**Freescale Semiconductor**
Application Note

Document Number: AN2950
Rev. 0.2, 07/2005

# MCF547x/8x Linux BSP Quick Start

This document provides instructions for getting started with the MCF547x/8x BSP.

**NOTE**

The 2.4 PCS BSP has been tested with RedHat 8 and 9, Fedora Core 3, SuSe 8.2, 9.0, 9.1, and 9.2, and Debian 3.0. The BSP *without* PCS has be tested with RedHat 9, SuSe 9.2, and Fedora Core 1.

**Neither BSP works with Cygwin.**

As of the writing of this document, there are two Linux BSPs for the MCF547x and MCF548x microprocessors.

1. The first BSP is a free, standard Linux BSP containing the kernel, core tools, and compilers for the target board. It requires no additional software to deploy on the target.

**Table of Contents**

2. The second BSP (PCS BSP) is configured to work with the Metrowerks Platform Suite (PCS) Tool. PCS is a tools framework into which Metrowerks Linux BSPs are integrated into a development environment. It allows users to configure, extend, build, and deploy Linux software with ease.

These BSPs currently support the following boards:

1. M5475EVB
2. M5485EVB
3. M5474LITE
4. M5484LITE

The M5475EVB and M5485EVB development systems provide everything required to quickly develop an embedded design with the MCF547x/8x processor family. The kit contains a system-on-module, called a Fire Engine, containing 64 MBytes of DDR-SDRAM, 16 MB of Flash, a graphics controller, and a 4-port USB host.

The M5474/84LITE development systems provide a more cost effective solution for getting started with the MCF547x/8x processor family. The kit contains a Fire Engine, containing 64 MBytes of DDR-SDRAM, 4 MB of Boot Flash, and a USB Device output.

All MCF547x/8x BSPs can be downloaded from any of the specific MCF547x/8x processor specific pages, or directly from the Metrowerks website: http://www.metrowerks.com/MW/download/default.asp.

**Table 1. MCF547x/MCF548x Linux BSP Components**

| | **Nov BSP** |
|---|---|
| Kernel | v2.4.26 |
| GCC | v3.4 |
| GDB | v6.2 |
| GLIBC | v2.3.2 |
| Ethernet Driver | included |
| I2C Driver | included |
| Serial Driver | included |
| PCI Driver | included |
| Multi-Channel DMA | included |
| Low-Latency Patch | included |
| Colilo Flash Boot | included |
| SEC Driver | included |
| NFS Deployment | included |
| TFTP Deployment (kernel from SDRAM) | included |
| TFTP Deployment (kernel from Flash) | included |
| GDBServer | included |
| User Land Support | included (binary and sources) |
| Application Packages | included |

Table 1. MCF547x/MCF548x Linux BSP Components (continued)

| | Nov BSP |
|---|---|
| Preemptible Kernel Patch | included |
| IPSEC | included |
| PCI WLAN (Prism54) | included |
| Flash/JFFS/JFFS2 Support | included |
| TFTP Deployment (root from flash) | included |
| IRDA Support | included |
| PCI LAN (native kernel drivers) | included |

## 1 Directory Contents

- colilo - contains colilo (linux loader) sources and.srec files.
- defconfig - contains default configuration files.
- toolchain - contains toolchain binaries.
- crosstool - contains the set of scripts for building toolchain.
- gdb - contains GDB source files.
- linux-2.4.26 - contains Linux kernel 2.4.26 source files.
- merge - contains files that are added to the kernel root filesystem.
- romfs - contains the root filesystem.
- tools - contains tools that perform some building operations.
- user - contains userland applications source files.
- images - contains image files (the image that should be loaded to the board is image.bin).

## 2 Linux BSP Installation Without PCS

To build the kernel and userland applications, follow these steps:

1. Unpack the tar.gz file:
   *tar xzvf <the_name_of_Linux_BSP.tar.gz>*
2. Go to LinuxBSP/ directory
3. Type *make fresh*

Then, choose one of the following processes:

a) If you want to use the default configuration for your board:

4. Type *make config-m547xevb*
   (config-m547xevb, config-m547xlite, config-m548xevb, config-m548xlite are supported)
5. Type *make dep*
6. Type *make*

b) If you want to manually configure the kernel and userland:

4. Type *make xconfig*

   Choose the necessary userland applications and kernel options. Make sure that correct CPU type is specified (for busybox configuration, 'menuconfig' is called instead of 'xconfig').
5. Type *make dep*
6. Type *make*

The compilation process will start.

As a result, the image.bin file will appear in./images directory. This file can be downloaded to the board using Colilo, described in Section 2.1, "Image loading Steps Using Colilo."

**NOTE**

If you have problems building the kernel with current toolchain binaries, try to rebuild the toolchain from sources. To do this type: *make toolchain.*

## 2.1 Image loading Steps Using Colilo

1. The Colilo directory contains RAM and Flash versions of Colilo. Download the appropriate Colilo.srec file to address 0x1000000 in RAM or 0xE0000000 in Flash using dBUG (serial or tftp download). Set up the dBUG network options, set the file name, and type

   *dBUG> dn.*

2. Run Colilo from 0x1000400 in RAM (0xE0000400 in Flash):

   *dBUG> g 0x1000400*
3. Set Colilo network parameters and file name (type '?' for help):

   *colilo> set ip <board_ip>*

   *colilo> set netmask <mask>*

   *colilo> set server <server_ip>*

   *colilo> set image <image_file>*
4. Download the *image.bin* file to address 0x1000:

   *colilo> tftp 0x1000*
5. Run image from address 0x2000:

   *colilo> g 0x2000*

This will run the kernel.

After these steps, you should see the login prompt.

# 3 Makefile Targets Description

- *fresh* - preparation before building (this target must be called before first build), restores default configuration.
- *images* - building of all images without romfs structure rebuilding.
- *linux* - building of Linux kernel.

- *user* - building of the userland applications.
- *gdb* - building of GDB.
- *romfs* - creating of romfs.
- *romfs_image* - building of the romfs image.
- *user_install* - installation of the userland applications.
- *modules_install* - installation of Linux modules.
- *clean* - deleting of all object and executable files.
- *user_clean* - deleting of userland object and executable files.
- *romfs_clean* - deleting of romfs object and executable files.
- *linux_clean* - deleting of linux object and executable files.
- *dep* – building of dependencies.
- *default* - copying of default configuration files.
- *colilo* - building of colilo srec files.
- *toolchain* - building of the toolchain (deletes the current version of the toolchain).

## 4 Linux BSP Installation With PCS

To set up the Linux BSP, make sure you have PCS installed and follow these steps:

As root:

1. Mount the supplied .iso image file to a location (e.g. /mnt/disk):
   m*ount mcf5485_5475-3.0.iso /mnt/disk -o loop*
2. Go to /mnt/ and run disk/install.sh file. This installs the new platform to PCS.

As user:

3. Run PCS (type 'tw' in the console).
4. Choose Project -> New from the menu bar.
5. Specify the project name, path, and other necessary options. On the last step of new project creation, choose 'm68k mcf5485_5475' as your target platform. Click ‘Finish.’
6. You should now see the project structure. Choose the components to build and install.

**NOTE**

To debug userland applications, make sure you have 'Programming->Debugger->gdb application debugging->Include /usr/bin/gdbserver?' option enabled.

7. Press the 'Build all' button (or press F8). This builds the kernel and userland applications.
8. Press the 'Deploy' button. This starts the deployment wizard. Follow these steps:
   1. During the first step, choose Deployment method, Board type, and Colilo options. For more information about available options, use the help system.

To debug userland applications, it is best to choose:

a. *Application Development Deploy*: this places the kernel into flash and allows use of nfsroot.

b. *Download, save and run colilo from Flash* for the first deployment only. This places colilo into flash and allows one to type 'go 0xE0000400' at the dBug prompt after board reset to start the kernel.

c. *Save dBUG vars in flash.*

d. *Configure colilo for autobooting of Linux after starting*. Options should be checked. This allows colilo to start the kernel automatically. Click 'Next.'

2. Build the target root file system in a directory on the host. After clicking 'Next' you will see rootfs creation process.

3. The next step allows you to remove debug information from the file system image. DO NOT DO THIS if you want to debug applications (the check box should be checked). Click 'Next.'

4. Enter host info and click 'Next.'

5. Enter target network options.

6. The next step allows you to update the root file system with new network options BUT if the nfs root was chosen during the first step, it is recommended to check 'Do not change the target root file system' box. Click 'Next' button.

7. Check the box here if you want to automatically bring up the network interface on the target board. Do not check this box if the nfs root was chosen during the first step. Click 'Next.'

8. This step configures the host tftp and nfs services. Check the 'Do not change host configuration' box if your host is already configured or if you want to configure it manually. Click 'Next.'

9. During this step, the target MAC address should be specified. The command line may remain default. Click 'Next.'

10. Next step prepares ROMFS and JFFS file systems.

11. Follow the instructions on the page. Click 'Next.'

12. On this page, choose the host serial port device and terminal program.

13. During the next steps, the terminal program is run and commands are sent to dBUG. It loads the colilo, which loads the kernel and executes it. Make sure the board is connected to the network.

After these steps, you should see the login prompt.

# 5 Userland Applications Debugging Without PCS

To debug userland applications using gdbserver, follow these steps:

1. gdb and gdbserver should be built. To build gdb, type: *make gdb*.
2. Upload image.bin to the target and run it.
3. romfs should contain the ELF file that you want to debug.
4. Run gdbserver on the target by typing gdbserver <board_ip>:<port> <path_to_application>

   For example: *gdbserver:3000 /bin/hello*

5. On the host PC, execute: *m68k-linux-gdb*

   The gdb prompt will appear. Type:

   *gdb> file <path_to_file_to_debug>*
   (file should be with debug information)
   *gdb> set solib-absolute-prefix <path_to_targets_filesystem>*
   (it needs it to search for shared libraries; it adds 'lib/' to this path)
   *gdb> set remotetimeout 60*
   (!!!very important)
   *gdb> target remote <ip_address_of_target>:3000*

   **NOTE**

   All these commands can be placed in the gdb command file. To run gdb type:
   *./m68k-linux-gdb --command=<command_file_name>*

   **NOTE**

   It can take up to 30 seconds to start executing/debugging the application.

6. You should now be able to *stepi,* insert breakpoints, print registers, etc.

## 6 Userland Applications Debugging With PCS

To debug userland applications using gdbserver without PCS, follow these steps:

1. The kernel should be built with 'Programming->Debugger->gdb application debugging->Include /usr/bin/gdbserver?' option enabled and network configured.
2. Upload the kernel to the target using the deployment process described above in Section 2, "Linux BSP Installation Without PCS."
3. Assume that romfs contains the ELF file that we want to debug (see Section 10, "Adding a New Userland Application to the Project Without PCS").
4. Run gdbserver on the target gdbserver <board_ip>:<port> <path_to_application>

   For example: *gdbserver 192.168.1.33:3000 /bin/hello*
5. On the host PC: execute *m68k-linux-gdb*

   This file is situated in <path_to_project>/emb-bin/. The gdb prompt will appear. Type:

   *target remote <board_ip>:<port>*

   *add-symbol-file <path_to_application_executable_on_the_host>*

   For example:

   *target remote 192.168.1.33:3000*

   *add-symbol-file /tftpboot/Linux_BSP/romfs/bin/hello*
6. You should now be able to *stepi,* insert breakpoints, print registers, etc.

# 7 Kernel Debugging Without PCS

## 7.1 To Debug the Kernel Using bdm-gdb

1. Build the kernel and all selected userland applications.
2. Build GDB for BDM using *gdb_bdm* target:
   make gdb_bdm
3. Connect the target board to the host using bdm-lpt cable.
4. Power up the board.
5. Execute (as root!):
   <project_dir>/m68k-gdb-bdm/debug.sh
   This runs GDB for BDM and load *5485.gdb* configuration file.
6. On the GDB command prompt type the following commands:
   ```
   c
   Ctrl + c
   load
   set $pc=0x2000
   hbreak *(<function name>)
   c
   ```
   Warning: You can use only hardware breakpoints.
7. Now you can continue debugging the kernel.

## 7.2 To Debug the Kernel Using Abatron BDI 2000

1. Prepare the Abatron BDI to work with the MCF547x/8x target boards:
   1. If your BDI does not support the new boards (547x/8x), you should update its firmware and .cfg files.
   2. Connect the BDI2000 to the 5485/75 board and network.
   3. Set the telnet connection to BDI2000 from the host PC:
      *telnet <ip_address_of_BDI>*
   4. When you see BDI prompt type:
      *reset*
      *go*
      *halt*
      This resets the board, runs dBug, and halts the execution. We use dBUG to initialize the hardware on the board when colilo is not used.
2. Build the kernel.
3. Copy your image/image.bin file to your tftp directory (e.g. /tftpboot).
4. Switch to the console with BDI telnet session running and type: *load 0x1000 image.bin*
   This loads the binary to the board.

5. Run DDD and specify the m68k-linux-gdb as debugger.
6. Open one more console window and make the serial connection to the board from the host PC:
   *tip -s 19200 /dev/ttyS0*
7. On the DDD command line type:
   *file <path_to_vmlinux>*
   *target remote <ip_address_of_BDI>:2001*
8. Now you can debug the kernel.

## 8 Kernel Debugging With PCS

This section describes the two different processes a user can employ to debug the kernel: one using bdm-gdb, and another using Abatron BDI 2000.

### 8.1 To Debug the Kernel Using bdm-gdb

1. Build the kernel and all the selected packages in PCS.
2. Connect the target board to the host using bdm-lpt cable.
3. Power up the board.
4. Execute
   *<project_dir>/emb-bin/bdm_linux_debug.sh*
   This builds the image for the bdm and runs the DDD.
5. On the DDD command line, type the following commands:
   *file <path_to_vmlinux>*
   *source 5485.gdb*
   *c*
   *load*
   *set $pc=0x2000*
   *hbreak *(start_kernel)*
   *c*
   Warning: You can use only hardware breakpoints (4).
6. Now you can debug the kernel.

### 8.2 To Debug the Kernel Using Abatron BDI 2000

1. Prepare the Abatron BDI to work with the MCF547x/8x target boards:
   1. If your BDI does not support the new boards (547x/8x), you should update its firmware and .cfg files.
   2. Connect the BDI2000 to the 5485/75 board and network.
   3. Set the telnet connection to BDI2000 from the host PC:
   *telnet <ip_address_of_BDI>*

4. When you see BDI prompt type:

   *reset*

   *go*

   *halt*

   This resets the board, runs dBug, and halts the execution (we use dBUG to initialize the hardware on the board when colilo is not used).

2. Build the kernel and all selected packages in PCS.
3. Run the deployment process. During the first step of the deployment process, select ‘ROMFS Deploy’ as Deployment Method. This combines kernel and romfs images into one binary file.
4. Stop when you reach the 11th step of deployment (Set Jumpers, Connect Cables, and Power ON). At this stage you will get the ‘linux_tftp.bin’ file in your /tftpboot directory. You can now cancel the deployment.
5. Switch to the console with BDI telnet session running and type:

   *load 0x1000 linux_tftp.bin*

   This loads the binary to the board.
6. Run ‘bdi_kernel_debug.sh’ script from the ‘<project_dir>/emb-bin/’. This script will run DDD.
7. Open one more console window and make the serial connection to the board from the host PC:

   *tip -s 19200 /dev/ttyS0*
8. On the DDD command line type:

   *target remote <ip_address_of_BDI>:2001*

   *file <path_to_vmlinux>*
9. Now you can debug the kernel.

# 9 Setting Up Device Drivers and Modules

## 9.1 FEC Driver

To include the FEC driver in compilation, the following options should be enabled:

- kernel->Network device support->
  - Fast Ethernet Controller (FEC) (with all dependencies)
  - Auto-Negotiation enable
  - Second FEC enable
- kernel->Networking options->
  - Packet socket
  - Unix domain sockets
  - TCP/IP networking
- Enable ifconfig compilation (it is included in system->utilities->busybox).

If your kernel is up and running from nfsroot, the FEC driver works perfectly. Additionally, you can set up the second FEC adapter by typing on the target console:

*/sbin/ifconfig eht1 <board_ip2>*

It is also possible to ping the board from the host PC.

## 9.2 I2C Driver

To include the I2C driver in compilation, the following options should be enabled:

- kernel->character devices->I2C support->
  - I2C ColdFire
  - I2C device interface
  - I2C /proc interface

Also, make sure that you have I2C tools included in compilation (Administration->Hardware->I2C-tools).

To check the I2C driver, you can type on the target's console:

- *i2cdetect* will show you the installed adapters and buses.
- *i2cdetect 0* will scan the bus for the clients. It should find the client with address 0x32. This is the RTC device.
- *i2cdump 0 0x32* will dump the registers of the RTC device (can take a long time).

## 9.3 IRDA Driver

To include the IrDA driver in compilation, the following option should be enabled:

- kernel->Character Devices->Support for serial IrDA port

The IrDA device file is /dev/ttyS3.

## 9.4 SEC Driver

To include the SEC driver in compilation, the following options should be enabled:

- kernel->Cryptographic options->
  - Security Encryption Controller (SEC) support
  - Cryptographic API->
    - MD5 digest algorithm
    - SHA1 digest algorithm
    - DES and Triple DES EDE cipher algorithms
    - AES cipher algorithms
    - ARC4 cipher algorithm

It is also possible to enable ‘Testing module’ compilation to test SEC support.

## 9.5 IPSEC Module

To include the IPSEC driver in compilation, the following options should be enabled:

- kernel->Network Device Support->
  — IPSEC tunnel device

To include *ipsecadm* you should enable:

- Administration-> Administration_Network->tcp_wrappers->ipsecadm

The IPsec tunnel implementation is divided into two parts: one kernel module called ipsec_tunnel.o, and a tool to administrate security associations and tunnels called ipsecadm.

The ipsecadm tool has a built-in brief help which is displayed if you execute it without parameters, or with unknown or invalid parameters. The tool has three main modes: the first is for adding and removing security associations (SAs), the second is for adding, modifying and removing tunnels, and the third mode is for displaying statistics. Before we start, we need an example scenario. Let's say that we are going to create a tunnel between the two hosts A and B. The public IP number of host A is 1.2.3.4 and its private IP address is 10.0.1.0/24.

IPsec uses Security Associations (SAs), which are another name for a security agreement between two hosts. The SA is uniquely described by two IP addresses and a 32-bit number called a SPI. The SPI allows more than one SA between a pair of hosts. When the SA was agreed upon, the two parties agreed upon the type of the SA which can basically be encryption and/or authorization, but also the algorithms, key sizes, and keys to be used.

Before we can create an SA, we need a key. The best way to create a key is to use *ipsecadm*. To create a 192-bit key (which corresponds to 24 bytes) and put it in the file /etc/ipsec/ipsec.key run the following:

```
mkdir /etc/ipsec
chmod 500 /etc/ipsec
ipsecadm key create 3des --file=/etc/ipsec/example.key
```

Before the previous command execution on the nfs file system, we need to make it available for writing:

```
mount –o remount,rw /dev/root /
```

For the romfs we have to place the generated key to ‘<project_dir>/merge/etc/ipsec/’. Now we can use *ipsecadm* to create the SA above. The name of the cipher is specified using its CryptoAPI name, which can be a little strange. The name for triple DES is des3_ede. You can see which ciphers you have installed by looking in the file /proc/crypto.

```
ipsecadm sa add --spi=0x1000 --dst=192.168.1.16 --src=192.168.1.15 \
      --cipher=des3_ede --cipher-keyfile=/etc/ipsec/example.key \
      --duplex
```

To create a tunnel you need two SAs, one in each direction. Since it is common to use the same security settings for both directions, you can create a pair of SAs at one time by using --duplex. Note that it is safer to supply the key as a file (using --cipher-keyfile) than to specify it on the command line (using --cipher-key), because it is easy for a local user to locate the key using w or ps.

To look at the two SAs you just created, run the following:

*ipsecadm sa show*

Note that the SAs can be used for all IPsec traffic, not only tunnels, but right now only tunnels are implemented. Also make sure you add the SAs on both hosts. The next step is to create the tunnels. The commands needed for host A will be shown. Run the following:

*ipsecadm tunnel add ipsec1 --local=192.168.1.15 --remote=192.168.1.16*

There is no need to specify the SPI. You can if you wish using the --spi option, but if unspecified, the local and remote addresses will be used to find a suitable SA.

Now we need to set an IP number on the newly created tunnel device, but what should it be? When a packet is created on a machine, the destination address is used by the routing table to find the outgoing device for the connection, and the source address is set to the address of the device. If you initiate a connection from host A to an address on the private network at host B, you probably want the source address of your packets to be the private address of host A, which is 10.0.0.1:

*ifconfig ipsec1 10.0.0.1 up*

The last step is to create an entry in the routing table that makes the packets to the private network at host B go out via the ipsec1 device:

*route add ipsec1 10.0.0.2*

Do the equivalent on host B and you should be set! Note that you should not generate a new key on host B, but copy the one you made for host A to host B.

## 9.6 WLAN Driver

To include the WLAN driver in compilation, the following options should be enabled:

- kernel->general setup->Support for hot-pluggable devices
- Code maturity level options->
  - Prompt for development and incomplete code/drivers
- Network device support->Wireless LAN->Intersil Prism GT/Duette/Indigo
- Library routines->Hotplug firmware loading support
- For wireless tools compilation:

  System->Network->Wireless-tools->(enable all)

Insert the Wi-Fi card in the PCI slot. Boot the kernel. Type on target console:

```
/sbin/ifconfig eth2 10.0.0.2
```

and the LED on the Wi-Fi card should light up. Now you can use the wireless tools to configure the network.

## 9.7 PCI Driver

To include the PCI driver in compilation, the following options should be enabled:

- kernel->general setup->ColdFire PCI support->PCI device name database
- Administration->Hardware->pciutils->include /sbin/setpci?
- Administration->Hardware->pciutils->include /sbin/lspci?

To make sure that PCI driver is loaded type:

```
lspci –vvv
```

This prints the list of all connected PCI devices.

## 9.8 Flash Driver (jffs, jffs2)

Before the first use of jffs, clear the flash using the board's dBUG utility. At the debug prompt, type:

```
fl e E0000000 200000
```

To enable the jffs2 support, the following options should be enabled:

```
kernel->Memory Technology Device (MTD) support->
    RAM/ROM/Flash chip drivers->
        Detect Flash chips by CFI probe
        Flash chip driver advanced configuration options->
            Flash cmd/query data swapping
            Specific CFI Flash geometry selection->
                Support 16-bit bus width
                Support 1-chip flash interleave
        Support for Intel/Sharp flash chips
    Mapping drivers for chip access->
        CFI Flash device in physical memory map->
            Physical start address of flash mapping = 0xf0000000
            Physical length of flash mapping = 0x200000
            Bus width in octets = 2
    MTD partitioning support
    MTD concatenating support
    Direct char device access to MTD devices
    Caching block device access to MTD devices
```

Boot up the kernel and type in the console:

```
mount –t jffs2 /dev/mtdblock/2 /mnt
```

to mount jffs2 file system to /mnt.

## 10 Adding a New Userland Application to the Project Without PCS

All paths are given relative to the BSP's folder. This example shows the creation of a ‘hello’ application.

1. Create 'hello' directory in the 'user/' directory.
2. Add configuration variable 'hello' to the "user/config.in' file:

   ```
   ...
   bool 'hello'        CONFIG_USER_HELLO   # define the 'hello' variable
   ...
   ```

   This adds the ‘hello’ menu option to userland configuration menu.
3. Add following lines to the 'user/Makefile' file:

   ```
   ...
   DIRSc$(CONFIG_USER_HELLO)           += hello       # Add the 'hello' directory
   ...
   ifeq ($(strip $(CONFIG_USER_HELLO)),y)
      make -C hello CC=$(shell cat ../cross-compile-prefix)gcc
   endif
   ...
   ifeq ($(strip $(CONFIG_USER_HELLO)),y)
      install -m 755 ./hello/hello /usr/bin       # Install the 'hello' application to romfs
   endif
   ```
4. The Makefile in ‘user/hello/’ should look like the following:

   ```
   EXEC = hello
   OBJS = hello.o
   CFLAGS += -Wall -O0 -g -mcfv4e
   LDFLAGS += -mcfv4e
   #LDLIBS += -lpthread

   all: $(EXEC)

   $(EXEC): $(OBJS)
   $(CC) $(LDFLAGS) -o $@ $(OBJS) $(LDLIBS)

   $(OBJS): hello.c
   $(CC) $(CFLAGS) -c hello.c

   clean:
   rm -f $(EXEC) *.elf *.gdb *.o
   ```

## 11 Adding a New Userland Application to the Project With PCS

All paths are given relative to the project's folder.

1. Go to 'config-data/buildcontrol/board/' and copy one of the .lbc files (e.g.'sh.lbc') to 'config-data/buildcontrol/local' with a new name (e.g. 'hello.lbc'). It will be used as the template.
2. Edit this file (hello.lbc) making corrections to all necessary fields (bld_dir_name, makeb, makei, pkg_file, etc.). This file is responsible for making and installing the application.
3. Go to 'config-data/ecds/board/' and copy one of the .ecd files (e.g.'sh.ecd') to 'config-data/ecds/local' with a new name (e.g. 'hello.ecd'). It will be used as the template.
4. Edit this file (hello.ecd) making corrections to all necessary fields. This file is responsible for creating the record in the project's configuration menu and connecting it with the application. It also defines all the dependencies and requirements for the application.
5. Copy the '.tar.gz' file with your application (e.g. 'hello.tar.gz') to 'packages/local/'.
6. Run PCS. Open the project. You should see you application in the component's list. Enable it and choose 'Reinstall and Rebuild Current Component' from the menu. It will unpack hello.tar.gz file, build it and install to romfs if necessary.

For more detailed information about .lbc and .ecd files format, refer to PCS documentation.

**NOTE**

To debug this application, make sure it is compiled with a ‘-g’ option. This option can be added either to the application’s Makefile or to the ‘makeb’ section of the ‘hello.lbc’ file.

## 12 Revision History

Table 2 provides a revision history of this document.

**Table 2. Revision History**

| Rev Number | Date of Release | Substantive Changes |
|---|---|---|
| 0.2 | 07/2005 | Added to NOTE on Page 1: PCS test info. |
| 0.1 | 04/2005 | Added Cygwin NOTE on Page 1. |
| 0 | 02/2005 | Initial customer-release version. |

THIS PAGE INTENTIONALLY LEFT BLANK

THIS PAGE INTENTIONALLY LEFT BLANK

THIS PAGE INTENTIONALLY LEFT BLANK

## How to Reach Us:

**Home Page:**
www.freescale.com

**E-mail:**
support@freescale.com

**USA/Europe or Locations Not Listed:**
Freescale Semiconductor
Technical Information Center, CH370
1300 N. Alma School Road
Chandler, Arizona 85224
+1-800-521-6274 or +1-480-768-2130
support@freescale.com

**Europe, Middle East, and Africa:**
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
support@freescale.com

**Japan:**
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064, Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

**Asia/Pacific:**
Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T., Hong Kong
+800 2666 8080
support.asia@freescale.com

**For Literature Requests Only:**
Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

Document Number: AN2950
Rev. 0.2
07/2005