec.
Output:	OK will be zero. The other has some error, to get the error code from errno().

int swtd_disable(int fd)

Description:	Disable the application to acknowledge WDT. And the kernel will it. User does not to do it at periodic.
Input:	int fd - the file handle from swtd_open() return value.
Output:	OK will be zero. The other has some error, to get error code from errno.

int swtd_get(int fd, int *mode, unsigned long *time)

Description:	Get current setting values. mode –1 for user application enable WDT: need to acknowledge. 0 for user application disable WDT: does not need to acknowledge. time – The time period to acknowledge WDT.
Input :	int fd - the file handle from swtd_open() return value. int mode - the function will return the status of user application need to . unsigned long time – the function will return the current time period.
Output:	OK will be zero. The other has some error, to get error code from errno().

int swtd_ack(int fd)

Description:	Acknowledge sWatchDog. When the user application enable sWatchDog. It need to call this function periodically with user predefined time in the application program.
Input:	int fd - the file handle from swtd_open() return value.
Output:	OK will be zero. The other has some error, to get error code from errno().

int swtd_close(int fd)

Description:	Close the file handle.
Input:	int fd - the file handle from swtd_open() return value.
Output:	OK will be zero. The other has some error, to get error code from errno().

4. Special Note

When you "kill the application with -9" or "kill without option" or "Ctrl+c" the kernel will change to auto ack the sWatchDog.

When your application enables the sWatchDog and does not ack, your application may have a logical error, or your application has made a core dump. The kernel will not change to auto ack. This can cause a serious problem, causing your system to reboot again and again.

5. User application example

Example 1:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <moxadevice.h>

int main(int argc, char *argv[])
{
  int fd;

  fd = swtd_open();
  if ( fd < 0 ) {
   printf("Open sWatchDog device fail !\n");
   exit(1);
  }
  swtd_enable(fd, 5000);// enable it and set it 5 seconds
  while ( 1 ) {
   // do user application want to do
   …..
   ….
   swtd_ack(fd);
   …..
   ….
  }
  swtd_close(fd);
  exit(0);
}
```

The makefile is shown below:

```
all:
  arm-linux-gcc –o xxxx xxxx.c –lmoxalib
```

Example 2:

```
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <string.h>
#include <sys/stat.h>
#include <sys/ioctl.h>
#include <sys/select.h>
#include <sys/time.h>
#include <moxadevice.h>

static void mydelay(unsigned long msec)
{
  struct timeval time;

  time.tv_sec = msec / 1000;
  time.tv_usec = (msec % 1000) * 1000;
  select(1, NULL, NULL, NULL, &time);
}
```

```
static int swtdfd;
static int stopflag=0;

static void stop_swatchdog()
{
 stopflag = 1;
}

static void do_swatchdog(void)
{
 swtd_enable(swtdfd, 500);
 while ( stopflag == 0 ) {
  mydelay(250);
  swtd_ack(swtdfd);
 }
 swtd_disable(swtdfd);
}

int main(int argc, char *argv[])
{
 pid_t  sonpid;

 signal(SIGUSR1, stop_swatchdog);
 swtdfd = swtd_open();
 if ( swtdfd < 0 ) {
  printf("Open sWatchDog device fail !\n");
  exit(1);
 }
 if ( (sonpid=fork()) == 0 )
  do_swatchdog();
 // do user application main function
 …..
 …..
 …..
 // end user application
 kill(sonpid, SIGUSR1);
 swtd_close(swtdfd);
 exit(1);
}
```

The makefile is shown below:

```
all:
 arm-linux-gcc –o xxxx xxxx.c –lmoxalib
```

UART

The normal tty device node is located at `/dev/ttyM0 … ttyM3`.

The IA3341 supports Linux standard termial control. The Moxa UART Device API allows you to configure ttyM0 to ttyM3 as RS-232, RS-422, 4-wire RS-485, or 2-wire RS-485. IA3341 supports RS-232, RS-422, 2-wire RS-485, and 4-wire RS-485.

You must **`include <moxadevice.h>`**.

```
#define RS232_MODE    0
#define RS485_2WIRE_MODE  1
#define RS422_MODE    2
#define RS485_4WIRE_MODE  3
```

1. Function: MOXA_SET_OP_MODE

```
int ioctl(fd, MOXA_SET_OP_MODE, &mode)
```

Description
Set the interface mode. Argument 3 mode will pass to the UART device driver and change it.

2. Function: MOXA_GET_OP_MODE

```
int ioctl(fd, MOXA_GET_OP_MODE, &mode)
```

Description
Get the interface mode. Argument 3 mode will return the interface mode.

There are two Moxa private ioctl commands for setting up special baudrates.

Function: MOXA_SET_SPECIAL_BAUD_RATE
Function: MOXA_GET_SPECIAL_BAUD_RATE

If you use this ioctl to set a special baudrate, the termios cflag will be B4000000, in which case the B4000000 define will be different. If the baudrate you get from termios (or from calling tcgetattr()) is B4000000, you must call ioctl with MOXA_GET_SPECIAL_BAUD_RATE to get the actual baudrate.

Example to set the baudrate

```
#include <moxadevice.h>
#include <termios.h>
struct termios term;
int  fd, speed;
fd = open("/dev/ttyM0", O_RDWR);
tcgetattr(fd, &term);
term.c_cflag &= ~(CBAUD | CBAUDEX);
term.c_cflag |= B4000000;
tcsetattr(fd, TCSANOW, &term);
speed = 500000;
ioctl(fd, MOXA_SET_SPECIAL_BAUD_RATE, &speed);
```

Example to get the baudrate

```
 #include <moxadevice.h>
 #include <termios.h>
 struct termios term;
 int  fd, speed;
 fd = open("/dev/ttyM0", O_RDWR);
 tcgetattr(fd, &term);
if ( (term.c_cflag & (CBAUD|CBAUDEX)) != B4000000 )
{// follow the standard termios baud rate define} else
{ioctl(fd, MOXA_GET_SPECIAL_BAUD_RATE, &speed);}
```

Baudrate error

Divisor = 921600/Target Baud Rate. (Only Integer part)
ENUM = 8 * (921600/Targer - Divisor) (Round up or down)
Inaccuracy = (Target Baud Rate – 921600/(Divisor + (ENUM/8))) * 100%

E.g.,

To calculate 500000 bps
Divisor = 1, ENUM = 7,
Error = 1.7%
*The error should less than 2% for reliable data transmission.

Special Note

1. If the target baudrate is not a special baudrate (e.g. 50, 75, 110, 134, 150, 200, 300, 600, 1200, 1800, 2400, 4800, 9600, 19200, 38400, 57600, 115200, 230400, 460800, 921600), the termios cflag will be set to the same flag.
2. If you use stty to get the serial information, you will get speed equal to 0.

Digital I/O

Digital Output channels can be set to high or low and are controlled by the function call set_dout_state( ). The digital input channels can be used to detect the state change of the digital input signal. The DI channels can also be used to detect whether or not the state of a digital signal changes during a fixed period of time. This can be done with the function call set_din_event( ).

Moxa provides five function calls to handle the digital I/O state change and events.

Application Programming Interface

Return error code definitions:

```
#define DIO_ERROR_PORT -1 // no such port
#define DIO_ERROR_MODE -2  // no such mode or state
#define DIO_ERROR_CONTROL -3 // open or ioctl fail
#define DIO_ERROR_DURATION -4 // The value of duration is not 0 or not in the range, 40 <= duration <= 3600000 milliseconds (1 hour)
#define DIO_ERROR_DURATION_20MS -5 // The value of duration must be a multiple of 20 ms
#define DIO_OK 0
```

The definition for DIN and DOUT:

```
#define DIO_HIGH 1
#define DIO_LOW 0
```

int set_dout_state(int doport, int state)

Description: To set the DOUT port to high or low state.

Input: int doport - which DOUT port you want to set. Port starts from 0 to 3.

int state - to set high or low state; DIO_HIGH (1) for high, DIO_LOW (0) for low.

Output: none.

Return: reference the error code.

int get_din_state(int diport, int *state)

Description: To get the DIN port state.

Input: int diport - get the current state of which DIN port. Port numbering is from 0 to 3.

int *state - save the current state.

Output: state - DIO_HIGH (1) for high, DIO_LOW (0) for low.

Return: reference the error code.

int get_dout_state(int doport, int *state)

Description: To get the DOUT port state.

Input: int doport - get the current state of which DOUT port.

int *state - save the current state.

Output: state - DIO_HIGH (1) for high, DIO_LOW (0) for low.

Return: reference the error code.

int set_din_event(int diport, void (*func)(int diport), int mode, long int duration)

Description: Set the event for DIN when the state is changed from high to low or from low to high.

Input: int diport - the port that will be used to detect the DIN event.

Port numbering is from 0 to 3.

void (*func) (int diport) - Not NULL

> Returns the call back function. When the event occurs, the call back function will be invoked.

NULL

> Clears this event

int mode DIN_EVENT_HIGH_TO_LOW

(1): from high to low

DIN_EVENT_LOW_TO_HIGH

(0): from low to high

DIN_EVENT_CLEAR

(-1): clear this event

unsigned long duration - 0: detect the din event > DIN_EVENT_HIGH_TO_LOW or DIN_EVENT_LOW_TO_HIGH> without duration

- Not 0

> detect the din event

DIN_EVENT_HIGH_TO_LOW or

DIN_EVENT_LOW_TO_HIGH with

duration. The value of "duration" must be a

multiple of 20 milliseconds. The range of

"duration" is 0, or 40 <= duration <= 3600000

milliseconds. The error of the measurement is

24 ms. For example, if the DIN duration is

200 ms, this event will be generated when the

DIN pin stays in the same state for a time

between 176 ms and 200 ms.

Output: none.

Return: reference the error code.

int get_din_event(int diport, int *mode, long int *duration)

Description: To retrieve the DIN event configuration, including mode

(DIN_EVENT_HIGH_TO_LOW or DIN_EVENT_LOW_TO_HIGH), and the value of "duration."

Input: int diport - which DIN port you want to retrieve.

- The port whose din event setting we wish to retrieve

int *mode - save which event is set.

unsigned long *duration - the duration of the DIN port is kept in high or low state.

- return to the current duration value of diport

Output: mode DIN_EVENT_HIGH_TO_LOW

(1): from high to low

DIN_EVENT_LOW_TO_HIGH(0): from low to high

DIN_EVENT_CLEAR(-1): clear this event

duration The value of duration should be 0 or 40 <= duration

<= 3600000 milliseconds.

Return: reference the error code.

Special Note

Don't forget to link the library libmoxalib for DI/DO programming, and also include the header file moxadevice.h. The DI/DO library only can be used by one program at a time.

Examples

DIO Program Source Code File Example

File Name: tdio.c

Description: The program indicates to connect DO1 to DI1, change the digital output state to high or low by manual input, then detect and count the state changed events from DI1.

```
#include <stdio.h>
#include <stdlib.h>
#include <moxadevice.h>
#include <fcntl.h>
#ifdef DEBUG
#define dbg_printf(x...) printf(x)
#else
#define dbg_printf(x...)
#endif
#define MIN_DURATION 40
static char *DataString[2]={"Low ", "High "};
static void hightolowevent(int diport)
{
printf("\nDIN port %d high to low.\n", diport);
}
static void lowtohighevent(int diport)
{
printf("\nDIN port %d low to high.\n", diport);
}
int main(int argc, char * argv[])
{
int i, j, state, retval;
```

```
unsigned long duration;
while( 1 ) {
printf("\nSelect a number of menu, other key to exit. \n\
1. set high to low event \n\
2. get now data. \n\
3. set low to high event \n\
4. clear event \n\
5. set high data. \n\
6. set low data. \n\
7. quit \n\
8. show event and duration \n\
Choose : ");
retval =0;
scanf("%d", &i);
if ( i == 1 ) { // set high to low event
printf("Please keyin the DIN number : ");
scanf("%d", &i);
printf("Please input the DIN duration, this minimun value must be over %d : ", MIN_DURATION);
scanf("%lu", &duration);
retval=set_din_event(i, hightolowevent, DIN_EVENT_HIGH_TO_LOW, duration);
} else if ( i == 2 ) { // get now data
printf("DIN data : ");
for ( j=0; j<4; j++ ) {
get_din_state(j, &state);
printf("%s", DataString[state]);
}
printf("\n");
printf("DOUT data : ");
for ( j=0; j<MAX_DOUT_PORT; j++ ) {
get_dout_state(j, &state);
printf("%s", DataString[state]);
}
printf("\n");
} else if ( i == 3 ) { // set low to high event
printf("Please keyin the DIN number : ");
scanf("%d", &i);
printf("Please input the DIN duration, this minimun value must be over %d :", MIN_DURATION);
scanf("%lu", &duration);
retval = set_din_event(i, lowtohighevent, DIN_EVENT_LOW_TO_HIGH, duration);
} else if ( i == 4 ) { // clear event
printf("Please keyin the DIN number : ");
```

```
scanf("%d", &i);
retval=set_din_event(i, NULL, DIN_EVENT_CLEAR, 0);
} else if ( i == 5 ) { // set high data
printf("Please keyin the DOUT number : ");
scanf("%d", &i);
retval=set_dout_state(i, 1);
} else if ( i == 6 ) { // set low data
printf("Please keyin the DOUT number : ");
scanf("%d", &i);
retval=set_dout_state(i, 0);
} else if ( i == 7 ) { // quit
break;
} else if ( i == 8 ) { // show event and duration
printf("Event:\n");
for ( j=0; j<MAX_DOUT_PORT; j++ ) {
retval=get_din_event(j, &i, &duration);
switch ( i ) {
case DIN_EVENT_HIGH_TO_LOW :
printf("(htl,%lu)", duration);
break;
case DIN_EVENT_LOW_TO_HIGH :
printf("(lth,%lu)", duration);
break;
case DIN_EVENT_CLEAR :
printf("(clr,%lu)", duration);
break;
default :
printf("err " );
break;
}
}
printf("\n");
} else {
printf("Select error, please select again !\n");
}
switch(retval) {
case DIO_ERROR_PORT:
printf("DIO error port\n");
break;
case DIO_ERROR_MODE:
printf("DIO error mode\n");
```

```
break;
case DIO_ERROR_CONTROL:
printf("DIO error control\n");
break;
case DIO_ERROR_DURATION:
printf("DIO error duratoin\n");
case DIO_ERROR_DURATION_20MS:
printf("DIO error! The duratoin is not a multiple of 20 ms\n");
break;
}
}
return 0;
}
```

DIO Program Make File Example

```
FNAME=tdio
CC=arm-linux-gcc
STRIP=arm-linux-strip
release:
$(CC) -o $(FNAME) $(FNAME).c -lmoxalib -lpthread
$(STRIP) -s $(FNAME)
debug:
$(CC) -DDEBUG -o $(FNAME)-dbg $(FNAME).cxx -lmoxalib -lpthread
clean:
/bin/rm -f $(FNAME) $(FNAME)-dbg *.o
```

Modbus

The IA3341-LX provides two AI (Analog Input) channels and two TC (Thermocouple Input) channels. The AI channels support voltage and current modes. The TC channels support J, K, T, E, R, S, B and N sensor types. The AI and TC channels are accessible through MODBUS TCP protocol or supported MODBUS TCP client API functions. A built-in MODBUS TCP gateway server in the IA3341-LX provides MODBUS TCP protocol service for AI/TC channels.

Getting Started

The MODBUS TCP gateway server is started when the IA3341-LX boots up. Once the server is ready, MODBUS TCP clients can communicate with it to access AI/TC channels using supported MODBUS TCP address mappings listed in the following section. The gateway server listens on TCP port 502.

MODBUS TCP Address Mapping

The following addresses of Input Registers and Holding Registers for MODBUS TCP are defined for AI/TC channels. AI/TC channel parameters are defined in Holding Registers and can be read and written. AI/TC channel data are defined in Input Registers and are read only. Each channel can be set to on or off and the former is the default value. The default value is the value used each time the IA3341-LX is cold booted. An AI value in C/C++ float data type is defined at two continuous MODBUS TCP addresses. The first address defines the high word of the AI value and the second defines the low word of the AI value. The data unit of an AI value depends on its corresponding

AI mode setting. TC values are C/C++ short data types and in 0.1 data unit of Celsius or Fahrenheit. A special TC value of 65535 (0xFFFF), which is -1 in C/C++ short data type, denotes that nothing is connected to the TC channel while the TC channel is set to on. When the channel is off, its channel value is meaningless. CJC (Cold-Junction Compensation) value could be read via address 0x0006 of the Input Register and updated using CJC offset via address 0x0010 of the Holding Register. The CJC value and offset value are in Celsius. A positive CJC offset value will increment the CJC value and a negative one will decrement it. The CJC offset value is in C/C++ short data type to represent positive and negative values from -128 to 127. Once the CJC value is updated via the CJC offset value, its effect is permanent. The CJC offset value gotten from MODBUS TCP is reset at each boot.

(1) 3xxxx Input (Read Only) Registers (Function Code 4)

Address	Data Type	Description
0x0000	1 16-bit word	Ch0 AI value in C/C++ float data type (Hi)
0x0001	1 16-bit word	Ch0 AI value in C/C++ float data type (Lo)
0x0002	1 16-bit word	Ch1 AI value in C/C++ float data type (Hi)
0x0003	1 16-bit word	Ch1 AI value in C/C++ float data type (Lo)
0x0004	1 16-bit word	Ch0 TC value in C/C++ short data type Unit:0.1 (Celsius or Fahrenheit) If the channel is set on, then its value 65535 (0xFFFF) denotes nothing connected to the channel.
0x0005	1 16-bit word	Ch1 TC value in C/C++ short data type Unit:0.1 (Celsius or Fahrenheit) If the channel is set on, then its value 65535 (0xFFFF) denotes nothing connected to the channel.
0x0006	1 16-bit word	CJC value in Celsius

(2) 4xxx Holding (Read/Write) Registers (Function Code 3, 6/16)

Address	Data Type	Description
0x0000	1 16-bit word	Ch0 AI On/Off 0:Off 1:On (default)
0x0001	1 16-bit word	Ch0 AI mode 0 : voltage mode (default) 1 : current mode
0x0002	1 16-bit word	Ch1 AI On/Off 0:Off 1:On (default)
0x0003	1 16-bit word	Ch1 AI mode 0 : voltage mode (default) 1 : current mode
0x0004	1 16-bit word	Ch0 TC On/Off 0:Off 1:On (default)
0x0005	1 16-bit word	Ch0 TC type 0:J type 1:K type (default) 2:T type 3:E type 4:R type 5:S type 6:B type 7:N type
0x0006	1 16-bit word	Ch0 TC data unit 0: Celsius (default) 1: Fahrenheit
0x0007	1 16-bit word	Ch1 TC On/Off 0:Off 1:On (default)
0x0008	1 16-bit word	Ch1 TC type 0:J type 1:K type (default) 2:T type 3:E type 4:R type 5:S type 6:B type 7:N type
0x0009	1 16-bit word	Ch1 TC data unit 0: Celsius (default) 1: Fahrenheit
0x0010	1 16-bit word	Last CJC offset (Celsius) in C/C++ short data type from -128 to 127 set since booted Positive value to increment CJC and negative value to decrement CJC

Support Ranges

The AI channels support 0-10 V in voltage mode and 4-20 mA in current mode. If the current of the AI channel source is below 4 mA the current value is still sent to MODBUS TCP clients. If AI channels are on but no device is connected to them, a small value close to zero might be sent to MODBUS TCP clients. The TC channels support the following Celsius value ranges for 8 different sensor types.

TC Sensor Type	Minimum Value	Maximum Value
J	0	750
K	-200	1250
T	-200	350
E	-200	900
R	-50	1600
S	-50	1760
B	600	1700
N	-200	1300

MODBUS TCP Client API Functions

To access IA3341-LX channels easily a set of C/C++ MODUBS TCP client API functions are provided. The API functions for IA3341-LX and Windows are both supported. The API functions are implemented in library libmbtcp.a for the IA3341-LX and static library mbtcp.lib and DLL mbtcp.dll for Windows. The declarations of the API functions are in mbapi.h, and all of them are in the IA3341-LX CD. There are six API functions, shown below. The sample program mbapi_demo.c uses the API functions and related files are also in the CD for reference.

Initialize MODBUS TCP connection environment
int mbtcp_init();
Input: none Output: none
Return: 0 for success and negative value for failure

Release MODBUS TCP connection environment
int mbtcp_release();
Input: none Output: none
Return: 0 for success and negative value for failure

Create MODBUS TCP connection handler and connect to MODBUS TCP server
int mbtcp_connect(char *ip, unsigned short port, void **hndl);
Input: <ip> IP address of MODBUS TCP server <port> port number of MODBUS TCP server Output: <hndl> MODBUS TCP connection handler

Return: 0 for success and negative value for failure -1 argument error -2 memory allocation error -3 failed to connect to MODBUS TCP server

Close MODBUS TCP connection and destroy connection handler
int mbtcp_disconnect(void *hndl);
Input: <hndl> MODBUS TCP connection handler Output: none
Return: 0 for success and negative value for failure -1 argument error

Read register values via function code 3 or 4
int mbtcp_read_regs(void *hndl, unsigned int tmout, unsigned char reg_type, unsigned short start_addr, unsigned short count, unsigned short reg_val[]);
Input: <hndl> MODBUS TCP connection handler <tmout> communication timeout in second <reg_type> register type for function code 3 or 4 3: read holding register 4: read input register <start_addr> starting address <count> quantity of registers Output: <reg_val> an array to hold register values, where the array size must be equal to or greater than the requested quantity of registers.
Return: 0 for success, negative value for failure from API, and positive value for failure from MODBUS TCP server -1 argument error -2 failed to send MODBUS TCP request to MODBUS TCP server -3 failed to receive MODBUS TCP response from MODBUS TCP server -4 disconnection from MODBUS TCP server -5 illegal MODBUS TCP response message received -6 unexpected MODBUS TCP response message received 1 bad function code 2 address out of range 3 bad read count 4 internal gateway failure 6 unable to accept the MODBUS request

Write register values via function code 6 or 16
int mbtcp_write_regs(void *hndl, unsigned int tmout, unsigned short start_addr, unsigned short count, unsigned short reg_val[]);
Input: <hndl> MODBUS TCP connection handler <tmout> communication timeout in second <start_addr> starting address <count> quantity of registers <reg_val> an array to hold register values Output: none
Return: 0 for success, negative value for failure from API, and positive value for failure from MODBUS TCP server -1 argument error -2 failed to send MODBUS TCP request to MODBUS TCP server -3 failed to receive MODBUS TCP response from MODBUS TCP server -4 disconnection with MODBUS TCP server -5 illegal MODBUS TCP response message received -6 unexpected MODBUS TCP response message received 1 bad function code 2 address out of range 3 bad write count or bad write value 4 internal gateway failure 6 unable to accept the MODBUS request

Programs that use the API functions must follow the proper usage. To begin with, call mbtcp_init() once to initialize the MODBUS TCP connection environment. Next, call mbtcp_connect() to create a MODBUS TCP connection handler and connect to the MODBUS TCP server. Then, call mbtcp_read_regs() and mbtcp_write_regs() to read register values and write register values, respectively. Before the program exits, call mbtcp_disconnect() to close its MODBUS TCP connection and destroy the specified connection handler, and then call mbtcp_release() to release the MODBUS TCP connection environment. The API functions cannot be used by multiple threads at the same time.