BAPI implementation for ELZET80 IPC–BIT900 PC plug-in boards Under Linux environment

Sergei B. Khvatov
Saint-Petersburg, Russia

April 23, 2002

Abstract

This document describes BAPI (BITBUS Application Programmer Interface) implementation for ELZET80 IPC–BIT900 family PC plug-in boards under Linux (with kernel version $\geq 2.4.0$) environment

Contents

1 BAPI library

The library provides BITBUS application programmers interface (BAPI), described in [1] with the following differences:

- *BirbusOpenSlave*() is not implemented. As the IPC–BIT900 boards can not be used as a slave, this function always returns *BAPI_ERR_NO_BOARD*

The library supports multi-tasking environment (in fact, this is provided with a kernel driver). It also multi-thread-safe.

1.1 How to use the library

To use the library you must include the header file named bapi.h in your source file

```
#include <bapi.h>
```

Note the lower case in file name.

After that you can put BAPI calls in your code, e.g.

```
 BBHANDLE bh;
```

```
 BitbusMsg msg;
 int ret;

 bh = BitbusOpenMaster ("My App", "BBUS0", NULL);
 if (bh < 0) {
     // process error here
 }

 // prepare message here
 ret = BitbusSendMsg (bh, &msg);
 // check errors here

 ret = BitbusWaitMsg (bh, &msg, -1);
 // check errors here, process received message

 (void) BitbusClose (bh);
```

Then compile your program and link it against *libbapi*:

```
cc -o prog prog.c -lbapi
```

You may need to add additional options if the header file and the library a installed in unusual places.

2 Driver of IPC–BIT900

The driver works with **ELZET80 IPC–BIT900** family of PC plug-in boards, which consists of **IPC–BIT900A** for ISA bus, **IPC–BIT900<104** for PC104 ISA bus and **IPC–BIT900<PCI** for PCI bus.

It supports all functions which are necessary to use the boards as BITBUS-Master.

The driver can drive up to 8 devices in any combination and permit up to 16 independent tasks per device (i.e. each device may be opened up to 16 times simultaneously)

2.1 Driver API

The driver use the boards as character devices. It supports *open*(), *close*(), *read*(), *write*(), *poll*() and *ioctl*(). But it ***does not*** supports *readv*() and *writev*().

open() operates as expected:

```
ret = open(name, mode);
```

where:

name is a file of a device. Usual names are /dev/bit900-N, where N is a device (board) number ($N = 0 \ldots 7$).

mode is a usual open mode, e.g. *O_RDWR|O_NONBLOCK*. Not all flags have a sense, in particular, there is no sense to open device only for reading or for writing only (with flag *O_RDONLY* or *O_WRONLY*), but driver permit you to do so if you want.

return value is a non-negative file descriptor or -1 in case of error

2.1.1 *read*() and *write*()

read() and *write*() are bit tricky. The driver attempts to exchange exactly one message per operation.

for *read*() driver fill the buffer with complete message (mCAT header followed by a data). If supplied buffer is shorter then received message (but not shorter when header), the message will be ***silently*** truncated, so it's better to have a buffer which is not shorter then maximum message length (*MCAT_MSG_LEN*, 512 bytes).

for *write*() driver expect complete message in a buffer. It ignores some fields in header: *len* (it takes length from the corresponding parameter of *write*()), *src* and *net* (they are taken from internal data of driver).

if the buffer length is greater then maximum length of message, only *MCAT_MSG_LEN* bytes will be sent.

Of course, both operations return number of transferred bytes.

readv() and *writev*() are ***not*** supported. In fact, LINUX kernel replace them with a series of *read*() and *write*() respectively. Since *read*() and *write*() exchange only whole messages, you got not what you expected.

2.1.2 *ioctl*()

The following IOCTL's are defined:

IOCTL	Arguments	description
BIT900_TASK	none	Returns a number of current task ($0 \ldots 15$)
BIT900_OQUEUE	none	Returns a number of packets in output queue
BIT900_IQUEUE	none	Returns a number of packets in input queue for current task
BIT900_IQUEUES	none	Returns a total number of packets in input queues for all tasks (for current device)
BIT900_SETAPPNAME	*bit900_buff_t*	Set name for current application
BIT900_GETAPPNAME	*bit900_buff_t*	Get name of current application. Return value is a length of an actual name.
BIT900_GETAPPNAMES	*bit900_buff_t*	Get list of newline–separated names of all applications using this device.

All them returns non-negative value in case of success

2.2 using /proc file system

The driver registers a read-only entry in /proc file system. It's name is driver/bit900 (usually /proc/driver/bit900). The contents of this file is look like this:

```
dev0:   hw=BITPCI       hwrev=22         swrev=2.0
        base=0xd400     irq=11
        status=0x24 (RX_EMPTY,TX_EMPTY)
        lstatus=0x80 ()
        control=0x8     hwver=0x11
        sent=0  recvd=0
        int=0   err=0   drop=0  trunc=0
```

For each installed device it contains the following entries:

name	value
hw	board type (from identification string). Possible values are BIT900 and BITPCI
hwrev	hardware revision (from identification string)
swrev	mCAT software revision
base	base I/O address
irq	IRQ
status	current status (register 2 of device)
lstatus	latched status (register 3 of device)
control	control
hwver	hardware version information (from register 5)
sent	counter of sent messages
recvd	counter of received messages
int	interrupt counter
err	error counter
drop	counter of dropped incoming messages
trunc	counter of truncated incoming messages

2.3 Invoking of module

Driver must be loaded before using, e.g. with *modprobe* You can (and for ISA card you must) supply some parameters to module:

parameter	type	default value	description
bit900_io	array of int	none	base I/O addresses of ISA cards
bit900_irq	array of int	none	IRQ of ISA cards (set to -1 to let module to choose IRQ automatically)

LINUX kernel (and BIOS) finds and correctly initializes all PCI cards automatically, but for each ISA card you must provide to driver I/O address and IRQ with above parameters, e.g.:

```
modprobe ipc-bit900 bit900_io=0x230,0x238 bit900_irq=9,11
```

You also must choose correct I/O address and set it in a board (see board manual).

While the driver can choose IRQ by it's own, it is unsafe, as IRQ may conflict with IRQ of an unused device. So it's better to provide IRQ explicitly. The latter may be one of $(3, 5, 7, 9, 10, 11, 12, 15)$

3 Installation

A kernel driver and a library have to be loaded.

All described actions should be executed as root.

3.1 Driver installation

First the bad news:

Linux kernel version must be 2.4.0 **or greater.** This mean you may need to upgrade your kernel.

Binary code of driver module depends on kernel. You can not use the same module for different kernel versions or even for kernels of same version but with different features. So you will probably need to recompile the driver

The driver made as separate module. This means to compile it you don't need to patch or recompile your kernel.

But you still need to have a proper kernel tree. This means the first you need, is the original tree in which you compiled your version kernel. But a freshly unpacked kernel will not cut it, because it miss some files that are needed. `make *config dep` creates some files that are needed. And even then, you will run into trouble, because you may not have selected the exact same configuration variables.

Plain advise: if you do not have your original kernel tree anymore, recompile your kernel first.

3.1.1 Compiling driver

If you have proper kernel tree you can compile and install driver.

1. Go into driver source tree and check top-level Makefile You can see a couple of variable there (with short description and examples):

   RELEASE sets the kernel version. It used mainly to install module in proper place. If you compile to current version, set this to `‘uname -r‘` to ask system for proper value.

   KERNELDIR sets the path to kernel tree. Usual value is /usr/src/linux

2. Type `make` to compile module.

3. As a root, type `make install` to install the module and public header file and to create device files /dev/bit900-0 ... /dev/bit900-7.

The driver is installed. To use it you must load the module first with command modprobe

To force automatic loading of driver module you can edit file /etc/modules.conf. Add the following lines into it:

```
alias char-major-242 ipc-bit900
#set parameters for ISA boards
options ipc-bit900 bit900_io=0x230,0x238 bit900_irq=9,11
```

The last line is need only for ISA boards.

3.2 Installation of library

not written yet...

3.3 Test program

not written yet...

References

[1] Mario Casali, Bassel Safadi, Matteo Mondada, Beggi Oskarsson, and Volker Goller. *BITBUS application programmers interface(BAPI). A BEUG recomendation.* World Wide Web, http://www.bitbus.org/dnl/bapi.pdf, 1999.