NIC-51240

Quad Copper Gigabit Ethernet

PCI-E Bypass Adapter

User’s Manual

Revision: 1.2

Table of Contents

1. GENERAL INTRODUCTION ..... …..2

1.1 INTRODUCTION 3
1.2 PRODUCT PACKING LIST.....3
1.3 FEATURES.....4
1.4 SPECIFICATIONS 4
1.5 BLOCK DIAGRAM 6

2. HARDWARE INSTALLATION.....7

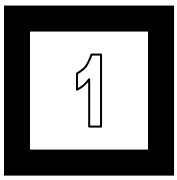
2.1 JUMPERS 8
2.2 CONNECTORS 8
2.3 LOCATING JUMPERS & CONNECTORS 9

3. ETHERNET INTERFACE.....10

3.1 ETHERNET DRIVER SUPPORT 11
3.2 INSTALLATION OF ETHERNET DRIVER ON WINDOWS XP 11
3.3 INSTALLATION OF ETHERNET DRIVER ON LINUX 15

4. PROGRAMMING BYPASS.....16

4.1 INTRODUCTION 17
4.2 COMPILATION 17
4.3 BYPASS UTILITY 20
4.4 PROGRAMMING GUILD 22



General Introduction

1.1 Introduction

The NIC-51240 is PCI Express x4 interface cards, contains four independent Gigabit Ethernet ports. To enhance Ethernet controller performance, it is designed with one Intel® 82580EB Gigabit Ethernet Controller to provide four Gigabit Ethernet ports.

1.2 Product Packing List

Before beginning installing, please make sure the following items have been included in the box.

1. NIC-51240 quad copper Gigabit Ethernet PCI-E bypass adapter
2. Quick Installation Guide

If any of these items is missing or damaged, contact you local dealer from whom you purchased the product.

1.3 Features

- Quad copper PCI-E Gigabit Ethernet ports via Intel® 82580EB controller
- Built-in Watchdog Timer (WDT) to bypass Ethernet ports on a host system hang or power failure
- Easy configuration of Normal/Bypass model and WDT timer
- Built with both onboard LED indicators and LED pin-out for LAN status and bypass mode, provides variable LED location for system integration

1.4 Specifications

Technical Specifications:

- **Standard:** IEEE 802.3z 1000BASE-SX Gigabit Standard; IEEE 802.3x Flow Control
- **Interface:** PCI-Express base specification Rev. 1.1
- **PCI-Express Bus Type**: x4
- **Installbale PCI Slot**: PCI Express x4/x8/x16
- **Controller:** Intel® 82580EB
- **Holder:** Metal bracket for both full height PCI-E x4 slots (metal bracket in low profile is optional)
- **Driver Support:** Windows, Linux, FreeBSD & VMware, from http://downloadcenter.intel.com/

- **Bypass API Sample code:** Linux Stable kernel version 2.6

Mechanical and Environmental:

- **Board Size:** 110 (W) x 100.7 (L) mm
- **Power Consumption:** 4.1W
- **Operating Temperature:** 0 to 60 ℃ (32 to 140℉)
- **Operating Storage:** -20 to 80 ℃ (-68 to 176℉)
- **Operating Humidity:** 5% to 90% RH(non-condensing)
- **Weight:** 77.7g

1.5 Block Diagram

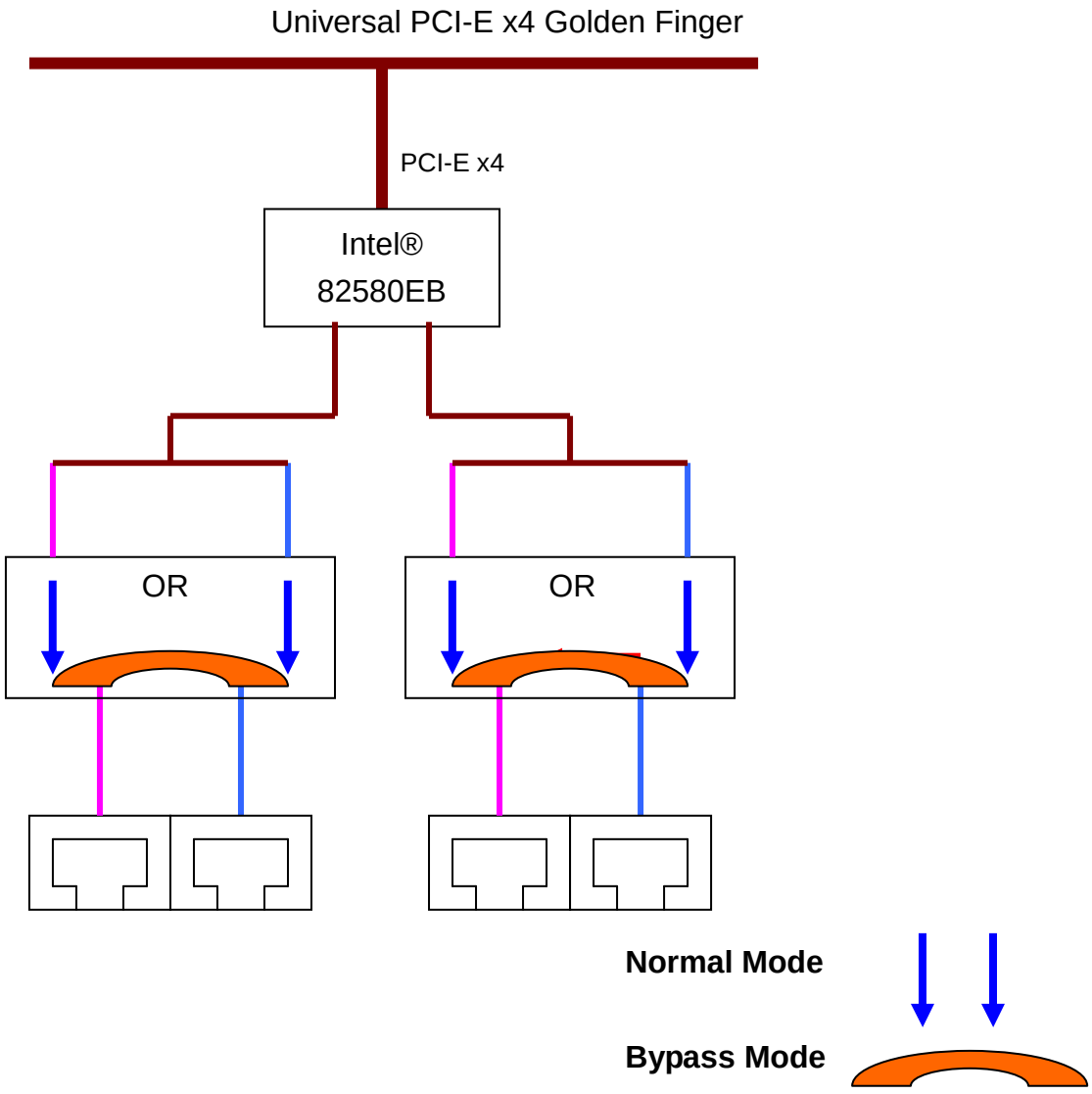


Figure 1.5: NIC-51240 Block Diagram

Hardware Installation

2.1 Jumpers

Label	Function
JP2	Bypass Function and Bypass Function Boot-up Setting for LAN1 / LAN2 (Segment 1)
JP1	Bypass Function and Bypass Function Boot-up Setting for LAN3 / LAN4 (Segment 2)

Bypass Function Boot-up Setting (JP1 / JP2)	
Setting	Function
1-3(default)	Active Bypass Function
3-5	Inactive Bypass Function
2-4	Enable Bypass Function before OS boot-up
4-6(default)	Disable Bypass Function before OS boot-up
Note: To active this function, jumper should be set on Bypass mode	

2.2 Connectors

Label	Function
J2	LAN1 and LAN2 Connector
J3	LAN3 and LAN4 Connector

2.3 Locating Jumpers & Connectors

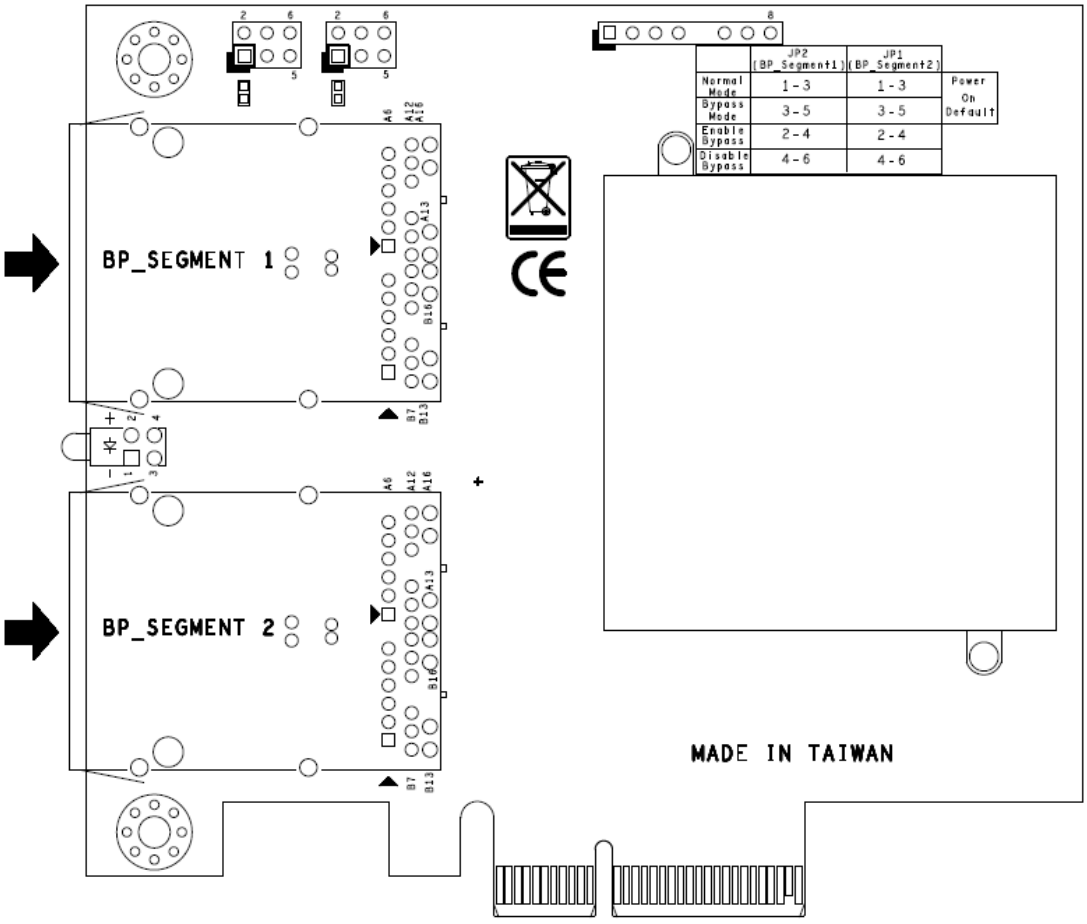


Figure 2.3.1: Jumper & Connector (component side)

Ethernet Interface

3.1 Ethernet Driver Support

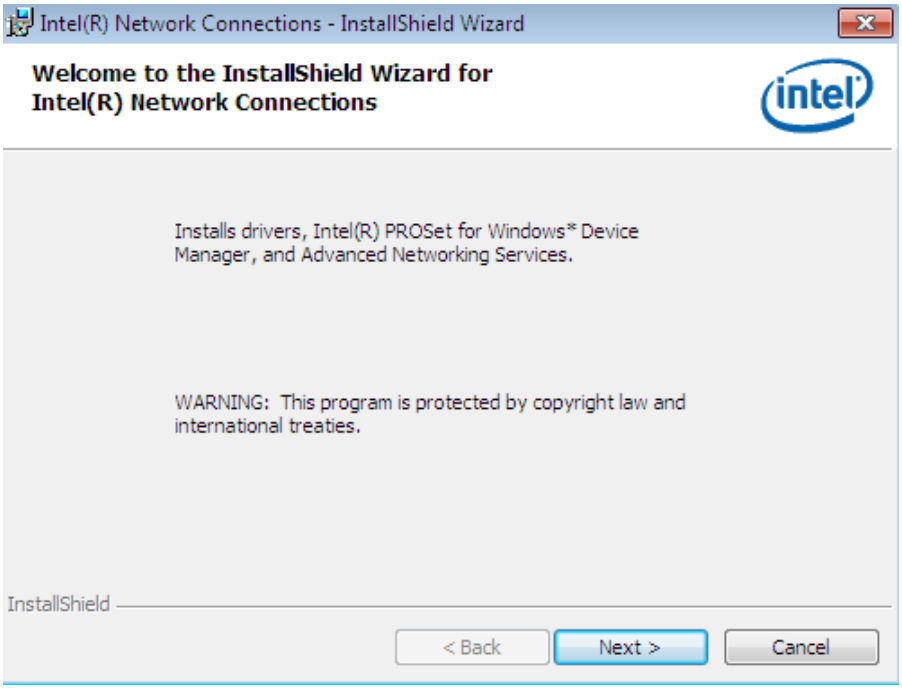
The Ethernet drivers are supported under Windows, Linux, FreeBSD & VMware. For other supported drivers, please contact your distributors or sales representative, or refer to Intel http://downloadcenter.intel.com.

3.2 Installation of Ethernet Driver on Windows XP

The following steps are manual installation for Windows XP

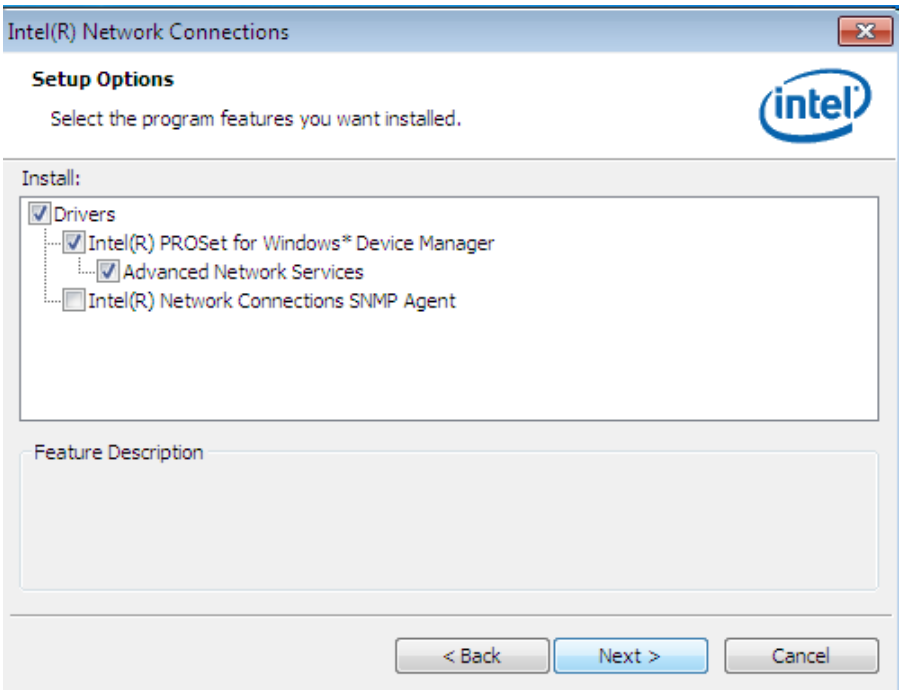
Step 1 Download the latest Intel 82580EB driver at http://downloadcenter.intel.com. If your operating system is 32 Bit, download and execute PROwin32.exe; If your operating system is 64 Bit, download and execute PROwinx64.exe

Step 2 Click “Next” to start installation

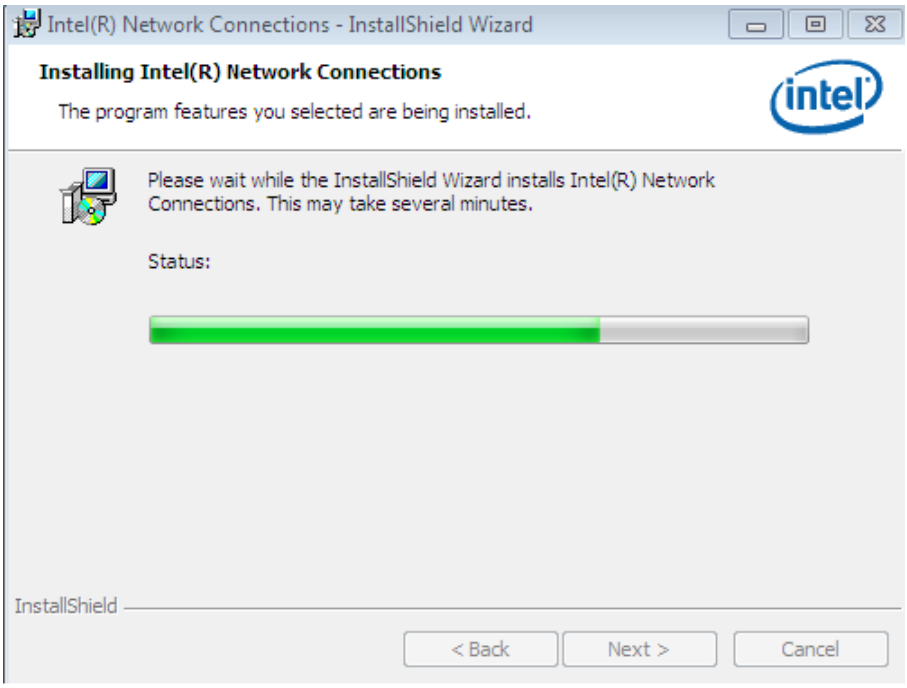


Step 3 Check the agreement box and click “Next”

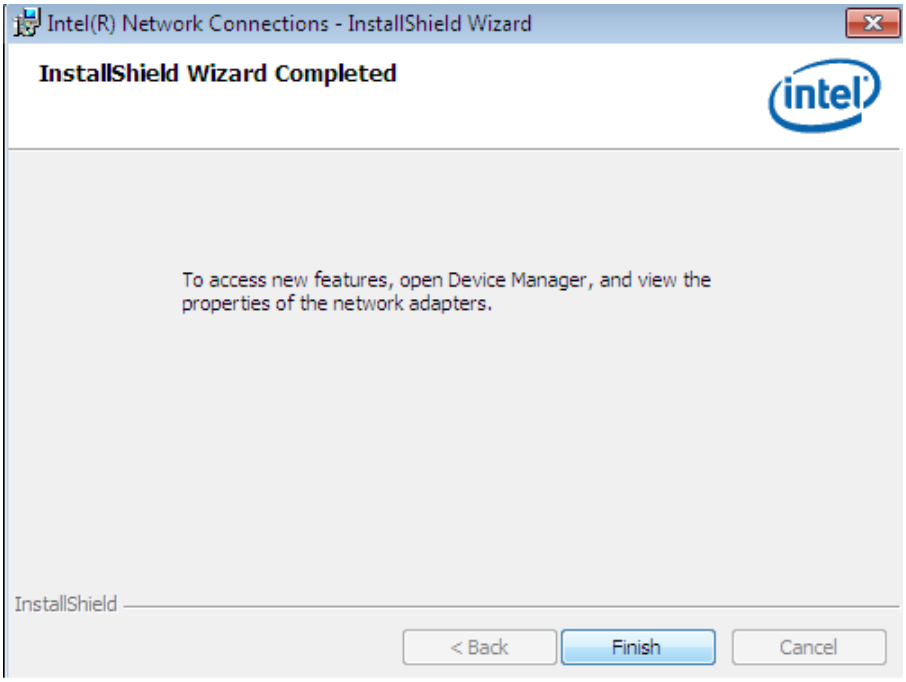
Step 4 Select “Default” and press ”Next” to continue



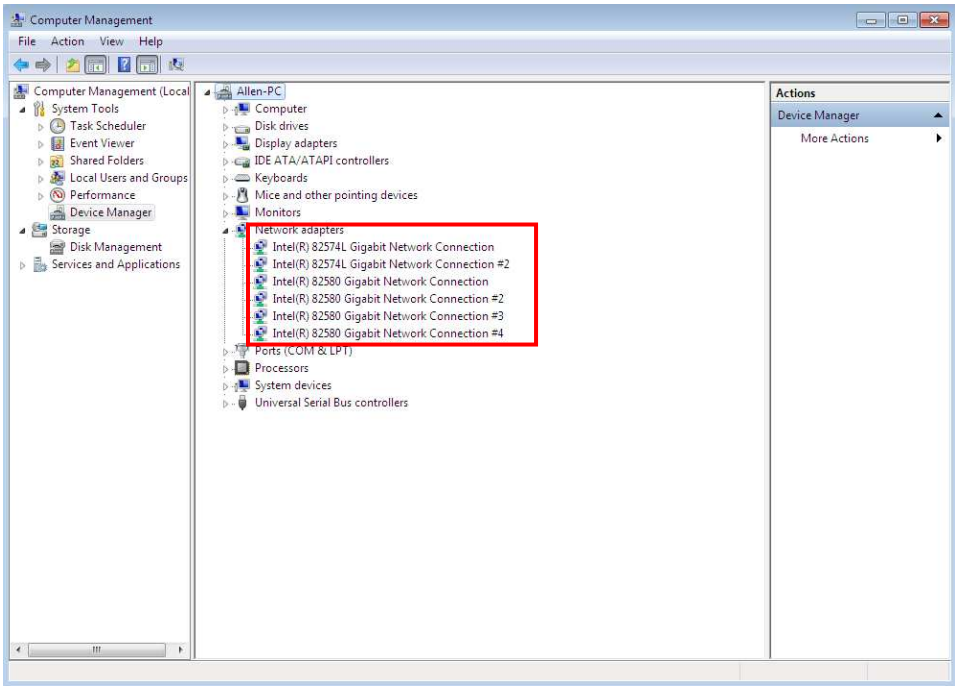
Step 5 Driver installing



Step 6 Click “Finish” to complete the installation



Step 7 Confirm installation successful in Control center > Hardware > device management



3.3 Installation of Ethernet Driver on Linux

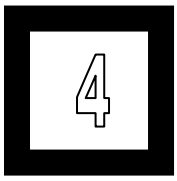
Most of the kernels contain the driver for NIC-51240, and these OS will automatically install the new hardware when booting up. If this doesn't happen, please follow the step to install.

Step 1 Download the latest Intel 82580EB driver at
http://downloadcenter.intel.com

Step 2 Run " tar zxvf igb-x.x.x.tar.gz " to extract
EX: #tar zxvf igb-x.x.x.tar.gz

Step 3 Change to the driver src directory: cd igb-x-x-x/src, and Enter "make install" to compile
EX: #cd igb-x-x-x/src
#make install

Step 4 Run "ifconfig -a" to ensure the installation successful
EX: #ifconfig -a



Programming Bypass

4.1 Introduction

The ‘bypass utility’ provides a user space tool ‘bpcmd’ for user to control bypass device and several API functions for programming. We describe the usage in the following section.

4.2 Compilation

This program has been tested under Fedora core 8 /12 and Red Hat Enterprise 5.4 with gcc version 4.1.2 20070925 (Red Hat 4.1.2-33) ,version 4.4.2 20091027 (Red Hat 4.4.2-7) and version 4.1.2 20080704 (Red Hat 4.1.2-46) (GCC)

All the makefiles are tested with GNU Make 3.81.

4.2.1 Uncompress the package

The user can uncompress the package according to the following command

Step 1: make a directory to place the source code

Ex/

```
[user1@host work]$ mkdir bypass
[user1@host work]$ cd bypass
```

Step 2: Copy the source code to the directory

Ex/

[user1@host work]$cp /mnt/sdb1 .

Step 3: Uncompress the source code

[user1@host work]$ unzip util_bypass-mac-sdp-x.x.x.zip

After uncompression, we can see two directories and several files

./src：All the source codes of bypass API and bpcmd are placed here

./sample：Several sample code developed by API functions can be found here

README：This file provide description for APIs and bpcmd

SUPPORT_LIST：This file provide the NIC card supported by this package

make_all.sh：A shell script that can help to compile all the source code

4.2.2 ./src

The targets of the Makefile are list in as the follows：

libso: Compile the bypass shared liberary and its soft link. The output file should be libbp_macspd.so.XXXX and a soft link libbp_macspd.so.

static_bpcmd: compile the utility 'bpcmd' statically.

all : compile the bypass shared liberary and its soft link.

'bpcmd' is compiled dynamically.
Note this is the first target in the Makefile.

clean: remove all the output files

4.2.2 ./sample

The targets of the Makefile are list in as the follows：
all: compile all the sample codes dynamically
clean: remove all the output files

4.2.3 compile.sh

Execute this script is equivalent to run ‘make clean’ and ‘make’ under ./src and ./sample

4.2.4 Dynamically compiling the source code and sample code

When we execute the executable which is compiled dynamically, the shared library has to set up properly. For example, after compilation

[user1@host src]$ cp –ar libbp_macsdp.so* /lib
[user1@host src]$ldconfig

4.3 Bypass Utility

The program 'bpcmd' is a sample utility written by these APIs. Users can use this utility to test and control bypass device. This program can be compiled in ./src by

[user1@host work]$ make

It will generate an executable which dynamically links libbp_macsdp.so in run time.

Moreover, an static compilation can be done by

[root@localhost src]# make static_bpcmd

A complete executable 'bpcmd' can be found under ./src
We list the complete usage of bpcmd in the following

[SYNOPSIS]

bpcmd <software info command> : -v/-h
bpcmd -i <interface> <command> : -V/-I/-H
bpcmd -i <interface> <command> [option]
bpcmd -i <interface> -s norm / non / read
bpcmd -i <interface> -t read / cl
bpcmd -i <interface> -w 0 / SEC, where SEC is a positive integer.
bpcmd -i <interface> -W SEC

[Comand Description]

-h shows a help message
-v show software version
-i specify network interface relative to bypass device
-s set/read the bypass device:
[norm] set bypass device to normal mode
[non] set bypass device to non-normal mode
[read] read the status of bypass device

-t clear/read the time-out flah
[cl] clear the time-out flag
[read] read the time-out flag
-w control the WDT timer

4.4 Programming Guild

All the APIs are defined in "mac_sdp_init.h" and "bypass_cmd.h". We describe all these APIs in the following context.

4.4.1 Initialize and release function

The initializing and releasing function are defined in "mac_sdp_init.h". When we try to control a bypass device, we have to initialize a variable of structure bp_seg. After all the work done, we have to release the resource of bp_seg (like semaphore).

Function name	init_seg
Description	Initial the structure bp_seg. The bp_seg structure is recorded all required data to control a bypass device and all bypass APIs need this object.
Format:	
int init_seg (bp_seg* seg, char const* iface_name, const unsigned char period_time) Input: seg — the pointer to the structure bp_seg to be initialized iface_name — the interface name of network device period_time — the timeout value for WDT Return: 0: If the initialization is successful	

Function name	release_seg
Description	This function is used to release resource of a bp_seg
Format:	
void release_seg(bp_seg* seg) Input: seg – the pointer to the structure bp_seg to be released	

4.4.2 Command APIs

All the APIs that control bypass device are define in "bypass_cmd.h". All of name require a initialized bp_seg. We list them as the follows/.

Function name	set_bp_normal
Description	Set the bypass device of seg to normal mode.
Format:	
void set_bp_normal(bp_seg* seg) Input: seg — the pointer to the structure bp_seg to be controlled	

Function name	set_bp_normal
Description	Set the bypass device of seg to normal mode.
Format:	
void set_bp_normal(bp_seg* seg) Input: seg — the pointer to the structure bp_seg to be controlled	

Function name	set_bp_nonormal
Description	Set the bypass device of seg to non-normal mode.
Format:	
void set_bp_nonormal (bp_seg* seg) Input: seg — the pointer to the structure bp_seg to be controlled	

Function name	read_bp
Description	Read the status of the bypass device of seg.
Format:	
void set_bp_nonormal (bp_seg* seg) Input: seg — the pointer to the structure bp_seg to be read	

Function name	stop_wdt
Description	Stop the WDT timer.
Format:	
void stop_wdt (bp_seg* seg) Input: seg — the pointer to the structure bp_seg to be controlled	

Function name	set_period
Description	Set the WDT timer with the period value seg->period.
Format:	
int set_period (bp_seg* seg) Input: seg — the pointer to the structure bp_seg to be controlled Return: -EINVAL: when the value of seg->period is invalid. 0: when the value of seg->period is valid.	
Function name	trigger_wdt
Description	Trigger the WDT with previous value. Note this function only issue command to hardware, and does not check the correctness of the period value. Make sure set_period is called at least once.
Format:	
void trigger_wdt (bp_seg* seg) Input: seg — the pointer to the structure bp_seg to be controlled	

Function name	read_timeout_bit
Description	Read the time-out flag
Format:	
int read_timeout_bit (bp_seg* seg) Input: seg — the pointer to the structure bp_seg to be controlled Return: 0: the WDT never time out. 1: the WDT has timed out at least once.	

Function name	clean_timeout_bit
Description	Clear the timeout flag to 0
Format:	
void clean_timeout_bit(bp_seg* seg) Input: seg — the pointer to the structure bp_seg to be controlled	

Function name	read_firmware_ver
Description	Read the firmware version
Format:	
unsigned char read_firmware_ver (bp_seg* seg) Input: seg — the pointer to the structure bp_seg to be controlled Return: 0: invalid firmware version. The firmware does not support this function. 1-254: the firmware version	

4.4.3 Note for Programming

1. Avoid terminating signal
Since these APIs are relative to hardware access, abnormal terminating above functions may cause some serious error. Do not send SIGKILL and SIGSTOP to the process issuing command to bypass device if not necessary.

2. Semaphore Lock
When the process is terminated abnormally, the semaphore to avoid concurrent access of hardware may not be unlocked. If this happens, the system may hang when any other bypass command is issue. Try to remove all the files under /dev/shm with prefix 'sem. Bypass-' and send SIGKILL signal to all the processes which hangs when issuing command to bypass device.