The ABCD Proto-Kernel™ Version 1.x Distribution Overview

(High-Level "Read-Me" Document)

Document Number C002394

2004/03/09

(C) 2003 CONNOTECH Experts-conseils inc.

Document Revision History

C-Number	Date	Explanation
C001994	2003/10/21	Initial release
C002221	2003/12/03	Update for ABCD Proto-Kernel release version 1.1
C002394	2004/03/09	Update for ABCD Proto-Kernel release version 1.2, Added reference to files abcd_maintcmd_dispatch.
C002394		Current version

Table of Contents

1. Introduction

1.1 Distribution Overview

The ABCD Proto-Kernel™ version 1.x distribution, should assist you for the development of *embedded software application* (hereafter *embedded application*) for a specific *embedded computer target hardware* (hereafter *target hardware*). This document gives background information on what the ABCD Proto-Kernel™ distribution can do for you, and where it stops in providing the valuable tools, software, and documentation that you need for your embedded application development project.

The ABCD Proto-Kernel™ is a minimal kernel ("A" interrupt dispatching, "B" fixed priority scheduling, "C" mutual-exclusion semaphores, "D" a queuing mechanism). We recommend the merging of the embedded application source code with the kernel source code, without special precautions to maintain a formal interface between the two, except for free software licensing reasons when the application source code is not free software. This makes a simple kernel model without an attempt to specify the application run-time behavior at any given API interface.

The ABCD Proto-Kernel™ is a niche software product that competes with proprietary and free kernels in the low end of the kernel sophistication spectrum.

The ABCD Proto-Kernel™ version 1.x distribution comes with a PowerPC implementation (see below for details), the FlashCnL library that helps with the flash memory management, and a couple of software loading utilities. In addition, a separate but related distribution provides the required cross-compiler tools (a port of the free software GCC tools) for the development of your embedded application. Together, these software and tools make a comprehensive embedded software development solution. However, please bear in mind that the documentation may describe features that are the left out of the ABCD Proto-Kernel™ version 1.x distribution.

CONNOTECH Experts-conseils inc. offers specialized consulting services related to the ABCD Proto-Kernel™ software.

1.2 The Embedded Software Development Model

The embedded software development is driven mainly by the specifications of the target hardware. We use the term *software image* for the outcome of the software development procedures, that is a file that contains *all* the required compiled and linked programs (in a binary format) ready to be loaded into the target hardware. Once the software image is loaded, the embedded system should operate as designed (hopefully). The term embedded system may encompass the each of the following elements:

- the embedded computer target hardware,
- the whole device hardware, including power source, analog electronics, mechanical components, enclosures, accessories, external connections, etc.,
- the embedded software application,
- device configuration data stored within the device,
- any software loading capability, usually in the form of special embedded loader software that is not used in the normal device operating conditions.

Note: This is a radical departure from the usual software development paradigm where an hardware abstraction layer, in the form of APIs (Application Programming Interfaces), is intended to make the software program portable.

Your first role as an embedded application developer is to use the software tools to create a software image from the source code that you develop for your application and the source code included in this free software distribution. Typically, the software image for the embedded application is *loaded* in the target hardware either

- in the context of system design and troubleshooting, in the laboratory environment,
- in the context of embedded system manufacturing, in the final stages of system assembly,
- in the context of fielded systems servicing, when a software fix or upgrade is "field loaded."

1.3 Distribution Contents Overview

The cross-compiler tools are in a separate distribution:

a) The GCC (Gnu Compiler Collection) as a C/C++/assembler/linker cross-compiler package for the MPC8xx PowerPC microprocessor family is to be found at http://www.connotech.com/gcc_mpc8xx/powerpc_eabi_mpc850.htm

The ABCD Proto-Kernel™ version 1.x distribution is a comprehensive solution including:

b) A post-link utility that converts the ELF format into the software image format expected by the loaders, items d) and g) below (project name `elf_post_ld`);

c) The ABCD Proto-Kernel™ merged with a sample application skeleton (project name `ppcmb850_init`);

d) The ABCD Proto-Kernel™ merged with the embedded loader application, enabling program loading through a serial port (project name `ppcmb850_devloader`);

e) The ABCD Proto-Kernel™ merged with a BDM port handler application, enabling

program loading in a remote target processor through its BDM port (project name ppcmb850_bdm);

f) An communications utility program, the lab-comm utility, for interfacing with the two embedded applications, items c), d) and e) above;

g) A very initial loader, that programs the flash memory, while running from RAM after having been loaded through the BDM port (project name ppcmb850_viloader).

1.4 What's New

1.4.1 ABCD Proto-Kernel Version 1.2

The ABCD Proto-Kernel distribution version 1.2 has the following improvements:

- The lab-comm communications utility replaces the commui utility present in previous versions. The new utility runs under Linux and provides a simpler source code base for implementing ad-hoc protocols to embedded systems.

- The power management modes of the PPCMB/850 are now supported as an ABCD Proto-Kernel™ useful addition.

- The maintenance commands (an ABCD Proto-Kernel™ useful addition) are now case-insensitive. The source code is now separated in a number of files instead of a single large file. It is thus easier to combine proprietary commands with some commands that are covered by the LGPL with a waiver on the opportunity to relink (provided the licensing conditions are met for merging proprietary application code with the ABCD Proto-Kernel™).

- Many bugs has been corrected.

- The documentation has been improved.

1.4.2 ABCD Proto-Kernel Version 1.1

The ABCD Proto-Kernel distribution version 1.1 had the following improvements:

- The BDM interface control software (project name ppcmb850_bdm) has been added. Correspondingly, the rproxy project has been removed.

- The makefiles has been improved for the handling of the various PPCMB/850 clock

frequencies.

- A few bugs has been corrected.

- The MPC8xx exception trapping handling has been refined so that corrupted exception vectors that cause a system reset are detected upon system startup (from the MPC8xx Reset Status Register and other clues).

- A draft white paper on the topic of Free Software Licensing has been added to the readme_etc documentation sub-directory.

2. Addressing the Embedded Software Certification Challenges

Embedded systems may control various types of devices, industrial processes, vehicles, airplanes, ... The potential security implications of a software malfunction may be significant, if not catastrophic. Accordingly, in the fields of aerospace, medical devices, nuclear and high energy industries, and electronic payment apparatuses, one will find stringent software certification requirements. Although the ABCD Proto-Kernel™ project is not directly subject to any certification requirements, it is definitely influenced by them.

- **Source code determinism.**

  With safety critical embedded software, it is usually required to keep a copy of the *complete and exact* source code compiled in the whole embedded system. This completeness requirement is readily met with the free software distribution used by the ABCD Proto-Kernel™ distribution (a much friendlier case than using proprietary software that is distributed in object or executable form only).

- **Dead code removal.**

  The above source code exactness requirement comes with a stricter criteria for *dead code removal*, i.e. the removal of any source code fragment that would never be actually run in any conceivable device operating condition. The ABCD Proto-Kernel™ development methodology uses the `prcpp` source code filter utility to remove dead code based on a specific embedded software configuration (see section on 6.3 page 18).

- **Dead code removal, API avoidance.**

  The safety related requirement for dead code removal and the merging of the kernel code

with the application software without a formal API boundary induce the liberal use of the C language conditional compilation directives, **`#if`**, **`#else`**, **`#elif`**, and **`#endif`**, where the usual coding practices would use logical identifiers, handles, or control block pointers created at run-time.

- **Software image creation determinism.**

  The software image format used by the The ABCD Proto-Kernel™ link and load utilities is such that it contains only the essential information (e.g. any debugging information is removed frm the intermediate software image files). This makes the software building procedures more reproducible: a second run of the software building procedures from the source files should give the exact same software image file (e.g. if neither the `__DATE__` nor the __TIME__ preprocessor macros are used).

- **Target hardware compatibility.**

  For the field loading of software, the software image file format is as close as possible to the target hardware characteristics. This extends up to the alignment of the software image segmentation with the flash memory sectors in the target hardware. (The usual strategy is to let the embedded loader do the segmentation). Accordingly, if there are two flavors of the fielded embedded hardware with a different flash memory sector arrangement, two software image files must be created. The idea is to handle software configuration issues as early as possible in the software creation and loading procedures.

  Note: With the field loading procedures, end-to-end checks are preferred to validations performed in intermediate steps of the procedure, as a consequence of the above strategy. Accordingly, the downloading utility merely sees a binary file with segmentation information, and transmits its contents without considering the binary file semantic beyond the segmentation information.

- **Secure loading option.**

  Moreover, a secure loader is in the development plans. It extends the concept of end-to-end checks with cryptographic integrity protection. This secure field loader would

  - verify that the loaded software is compliant with the target hardware and the embedded system configuration, helping to prevent software compatibility problems,

  - check the integrity of the loaded software, helping to prevent software image corruption,

- check an electronic signature endorsing the loaded software, helping to prevent the loading of valid but otherwise unauthorized software image (e.g. software sabotage by disgruntled former employees), and

- prevent the embedded system from entering any seemingly normal operating mode if the loaded software isn't completely and properly loaded, as a last line of defense against the former risks and threats.

If embedded software sabotage protection is a genuine requirement in your project, you should contact CONNOTECH Experts-conseils inc. to discuss the secure loader project.

- System configuration support.

  In support of production inventory traceability procedures, the FlashCnL software library is provided for permanent storage of system configuration data in the flash, so that part numbers and serial numbers may be electronically recorded immediately after the completion of manufacturing tests for the target hardware units. The FlashCnL software library integration with the ABCD Proto-Kernel™ is designed such that the configuration data survive the loading of a new software version (while avoiding the complexity of a full-featured flash file system).

- Service history data recording.

  For the laboratory environment where software certification requirements usually mandate the recording of runtime behavior evidence, the FlashCnL library also supports the unintrusive recording of system log data. This capability is advantageously kept in production systems for service history data collection, because embedded systems are usually operated in an unattended mode.

3. The Version 1.x Target Hardware

With the ABCD Proto-Kernel™ version 1.x distribution, the embedded computer target hardware is based on the PPCMB/850 PowerPC mini-board sold by CONNOTECH experts-conseils inc. The PPCMB/850 contains a fully functional Motorola MPC850, which is the entry-level PowerPC family member for the embedded market. The PPCMB/850 is easily expandable, so that the hardware designer of an application-specific electronics sub-assembly can save the time and expense of designing and troubleshooting the microprocessor and memory subsystem. In addition, the early availability of the PPCMB/850 with the ABCD Proto-Kernel™ can kick-start the embedded application development using the core target hardware.

Note: The ABCD Proto-Kernel™ adaptation to other members of the PowerPC family, or even

to other microprocessor architectures, represents a reasonable undertaking. The binding of the ABCD Proto-Kernel™ version 1.x distribution to the PPCMB/850 hardware occurs mainly as a matter of effectiveness of free software distribution: It appears better to distribute something that was tested in a given context than to distribute something that will eventually work in a large set of contexts.

4. What You Should Know

4.1 CPU Architecture

Before you undertake embedded software development, you should have an understanding of CPU architectures, assembler programming, and memory address computations done by compilers, assemblers and linkers. With the ABCD Proto-Kernel™ version 1.x, this extends to the Motorola MPC8xx embedded microprocessor family architecture, with its rich set of integrated peripherals. This knowledge is useful to understand the consequences of programming constructs that you will find necessary to develop the application. Although the ABCD Proto-Kernel™ provides a programmer-friendly environment for embedded application development and troubleshooting, you will have to dig into the technical implementation details at one point or another. If the development is done by a team of programmers, one team member may be assigned the kernel support assignment, provided this person is given the opportunity to review the other programmers' source code before it is considered ready for integration testing.

You need not learn the complete PowerPC instruction set; assembler programming skills are usually applied when looking at the assembler code generated by the compiler in attempting to diagnose a bug. In the ABCD Proto-Kernel™ source code, assembler source code is limited to the execution sequences that breach some of the run-time properties assumed by high level compilers (e.g. task context switching and interrupt signal dispatching). Processing-time optimization is not a valid rationale for using assembler programming. Despite this, the typical ISR (Interrupt Service Routine) can conveniently be programmed in assembler with the ABCD Proto-Kernel™ version 1.x: an ISR is coded as a few assembler macro calls for ISR prologue, event queue entry insertion, and ISR epilogue. The only explicit assembler instructions in a typical ISR records a peripheral status register contents and clears a few peripheral register bits.

4.2 Linker Sections

If you are already familiar with linker sections and how they are handled in the assembler source code syntax and the linker command files, you may skip this section. Otherwise, your learning experience may be much more efficient if you quickly understand the basic ideas behind linker sections. The following explanations apply to the ELF linker output format that is used by the GCC tools for the PowerPC.

The linker sections is the main mechanism used by compilers, assemblers and linkers to collect machine instructions and static memory allocation for variables and constants from multiple source files. Linker sections are identified by name. An object file contains object code and memory space reservation assigned to various linker sections. Traditionally, the linker section ".text" contains the machine instructions, the linker section ".data" contains initialized variables (e.g. int var_i=3;), the linker section ".bss" contains un-initialized variables (e.g. int var_j;), and so forth. In C or C++, the compiler implicitly selects the proper linker section for each portion of the object file it produces. Command-line arguments to the compiler may alter the default section names. The programmer may use section names for which no default processing is implied by the tools. With the GCC compilers, non-standard language extensions can alter the section name of a specific variable. With the assembler language, the programmer is given a much finer control over linker sections names (see the assembler directives .text, .data, and .section).

The default behavior for the linker is to group the linker section contributions by the various object files, e.g. the linker output file has every ".text" sections grouped in a contiguous memory area. In practice however, a software building procedure usually contains a linker script file that commands special processing for special section names, or traditional section names from specific object files. This is used e.g. to force some explicit address locations for variables and machine instruction sequences, or to control the order in which things get loaded in the embedded target memory. Once the linker sections placement in the memory map is complete, another important linker role is fulfilled: the symbol references are *relocated*. The symbol relocation by the linker is a fully automated process, but may produce cryptic error messages when the linker section placement conflicts with symbol relocation rules.

The contiguous memory areas filled by the linker are called program sections. The output of the linker process can thus be seen as a collection of program sections. The ABCD Proto-Kernel™ version 1.x distribution comes with a post-link utility (elf_post_ld) that converts the .ELF format into the software image format expected by the loaders. The main role of this elf_post_ld utility is to match the linker output program sections to the flash memory sector arrangement and provide flash-to-RAM control information for the machine instructions that are copied from the flash to the RAM at system startup.

You are advised to refer to the listings and other intermediate files produced by the software development tools to see in practice what is going on. This includes the .MAP file produced by the linker, the .LST files produced by assembler, and the assembler code file generated by the compiler upon request. If your embedded application fails on a processor exception (e.g. illegal memory access), the ABCD Proto-Kernel™ attempts to record the CPU register contents at the time of failure, and you should be able to gather clues about which part of the software triggered the failure.

5. Documents

5.1 Documentation for GNU Tools

- Free Software Foundation, *Using the GNU Compiler Collection (GCC)*
  Distribution file documents/gnu_tools_ug/gcc.html

- Free Software Foundation, *The C Preprocessor*
  Distribution file documents/gnu_tools_ug/cpp.html

- Free Software Foundation, *Using as*, a user guide to the GNU assembler as version 2.13.90, 2003
  Distribution file documents/gnu_tools_ug/as.html

- Free Software Foundation, *Using ld*, documenting the GNU linker ld version 2.13.90, 2003
  Distribution file documents/gnu_tools_ug/ld.html

- Free Software Foundation, *GNU Binary Utilities*, documentation for the GNU binary utilities (collectively version 2.13.90), 2003
  Distribution file documents/gnu_tools_ug/binutils.html

- Free Software Foundation, *The GNU Make Manual, for make, Version 3.79*, Edition 0.55, 2000/04/04
  Distribution file documents/gnu_tools_ug/make.html

- Paul D. Smith, *Paul's Rules of Makefiles*, http://www.paulandlesley.org/, 2002/11/02
  Distribution file documents/gnu_tools_ug/make_rules.html

5.2 PowerPC Architecture Documentation

The documentation for the PowerPC CPU and instruction set is somehow ill-organized, and the present distribution can not help much with this situation. For instance, there is no single document that reliably describes the MPC8xx instruction set and the simplified mnemonics that can be expected from the assembler, and the @sda21 address relocation operator appears documented nowhere. There is even a document for which the pdf file is prepared for reverse printing (ppcabi.pdf).

5.2.1 Object Files and Linker Specifications

[PPC_ABI]
Steve Zucker, SunSoft, and Kari Karhi, IBM, *System V Application Binary Interface, PowerPC Processor Supplement*, September 1995
Distribution file documents/powerpc_cpu/obj_link/ppcabi.pdf

[PPC_EABI]
Stephen Sobek, Motorola, and Kevin Burke, IBM, *PowerPC Embedded Application Binary Interface*, 1995/01/10
Distribution file documents/powerpc_cpu/obj_link/ppceabi.pdf

[ELF_SPEC]
Tool Interface Standards (TIS), *Executable and Linkable Format (ELF), Portable Formats Specification*, Version 1.1
Distribution file documents/powerpc_cpu/obj_link/elf.txt

5.2.2 PowerPC Reference Documents

[MPC8XX_INSTR]
Motorola, *MPCxxx Instruction Set*
Distribution file documents/powerpc_cpu/cpu_ref/mpc82xinset.pdf

[PPC_SIMPL_MNEMONICS]
Motorola, *Simplified Mnemonics for PowerPC™ Instructions*, Application note AN2491, Rev. 0, 9/2003
Distribution file documents/powerpc_cpu/cpu_ref/an2491.pdf

[PPC_ARCH32]
Motorola, *Programming Environments Manual For 32-Bit Implementations of the PowerPC Architecture*, document MPCFPE32B/AD, 12/2001, revision 2
Distribution file documents/powerpc_cpu/cpu_ref/mpcfpe32b.pdf

[PPC_ARCH32_ERRATA]
Motorola, *Errata to Programming Environments Manual for 32-Bit Implementations of the PowerPC Architecture, Rev. 2*, document MPCFPE32BAD/AD, revision 0, 10/2002
Distribution file documents/powerpc_cpu/cpu_ref/mpcfpe32bad.pdf

5.2.3 PowerPC Programming Guides

- Motorola and IBM, *PowerPC™ Microprocessor Family: The Programmer's Reference Guide*, document MPCPRG/D, 10/1995
  Distribution file documents/powerpc_cpu/cpu_guides/mpcprg.pdf

- Motorola, *MPC8xx Exception Processing*, application note 2055
  Distribution file documents/powerpc_cpu/cpu_guides/an2055.pdf

- Motorola, *SIU Interrupt Controller*, application note 2056
  Distribution file documents/powerpc_cpu/cpu_guides/an2056.pdf

5.3 Hardware Documentation

5.3.1 PPCMB/850 Documents

[PPCMB850_HW_UG]
CONNOTECH Experts-conseils inc., *PPCMB/850 Hardware User's Guide*, Document Number C001954, 2003/09/20
Distribution file documents/hardware/ppcmb850/ppcmb850_hug_1.pdf

5.3.2 Motorola MPC850 Embedded Processor Documentation

- Motorola, *MPC850 Communications Processor Technical Summary*, document MPC850TS/D, Rev. 2.4 11/2001
  Distribution file documents/hardware/mpc850/mpc850ts_2_4.pdf

[MPC850_UG]
Motorola, *MPC850 Family User's Manual, Integrated Communications Microprocessor*, document MPC850UM/D, Rev. 1, 1/2001
Distribution file documents/hardware/mpc850/mpc850umr1.pdf

[MPC850_UGERRATA]
Motorola, *MPC850 Communications Microprocessor User's Manual Errata*, document MPC850UMAD/D, Rev. 0.3, 6/2002
Distribution file documents/hardware/mpc850/mpc850r1umad.pdf

- Motorola, *MPC850 Device Errata Reference*, document MPC850CE/D, Rev. 3, 2/2003
  Distribution file documents/hardware/mpc850/mpc850ce.pdf

- Motorola, *Revision 2.1 MPC850 Family Device Errata Summary*, 2/24/03
  Distribution file documents/hardware/mpc850/mpc850cesumm.pdf

- Motorola, *MPC850 (Rev. A/B/C) Family Communications Controller Hardware Specifications*, document MPC850ABEC/D, Rev. 1, 10/2002
  Distribution file documents/hardware/mpc850/mpc850abec.pdf

5.3.3 PPCMB/850 Components Documents

- AMD datasheet, *Am29LV160D, 16 Megabit (2 M x 8-Bit/1 M x 16-Bit), CMOS 3.0 Volt-only Boot Sector Flash Memory*, Publication# 22358 Rev: B Amendment/+3, Issue Date: November 10, 2000
  Distribution file documents/hardware/ppcmb850_comp/104-0001-01-amd29lv160d.pdf

- Sipex Corporation datasheet, *SP3490/SP3491 +3.3V Low Power Full Duplex RS-485 Transceivers with 10Mbps Data Rate*, document SP3490/3491DS/23
  Distribution file documents/hardware/ppcmb850_comp/102-0001-01-sp3490_3491.pdf

- Samtec, *2mm SQ Tail Socket SQT Series*, Factsheet F-202-1
  Distribution file documents/hardware/ppcmb850_comp/600-0005-02-sqt_th.pdf

5.4 Software Tools and Utilities Documents

[GNU_GCC_PORT]
CONNOTECH Experts-conseils inc., *MPC8xx-POMP, Port of the GNU Compiler Collection to the Motorola MPC8xx Processor Family*, Release Notes, Document number C002008, 2003/10/07
Distribution file documents/sw_tools/gccmpc8xx_tools_pomp.pdf

[ABCD_LINK_N_LOAD]
CONNOTECH Experts-conseils inc., *ABCD Proto-Kernel™ Software Link and Load Process*, Document Number C002468, 2004/03/09
Distribution file documents/sw_tools/ppcmb850_link_and_load_1_2.pdf

[LAB_COMM_UG]
CONNOTECH Experts-conseils inc., *The ABCD Proto-Kernel Networking Specifications, including the LAB-COM Utility Guide*, Document Number C002424, 2004/03/09
Distribution file documents/sw_tools/labcomm_ug.pdf

5.5 ABCD Proto-Kernel™ and Useful Additions Documents

[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
Distribution file documents/abcd_software/abcd_proto_kernel_1x.pdf

[PPCMB850_INIT]
CONNOTECH Experts-conseils inc., *PPCMB/850 Initialization Sequence, (Embedded Software Document)*, Document Number C001804, 2003/09/29
Distribution file documents/abcd_software/ppcmb850_initialization.pdf

[FLASHCNL]
CONNOTECH Experts-conseils inc., *The FlashCnL API, A Flash Memory Configuration and Log Application Programming Interface*, Document Number C001270, 2003/06/17
Distribution file documents/abcd_software/flashcnlapi.pdf

5.6 Other Documents

[RFC1549]
Network Working Group, *PPP in HDLC Framing*, Internet Request for Comments 1549, RFC1549, December 1993
Distribution file documents/misc/rfc1549.txt

6. The Version 1.x Distribution Limitations

The primary goal of this distribution is to share a *comprehensive solution* for embedded software development. It is expected that future distributions from CONNOTECH will provide local improvements in specific areas. It is thus possible that you need this distribution to get the complete set of files in addition to a future distribution for a specific aspect within the distribution.

6.1 Operating System Requirements

The GCC tools are built to run on a PC running the Linux operating system. The GCC tools represent an incredible value for industrial-strength software development tools. It is worth installing a Linux server just for them.

The lab-comm utility is also running on the Linux operating system, so a Linux laboratory PC is recommended.

Note: The source code of a previous MS-Windows GUI utility achieving the same connectivity as the Linux lab-comm utility is no longer supported nor distributed by CONNOTECH Experts-conseils inc. If you are interested in supporting this embedded system communications software, you should contact US.

Here are a few tips related to the operating system requirements:

- We didn't attempt to install Cygwin (the foremost Linux emulation system under MS-Windows), because native Linux execution was assumed easier to support for GNU development tools.

- It is assumed that the Linux system is already fitted with the native GCC (GNU Compiler Collection), so that simple software utilities written in C or C++ can be easily compiled from the source code distribution.

- The SAMBA file server system is a suitable alternative for file sharing between the Linux and the MS-Windows environments. It is a free software package available on Linux. Because the file names on the MS-Windows systems are case-insensitive, the ABCD Proto-Kernel™ distribution uses lower-case file names.

- In a mixed Linux/MS-Windows environment, on the MS-Windows side, you should install the MinGW/msys execution environment, so that some of the Linux shell commands are available in the MS-Windows environment (notably GNU make). The msys package can be found at http://www.mingw.org/download.shtml. Like any good GPL'ed package, the source for the msys package is also readily available from the same location.

  Notes: 1) We observed that copying the msys file `make.exe` to a directory already in the current path didn't work. Instead, we use the more explicit command name `x:\msys\bin\make`.

  2) The same source provide a port of the GNU GCC for the MS-Windows environment known as MinGW. A quick attempt to build a cross-compiler using this compiler was abandoned after encountering a failure on building a linker-related utility called `collect2`.

There is no instruction set simulator for the PowerPC in the ABCD Proto-Kernel version 1.x distribution, so the execution environment for the ABCD Proto-Kernel™ is the target hardware itself.

6.2 Interactive Debugging

There is yet no interactive debugger adapted to the ABCD Proto-Kernel™ execution environment. The adaptation of the GNU debugger (GDB) to the ABCD Proto-Kernel™ should be a reasonable undertaking.

Nonetheless, the ABCD Proto-Kernel™ implementation for the Motorola MPC8xx processor family has many built-in features that ease the debugging:

- the FlashCnL log recording capability (traces to flash) that is independent of any communications channel,
- PowerPC exception catching with traces to flash,
- troubleshooting LED under software control, and
- traces to serial port, that can share the serial port with other protocols.

6.3 Source Files Status

Any serious software development project requires the use of source code control procedures and version control. In the case of the ABCD Proto-Kernel™ project, these procedures are left out of the version 1.x distribution.

Here are the highlights of the original ABCD Proto-Kernel™ source code control procedures and version control.

- The original ABCD Proto-Kernel™ source files are controlled by a CVS database (the CVS program is the foremost source code control package in the free software world).

- In the ABCD Proto-Kernel™ software building procedures for a given software image file, there are two directory structures, respectively the master directory structure present in the CVS database (where source files are approximately grouped by functional areas), and a build directory (no sub-directory structure).

- The selection of source files from the master directory structure to the build directory is driven by a list of relevant source files in the makefile for the software image file.

- Version control is assisted by a unique procedure for deriving variants of the master source code based on a configuration file. This unique procedure removes unused source code lines between the conditional compilation directives (**`#if`** / **`#ifdef`** / **`#ifndef`** / **`#elif`** / **`#else`** / **`#endif`**) so that the *filtered source files* are an exact representation of the compiled programs. A C preprocessor utility called `prcpp` is used to filter the source files (this utility cleverly implements the C language conditional compilation directives on a *set* of source files to create the set of source files exactly as they are seen by the compiler). The `prcpp` source code filter utility is *not* part of the ABCD Proto-Kernel™ version 1.x distribution, but it is being used to prepare this distribution.

  Note: This source code filtering concept originates from the need to eliminate *dead code* from embedded software subject to security certification (dead code is any instruction sequence that is never exercised under any embedded device operating condition).

7. Specific Licensing Terms of Various Software Components

This section describes the licensing issues at the detailed level.

7.1 The ABCD Proto-Kernel™ Trademark Usage

The ABCD Proto-Kernel™ trade mark belongs to CONNOTECH Experts-conseils inc. Provided that they agree to the GPL-style licenses attached to the software copyrighted by CONNOTECH Experts-conseils inc., free software users and developers who are hereby granted the right to use the ABCD Proto-Kernel™ trade mark according to the following guidelines.

The primary purpose of the ABCD Proto-Kernel™ trade mark is to refer to the set of software control functions, as the name ABCD implies, that are kernel elementary services from which a real-time application software can be build:
a) the original ABCD Proto-Kernel™ interrupt dispatching,
b) the original ABCD Proto-Kernel™ fixed priority scheduling,
c) the original ABCD Proto-Kernel™ mutual-exclusion semaphores, and
d) the original ABCD Proto-Kernel™ queuing mechanism.
The essence of the ABCD Proto-Kernel™ is a matter of simplicity and proper selection of kernel elementary services, and a breach of this simplicity characteristic invalidates the use of the mark to refer to the modified kernel software.

By extension, the ABCD Proto-Kernel™ trade mark may apply to

- the implementation of the above kernel elementary services on a specific processor architecture or processor type, including its system startup sequence, and
- the specialized software development tools required for software image creation and downloading in a target system that supports the ABCD Proto-Kernel™ software execution environment.

A change in the run-time behavior of any of the above kernel elementary services invalidates your right to use the ABCD Proto-Kernel™ trade mark. For instance:

- if the embedded system kernel supports nested interrupts, the ABCD Proto-Kernel™ trade mark is no longer applicable (breach of the simplicity rule),
- if the embedded system kernel supports changing a task's run-time priority other than implicitly through the mutual-exclusion semaphore mechanism, the ABCD Proto-Kernel™ trade mark is no longer applicable,
- if the embedded system kernel supports a queue peek function that would allow an application to read the queue entry beyond the oldest one, the ABCD Proto-Kernel™ trade mark is no longer applicable, or

- if an embedded system kernel controls the dispatching of task execution among two or more identical CPUs, the ABCD Proto-Kernel™ trade mark is no longer applicable. The ABCD Proto-Kernel™ is limited to the control of a single central processor unit (CPU). However, there may be cases where the processor controlled by the ABCD Proto-Kernel™ is coupled with a specialized processor or co-processor (having either a different CPU type or the same CPU type with a different peripheral set dictated by system application requirements) with mechanisms like shared memory or high speed serial communications. The ABCD Proto-Kernel™ branding applies to such schemes.

The following examples are changes that do not invalidate the use of the ABCD Proto-Kernel™ mark:

- a developer may drop, add, or modify, software functions what is called *Useful Additions to the ABCD Proto-Kernel*™, including system tick timer, flash memory interface, trapping processor exceptions, serial port trace functions, maintenance commands, and the like,
- a developer may use other standard C and C++ run-time library implementations, e.g. alternate dynamic memory allocation routines or the addition of file I/O support, or
- a developer may alter the processor initialization sequence, e.g. when required by the above allowed changes.

This is the end of the ABCD Proto-Kernel™ trade mark usage guidelines

7.2 Software Images that Runs on the Target

7.2.1 The ABCD Proto-Kernel™ Application

7.2.1.1 Overview

A typical ABCD Proto-Kernel™-based application is made of ABCD Proto-Kernel™ software source files plus application software source files. Most of the ABCD Proto-Kernel™ software source files are distributed with the LGPL with a Waiver on the Opportunity to Relink, so that under certain circumstances,

- the application source files need not be distributed in source form (as with the LGPL), and
- the application source files need not even be distributed in object form that would allow the software users to re-link the application with a newer version of the ABCD Proto-Kernel™ software.

Some ABCD Proto-Kernel™ software source files are distributed with either the GPL or the LGPL, these files are usually part of the software creation procedures for which the free software model is adopted, and none of these are part of the sample ABCD Proto-Kernel application.
Note: The LGPL with a Waiver on the Opportunity to Relink can be seen as a license stricter

than the (L)GPL with a "link-with-anything exception," but slightly less than the LGPL itself.

7.2.1.2 Notice for LGPL with a Waiver on the Opportunity to Relink

The LGPL with a Waiver on the Opportunity to Relink notice is reproduced below:

Copyright (C) 2003 CONNOTECH Experts-conseils inc.

. . . . . .

Note: The present legal notice is based on the GNU LGPL (Lesser General Public License) with a Waiver on the Opportunity to Relink.

. . . . . .

This library is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License version 2.1 as published by the Free Software Foundation, except for its preamble and clause 6 that are modified as stated herein:

A) DELETE from the preamble fifth paragraph the sentence that reads ``If you link other code with the library, you must provide complete object files to the recipients, so that they can relink them with the library after making changes to the library and recompiling it."

B) DELETE the preamble before last paragraph that reads ``Although the Lesser General Public License is Less protective of the users' freedom, it does ensure that the user of a program that is linked with the Library has the freedom and the wherewithal to run that program using a modified version of the Library."

C) At the end of the first paragraph of clause 6, REPLACE the phrase ``, provided that the terms permit modification ... ... debugging such modifications." by the following phrase ``, provided that the provisions of the following paragraph are followed."

D) At the end of the text in the bullet a) in clause 6, DELETE the phrase ``; and, if the work is an executable linked with ... ... use the modified definitions.)"

E) DELETE the complete paragraph after the bullet e) in clause 6, this paragraph reading ``For an executable, the required form ... ... itself accompanies the executable."

These modifications do not however invalidate any other reasons why derived work might be covered by the exact GNU Lesser General Public License.

This library is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License version 2.1 for more details.

You should have received a copy of the GNU Lesser General Public License version 2.1 along with this library; if not, you may write to the Free Software Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.

HOW TO CONTACT THE COPYRIGHT HOLDER:
CONNOTECH Experts-conseils inc.
home page: http://www.connotech.com
e-mail to: info@connotech.com
mailing address: 9130 Place de Montgolfier, Montreal, Quebec, Canada, H2M 2A1.

Send bug reports to the above address.

7.2.1.3 Non-Free Application Source Files

The explanation in the rest of this document section applies when every ABCD Proto-Kernel™ software source files used in an application software image are distributed with the LGPL with a Waiver on the Opportunity to Relink. Moreover, this explanation is not applicable if any source file uses either the unmodified GPL or the LGPL (e.g. in a specialized library).

The ABCD Proto-Kernel™ documentation does not specify a formal API (Application Programming Interface), but its licensing allows linking with proprietary application source files. It is thus useful to list the source files that can be replaced by proprietary versions.

- Configuration-type files required by the kernel:

  abcd_config.h, for a number of configuration items selected by C preprocessor **`#define`**s,

  mpc8xx_clk_cfg.txt, for the processor clock distribution configuration, an input file to the clock_mpc8xx utility

  mpc8xx_syst_hw_cfg.cpp, for the system startup time configuration of parallel input-output pins in the MPC8xx integrated peripherals.

- Registry-type files:

  ppcmb850_init.mak (the actual name of this file is the software image name as known in the makefile syntax as $(TARGET_PRGM).mak), where the application-specific source files are listed,

  abcd_applic_tasklst.h, where the kernel tasks are declared,

abcd_appl_mutexid.h, where the application-specific mutex rank identifiers are listed

abcd_appl_sw_diagnostics.h, where the application-specific software diagnostics codes are listed.

- Files required by some useful additions to the ABCD Proto-Kernel

  The maintenance commands and the trace mechanism implement a built-in command monitor facility in an embedded application. The following two files allow the customization of this facility:

  abcd_maintcmd_dispatch.cpp, holding an application-specific dispatching of maintenance commands, organized in groups of commands,

  abcd_maintcmd_dispatch.h, holding application-specific definitions that select the groups of maintenance command classes that are to be included in the software image.

  The flashCnL library is intended to manage application-specific configuration items:

  abcd_appl_flashcfgid.h, where the application-specific flash configuration items are listed, along with the relevant data definitions.

- Genuine application source files for interrupt service routines and application code running as ABCD Proto-Kernel Tasks. The relevant source file names should be listed in the abovementioned file ppcmb850_init.mak.

Furthermore, it is normal to adapt the kernel source files to application-specific requirements, and in some cases, these modifications may appear incompatible with GPL-style licensing provisions. For instance, if the target system has an FPGA integrated circuit, the system startup sequence might need a call to a function called e.g. **`fpga_bitstream_download()`**. Then, in order to fulfill the free software distribution obligations, it is best to allow the whole application to compile and link even if the proprietary source files are missing, using dummy functions and/or C preprocessor conditional compilation directives. Using the above example, the following code fragment might achieve the error-free link:

```
#if HAS_PROPRIETARY_APPLICATION
fpga_bitstream_download();
```

```
#endif
```

7.2.2 The PPCMB/850 Target Embedded Loader

The target embedded loader is an ABCD Proto-Kernel™ application, and is distributed under the GPL. This GPL distribution is important to promote the shared support of embedded development tools. The GPL is introduced notably through the files abcd_bin_file_receiv.cpp and abcd_bin_file_receiv.h that implement the receiver portion of the download protocol logic, and more generally through the following application source files:

- registry-type files abcd_applic_tasklst.h, abcd_appl_mutexid.h, and abcd_appl_sw_diagnostics.h,
- file required by some useful additions to the ABCD Proto-Kernel: abcd_appl_flashcfgid.h, and
- genuine application source files for application code running as ABCD Proto-Kernel™ tasks: abcd_loader_comm.cpp, abcd_loader_comm.h, abcd_flwrite_backg.cpp, and abcd_flwrite_backg.h.

With the current system startup procedures and the current flash organization, there is an arm's length programmatic interface between the target embedded loader and the normal ABCD Proto-Kernel™ application (or a system operational application written using other kernel or operating system). Accordingly, the fact that the target embedded loader is used to load the system operational application (and the fact that loader software image remains in the flash memory and might be re-activated) does not force the target system operational application to be distributed under the GPL.

7.2.3 The PPCMB/850 Very Initial Loader

The very initial loader is a special software image that runs from the RAM and writes the embedded loader software image to the flash memory. It is normally used in the PPCMB/850 manufacturing procedures and relies on the BDM port for its own loading (see the description of the PPCMB/850 BDM Handler). The data for the very initial loader is the embedded loader software image, which is obtained as C static initializations produced when the embedded loader is itself created (output from the elf_post_ld utility for the project ppcmb850_bdm).

The very initial loader is made of a small portion of the system startup sequence, the embedded loader software image in the form of C static initializations, and the flash low level programming algorithm. It is distributed with the LGPL with a Waiver on the Opportunity to Relink, although it makes little sense to link a proprietary application with the very initial loader.

7.2.4 The PPCMB/850 BDM Handler

The PPCMB/850 BDM Handler (project name ppcmb850_bdm) is an application program that

runs on the PPCMB/850 hardware hosted on a simple special-purpose circuit (CONNOTECH part number 752-0002-01) that enables the PPCMB/850 to operate as a BDM interface. See the Annex A on page 34 for details. This is the current mechanism for manufacturing initialization of brand new PPCMB/850.

Note: The first few PPCMB/850 units has been initialized with a third-party BDM interface that is broken and a PC-based utility that is no longer supported.

The PPCMB/850 BDM Handler is an ABCD Proto-Kernel™ application, and is distributed under the GPL. This GPL distribution is important to promote the shared support of embedded development tools.

7.3 The lab-comm Utility

The lab-comm utility was written by CONNOTECH and is distributed as a GPL'ed free software, in part because it uses the readline library, which is licensed by the Free Software Foundation with the explicit purpose of attracting additional GPL'ed software contributions. You may refer to the document [LAB_COMM_UG] for more information about conditions for implementing proprietary applications with the ABCD Proto-Kernel™ networking specifications.

7.4 Software Image Creation Utilities

7.4.1 The clock_mpc8xx Utility

The clock_mpc8xx utility was entirely written by CONNOTECH and is distributed as a GPL'ed free software.

7.4.2 The elf_post_ld Utility

The elf_post_ld utility was written by CONNOTECH starting from the FSF binutils software for interfacing with the ELF file format. Specifically, we used some definition files for the ELF data structures. We thus accepted the GPL licensing terms to be applicable to the elf_post_ld utility as a whole. Source files that are specific to the elf_post_ld utility are also affixed with the genuine GPL terms, but the definition files that are shared with the other software components are distributed with the LGPL with a Waiver on the Opportunity to Relink (e.g. files abcd_link_and_load.h, abcd_load_file_hdr.h, and abcd_network_defs.h).

8. Overview of Software Installation and Test

This section gives an overview of the steps in the software installation and test procedure. Some

of the procedure elements described below may be already done at your site, or the intended functions may be provided by slightly different means. Obviously, it is up to you to adapt the procedure accordingly.

8.1 Getting the Distribution files

In this step, you should gather the distribution files, for

- the PowerPC cross-compiler distribution, see http://www.connotech.com/gcc_mpc8xx/powerpc_eabi_mpc850.htm
- the ABCD Proto-Kernel version 1.x distribution, including
  - source files,
  - documentation files,
  - a few executables or software image files,
  - WinPcap library, version 2.3, files `wpdpack_2_3.zip` (sha 330DD75B 22C89218 8EEEA72C 5A421818 0CBA64E3) and `wpcapsrc_2_3.zip` (sha A47DC0CD 6EAEAF26 83940114 C8D1D9DE 7B4D465F).

Unzip and untar the distribution files as is usual.

8.2 Installing PowerPC Cross-Compiler Tools

See the CONNOTECH web site at http://www.connotech.com/gcc_mpc8xx/powerpc_eabi_mpc850.htm to find the cross-compiler distribution. These cross-compiler tools should be installed on the Linux system.

Some files in the ABCD Proto-Kernel™ distribution are actually more relevant to the cross-compiler tools. This includes the file `documents/sw_tools/gccmpc8xx_tools_pomp.pdf` and the user documentation in html format for the cross-compiler tools (files in the distribution directory `documents/gnu_tools_ug/`). Along the same line of thought, the elf_post_ld utility, to be created later in section 8.5 on page 29, could also be considered as part of the cross compiler distribution.

8.2.1 Installing Miscellaneous Utilities

It should be trivial to install two small utilities, `crlf` and `sha`, that are provided in source form, without a corresponding makefile because their compile-and-link procedure. Both are simple enough to be compiled and linked by any minimally standard C compilers.

Utility	Source files	Executable, Linux	Executable, MS-Windows
crlf	source/utils/crlf.c	binaries/i386_linux/crlf	binaries/win32/crlf.exe

sha	source/utils/sha_fertile/sha.c, source/utils/sha_fertile/sha.h	binaries/i386_linux/sha	binaries/win32/sha.exe

Change the execute permissions for the two executables crlf and sha, and move them to the directory /usr/bin.

8.3 Installing Laboratory Tools

A serial port must be reserved for the lab-comm utility on the Linux box that connects to the target hardware. For the PPCMB/850 target hardware, a very simple RS422 adapter (the Kontron CONV422/DB9) and the special pinout cable (CONNOTECH part number 616-0001-01) should be used for this connection.

Note: Ethernet connectivity is supported by the lab-comm utility, but not yet by the PPCMB/850 target hardware.

This single communications channel to the PPCMB/850 target hardware fulfills the BDM interfacing requirement with the use of a PPCMB/850 as a BDM port controller.

8.4 Getting the Source Files in Directories Where the Software Creation Procedures Will Occur

The ABCD Proto-Kernel™ distribution locates source files in a "master" directory structure from where each "project work directory" retrieves the required files (the notation below uses respectively master and work as placeholders for the actual directory locations). There are 7 projects, and many source files are shared between two or more of them:

distribution file master/source/projects/utils/clock_cfg/linux/makefile,
distribution file master/source/projects/utils/lab_comm/makefile,
distribution file master/source/projects/utils/elf_post_ld/linux/makefile,
distribution file master/source/projects/masters/ppcmb850_devloader/makefile,
distribution file master/source/projects/masters/ppcmb850_init/makefile,
distribution file master/source/projects/masters/ppcmb850_bdm/makefile, and
distribution file master/source/projects/masters/ppcmb850_viloader/makefile,

including the last four projects that are cross-compiled for the target execution environment. Each project should have its own build directory, to which the project file makefile is manually copied as a starting point.

Make subdirectories to receive the various projects:

mkdir work/lab_comm
mkdir work/clock_mpc8xx

mkdir work/elf_post_ld
mkdir work/mpc/ppcmbdevload
mkdir work/mpc/ppcmbinit
mkdir work/mpc/ppcmbbdm
mkdir work/mpc/ppcmbviloader

The directory names above are arbitrary, except for the subdirectory name ppcmbviloader that is present in the file makefile in the directory ppcmbdevload in the same parent directory.

In each of the above directory, copy the relevant makefile from the filtered sources database

cd work
cp master/source/projects/utils/lab_comm/makefile lab_comm/.
cp master/source/projects/utils/clock_cfg/linux/makefile clock_mpc8xx/.
cp master/source/projects/utils/elf_post_ld/linux/makefile elf_post_ld/.
cp master/source/projects/masters/ppcmb850_devloader/makefile mpc/ppcmbdevload/.
cp master/source/projects/masters/ppcmb850_init/makefile mpc/ppcmbinit/.
cp master/source/projects/masters/ppcmb850_bdm/makefile mpc/ppcmbbdm/.
cp master/source/projects/masters/ppcmb850_viloader/makefile mpc/ppcmbviloader/.

For the ppcmb850_init and the ppcmb850_bdm projects, since proprietary source files can be linked with the ABCD Proto-Kernel™ software in a single software image (as is allowed by the applicable licenses in some cases) this step is where the proprietary source files list needs to be copied along with the free software files list.

As a special step (to accommodate proprietary applications), do

cp master/source/projects/masters/ppcmb850_init/ppcmb850_init.mak mpc/ppcmbinit/.
cp master/source/projects/masters/ppcmb850_bdm/ppcmb850_bdm.mak mpc/ppcmbbdm/.

Note: the actual name of this file ppcmb850_init.mak and ppcmb850_bdm.mak is the software image name as known in the makefile syntax as $(TARGET_PRGM).mak.

Now you are ready to copy the source files from the master directory to the project directories, according to each project specifications found in the respective makefiles.

In each of the above project work directories, do

make checkout.make
make -f checkout.make

Note: Modifying a source file should be done at the project directory level, and the (validated) changes may be committed to the master directory structure. As it should makes sense to the reader if is familiar with the makefile, the commands to retrieve the master source files, check the differences, and commit the changes are, respectively:

make -f checkout.make

make -f diff.make
make -f commit.make

The three files checkout.make, diff.make, and commit.make are created by the project file makefile, using "make sub_make."

8.5 Creating the Software Images

In each build directory made in the above step, a specific software creation procedure must be run. Generally, the software creation procedure is nominally a "make all" command.

Some files installed in a previous step, or created by a "make all" in another build directory, may be prerequisite to a build directory, and attention should be paid to these interdependencies (they are not always handled automatically).

Using the Linux command syntax, the "make all" can be replaced by "make all 2>&1 | tee tmp.log" that records the screen output to the local file tmp.log.

```
cd work/clock_mpc8xx
make all
cd work/elf_post_ld
make all
cd work/lab_comm
make all
```

Change the execute permissions for the three executables created above, and move them to the directory /usr/bin.

```
mv work/clock_mpc8xx/clock_mpc8xx /usr/bin/.
chmod ugoa+x /usr/bin/clock_mpc8xx

mv work/elf_post_ld/elf_post_ld /usr/bin/.
chmod ugoa+x /usr/bin/elf_post_ld

mv work/lab_comm/lab-comm /usr/bin/.
chmod ugoa+x /usr/bin/lab-comm
```

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
```

Note: The lab-comm software starts by restricting the system permissions to the single one it actually needs, namely CAP_NET_RAW (numeric value **0x2000**). In the Linux operating system, because this precise system permission assignment can not be recorded as a file attribute, it has to be set by the program.

Now, turning to the software images for the embedded system:

```
cd work/mpc/ppcmbdevload
make CPU_MODEL=850de PPCMB850_OSC_FREQ=7372800 all
```

This will bring the big file image_ppcmb850_devloader.cpp in the directory work/mpc/ppcmbviloader.

```
cd work/mpc/ppcmbviloader
make CPU_MODEL=850de PPCMB850_OSC_FREQ=7372800 all
```

The make command parameter

CPU_MODEL=850de

gives the specific model in the MPC8xx microprocessor family. Currently valid selections are 821, 823, 823e, 850, 850de, 850dsl, 850sr, 852t, 855t, 857dsl, 857t, 859dsl, 859t, 860, 860de, 860dp, 860dt, 860en, 860mh, 860p, 860sar, 860sr, 860t, 862p, 862t, 866p, and 866t. This list is built-in the GCC compiler port (see [GNU_GCC_PORT]) and can be extracted with the command

powerpc-eabi-c++ -dumpspecs

(look for a long line starting with the cryptic characters string "%{!mpcu*:"). The specification CPU_MODEL=850de defines the preprocessor symbol _MPC8XX to MPC850de as a pre-defined symbol, and the like.

Similarly, the make command parameter

PPCMB850_OSC_FREQ=7372800

gives the oscillator clock frequency in the target system. The possible values are 10000000, 8000000, 7372800, 6176000, 4000000, and 3686400. The makefile uses this parameter value to alter the input file to the clock_mpc8xx utility.

For the software images created in the directories work/mpc/ppcmbdevload, work/mpc/ppcmbinit and work/mpc/ppcmbbdm, the utility elf_post_ld reports a master CRC for the software image as an hexadecimal value. This master CRC is independently reported by the ABCD Proto-Kernel™ system initialization sequence, through the serial port traces that can be viewed with the utility lab-comm.

```
cd work/mpc/ppcmbinit
```

make CPU_MODEL=850de PPCMB850_OSC_FREQ=7372800 all

cd work/mpc/ppcmbbdm
make CPU_MODEL=850de PPCMB850_OSC_FREQ=7372800 all

8.6 Initial Flash Programming: Loading of the Target Embedded Loader, Sample Application

This step is normally done by CONNOTECH as part of the PPCMB/850 target hardware manufacturing process. It is not needed by embedded application developers unless the flash memory contents is seriously damaged, that is the embedded loader software image in the flash is corrupted or erased.

Note: In the more usual case where a faulty application is loaded in the flash, there exists a backup procedure, which is a momentary wire jumper connection on the PPCMB/850 when the system is reset (as described in the reference [PPCMB850_HW_UG]).

The initial flash programming can be done by the developer if the proper hardware is available. See the Annex A on page 34 for details. In the near future, an operator guide document for the initial test, flash loading, and manufacturing configuration data might be released. In the meantime, the BDM control software is distributed as a sample ABCD Proto-Kernel™ application (project name ppcmb850_bdm).

8.7 Development of Application Software Revisions

This section describes the routine compile-link-load-and-try cycle that occurs in developing a new version of an application software.

8.7.1 Selection Between Load-and-Run or Boot-and-Run

An application software image can be prepared either for the *load-and-run* memory model or the *boot-and-run* memory model. During the initial troubleshooting period for a new software version, it is recommended to use the *load-and-run* memory model. When the developer is relatively confident that the application runs without corrupting the system, he should proceed with tests using the *boot-and-run* memory model.

The selection of the loading method is done in the project files makefile and abcd_config.h. In the file abcd_config.h, the following source line selects to the *load-and-run* memory model:

```
#define ABCD_TARGET ABCD_TARGET_LOAD_AND_RUN
```

whereas the following source line selects to the *boot-and-run* memory model:

```
#define ABCD_TARGET ABCD_TARGET_BOOT_AND_RUN
```

In the file makefile, the following specification line selects to the *load-and-run* memory model:

```
LD_SCRIPT=ppcmb850_ram.ld
```

whereas the following specification line selects to the *boot-and-run* memory model:

```
LD_SCRIPT=ppcmb850.ld
```

Note: This double entry (files makefile and abcd_config.h) for a configuration element infringes the best practice rule saying that a configuration element should be specified in a single place.

8.7.2 Loading of Application Software Revisions

In the sample ABCD Proto-Kernel™ application, the maintenance command "/LD LOADER " triggers a system reset followed by entry into the embedded loader. This command is part of the routine loading method for a new software version, as long as the maintenance command "/LD LOADER " remains in future application software revisions. It is perfectly acceptable to implement another command or condition (that makes more sense in the new application context) that triggers the entry into the embedded loader.

Here is the step-by-step sequence for an application software image load:

- Connect the serial port as described in section 8.3 on page 27.
- Start the executable file lab-comm.
- Then use the appropriate command to indicate the serial port selection, as in ".COM PORT "/dev/ttyS0"".
- Set the serial port RTS signal mode with the command ".COM RTS 1".
- Set the serial port connection communications speed at 115200 bauds, using the command ".COM SPEED 115200".
- Open the serial port connection with the command ".COM CONNECT".
- Reset the target system, check the software image that runs (e.g. from the master CRC for the software image).
- Make sure that the embedded loader software image runs, e.g. send the command "/LD LOADER" if the application software is running.
- Download the software image, e.g. by sending the command

"/LOAD work\mpc\ppcmbinit\image_ppcmb850_init.bin" and you are now ready to try the application.

- Actually, the application starts automatically after a software image download. In the case of the *load-and-run* memory model, the application must be loaded each time it is ran. With the *boot-and-run* memory model, a system reset starts the loaded application since it resides in flash.

8.7.3 Preserving the Application Loading Capabilities

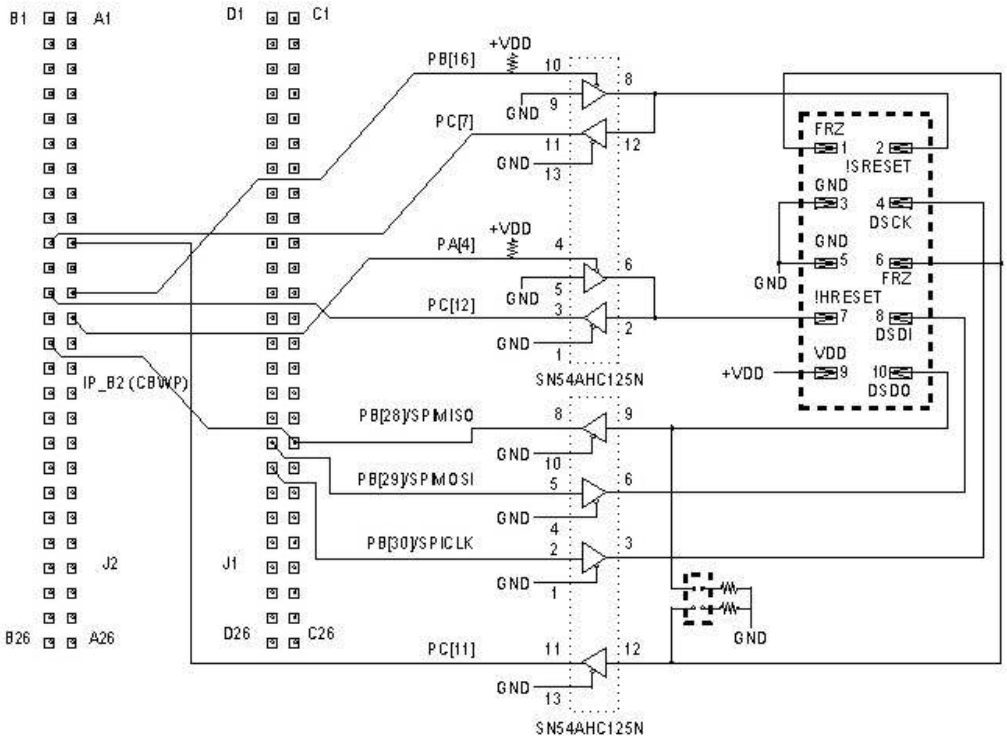
Loading to the flash memory should be done only when it has been verified that some loader entry command is operational within the new software version (either the "/LD LOADER" or another command or condition that makes sense in the new application context). Such a command is the recommended means for field-loading of new software versions, for system designs where this capability is desired.

However, if an application software fails to implement a loader-entry command, the loading method will revert to the backup procedure, which is a momentary wire jumper connection on the PPCMB/850 when the system is reset (as described in the reference [PPCMB850_HW_UG]).

Annex A - Operating Principles for the BDM Hardware

This annex is a brief description of the CONNOTECH part number 752-0002-01. It gives the electronic circuit diagram with sufficient level of details for software development and compatible hardware construction as well.

The J1 and J2 connectors shown on the diagram is for the PPCMB/850 unit that controls the BDM connector. The BDM connection goes from the BDM header connector on the 752-0002-01 to the remote PPCMB/850 connector J4. The power and ground for the two PPCMB/850 units are not shown on the diagram. Actually, both of them are powered from a single +3.3 VDC source.



When the BDM handler board is plugged and the BDM connector is left un-connected, the two jumpers should be installed. In this case, the input values read on the pins PC[28]/SPIMISO, PC[10], PC[7], PC[6], are all logical '0'.

Compatibility with other BDM targets:

The above diagram has been designed for the MPC8xx BDM port where the VFLS0 and VFLS1 pins are not used (hence the FRZ signal that occurs on pins 1 and 6 of the BDM connector). Compatibility with other BDM port configuration and pinouts is unknown. Moreover, there is no power and ground protection and/or isolation between the remote target unit and the BDM port control unit. The reader is advised to consider the implications of these facts before using the above diagram with other BDM targets.