Xport 2.0 eCos and RedBoot

Version 1.2, June 29, 2004

Summary

This application note explains how to use eCos and Redboot with the Xport 2.0. Remote debugging using GDB is also explained.

Introduction

eCos is a powerful, free, open source, real-time OS for embedded applications. Some eCos features:

- Preemptive multi-threading
- Configurable scheduling
- Synchronization objects
- Timers, counters, alarms, mailboxes
- ISR/DSR interrupt handling
- Efficient implementations of ISO C and math libraries
- GDB debug support
- Very small memory footprint (30K typical)

Using eCos simply entails linking to the runtime libraries.

Quickstart

Following these steps provides a quick demo of source level debugging of a simple eCos multi-threaded example. Execute the following steps from within the Xport Shell.

1. Upload RedBoot into the Xport.

   ```
   cd $XPORTDIR/examples/ecos/redboot
   make upload
   ```

2. Run GDB.

   ```
   cd $XPORTDIR/examples/ecos/twothreadsram
   arm-agb-elf-gdb twothreads.elf
   ```

3. At the GDB prompt type

   ```
   target remote | xpcomm
   ```

4. Wait for the response (or similar)

   ```
   0x08004010 in ?? ()
   ```

5. Load the program into RAM. At the GDB prompt type

   ```
   load
   ```

6. Wait for the program to load.

7. Make a breakpoint. For example, type

   ```
   break printf
   ```

8. Run the twothreads example by typing

   ```
   continue
   ```

9. You are now debugging within GDB. After typing "continue" (or simply "c") several times, the twothreads example will create two threads and begin printing a steady stream of simple messages. At this point you may type

   ```
   info threads
   ```

   to view the thread table. (The GDB stubs in RedBoot are thread-aware.)

10. Typing "quit" will exit GDB.

The above steps don't even scratch the surface when it comes to exploiting the power of GDB. If you are unfamiliar with GDB, you can find the documentation here:

http://www.gnu.org/manual/GDB/

You may have noticed when debugging twothreads that some of the debugging information is missing – namely the source code. Since eCos and twothreads were compiled elsewhere, GDB cannot locate the source code, which has surely moved. Recompiling will update the source code paths and allow you to view the source code while debugging.

You can recompile the eCos RAM libraries by typing:

```
cd $XPORTDIR/src/ecos
make ecosram install
```

To recompile twothreads:

```
cd $XPORTDIR/examples/ecos/twothreadsram
make clean
make
```

You can now go back to step 2 to see how things look with source code (much better!)

You can use a GDB initialization file that performs the connection and loading automatically upon invoking GDB. For example, create a file in your home directory ~/gdb.ini that contains the text:

```
target remote | xpcomm
load
```

Then upon invoking gdb run it with the –x option, such as:

```
arm-agb-elf-gdb twothreads.elf –x
~/gdb.ini
```

But don't bother with this! We have included this initialization file and a shell script that does all of this automatically. Simply run it as follows:

```
rungdb twothreads.elf
```

These files are included in Xport eCos releases 2.0.12 and higher.

Insight

Running GDB from within a shell (without a GUI) may be too archaic for many folks. Insight offers a nice GUI to more intuitively highlight the many features of GDB. To debug the twothreads example using Insight, execute the following steps from within the Xport Shell.

1. Run Insight.

   ```
   cd $XPORTDIR/examples/ecos/twothreadsram
   arm-agb-elf-insight twothreads.elf
   ```

2. Bring up the console window by selecting View→Console from within the toolbar.

3. From within the console window type:

   ```
   target remote | xpcomm
   ```

4. Wait for the response (or similar)

   ```
   0x08004010 in ?? ()
   ```

5. Load the program into RAM. In the console window type

   ```
   load
   ```

6. Wait for the program to load.

7. Make a breakpoint. For example, type

   ```
   break printf
   ```

8. Run the twothreads example by typing

   ```
   continue
   ```

You are now debugging within Insight. When the program reaches a breakpoint, you can see the current line of execution in the source window highlighted in green. Other windows such as the call stack, thread list, etc can be displayed by selecting them in the View menu. Having multiple windows simultaneously displaying the stack, memory, variables, etc. is one of the most attractive features of Insight.

We have also included a shell script for Insight that is similar to rungdb. Simply run it as follows:

```
runinsight twothreads.elf
```

RedBoot

RedBoot is a bootstrap environment that is designed to work with eCos. It includes GDB stubs for debugging, a flash file system, and facilities for examining and modifying memory.

RedBoot can be uploaded into the Xport 2.0 by running the following commands from within the Xport Shell:

```
cd $XPORTDIR/examples/ecos/redboot
make upload
```

You can communicate with Redboot through the Cport by using Xpcomm in console mode. That is, run Xpcomm as

```
xpcomm –console
```

when RedBoot is running on the Xport. Typing "help" at the RedBoot prompt provides a brief synopsis of the available commands. Refer to the RedBoot documentation for more detailed information. The documentation is located in the Start menu (Start→Programs→Xport→eCos→Redboot User Guide).

Note, when first running RedBoot, it will complain of a "flash configuration checksum error". This is because the flash file system has not been initialized yet. Type "fconfig –i" at the redboot prompt through the xpcomm console to initialize the flash file system.

RAM vs ROM targets

In the example above we ran eCos from RAM using example code linked with the eCos RAM libraries (e.g. twothreadsram, located in $XPORTDIR/examples/ecos/twothreadsrom). GDB needs to modify the code to set breakpoints, so it is necessary that the code reside in RAM in order for it to be debugged. Running from RAM has the advantage of being debuggable, but requires that the code be uploaded upon power-up through RedBoot.

Alternatively, the same example code can be linked with the eCos ROM libraries and run from ROM (e.g. twothreadsrom, located in $XPORTDIR/examples/ecos/twothreadsrom). This has the advantage of being automatically executed upon power-up, but it cannot be debugged with GDB.

Both RAM and ROM libraries are provided precompiled in the Xport eCos distribution. They are located in $XPORTDIR/lib/ecos.

Rebuilding eCos and RedBoot

Rebuilding the complete (default) eCos runtime libraries and RedBoot can be easily accomplished by running the following in the Xport Shell:

```
cd $XPORTDIR/src/ecos
make ecosram install
```

Here, you can replace ecosram with ecosrom or redboot depending on what you would like to rebuild (or you can build them all by listing all three.)

eCos Configuration Tool

The eCos Configuration Tool is installed as part of the Xport eCos distribution. It allows you to customize eCos to your needs (e.g. change kernel scheduling policy, memory allocation method, add or remove compilation options, etc.) The sheer number of options is impressive, and the fact that they can be modified from within a nice GUI environment is even more impressive. The tool is located in the Start menu (Start→Programs→Xport→eCos→Configuration Tool)

To create a custom version of eCos, bring up the Configuration Tool. From the toolbar, select Build→Templates to bring up the Templates dialog. In the Hardware pulldown selection choose Game Boy Advance with Xport and click OK. By default, this will choose a RAM configuration for the GBA. If this is the desired configuration, you can save the new project in an appropriate location (File→Save). It is recommended that you save new projects in their own directories as many files and subdirectories are created.

If you want a ROM configuration, for example, from within the Configuration pane (the subwindow toward the upper-left corner of the ConfigTool window) expand eCos HAL→ARM architecture→GameBoy Advance with Xport. Then from the Startup type field (under Game Boy Advance with Xport) you can click on RAM (the current value) and then change it to ROM. Saving this configuration will result in a ROM-based eCos configuration. Note, other eCos configuration parameters can be modified in a similar manner, if you wish to customize eCos to your needs.

Building newly-created eCos projects is simply a matter of running "make build" from the <projectname>-build directory, where <projectname> is the name you assigned to the project when saving it.

Some notes on printf()

When making calls to printf(), output automatically goes to the GBA LCD and the Cport. Also note:

- The LCD output can be disabled from the eCos Configuration Tool (in case you wish to use the LCD for other purposes, e.g. graphics.)
- For ROM targets, the console data from calls made to printf(), for example, can be viewed by running Xpcomm in console mode (xpcomm –console). (Try

uploading the twothreadsrom example and running "xpcomm –console" for a quick demo of this.)

- Unlike serial ports, outputting to the Cport incurs negligible overhead when the PC is not listening (that is, when you are not running Xpcomm in console mode, handshaking is turned off and writing data to the Cport from the GBA side is immediate – only the overhead of a bus cycle is incurred.)

- For RAM targets, the console output appears in the GDB console window. (That is, console information is still passed through the Cport, but is mangled so GDB can parse and display the information.)

Other notes

We like to think of the eCos libraries as a direct replacement for libc. It provides all of the facilities you usually want (heap, static constructors/destructors, floating point, math libraries) and facilities that are important for more complex applications (multi-threading, synchronization primitives, timers, alarms, mailboxes.)

Facilities not used are not linked in and do not wind up in the final elf binary. Thus, no unnecessary overhead is typically incurred by linking to eCos. Also note, if you plan on not using multiple threads and do not want the CPU overhead incurred by preemptive scheduling (granted, the overhead is small <2%), simply implement the cyg_user_start() function as your main() function. Not returning from cyg_user_start() prevents eCos from launching the scheduler.

Documentation

eCos is well documented. Look in the Start menu (Start→Programs→Xport→eCos). Here, you will find several PDF documents as well as an HTML document.

A good starting point is to go to the eCos HTML documentation and select eCos Reference. This page contains a good index of the many facilities offered.

Contacting Us

Comments or questions? Please email them to support@charmedlabs.com.