CONNOTECH Experts-conseils inc.
PPCMB/850 Product Family Documentation

The ABCD Proto-Kernel Networking Specifications,

including the LAB-COM Utility Guide

Document Number C002424

2004/03/09

Document Revision History

C-Number	Date	Explanation
C002424	2004/03/09	First release
C002424		Current version

Table of Contents

1. Background

1.1 The Networking Specifications

The present document addresses the data communications requirements between an embedded system and a personal computer. CONNOTECH offers the PPCMB/850 (an embedded microprocessor piggy-back module) that qualifies as an embedded system, but the present document has obviously many other potential applications.

The ABCD Proto-Kernel™ networking specifications is established as a support function for the ABCD Proto-Kernel™. The kernel itself does not require any data communications, but the software image loading does, as some of the useful additions to the ABCD Proto-Kernel™ (i.e. the diagnostics trace function and the maintenance commands).

The implied main purpose for the embedded system data communications includes systems development, troubleshooting, production testing and field servicing. The present specifications can nonetheless be used as a base for an operational protocol requirement, such as in a data-intensive sensor acquisition application.

These networking specifications provide a commonality among a serial port connection and an Ethernet connection, with minimal burden on the embedded system software setup. The networking specifications are meant to be more robust than an ad-hoc serial port protocol, but simpler than a subset of the IP protocol suite.

1.2 The lab-comm Utility

The present document describes the lab-comm utility. This program implements the networking specifications on a personal computer running the Linux operating system. A prior version has been developed for the MS-Windows environment (the commui utility program) but it is no longer supported nor distributed. In either case, the ease with which a protocol enhancement can be developed was an important design factor (perhaps sacrificing user convenience). The lab-comm utility is a simple command-line utility.

This software development initiative is part of the GPL-inspired free software trend, but attention is paid to the embedded system practicalities. An embedded system being a product more than software, it is legitimate to protect the embedded software application by some secrecy, intellectual property, and/or, operational controls (e.g. preventing unauthorized field upgrade of software when safety is at stake). This balance between the free software benefits and the protection of proprietary product designs extends to the implementation of the lab-comm utility.

2. Document References

[ABCD_KERNEL]
CONNOTECH Experts-conseils inc., *The ABCD Proto-Kernel™ Guide (Embedded Software Document)*, PPCMB/850 Product Family Documentation, Document Number C001534, 2003/10/17

[ABCD_LINK_N_LOAD]
CONNOTECH Experts-conseils inc., *ABCD Proto-Kernel™ Software Link and Load Process (Embedded Software Document)*, PPCMB/850 Product Family Documentation, Document Number C002468, 2004/03/09

3. Overview of Protocol Framing Specification

3.1 Serial Port Framing

With the serial port connection, the PPP framing specification (a subset of Internet RFC 1549 as framing specifications) makes the serial link very similar to packet-oriented transmission such as Ethernet or synchronous HDLC. It provides the following services which in a standardized way:

- data frame delineation (the length of each data frame is implicit in the protocol and conveyed to the receiving entity),
- data transparency (no control character in the data to be transmitted can interfere with the protocol operation),
- end-to-end error detection (using a CRC-16 error detection at the end of the frame).

These services are typical of frame-oriented protocols, but their software implementation is fairly simple. Note that station addressing is not listed above. That's because the basic scenario is a single embedded device connected to a personal computer.

Field	Flag	Information	CRC	Flag
Width	8 bits / 0x7E	M * 8 bits	16 bits	8 bits / 0x7E
Application payload		▨		
Transparency mechanism	no	yes	yes	no

3.2 Ethernet Frame Format

A specific protocol identifier is selected among the unused ones (default value 0xBA00) in order to distinguish the `lab-comm` Ethernet traffic on the LAN from the normal protocol (e.g. 0x0800 for Internet Protocol packets, see the include file `linux/if_ether.h`). Accordingly, the Ethernet connectivity model is limited to the local area network to which the personal computer is connected.

Here are some of the Ethernet protocol standard implementation options:

- either the Ethernet broadcast address is used for transmission and reception (assuming a single embedded device connected on the LAN and a single PC involved in `lab-comm` communications with the selected protocol identifier), or the embedded device Ethernet addresses is configured by the `lab-comm` user,
- a single byte length field is inserted between Ethernet header (destination address, source address, and protocol identifier) and the frame contents, indicating the length of the valid bytes when the 64-bytes minimal frame requirement forces a larger than necessary frame; the length field excludes the header, checksum, and length field itself, it is thus significant only when its value is between 0 and 45 inclusive,
- the physical layer interfacing characteristics must agree between the embedded device and the personal computer Ethernet adapter. This includes issues such as the media type, full duplex or half duplex mode in the case of twisted pair media, the transmission and detection of link integrity pulses.

Field	Destination Address	Source Address	Protocol Type	Informa-tion Length	Informa-tion	Padding	CRC
Size	16 or 48 bits	16 or 48 bits	16 bits	8 bits	n * 8 bits		32 bits
Controlled by adapter						▨	▨
Controlled by driver	▨	▨	▨				
Application payload					▨		

3.3 Protocol Identifiers

With either the serial port framing or the Ethernet framing, lab-comm specific frame contents starts with a protocol identifier (two bytes, in network order, i.e. most significant byte first).

3.3.1 Trace Protocol

The trace protocol has the following protocol identifier value:

ABCD_TRACE_PROT_IDENT (0xEA09)

With the current implementation in the ABCD Proto-Kernel™ useful additions, the trace protocol is expected to work as follows:

- the “trace” protocol frames *from* the embedded system carry trace entries issued by the embedded system (presumably in human-readable format, preferably without any cursor movement character or national characters), whereas

- the “trace” protocol frames *to* the embedded system carry maintenance commands if the first character starts with a slash (/) character (other trace protocol frames are quietly ignored).

The syntax and processing of such maintenance commands is described in a section of the document [ABCD_KERNEL].

Normally, a lack of compliance with the above provisions does not affect the other protocols. However, the capability to re-load a new software image into the embedded device may depend on the proper recognition of a maintenance command (e.g. "/ld loader") sent to the embedded system through the trace protocol as described above.

3.3.2 Download Protocol

The download protocol has the following protocol identifier value:

`ABCD_LOAD_PROT_IDENT` (0x7A9F)

The download protocol is described in a section of the document [ABCD_LINK_N_LOAD].

3.3.3 Future Directions for New Protocols

Other protocol identifier values may be defined for embedded system protocols in the future.

Moreover, it is envisioned that the currently defined protocol identifier be prefixed by encapsulation protocol headers. Such encapsulation protocols should be identified by protocol identifier values not conflicting with the currently defined protocol identifiers.

4. The lab-comm Software Utility

The lab-comm utility is a free software that allows interactive communications with a *target system*, much like a remote terminal emulation software. It was developed as an electronics laboratory tool for microprocessor-based product development, troubleshooting, and testing (including production testing at the end of the manufacturing process).

4.1 Note on the Free Software License for the lab-comm Utility.

The lab-comm as a whole is covered by the GPL (notably, the lab-comm utility relies on the readline library which is purposely GPL'ed by the Free Software Foundation). Part of the lab-comm source code is borrowed from the ABCD Proto-Kernel™ source and is covered by the "LGPL with a Waiver on the Opportunity to Relink" (i.e. a license intended to allow, under certain conditions, proprietary embedded applications to be developed with the free software ABCD Proto-Kernel™ in view of some practicalities of embedded system development). The LGPL license is affixed to yet another section of the lab-comm utility source code (more or less the low-level communications stuff with the serial port and the Ethernet port).

If you whish to develop a proprietary protocol to your embedded devices, you might consider

- having a defined programmatic interface between the lab-comm executable and another executable implementing your proprietary protocol (e.g. through a socket interface), or
- start with the LGPL'ed portion of the source code, and distribute your software accordingly.

The intent is to keep the lab-comm utility, as it is organized, a GPL'ed whole. If you adopt the lab-comm mode of linking personal computers to your embedded systems, you agree that this link technique is not a place where proprietary restrictions may by put. See the text of relevant licences for the exact terms.

4.2 The lab-comm Main Features

4.2.1 Design Goals

A simple command-line utility developed on the Linux operating system. Design simplicity is somehow preferred to operation ease of use.

A built-in software download capability, so that the target system software can be updated.

A capture-to-file capability for the data exchanged between the terminal software and the target system.

A command scripting capability, admittedly somehow primitive

Re-use of command parsing source code from an ABCD Proto-Kernel™ useful addition, with a uniform command syntax when the lab-comm utility is connected to a target system based on the ABCD Proto-Kernel™.

Simple software source code, with the hope that special-purpose communications requirements can be added to the software without too much effort.

4.2.2 Limitations

Little status is available on the screen (e.g. whether the target system connection is established), as would be expected from a terminal emulation utility. The current state of the program can be queried with commands.

The synchronization among various input and output channels is achieved with the same queue mechanism as the ABCD Proto-Kernel™, with a fixed maximum number of queue elements for each queue. Although these maximum numbers are generously specified, this might eventually lead to data losses that would not be expected from a utility that runs on an operating system with virtual memory and large swap files.

The lab-comm source code is a single-thread application. As a software source code development base, a developer may have a preference for multi-threaded applications. This can be perceived as a restriction of possible coding style for the ad-hoc protocols that can be added to the lab-comm source code.

4.3 Development Plan

4.3.1 RS422 Multi-point Network Topology

The serial interface should support the master station role in an RS422 multi-point network topology (single master, multiple slaves). This support for this network topology should at first include operator-controlled selection of the target system that is allowed to transmit (i.e. automated polling is not considered a development priority).

Note: With the RS422 multipoint network topology, the personal computer (PC) serial port is connected to a number of target systems. The transmit signal pair from the PC is connected in parallel to the receive signal pairs of every target system, and the transmit signal pairs of every target systems are connected in parallel to the receive signal pair on the PC (a separate ground connection is also needed). The master station software logic (the lab-comm utility) must allow a single target system to transmit at any time. The RS422 multipoint network topology is a simple LAN implementation that predates the Ethernet topology and is still in common use in applications where its simple cabling requirements are appreciated.

4.3.2 Other Development Directions

The other development directions include the following items:

- Transparent and automated polling of RS422 slave stations, which can be attractive for operational protocols in sensor data collection application at low speed.

Note: For higher speeds, i.e. up to the 2.5-7.5 Mbps range, the same topology and protocol can be easily achieved with the synchronous protocol support built-in the MPC8xx processors used in the PPCMB/850 processor board. In this case, the master station role would be implemented by a PPCMB/850 that would be connected to the personal computer with e.g. a poit-to-point Ethernet link.

- USB support.

- ATM support, since a variant of the MPC850 used in the PPCMB/850 processor board is the entry-level processor with ATM support. The ATM connectivity has the potential to support aggregate throughput above 10Mbps with predictable latency in transmission.

5. Sources of Information for a User Guide

5.1 Generic Command Processing

5.1.1 Interactive Command Input

Once started, the `lab-comm` utility expects commands from the console, and prompts the user with "LAB-COMM>". Commands are interpreted as follows:

- Commands lines starting with a dot (.) character in the first position are interpreted by the `lab-comm` utility itself.
- Other non-empty lines are sent to the target system if the `lab-comm` utility currently has an open channel. Each line contents, without the line-terminating indication, is prefixed by the trace protocol identifier (see section 3.3.1) and packetized for transmission.

  Note: With the "maintenance commands" implemented as an ABCD Proto-Kernel™ useful addition following the present networking specifications, command lines starting with a slash (/) character are interpreted as a command by the target system.
- Only in script lines read from a script file, a line starting with a number sign (#) character is ignored as a comment.
- An empty line gives a download operation status.

The command line input uses the readline library and the readline history facility, so the keystrokes typed at the keyboard should generally react as usual in the Linux shell script. There is an exception for command scripting described below, which is implemented as a modified behavior of the readline history facility.

5.1.2 Command Help

The user may get cryptic help with the ".HELP" command for the commands that are interpreted locally, and the "/help" command for commands that are interpreted by a connected and compliant target system, if any.

In addition, the lab-comm licensing information and warranty information can be obtained respectively with the commands ".GPL ABCD-W" and ".GPL ABCD-C".

5.1.3 Command Syntax

The lab-comm command syntax is inherited from the ABCD Proto-Kernel™ useful addition called maintenance commands. For sake of brevity, we refer the reader to the section describing the maintenance commands in the document [ABCD_KERNEL].
Note: This document reference applies to commands interpreted by the target system, i.e. the lines starting with a slash (/) character.

5.2 Basic Command Set

5.2.1 Opening and Closing Communications Channels

The lab-comm utility can be connected either to a serial port or an Ethernet adapter on the local personal computer. There is no concept of connection to a target system as in a connections-oriented end-to-end protocol.

5.2.1.1 Serial Connection Channel

The command ".COM SHOW" displays the current serial communications port configuration.

The following commands allow the serial port configuration:

- ".COM PORT" sets the serial communications port name as the string literal parameter (e.g. "/dev/ttyS0")."
- ".COM SPEED" sets the baud rate for the serial communications port as the integer argument, with the usual baud rate options from 1200 to 115200 bauds.
- ".COM RTS" sets the serial communications port RTS configuration (0 or 1).
- ".COM DTR" sets the serial communications port DTR configuration (0 or 1).

When the serial port is currently disconnected, the changes to the configuration are retained for the next connect command.

Two additional commands allow the serial port to be connected and disconnected:

- ".COM CONNECT" connects the serial communications port.
- ".COM DISC" disconnects the serial communications port.

5.2.1.2 Ethernet Adapter Channel

The Ethernet connectivity requires a system permission (see the technical details in section 7.3). If the `lab-comm` utility software does not have the required permission, many Ethernet adapter commands fail.

Two commands are provided to query the Ethernet status:

- ".ETH SHOW" displays the current Ethernet connectivity status.
- ".ETH IF" without a parameter value, displays the list of Ethernet interface cards on the machine.

The following commands allow the Ethernet port configuration:

- ".ETH IF" with a string literal parameter, selects the local Ethernet interface card to use.
- ".ETH FRAMETYPE" sets the frame type number to use as Ethernet connectivity protocol indication (see section 3.2 for explanation).
- ".ETH REMOTE" sets the remote Ethernet link layer address to the string literal parameter, which must be exactly 12 hexadecimal digits."
- ".ETH BROADCAST" selects the use of broadcast Ethernet addressing instead of any specific remote address indicated by the previous command.

When the Ethernet port is currently turned off, the changes to the configuration are retained for the next connect command.

Two additional commands allow the Ethernet port to be turned on and off:

- ".ETH ON" turns on (connects) the Ethernet connectivity.
- ".ETH OFF" turns off (disconnects) the Ethernet connectivity.

5.2.2 The Remote Transmission Capture Facility

The remote transmission capture facility allows the recording of protocol transmission to and from the target embedded system. This recording follows the file format specified in section 7.2.

- ".LOG SHOW" displays the current log recording configuration.
- ".LOG" with a string literal parameter indicating a file name, sets the log file name.
- ".LOG ON" turns on the log capability.
- ".LOG OFF" turns off the log capability.

5.2.3 The Exit Command

The command .EXIT exits the lab-comm utility.

5.3 The Command Scripting Facility

The lab-comm command scripting facility provides a rudimentary mechanism for running commands stored in a file.

- ".SCRIPT" with a string literal parameter indicating a file name, reads a script file (stored command sequence) into the readline's command history buffer. This command can be nested.
- ".SCRIPT OFF" turns off the special handling of readline's command history buffer when a script file has been read but not completely processed.
- ".SCRIPT SAVE" with a string literal parameter indicating a file name, saves the current readline's command history buffer in a file that may be edited to create a useful lab-comm script file.

Here is how the readline's handling of command history buffer is altered to provide the script logic: When a script file is read by the command ".SCRIPT "filename"", its contents is uploaded to the command history buffer, except for the command lines starting with a number sign (#) character, and the script mode is entered. In this script mode, the lines typed by the user (or recalled from the history buffer) are not added to the history buffers. Instead, if a command line (typed or recalled) is identical to a line in the history buffer, it is removed from the history buffer (only the most recent one is removed in case the line occurs more than once).

With this behavior, the script file lines are quickly executed by the user by repeatedly pressing the up arrow key and the enter key. The command ".SCRIPT OFF" allows the user to exit the script mode before every lines from the script file are executed.

5.4 The Software Image Download Operation

The lab-comm user interface to the software image download operation provides the capability to transfer a file to an embedded system using the protocol identified in section 3.3.2 and specified in a section of the document [ABCD_LINK_N_LOAD]. The required file format is also described in the referenced document.

- ".LOAD SHOW" displays the current status for the file transmission operation. An empty command line to the lab-comm utility performs likewise.

- ".LOAD" with a string literal parameter indicating a file name, starts a file transmission operation for the specified file.
- ".LOAD HALT" halts any current file transmission operation.

As far as the lab-comm utility is concerned, the contents and purpose of the downloaded file is unknown, and the some the status codes are meaningless. The operator feedback on the file transmission operation is ugly. For an error-free operation, it was intended to drive two progress bars, respectively for small segment transmission progress and big segment transmission progress in a GUI operator interface. For transmission operation failures or stalls, the operator feedback details were intended to provide troubleshooting data that can be forwarded by field-service personnel to support personnel in a position to diagnose the transmission problem.

6. Installation Guide

A hand-crafted makefile is included in the distribution.

The lab-comm utility requires the "setuid" file permission if it is to be used with the Ethernet connectivity. This is granted by the following commands done by a root user:

```
chown root:root/usr/bin/lab-comm
chmod ugoa+s /usr/bin/lab-comm
```

See section 7.3 for more information.

7. The lab-comm Utility Software Internals

7.1 Software Organization

The software is a simple loop based on the select() function. It is a single thread application. It uses the queue mechanism from the ABCD Proto-Kernel™, for sake of simplicity, expeditiousness, and commonality of source code among the embedded application and host utilities.

Note: The queue mechanism from the ABCD Proto-Kernel™ might be used in a multi-thread application, but we didn't look at the implied synchronization requirements.

There are three instances of a queue, each one ensuring that the lab-comm software is not blocked by a slow output function:

a capture queue for logging protocol exchanges to a file
a target command queue for sending lines to the target
a console write queue for user feedback

A basic periodic timer logic is included as a software example, but the lab-comm software currently makes no use of it.

The Ethernet connectivity implemented with (more or less Linux-specific) low level packet interface. We decided to avoid the pcap library, because we would have used a very small subset of its functionality, with the risk of reduced portability.

7.2 Format of lab-comm Communications Log File

Thee format of the lab-comm communications log file is specified in this document subsection.

General rules:

- binary values are stored most significant byte first and occupy the number of bytes indicated below,
- a log record length is indicated by field values present in the log record itself, as indicated below.

Log record contents:

Field Name	Field Size, in bytes	Description
tv_seconds	4	time of frame transmission or receipt, second value from the tv_sec field in the struct timeval value returned by gettimeofday
tv_usec	4	time of frame transmission or receipt, micro-second value from the tv_usec field in the struct timeval value returned by gettimeofday
direction_outg	1	0: incoming frame, 1: outgoing frame
type_of_address	1	0: no address field, 1: 48 bits Ethernet link layer address
address_length	4	Length of the address field that follows
frame_length	4	Length of the frame field that follows
address	address_length	Address field
frame	frame_length	Frame field

7.3 System Privilege Needed for Ethernet Connectivity

The packet interface to Ethernet requires permission (a capability) that is normally not available to non-root users. The Linux system administration tools does not readily allow an executable file to have a specific capability subset (from the set of root user capabilities), so the `lab-comm` utility starts by dropping every capability that it does not need (it actually needs only one) and changes the effective UID to the non-root parent UID. With this scheme, the security risk of starting the `lab-comm` utility with the setuid flag is reduced to the strict minimum.

To validate that the capabilities are sufficient for using the `lab-comm` utility by non-root users, the ethernet connectivity must be turned on with the command ".eth on" (an error message appears only when attempting to open the Ethernet interface, even if the capabilities cooking is done upon program entry).

To validate that the capabilities are indeed restricted in a running version of `lab-comm`, the following commands can be used, from a root command window:

```
ps -A | grep lab-comm
```

this will show a process id number ### that is then used in the following command:

```
cat /proc/ ### /status
```

in the listing, the last three lines give the current capability status for this task as 64 bit masks. The value 0000000000002000 (mnemonic `CAP_NET_RAW`) is needed in the CapEff field for the raw packet interface to work.