Getting started with EPICS on RTEMS

W. Eric Norum

October 19, 2009

Contents

Chapter 1

Introduction

This tutorial presents the steps needed to obtain and install the development tools and libraries required to run EPICS IOC applications using RTEMS. As you can see by the size of this document the process isn't trivial, but it's not terribly difficult either.

Chapter two deals with the problem of getting all the tools in place. This is the most difficult task. Once the tools and operating system are working most of the work is complete. Chapter three shows the steps needed to configure and build your first EPICS application for RTEMS. After you've completed those steps you can forget about this document and use the generic EPICS documentation.

This is a living document. Please let me (norume@aps.anl.gov) know which of these instructions worked for you and which did not.

Chapter 2

Infrastructure – Tools and Operating System

If you will be using Linux as your development platform you might be able to skip this entire chapter. RPMs of the toolchain and RTEMS are available from www.rtems.com.

2.1 Create the RTEMS source and installation directories

There should be at least 300 Mbytes of space available on the drive where these directories are located. I used `/usr/local/rtems/rtems-4.9.2` as the installation target directory. The location of the RTEMS source is not critical. This document assumes that the root of the RTEMs source tree is `/usr/local/rtems/source`.

Create the directories where the source will be placed and the results of the build installed:

```
/usr/local/rtems/source
/usr/local/rtems/source/tools
/usr/local/rtems/rtems-4.9.2
```

2.2 Add the directory containing the tools to your shell search path

The following sections assume that the directory into which you will install the cross-development tools (`/usr/local/rtems/rtems`
is on your shell search path. For shells like sh, bash, zsh and ksh you can to this with

```
PATH="$PATH:/usr/local/rtems/rtems-4.9.2/bin"
```

For shells like csh and tcsh you can

```
set path = ( $path /usr/local/rtems/rtems-4.9.2/bin )
```

2.3 Get and build the development tools

RTEMS uses the GNU toolchain to build the executive and libraries. Information about the GNU tools can be found on the GNU home page. If you're feeling brave you can skip the following sections and turn loose the script included in appendixA. In either case, if you're building on Solaris you'll need to ensure that you have GNU make (gmake) installed on your system and also set a couple of environment variables for things to build properly:

```
MAKE=gmake
INTLLIBS=-lintl
```

The script attempts to download, unpack, configure, build and install the GNU cross-development tools and libraries for one or more target architectures. To use the script, set the `ARCHS` environment variable to the architectures you wish to support, then

```
sh getAndBuildTools.sh
```

Set the MAKE environment variable to the name of whatever make program you need for your system.

2.3.1 Download the tool source files

The source for the GNU tools should be obtained from the On-line Applications Research (OAR) FTP server since that server provides any RTEMS-specific patches that may have to be applied before the tools can be built.

The files in the OAR FTP server directory ftp://www.rtems.com/pub/rtems/SOURCES/4.9 should be downloaded to the RTEMS/tools directory created above. The files can be downloaded using a web browser or a command-line program such as `curl` or `wget`. (note that the command examples have been split to help them fit on the page):

```
curl --remote-name
    ftp://www.rtems.com/pub/rtems/SOURCES/4.9/binutils-2.19.tar.bz2
curl --remote-name
    ftp://www.rtems.com/pub/rtems/SOURCES/4.9/gcc-core-4.3.2.tar.bz2
curl --remote-name
    ftp://www.rtems.com/pub/rtems/SOURCES/4.9/gcc-g++-4.3.2.tar.bz2
curl --remote-name
    ftp://www.rtems.com/pub/rtems/SOURCES/4.9/newlib-1.16.0.tar.gz

curl --remote-name
    ftp://www.rtems.com/pub/rtems/SOURCES/4.9/gcc-core-4.3.2-rtems4.9-20081214.diff
curl --remote-name
    ftp://www.rtems.com/pub/rtems/SOURCES/4.9/newlib-1.16.0-rtems4.9-20090324.diff
```

or

```
wget --passive-ftp --no-directories --retr-symlinks
    ftp://www.rtems.com/pub/rtems/SOURCES/4.9/binutils-2.19.tar.bz2
wget --passive-ftp --no-directories --retr-symlinks
    ftp://www.rtems.com/pub/rtems/SOURCES/4.9/gcc-core-4.3.2.tar.bz2
...
```

Depending on the type of firewall between your machine and the OAR FTP server you may need to remove the `--passive-ftp` option from the `wget` commands.

2.3.2 Unpack the source archives:

The following commands will extract the GNU tool sources from the downloaded tar archive files.

```
bzcat binutils-2.19.tar.bz2 | tar xf -
bzcat gcc-core-4.3.2.tar.bz2 | tar xf -
bzcat gcc-g++-4.3.2.tar.bz2 | tar xf -
zcat <newlib-1.16.0.tar.gz | tar xf -
```

To build the newlib libraries needed by RTEMS you must make a symbolic link to the newlib source directory from the gcc source directory.

```
cd gcc-4.3.2
rm -rf newlib
ln -s ../1.16.0/newlib newlib
cd ..
```

2.3.3 Apply any RTEMS-specific patches

If any patch files (those with a `.diff` suffix) were downloaded from the OAR FTP server the patches in those files must be applied before the tools can be compiled.

Here is how the patches can be applied to the gcc sources:

```
cd gcc-4.3.2
patch -p1 <../gcc-core-4.3.2-rtems4.9-20081214.diff
cd ..
```

Here is how the patches can be applied to the newlib sources:

```
cd newlib-1.16.0
patch -p1 <../newlib-1.16.0-rtems4.9-20090324.diff
cd ..
```

2.3.4 Configure, build and install the 'binutils':

The commands in this section must be repeated for each desired target architecture. The examples shown build the tools for Motorola Power PC targets.

1. Create a directory in which the tools will be built and change to that directory.

   ```
   rm -rf build
   mkdir build
   cd build
   ```

2. Configure the tools.

   ```
   ../binutils-2.19/configure --target=powerpc-rtems4.9.2 \
               --prefix=/usr/local/rtems/rtems-4.9.2 \
               --verbose --disable-nls \
               --without-included-gettext \
               --disable-win32-registry \
               --disable-werror
   ```

   You should replace the '`powerpc`' with the name of the architecture for which you're building the tools. Common alternatives are '`m68k`' and '`i386`' for the Motorola M68k and Intel x86 family of processors, respectively.

3. Build and install the tools.

   ```
   make -w all install
   ```

   In this and all subsequent cases the use of a GNU make program is required. On some hosts you'll have to use `gmake` instead of `make`.

4. Return to the directory containing the tool and library sources.

   ```
   cd ..
   ```

2.3.5 Configure, build and install the cross-compiler and libraries

1. Create a directory in which the tools will be built and change to that directory.

   ```
   rm -rf build
   mkdir build
   cd build
   ```

2. Configure the compiler and libraries.

   ```
   ../4.3.2/configure --target=powerpc-rtems4.9.2 \
               --prefix=/usr/local/rtems/rtems-4.9.2 \
               --disable-libstdcxx-pch \
               --with-gnu-as --with-gnu-ld --verbose \
               --with-newlib \
               --with-system-zlib \
               --disable-nls --without-included-gettext \
               --disable-win32-registry \
               --enable-version-specific-runtime-libs \
               --enable-threads \
               --enable-newlib-io-c99-formats \
               --enable-languages="c,c++"
   ```

   You should again replace the '`powerpc`' with the name of the architecture for which you're building the cross-compiler and libraries.

3. Build and install the cross-compiler and libraries by.

   ```
   make -w all install
   ```

4. Return to the directory containing the tool and library sources.

   ```
   cd ..
   ```

2.4 Get, build and install RTEMS

2.4.1 Download the RTEMS source from the OAR web server.

The source releases are available at

`http://www.rtems.com/ftp/pub/rtems/4.9.2/rtems-4.9.2.tar.bz2`

The compressed *tar* archive in this directory can be downloaded using a web browser or a command-line program such as `curl` or `wget`:

```
curl --remote-name
   http://www.rtems.com/ftp/pub/rtems/4.9.2/rtems4.9.2.tar.bz2
```

or

```
wget --passive-ftp --no-directories --retr-symlinks
   http://www.rtems.com/ftp/pub/rtems/4.9.2/rtems4.9.2.tar.bz2
```

Depending on the type of firewall between your machine and the OAR FTP server you may need to remove the `--passive-ftp` option from the `wget` command.
When you are done you should have the compressed archive with a name something like

```
rtems-4.9.2.tar.bz2
```

2.4.2 Unpack the RTEMS sources

Change to your RTEMS source directory and unpack the RTEMS sources by:

```
bzcat rtems-4.9.2.tar.bz2 | tar xf -
```

This will create the directory `rtems-4.9.2` and unpack all the RTEMS source into that directory.

2.4.3 Make changes to the RTEMS source to reflect your local conditions.

Some of the board-support-packages distributed with RTEMS may require modifications to match the hardware in use at your site. The following sections describe changes commonly made to two of these board-support-packages.

MVME167

The linker script distributed with RTEMS assumes an MVME167 with 4 Mbytes of on-board memory starting at location 0x00800000. A more common configuration is 16 Mbytes of memory starting at location 0x00000000. To reflect this configuration make the following changes to

```
rtems-4.9.2/c/src/lib/libbsp/m68k/mvme167/startup/linkcmds
```

```
@@ -24,8 +24,8 @@
 /*
  * Declare some sizes. Heap is sized at whatever ram space is left.
  */
-_RamBase = DEFINED(_RamBase) ? _RamBase : 0x00800000;
-_RamSize = DEFINED(_RamSize) ? _RamSize : 4M;
+_RamBase = DEFINED(_RamBase) ? _RamBase : 0x0;
+_RamSize = DEFINED(_RamSize) ? _RamSize : 16M;
 _HeapSize = DEFINED(_HeapSize) ? _HeapSize : 0;
 _StackSize = DEFINED(_StackSize) ? _StackSize : 0x1000;

@@ -35,7 +35,7 @@
       This is where we put one board. The base address should be
       passed as a parameter when building multiprocessor images
       where each board resides at a different address. */
-  ram  : org = 0x00800000, l = 4M
+  ram  : org = 0x00000000, l = 16M
   rom  : org = 0xFF800000, l = 4M
   sram : org = 0xFFE00000, l = 128K
 }
```

PC-x86

A change I like to make to the RTEMS pc386 source is to increase the number of lines on the console display from 25 to 50 since I find that the output from some EPICS commands scrolls off the display when only 25 lines are present. To make this change, add the '`#define`' line shown below

rtems-4.9.2/c/src/lib/libbsp/i386/pc386/start/start16.S

```
 +-------------------------------------------------------*/

 #include <bspopts.h>
 #define RTEMS_VIDEO_80x50

 /*-------------------------------------------------------+ | Constants
```

Another change I make is to automatically fall back to using COM2: as a serial-line console (9600-8N1) if no video adapter is present. This allows the pc386 BSP to be used on conventional PCs with video adapters as well as with embedded PCs (PC-104) which have no video adapters. To make this change, add the '`#define`' line shown below

rtems-4.9.2/c/src/lib/libbsp/i386/pc386/console/console.c

```
   */
  rtems_termios_initialize ();

#define RTEMS_RUNTIME_CONSOLE_SELECT
#ifdef RTEMS_RUNTIME_CONSOLE_SELECT
  /*
   * If no video card, fall back to serial port console
```

2.4.4 Build and install RTEMS

1. It is best to start with a clean slate. Create a new directory in which to build or clean out all files in your existing build directory.

2. Configure RTEMS for your target architecture:

   ```
   cd /usr/local/rtems-4.9.2/build
   .../rtems-4.9.2/configure --target=powerpc-rtems4.9.2 \
     --prefix=/usr/local/rtems/rtems-4.9.2 \
     --enable-cxx --enable-rdbg --disable-tests --enable-networking \
     --enable-posix --enable-rtemsbsp=mvme2100 \
   ```

   You should replace the '`powerpc`' with the name of the architecture for which you're building RTEMS. Common alternatives are '`m68k`' and '`i386`' for the Motorola M68k and Intel x86 family of processors, respectively. You should replace the '`mvme2100`' with the board-support packages for your particular hardware.

   If you've got lots of free time and disk space you can omit the `--enable-rtemsbsp` argument in which case all possible board-support packages for that architecture will be built. You can build for more than one board-support package by specifying more names on the command line. For example, you could build for a Arcturus uCDIMM ColdFire 5282 system and an MVME-167 system by:

   ```
   cd /usr/local/rtems-4.9.2/build
   .../rtems-4.9.2/configure --target=m68k-rtems4.9.2 \
         --prefix=/usr/local/rtems/rtems-4.9.2 \
         --enable-cxx --enable-rdbg --disable-tests --enable-networking \
         --enable-posix --enable-rtemsbsp="uC5282 mvme167" \
   ```

3. Compile and install:

   ```
   make -w
   make -w install
   ```

2.5 Get, build and install some RTEMS add-on packages

The EPICS IOC shell uses the the libtecla or GNU readline package to provide command-line editing and command history. While the IOC shell can be compiled without these capabilities I think they're important enough to warrant making the effort to download and install the extra packages. GNU readline is more well-tested, but libtecla does not bring along the issues associated with the GNU Public License.

2.5.1 Download the add-on package sources

The latest versions of these files are in

```
http://www.rtems.com/ftp/pub/rtems/4.9.2/rtems-addon-packages-4.9.2.tar.bz2
```

The compressed *tar* archive in this directory can be downloaded using a web browser or a command-line program such a `wget` (note that the wget command example has been split to make it fit on this page):

```
wget --passive-ftp --no-directories --retr-symlinks
    "http://www.rtems.com/ftp/pub/rtems/4.9.2/rtems-addon-packages-4.9.2.tar.bz2"
```

Depending on the type of firewall between your machine and the OAR FTP server you may need to remove the `--passive-ftp` option from the `wget` command.
When you are done you should have the compressed archive with a name something like

```
rtems-addon-packages-4.9.2.tar.bz2
```

2.5.2 Unpack the add-on package sources

Change to your RTEMS source directory and unpack the RTEMS sources by:

```
cd /usr/local/rtems/source
bzcat rtems-addon-packages-4.9.2.tar.bz2 | tar xf -
```

This will unpack the source for all the RTEMS packages into a directory named

```
rtems-addon-packages-4.9.2
```

2.5.3 Set the RTEMS_MAKEFILE_PATH environment variable

The makefiles in the RTEMS packages use the `RTEMS_MAKEFILE_PATH` environment variable to determine the target architecture and board-support package. For example,

```
export RTEMS_MAKEFILE_PATH=/usr/local/rtems/rtems-4.9.2/powerpc-rtems4.9.2/mvme2100
```

will select the Motorola Power PC architecture and the RTEMS mvme2100 board-support package.

2.5.4 Build and install the add-on packages

The *bit* script in the packages source directory builds and installs all the add-on packages. To run the script change directories to the add-on packages directory and execute:

```
sh bit
```

If you are building for more than one architecture or board-support package, you must run the *bit* script once for each variation with `RTEMS_MAKEFILE_PATH` set to the different architecture and board-support package.

2.6 Try running some RTEMS sample applications (optional)

The RTEMS build process creates some sample applications. If you're just getting started with RTEMS it's probably a good idea to verify that you can actually run a simple RTEMS application on your target hardware before trying to run a full-blown EPICS IOC application.

The actual method of loading an application into a target processor is hardware-dependent. Section 4.1.3 describes a method which may be used with RTEMS mvme2100 targets.

2.7 Extended BSP routines

Some additional support routines are necessary to use EPICS/RTEMS with PowerPC VME cards such as the MVME2100 and MVME3100.

1. Download the libbspExt sources:

   ```
   wget http://www.slac.stanford.edu/~strauman/rtems/rtems_libbspExt_1.3.beta.tgz
   ```

2. Unpack

   ```
   cd /usr/local/rtems/source
   tar -xzf rtems_libbspExt_1.3.beta.tgz
   ```

3. Patch 2 files (add support for mvme2100 and correct install location)

   ```
   cd rtems_libbspExt_1.3.beta
   patch -p1 < libbspExt-1.3.beta.patch
   ```

4. Set the `RTEMS_MAKEFILE_PATH` environment variable

   ```
   export RTEMS_MAKEFILE_PATH=/usr/local/rtems/rtems-4.9/powerpc-rtems4.9/mvme2100
   ```

5. Build and install the add-on packages

   ```
   make
   make install
   ```

Chapter 3

EPICS Base

The first step in building an EPICS application is to download the EPICS base source from the APS server and unpack it. The details on how to perform these operations are described on the APS web pages and will not be repeated here. Make sure you get the R3.14.9 or later release of EPICS.

3.1 Specify the location of RTEMS tools and libraries

You must first let the EPICS Makefiles know where you've installed the RTEMS development tools and libraries. The default location is

```
/usr/local/rtems/rtems-4.9.2
```

If you've installed the RTEMS tools and libraries in a different location and have not created a symbolic link from

```
/usr/local/rtems/rtems-4.9.2
```

to wherever you've installed RTEMS you need to edit the EPICS configuration file

```
configure/os/CONFIG_SITE.Common.RTEMS
```

In this file you'll find the lines

```
RTEMS_BASE=/usr/local/rtems/rtems-4.9.2
RTEMS_VERSION=4.9.2
```

while will have to be changed to reflect the directory where you installed RTEMS.
If you installed the RTEMS readline or tecla add-on packages you can change the `EPICSCOMMANDLINE_LIBRARY` definition from `EPICS` to `READLINE` or `LIBTECLA`, respectively. If you don't want to use NFS to access remote files you can add

```
OP_SYS_CFLAGS += -DOMIT_NFS_SUPPORT
```

3.2 Specify the network domain

If you're using neither DHCP/BOOTP not non-volatile RAM to provide network configuration information to your RTEMS IOCs you should specify your Internet Domain Name as:

```
OP_SYS_CFLAGS += -DRTEMS_NETWORK_CONFIG_DNS_DOMAINNAME=your.dnsname.here
```

3.3 Specify the network interface

Some RTEMS board support packages support more than one type of network interface. The pc386 BSP, for example, can be configured to use several different network interface cards. By default the EPICS network configuration for the pc386 BSP loads network drivers for all network interfaces which support run-time probing so if you've got one of these network interfaces you don't need to make any changes to the EPICS network configuration. If not, see the comments in

`src/RTEMS/base/rtems_netconfig.c`

for instructions on selecting a network interface card when building your EPICS application. Most RTEMS board support packages support only a single network interface and need no changes to rtems_netconfig.c.

3.4 Specify the target architectures

The `configure/os/CONFIG_SITE.<host_architecture>.Common` file specifies the target architectures and board support packages to be supported. For example, I regularly build for a single target:

```
CROSS_COMPILER_TARGET_ARCHS=RTEMS-mvme2100
```

If you want to build for multiple RTEMS targets you would change this line to something like

```
CROSS_COMPILER_TARGET_ARCHS=RTEMS-mvme2100 RTEMS-uC5282 RTEMS-pc386
```

The format of the target architecture names is `RTEMS-`*bspname*, where `RTEMS-` indicates that the RTEMS development tools and libraries should be used, and *bspname* is the name of the RTEMS target architecture and board support package used back in section 2.4.4.

3.5 Build EPICS base

This step is very simple. Just change directories to the EPICS `base` directory and run

```
make
```

After a while you'll end up with a working set of EPICS base libraries and tools.

Chapter 4

EPICS Applications

Now that you've built the EPICS base libraries you're ready to build and run your first EPICS application. Once you've got this application running you can forget about this tutorial and revert to using the standard EPICS documentation. You can start with your own special application or you can start with the example application that is provided with the EPICS distribution. The following sections describe the procedure to create, build and run this example application.

4.1 The EPICS example application

4.1.1 Build the example application

1. Create a new directory to hold the application source and then 'cd' to that directory.
2. Run the `makeBaseApp.pl` program to create the example application:

   ```
   makeBaseApp.pl -t example test
   makeBaseApp.pl -i -t example -a RTEMS-mvme2100 test
   ```

   If you get complaints about not being able to run these commands you've probably forgotten to put the 'bin' directory created in the previous section on your shell executable search path.

   The '`test`' in the two `makeBaseApp.pl` commands can be replaced with whatever name you want to give your example application. The '`RTEMS-mvme2100`' can be replaced with whatever target architecture you plan to use to run the example application.
3. Build the example application by running

   ```
   make
   ```

4.1.2 Install the EPICS IOC files on the TFTP/NFS server

The application build process creates `db` and `dbd` directories in the top-level application directory and produces a set of IOC shell commands in the `st.cmd` file in the `iocBoot/ioctest` directory. If you chose an application name different than `test` in the previous step, the directory name will change accordingly. These directories and their contents must be copied to your TFTP/NFS server. The actual location depends upon the remote file access technique being used as described in the following section.

4.1.3 Run the example application on an RTEMS IOC

Everything's now ready to go. The only item left is arranging some way of loading the RTEMS/EPICS application executable image into the IOC. There are many ways of doing this (floppy disks, PROM images, etc.), but I find using a BOOTP/DHCP/TFTP server to be the most convenient. The remainder of this section describes how I load executables into my RTEMS-mvme2100 and RTEMS-pc386 IOCs. If you're running a different type of IOC you'll have to figure out the required steps on your own. The RTEMS documentation may provide the required information since an EPICS IOC application is an RTEMS application like any other.

Some RTEMS board-support packages require an NTP server on the network. If such an IOC doesn't receive a time-synchronization packet from an NTP server the IOC time will be set to 00:00:00, January 1, 2001.

4.1.4 Location of EPICS startup script

If you're using BOOTP/DHCP to provide network configuration information to your IOC you should use DHCP site-specific option 129 to specify the path to the IOC startup script. If you're using PPCBUG you should set the NIOT 'Argument File Name' parameter to the IOC startup script path.

If you're using NFS for remote file access the EPICS initialization uses the startup script pathname to determine the parameters for the initial NFS mount. If the startup script pathname begins with a '/' the first component of the pathname is used as both the server path and the local mount point. If the startup script pathname does not begin with a '/' the first component of the pathname is used as the local mount point and the server path is "`/tftpboot/`" followed by the first component of the pathname. This allows the NFS and TFTP clients to have a similar view of the remote filesystem.

If you're using TFTP for remote file access the RTEMS startup code first changes directories to `/epics/`*hostname*`/` within the TFTP server, where *hostname* is the Internet host name of the IOC. The startup code then reads IOC shell commands from the `st.cmd` script in that directory. The name (`st.cmd`) and location of the startup script are fixed from the IOCs point of view so it must be installed in the corresponding location on the TFTP server. Many sites run the TFTP server with an option which changes its root directory. On this type of system you'll have to copy the startup script to the `/epics/`*hostname*`/` directory within the TFTP server's root directory. On a system whose TFTP server runs with its root directory set to `/tftpboot` the startup script for the IOC whose name is `norumx1` would be placed in

```
/tftpboot/epics/norumx1/st.cmd
```

The application build process creates `db` and `dbd` directories in the top-level application directory. These directories and their contents must be copied to the IOC's directory on the TFTP server. For the example described above the command to install the files for the `norumx1` IOC is

```
cp -r db dbd /tftpboot/epics/norumx1
```

MVME2100 Using PPCBUG

1. Use the PPCBUG ENV command to set the 'Network PReP-Boot Mode Enable' parameter to 'Y'.
2. Use the PPCBUG NIOT command to set the network parameters. Here are the parameters for a test IOC I use:

```
Controller LUN =00
Device LUN     =00
Node Control Memory Address =FFE10000
Client IP Address      =www.xxx.yyy.8
Server IP Address      =www.xxx.yyy.131
Subnet IP Address Mask =255.255.252.0
Broadcast IP Address   =192.168.11.255
Gateway IP Address     =0.0.0.0
Boot File Name ("NULL" for None)     =/epics/test/bin/RTEMS-mvme2100/example.boot
```

```
Argument File Name ("NULL" for None) =www.xxx.yyy.98:/home/epics:test/iocBoot/iocexample/st.cmd
Boot File Load Address         =001F0000
Boot File Execution Address    =001F0000
Boot File Execution Delay      =00000000
Boot File Length               =00000000
Boot File Byte Offset          =00000000
BOOTP/RARP Request Retry       =00
TFTP/ARP Request Retry         =00
Trace Character Buffer Address =00000000
BOOTP/RARP Request Control: Always/When-Needed (A/W)=W
BOOTP/RARP Reply Update Control: Yes/No (Y/N)       =Y
```

- The `Server IP Address` is used as the address of the TFTP, NTP and domain name servers.
- On the TFTP server the path to the executable file is

  `/tftpboot/epics/test/bin/RTEMS-mvme2100/example.boot`
- In the example above I have shown how to use a different address for the NFS server. On the NFS server the path to the startup script would be

  `/home/epics/test/iocBoot/iocexample/st.cmd`
- The Boot File Name and Argument File Name strings can be at most 64 characters long. You may have to shuffle files around on your servers to accomodate this restriction.

3. Set up your TFTP/NFS servers. PPCBUG uses TFTP to load the executable image then the EPICS initialization uses NFS to read the EPICS startup script (the 'Argument File Name' in the NIOT parameters). I set the TFTP server root to /tftpboot and arrange for the NFS server to export /tftpboot/epics to the IOCs. This arrangement lets me simply copy the application tree, beginning at the `<top>` directory to the TFTP/NFS server area.

Motorola Processors Using MOTLOAD

The following 'Global Environment Variables' are used. These are set using the MOTLOAD `gevEdit` command.

mot-script-boot These commands are run by MOTLOAD when the board is booted. A typical example is shown below:

```
tftpGet -cww.ww.ww.ww -sxx.xx.xx.xxx -myy.yy.yy.yy -d/dev/enet0 -fpath
netShut
go
```

where *ww.ww.ww.ww* is the IP number of the client (VME card), *xx.xx.xx.xx* is the IP number of the TFTP server, *yy.yy.yy.yy* is the IP subnet mask, and *path* is the path to the executable image file on the TFTP server.

The standard MOTLOAD download buffer may be too small to hold your application. If this is the case you can call malloc to allocate a larger buffer and then use the -a option to pass the address of this buffer to the tftpGet and go commands:

```
dla=malloc 0x280000
tftpGet -cww.ww.ww.ww -sxx.xx.xx.xxx -myy.yy.yy.yy -d/dev/enet0 -fpath -adla
netShut
go -adla
```

mot-/dev/enet0-cipa The client (VME card) IP address. If this variable is not set the client address is set to the value of the '-c' argument in the mot-script-boot variable.

mot-/dev/enet0-sipa The server IP address. If this variable is not set the server address is set to the value of the '-s' argument in the mot-script-boot variable.

mot-/dev/enet0-gipa The gateway IP address. If this variable is not set the gateway IP address is set to the value of the '-g' argument in the mot-script-boot variable.

mot-/dev/enet0-snma The subnet mask. If this variable is not set the subnet mask is set to the value of the '-m' argument in the mot-script-boot variable.

mot-/dev/enet0-file The name of the executable image. If this variable is not set the name is set to the value of the '-f' argument in the mot-script-boot variable. The name is passed as the 'argv[0]' to the main() function.

rtems-dns-server The domain name server IP address. If this variable is not set the server address as described above is used.

rtems-dns-domainname The domain name. If this variable is not set the value compiled into the executable image is used.

rtems-client-name The client host name. If this variable is not set the client address as described above is used.

epics-script The path to the IOC startup script on the TFTP or NFS server. The value can be a simple path or be of the form *nfsServer*:*nfsExport*:*nfsPath*. The *nfsExport* component is the directory exported from the NFS server and is also used as the local mount point and as a prefix to *nfsPath*.

epics-nfsmount If set, this variable should be of the form *nfsServer*:*nfsExport*:*nfsMount*. The *nfsExport* component is the directory exported from the NFS server and the *nfsMount* is the local mount point.

epics-ntpserver The NTP server IP address. If this variable is not set the server address as described above is used.

epics-tz Set the value of the TZ environment variable. This is useful for handling daylight-savings-time changes. A value of CST6CDT5,M3.2.0,M11.1.0 is appropriate for Chicago in 2007 and perhaps subsequent years.

PC386

1. Install an EtherBoot bootstrap PROM image obtained from the 'ROM-o-matic' server (http://www.rom-o-matic.net/) on the IOC network interface cards.
2. Set up your BOOTP/DHCP server to provide the network configuration parameters to the IOC.
3. The TFTP and NFS servers can be configured as noted above.

Arcturus uCDIMM ColdFire 5282

Use the bootstrap `setenv` command to set the EPICS and network configuration parameters:

```
IPADDR0=www.xxx.yyy.27
HOSTNAME=ioccoldfire
BOOTFILE=epics/ucdimm/bin/RTEMS-uC5282/ucdimm.boot
NAMESERVER=www.xxx.yyy.167
NETMASK=255.255.252.0
CMDLINE=epics/i2c/iocBoot/ioci2c/st.cmd
SERVER=www.xxx.yyy.161
NFSMOUNT=106.74@nfssrv:/export/nfssrv:/home/nfssrv
```

The environment variables used by the EPICS startup code are:

IPADDR0 The client (Coldfire processor) IP address.

HOSTNAME The client host name.

SERVER The server IP address. If this variable is not set the specific server addressses as described below must be set.

GATEWAY The gateway IP address. If this variable is not set the Coldfire will be able to communicate only with hosts on its local network.

NETMASK The subnet mask.

BOOTFILE The name of the executable image. The name is passed as the 'argv[0]' to the main() function.

NTPSERVER The NTP server IP address. If this variable is not set the server address as described above is used.

NAMESERVER The domain name server IP address. If this variable is not set the server address as described above is used.

DOMAIN The domain name. If this variable is not set the value compiled into the executable image is used.

CMDLINE The path to the IOC startup script on the TFTP or NFS server. The value can be a simple path or be of the form *nfsServer*:*nfsExport*:*nfsPath*. The *nfsExport* component is the directory exported from the NFS server and is also used as the local mount point and as a prefix to *nfsPath*.

NFSMOUNT If set, this variable should be of the form *nfsServer*:*nfsExport*:*nfsMount*. The *nfsExport* component is the directory exported from the NFS server and the *nfsMount* is the local mount point.

TZ Set the value of the TZ environment variable. This is useful for handling daylight-savings-time changes. A value of CST6CDT5,M3.2.0,M11.1.0 is appropriate for Chicago in 2007 and perhaps subsequent years.

The uCDIMM ColdFire 5282 module is distributed with two types of bootstrap ROMs. One type provides a TFTP server and the other provides a TFTP client. The steps required to boot an EPICS application differ depending on the the bootstrap type.

Arcturus uCDIMM ColdFire 5282 with boostrap ROMs providing TFTP server Use the bootstrap `tftp` command to load the IOC application image (which may include a tar image of the in-memory filesystem contents in which case the CMDLINE will likely look something like `/st.cmd` and the NFSMOUNT need not be present). You'll need to run a TFTP client on your host machine to transfer the file. An example using *curl* is:

```
curl -T bin/RTEMS-uC5282/example.boot tftp://192.168.1.93
```

where 192.168.1.93 is the IP address of the ColdFire target.

Use the bootstrap `goram` command to start the application or the `program` command to burn the image into the on-board flash memory. In the latter case you may want to also use the `setenv` command to set the `AUTOBOOT` environment variable.

Arcturus uCDIMM ColdFire 5282 with boostrap ROMs providing TFTP client The bootstrap procedure in this case is similar to that for the Motorola VME processors in section 4.1.4.

The cexp package can be used to incrementally load your application components.

Appendix A

Script to get and build the cross-development tools

If you're feeling brave you can turn loose the following script. It attempts to download, unpack, configure, build and install the GNU cross-development tools and libraries for one or more target architectures. To use the script, set the `ARCHS` environment variable to the architectures you wish to support, then run the script.

A.1 getAndBuildTools-4.9.2.sh

```
#!/bin/sh

#
# Get, build and install the latest cross-development tools and libraries
#

#
# Specify the architectures for which the tools are to be built
# To build for single target: ARCHS="m68k"
#
ARCHS="${ARCHS:-m68k i386 powerpc}"

#
# Specify the versions
#
GCC=4.3.2
BINUTILS=2.19
NEWLIB=1.16.0
GDB=6.8
#BINUTILSDIFF=binutils-2.18-rtems4.9-20080211.diff
GCCDIFF=gcc-core-4.3.2-rtems4.9-20081214.diff
NEWLIBDIFF=newlib-1.16.0-rtems4.9-20090324.diff
GDBDIFF=gdb-6.8-rtems4.9-20090923.diff
RTEMS_BASE_VERSION=4.9
RTEMS_VERSION=4.9.2

#
```

```
# Where to install
#
PREFIX="${PREFIX:-/usr/local/rtems/rtems-${RTEMS_VERSION}}"

#
# Where to get the GNU tools
#
RTEMS_SOURCES_URL=ftp://www.rtems.com/pub/rtems/SOURCES/${RTEMS_BASE_VERSION}
RTEMS_BINUTILS_URL=${RTEMS_SOURCES_URL}/binutils-${BINUTILS}.tar.bz2
RTEMS_GCC_CORE_URL=${RTEMS_SOURCES_URL}/gcc-core-${GCC}.tar.bz2
RTEMS_GCC_GPP_URL=${RTEMS_SOURCES_URL}/gcc-g++-${GCC}.tar.bz2
RTEMS_NEWLIB_URL=${RTEMS_SOURCES_URL}/newlib-${NEWLIB}.tar.gz
RTEMS_GDB_URL=${RTEMS_SOURCES_URL}/gdb-${GDB}.tar.bz2
RTEMS_BINUTILS_DIFF_URL=${RTEMS_SOURCES_URL}/${BINUTILSDIFF}
RTEMS_GCC_DIFF_URL=${RTEMS_SOURCES_URL}/${GCCDIFF}
RTEMS_NEWLIB_DIFF_URL=${RTEMS_SOURCES_URL}/${NEWLIBDIFF}
RTEMS_GDB_DIFF_URL=${RTEMS_SOURCES_URL}/${GDBDIFF}

#
# Uncomment one of the following depending upon which your system provides
#
GET_COMMAND="curl --remote-name"
#GET_COMMAND="wget --passive-ftp --no-directories --retr-symlinks "
#GET_COMMAND="wget --no-directories --retr-symlinks "

#
# Solaris likely needs gmake and /bin/bash here.
#
MAKE="${MAKE:-make}"
export MAKE
SHELL=/bin/sh
export SHELL

#
# Get the source
# If you don't have curl on your machine, try using
#     wget --passive-ftp --no-directories --retr-symlinks <<url>>
# If that doesn't work, try without the --passive-ftp option.
#
getSource() {
    ${GET_COMMAND} "${RTEMS_BINUTILS_URL}"
    if [ -n "$BINUTILSDIFF" ]
    then
        ${GET_COMMAND} "${RTEMS_BINUTILS_DIFF_URL}"
    fi
    ${GET_COMMAND} "${RTEMS_GCC_CORE_URL}"
    ${GET_COMMAND} "${RTEMS_GCC_GPP_URL}"
    if [ -n "$GCCDIFF" ]
    then
        ${GET_COMMAND} "${RTEMS_GCC_DIFF_URL}"
    fi
```

```
     ${GET_COMMAND} "${RTEMS_NEWLIB_URL}"
    if [ -n "$NEWLIBDIFF" ]
    then
        ${GET_COMMAND} "${RTEMS_NEWLIB_DIFF_URL}"
    fi
     ${GET_COMMAND} "${RTEMS_GDB_URL}"
    if [ -n "$GDBDIFF" ]
    then
        ${GET_COMMAND} "${RTEMS_GDB_DIFF_URL}"
    fi
}

#
# Unpack the source
#
unpackSource() {
    rm -rf "binutils-${BINUTILS}"
    bzcat "binutils-${BINUTILS}.tar.bz2" | tar xf -
    for d in "binutils-${BINUTILS}"*.diff
    do
        if [ -r "$d" ]
        then
            cat "$d" | (cd "binutils-${BINUTILS}" ; patch -p1)
        fi
    done

    rm -rf "gcc-${GCC}"
    bzcat "gcc-core-${GCC}.tar.bz2" | tar xf -
    bzcat "gcc-g++-${GCC}.tar.bz2" | tar xf -
    for d in gcc*.diff
    do
        if [ -r "$d" ]
        then
            cat "$d" | (cd "gcc-${GCC}" ; patch -p1)
        fi
    done

    rm -rf "newlib-${NEWLIB}"
    zcat <"newlib-${NEWLIB}.tar.gz" | tar xf -
    for d in "newlib-${NEWLIB}"*.diff
    do
        if [ -r "$d" ]
        then
            cat "$d" | (cd "newlib-${NEWLIB}" ; patch -p1)
        fi
    done
    (cd "gcc-${GCC}" ; ln -s "../newlib-${NEWLIB}/newlib" newlib)

    rm -rf "gdb-${GDB}"
    bzcat <"gdb-${GDB}.tar.bz2" | tar xf -
    for d in "gdb-${GDB}"*.diff
```

```
    do
        if [ -r "$d" ]
        then
            cat "$d" | (cd "gdb-${GDB}" ; patch -p1)
        fi
    done
}

#
# Build
#
build() {
    PATH="${PREFIX}/bin:$PATH"
    for arch in $ARCHS
    do
        rm -rf build
        mkdir build
        cd build
        "${SHELL}" "../binutils-${BINUTILS}/configure" \
            "--target=${arch}-rtems${RTEMS_VERSION}" "--prefix=${PREFIX}" \
            --verbose --disable-nls \
            --without-included-gettext \
            --disable-win32-registry \
            --disable-werror
        ${MAKE} -w all install
        cd ..

        rm -rf build
        mkdir build
        cd build
       "${SHELL}" "../gcc-${GCC}/configure" \
            "--target=${arch}-rtems${RTEMS_VERSION}" "--prefix=${PREFIX}" \
            --disable-libstdcxx-pch \
            --with-gnu-as --with-gnu-ld --verbose \
            --with-newlib \
            --with-system-zlib \
            --disable-nls --without-included-gettext \
            --disable-win32-registry \
            --enable-version-specific-runtime-libs \
            --enable-threads \
            --enable-newlib-io-c99-formats \
            --enable-languages="c,c++" \
            --with-gmp="${PREFIX}" --with-mpfr="${PREFIX}"
        ${MAKE} -w all
        ${MAKE} -w install
        cd ..

        rm -rf build
        mkdir build
        cd build
       "${SHELL}" "../gdb-${GDB}/configure" \
```

```
            "--target=${arch}-rtems${RTEMS_VERSION}" "--prefix=${PREFIX}" \
            --verbose --disable-nls --without-included-gettext \
            --disable-win32-registry \
            --enable-version-specific-runtime-libs \
            --disable-win32-registry \
            --disable-werror \
            --enable-sim \
            --with-expat
        ${MAKE} -w all
        ${MAKE} -w install
        cd ..
    done
}

#
# Do everything
#
# Comment out any activities you wish to omit
#
set -ex
getSource
unpackSource
export LD_LIBRARY_PATH="${PREFIX}/lib"
build
```

A.2 getAndBuildTools-4.10.sh

```
#!/bin/sh

#
# Get, build and install the latest cross-development tools and libraries
#

#
# Specify the architectures for which the tools are to be built
# To build for single target: ARCHS="m68k"
#
ARCHS="${ARCHS:-m68k i386 powerpc}"

#
# Specify the versions
#
GCC=4.4.2
BINUTILS=2.20
NEWLIB=1.17.0
GDB=7.0
#BINUTILSDIFF=binutils-2.19.1-rtems4.10-20090203.diff
GCCDIFF=gcc-core-4.4.2-rtems4.10-20091015.diff
NEWLIBDIFF=newlib-1.17.0-rtems4.10-20091009.diff
GDBDIFF=gdb-7.0-rtems4.10-20091007.diff
RTEMS_BASE_VERSION=4.10
```

```
RTEMS_VERSION=4.10

#
# Where to install
#
PREFIX="${PREFIX:-/usr/local/rtems/rtems-${RTEMS_VERSION}}"

#
# Where to get the GNU tools
#
RTEMS_SOURCES_URL=ftp://www.rtems.com/pub/rtems/SOURCES/${RTEMS_BASE_VERSION}
RTEMS_BINUTILS_URL=${RTEMS_SOURCES_URL}/binutils-${BINUTILS}.tar.bz2
RTEMS_GCC_CORE_URL=${RTEMS_SOURCES_URL}/gcc-core-${GCC}.tar.bz2
RTEMS_GCC_GPP_URL=${RTEMS_SOURCES_URL}/gcc-g++-${GCC}.tar.bz2
RTEMS_NEWLIB_URL=${RTEMS_SOURCES_URL}/newlib-${NEWLIB}.tar.gz
RTEMS_GDB_URL=${RTEMS_SOURCES_URL}/gdb-${GDB}.tar.bz2
RTEMS_BINUTILS_DIFF_URL=${RTEMS_SOURCES_URL}/${BINUTILSDIFF}
RTEMS_GCC_DIFF_URL=${RTEMS_SOURCES_URL}/${GCCDIFF}
RTEMS_NEWLIB_DIFF_URL=${RTEMS_SOURCES_URL}/${NEWLIBDIFF}
RTEMS_GDB_DIFF_URL=${RTEMS_SOURCES_URL}/${GDBDIFF}

#
# Uncomment one of the following depending upon which your system provides
#
GET_COMMAND="curl --remote-name"
#GET_COMMAND="wget --passive-ftp --no-directories --retr-symlinks "
#GET_COMMAND="wget --no-directories --retr-symlinks "

#
# Solaris likely needs gmake and /bin/bash here.
#
MAKE="${MAKE:-make}"
export MAKE
SHELL=/bin/sh
export SHELL

#
# Get the source
# If you don't have curl on your machine, try using
#     wget --passive-ftp --no-directories --retr-symlinks <<url>>
# If that doesn't work, try without the --passive-ftp option.
#
getSource() {
    ${GET_COMMAND} "${RTEMS_BINUTILS_URL}"
    if [ -n "$BINUTILSDIFF" ]
    then
        ${GET_COMMAND} "${RTEMS_BINUTILS_DIFF_URL}"
    fi
    ${GET_COMMAND} "${RTEMS_GCC_CORE_URL}"
    ${GET_COMMAND} "${RTEMS_GCC_GPP_URL}"
    if [ -n "$GCCDIFF" ]
```

```
    then
        ${GET_COMMAND} "${RTEMS_GCC_DIFF_URL}"
    fi
     ${GET_COMMAND} "${RTEMS_NEWLIB_URL}"
    if [ -n "$NEWLIBDIFF" ]
    then
        ${GET_COMMAND} "${RTEMS_NEWLIB_DIFF_URL}"
    fi
     ${GET_COMMAND} "${RTEMS_GDB_URL}"
    if [ -n "$GDBDIFF" ]
    then
        ${GET_COMMAND} "${RTEMS_GDB_DIFF_URL}"
    fi
}

#
# Unpack the source
#
unpackSource() {
    rm -rf "binutils-${BINUTILS}"
    bzcat "binutils-${BINUTILS}.tar.bz2" | tar xf -
    for d in "binutils-${BINUTILS}"*.diff
    do
        if [ -r "$d" ]
        then
            cat "$d" | (cd "binutils-${BINUTILS}" ; patch -p1)
        fi
    done

    rm -rf "gcc-${GCC}"
    bzcat "gcc-core-${GCC}.tar.bz2" | tar xf -
    bzcat "gcc-g++-${GCC}.tar.bz2" | tar xf -
    for d in gcc*.diff
    do
        if [ -r "$d" ]
        then
            cat "$d" | (cd "gcc-${GCC}" ; patch -p1)
        fi
    done

    rm -rf "newlib-${NEWLIB}"
    zcat <"newlib-${NEWLIB}.tar.gz" | tar xf -
    for d in "newlib-${NEWLIB}"*.diff
    do
        if [ -r "$d" ]
        then
            cat "$d" | (cd "newlib-${NEWLIB}" ; patch -p1)
        fi
    done
    (cd "gcc-${GCC}" ; ln -s "../newlib-${NEWLIB}/newlib" newlib)
```

```
    rm -rf "gdb-${GDB}"
    bzcat <"gdb-${GDB}.tar.bz2" | tar xf -
    for d in "gdb-${GDB}"*.diff
    do
        if [ -r "$d" ]
        then
            cat "$d" | (cd "gdb-${GDB}" ; patch -p1)
        fi
    done
}

#
# Build
#
build() {
    PATH="${PREFIX}/bin:$PATH"
    for arch in $ARCHS
    do
        rm -rf build
        mkdir build
        cd build
        "${SHELL}" "../binutils-${BINUTILS}/configure" \
            "--target=${arch}-rtems${RTEMS_VERSION}" "--prefix=${PREFIX}" \
            --verbose --disable-nls \
            --without-included-gettext \
            --disable-win32-registry \
            --disable-werror
        ${MAKE} -w all install
        cd ..

        rm -rf build
        mkdir build
        cd build
       "${SHELL}" "../gcc-${GCC}/configure" \
            "--target=${arch}-rtems${RTEMS_VERSION}" "--prefix=${PREFIX}" \
            --disable-libstdcxx-pch \
            --with-gnu-as --with-gnu-ld --verbose \
            --with-newlib \
            --with-system-zlib \
            --disable-nls --without-included-gettext \
            --disable-win32-registry \
            --enable-version-specific-runtime-libs \
            --enable-threads \
            --enable-newlib-io-c99-formats \
            --enable-languages="c,c++" \
            --with-gmp="${PREFIX}" --with-mpfr="${PREFIX}"
        ${MAKE} -w all
        ${MAKE} -w install
        cd ..

        rm -rf build
```

```
        mkdir build
        cd build
       "${SHELL}" "../gdb-${GDB}/configure" \
            "--target=${arch}-rtems${RTEMS_VERSION}" "--prefix=${PREFIX}" \
            --verbose --disable-nls --without-included-gettext \
            --disable-win32-registry \
            --enable-version-specific-runtime-libs \
            --disable-win32-registry \
            --disable-werror \
            --enable-sim \
            --with-expat
        ${MAKE} -w all
        ${MAKE} -w install
        cd ..
    done
}

#
# Do everything
#
# Comment out any activities you wish to omit
#
set -ex
getSource
unpackSource
export LD_LIBRARY_PATH="${PREFIX}/lib"
build
```