RedBoot and eCos an Application Specific Operating System

Weqaar Janjua
Arizona State University, Tempe, AZ
<janjua@asu.edu>

Abstract

This paper focuses on efficient implementation of eCos (Embedded Configurable Operating System)[1] Real-Time Operating System on embedded hardware systems and RedBoot[2] boot loader as a simulated ROM monitor. Target board used as an example is Intel architecture based computer system chosen as a demonstration platform. The paper describes the usefulness of using an application specific operating system in highly resource constrained embedded environments. It starts with building the host environment for developing the toolchain and delves down further into developing an application on the host machine, transferring it to the target board and executing it.

1. Problem Statement

There are many open-source and close-source Real-Time embedded operating systems available in the market. Most of them are based on the Linux kernel providing hard and soft real-time capabilities. These operating systems are generic in nature and are not application specific. Systems based on such operating systems require software applications that are operating system specific i.e. a graphics application needs to

be specifically designed for the Linux operating system in order to execute it. Though the Linux kernel can be customized for application specific purposes by writing or using a Linux micro-kernel but porting it to different target boards is a complex task and the kernel can be shrunk up to a certain level of functionality. Even at that certain level of functionality the operating system does not become application specific and requires a minimum set of system resources to function at a required level. Therefore, the objectives of this project are to:

- Conduct literature research on eCos operating system.
- Compare eCos operating system's functionality to traditional Real-Time operating systems.
- Define a procedure for developing a proper host environment for developing applications for eCos RTOS.
- Show the benefits of choosing eCos RTOS as an embedded OS over traditional embedded operating systems.
- Present shortcomings in eCos RTOS current version i.e. 2.0.
- Suggest areas of future work.

2. Introduction to eCos

The eCos (Embedded Configurable Operating System) is an open-source Real-Time embedded operating system ported to a variety of architectures[3]. It gives developers a low-cost, royalty-free embedded software development solution that works in highly constrained hardware environments.

“eCos is an open source, royalty-free, real-time operating system intended for embedded applications. The highly configurable nature of eCos allows the operating system to be customized to precise application requirements, delivering the best possible run-time performance and an optimized hardware resource footprint.[4]”

eCos uses RedBoot for

bootstrapping. RedBoot is a complete bootstrap environment for embedded systems based on the eCos HAL (Hardware Abstraction Layer)[5].

2.1. Overview of eCos, an ASOS (Application Specific Operating System)

Traditional operating systems need applications that are written specifically for them but in the case of an application specific operating system the application is written using a set of eCos supported APIs are linked against the kernel which results in a single binary containing the combined application and kernel code. Fig.1 shows the process flow.

eCos kernel does not need to run on the target board in order to execute the application i.e. running an eCos kernel is optional.

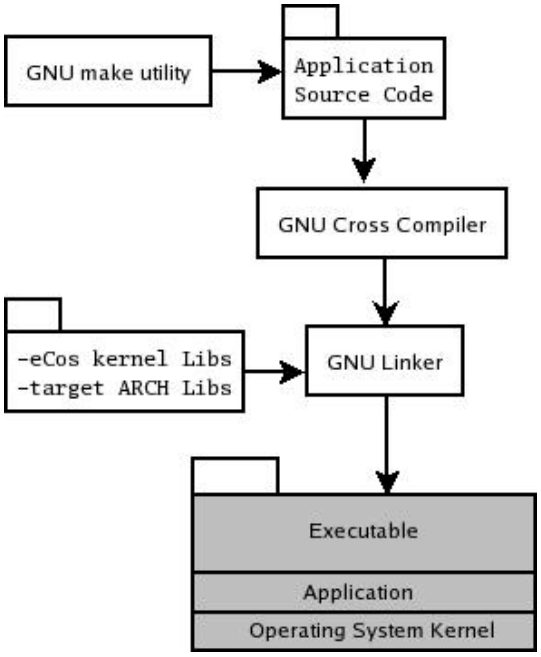

Fig.1

2.2. eCos HAL

The first component in the eCos system architecture is the hardware abstraction layer, which can be broken down into three sub-modules:

1. The first sub-module defines the architecture. Each processor family supported by eCos is said to be a different architecture. Each architecture sub-module contains the code necessary for CPU startup, interrupt delivery, context switching,

and other functionality specific to the instruction set architecture of that processor family.

2. The second HAL sub-module defines the variant. A variant is a specific processor within a processor family.
3. The third HAL sub-module defines the platform or board that includes the selected processor architecture and variant. This module includes code for startup, chip selection configuration, interrupt controllers, and timer devices.

2.3. eCos Kernel

Unlike traditional embedded operating systems, an application does not need to be executed on top of a running kernel. The application does not need to be written specifically using eCos APIs since the operating system supports µITRON and POSIX APIs. It also includes fully featured and thread-safe ISO standard C and math libraries[6].

Just like UNIX where everything is a file every part of eCos is a package. Memory allocation is handled by a separate package, device drivers will typically be a separate package. Various packages are combined and configured using the eCos configuration tool to meet the requirements of the application.

eCos incorporates almost all the functionalities of modern operating systems. Since, running the eCos kernel is optional, It is possible to write single-threaded applications which do not use any kernel functionality[7].

3. Methodology

The author starts with building and configuring the host development environment, and building the tool chain. Then covers configuring and building the boot loader, and installing it on the target board. A sample application is developed on the host system which is then transferred to and executed on the target board.

The target board used is a

Pentium-IV Desktop PC which is simulated as an Intel based embedded board. The PC is equipped with an onboard Ethernet controller with 82559 chipset which is the most suitable chipset for RedBoot as tested by the author. It is booted from a floppy disk, it then loads the boot loader into RAM. The host machine used is also an Intel based desktop computer running Linux OS.

3.1. Building the host development environment

Before we start building the applications for eCos we need to have a complete embedded software development environment i.e. a set of compiler, debugger and utilities that will enable us to compile our programs from source, load and execute them on the target board. We focus on the target architecture specific compiler and debugging tools.

3.1.1. Required tools and utilities

The platform specific cross-development tools can be broken down into three categories:

- GNU Binary Utilities – http://sources.redhat.com/binutils
- GNU C/C++ Compilers – http://gcc.gnu.org
- GNU GDB Debugger - http://www.gnu.org/software/gdb/gdb.html

In addition, TCL/TK wish shell and an Xserver running on the host machine are required to execute eCos installation and configuration tools. These tools are available on Linux operating system distributions by default. Installation and configuration of these tools is beyond the scope of this paper. For further information regarding building from source please refer to README and INSTALL files in each package.

3.2. Acquiring eCos and Building the toolchain on host development platform

The latest eCos release at the point of this writing is version 2.0 which features an installation tool which simplifies the downloading and installation of the eCos sources, host tools and documentation. The installation tool can be used to download GNU cross toolchains(courtesy of eCosCentric Limited[8]) for use in conjunction with eCos.

Download the eCos installation tool by using the following command at a shell command prompt: '*wget –passive-ftp ftp://ecos.sourceware.org/pub/ecos/ecos-install.tcl*'

Execute the installation tool as '*sh ecos-install.tcl*'. The installation tool when executed presents a list of mirror sites from which the software may be downloaded. Select the closes mirror site according to your geographical region. The tool will then prompt for an installation location. We refer to the installation location as '$*ECOS*'. Next, the tool will present a list of pre-built GNU toolchains available for download. Select 'i386-elf' and then 'q' to finish selection and start downloading.

3.3. Using 'configtool' to configure and build RedBoot on the host system

Once the installation tool is done downloading the toolchain and eCos distribution, the next step is to use the configuration tool to configure the boot loader and ROM monitor i.e. RedBoot. In our case, we use the FLOPPY startup type configuration since we use a desktop pc as the target.

Before we begin, let’s get an overview of the build procedure. Figure.2 shows the flow of the build and install procedure. Execute 'configtool' which is located under '*$ECOS/ecos-2.0/tools/bin*'.

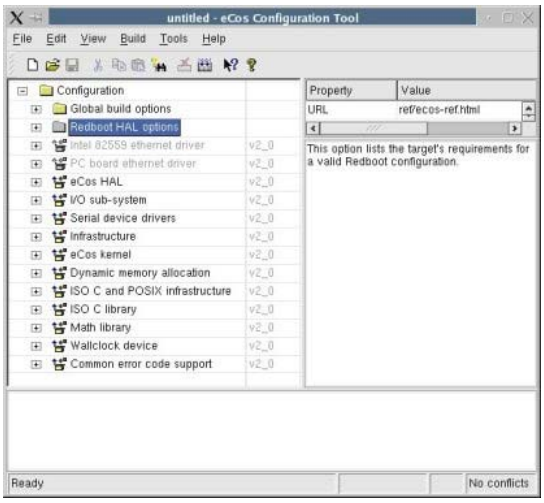

Fig.2

The first step is to load the packages we need to build the RedBoot image. We use a template for this by selecting *Build -> Templates* from the configuration tool. In the templates dialog box we select *i386 PC target* from the hardware drop-down list and *redboot* package from packages drop-down list and then click on *OK* button to continue. If Resolve Conflict dialog box pops up press *continue*. We don't need to worry about it at this point.

The second step is to set up the configuration options for our RedBoot build. We do this by importing the minimal configuration file we select *File -> Import* from the menu. Then select the appropriate ecm(eCos minimal configuration file)

'*$ECOS/ecos_2.0/packages/hal/i386/pc/v2_0/misc/redboot_FLOPPY.ecm*'. If Resolve Conflict dialog box pops up again press *continue*. We don't need to worry about it at this point.

The third step is to select configuration options and verify them. We set the IP Address and Serial port configurations for RedBoot ROM monitor. From the configuration tool select *redboot ROM monitor -> Build Redboot ROM ELF image -> Redboot networking -> Default IP address* and set it to the IP Address that the host machine can connect to, in the format 192, 168, 0, 1 (note commas instead of dots), preferably from the same subnet as the host machine. Make sure the option *Do not try to use BOOTP* is unchecked since we are remote booting. Then, from *Redboot networking* select

DNS support -> Default DNS IP and set it to the IP Address of any DNS that is accessible from the subnet, in the format 129.219.17.5 (note dots instead of commas).

Next we select the communication channel for uploading binary files to RedBoot ROM monitor from *eCos HAL -> i386 architecture -> i386 PC Target*. For our set up we use the default communication channel 0 which is serial port 0 i.e. /dev/ttyS0 and leave the option Output to PC screen checked since we may use a monitor connected to the PC for troubleshooting purposes though we do not discuss this explicitly in the paper.

Next, we save our configuration file by selecting *File -> Save as* and picking a random name for it, say '*ecos_config.ecm*' in the directory '*$ECOS'* that we created in section 3.2. of this paper. The configuration tool then creates following three sub-directories under '*$ECOS*':

1. *ecos_config_build:* Contains the build tree and is used by the configuration tool to store files used and created in the build process such as make and object files.
2. *ecos_config_install:* Contains the install tree, output binary images and header files used for the build process.
3. *ecos_config_mlt:* Contains memory layout files used by the Memory Layout Tool, which is a subset of

   the configuration tool.

Next step is to build the RedBoot image from the configuration tool using our configuration settings. For this purpose we select Build -> Library from the menu. The output window on bottom of the configuration tool shows output from the compiler. The output window then displays *build finished* as soon as the compilation and linking is done successfully. In the case of errors you need to go back and check for conflicts

and repeat the build procedure.

Now that the build is done we should have a binary image named '*redboot.bin*' under the directory '*$ECOS/ecos_config_install/bin*'.

3.4. Loading the boot loader to the Floppy Disk

We use the *dd* utility to write the image onto the floppy disk. From the shell prompt we change the directory i.e. *cd* to '*$ECOS/ecos_config_install/bin*' and issue the command '*dd conv=sync if=redboot.bin of=/dev/fd0*'. After the process is complete, a message is displayed showing the records converted in the format:

xxx+1 records in

xxx+0 records out

3.5. Booting the target board

Now that we have a bootable floppy containing the binary image of RedBoot our next step is to boot the target system. Boot the target system from the floppy disk by using standard floppy boot procedures which are not covered in this paper. When the system boots it displays Ethernet configuration on the top and displays a command prompt if everything works as expected.

3.6. Connecting to the target board over IP

On the host system, use a telnet client to connect to the target board by issuing the command 'telnet <ip_address>" where the <ip_address> is the IP Address that was set previously in section 3.3. of this paper. As soon as you connect you will see the RedBoot prompt awaiting input.

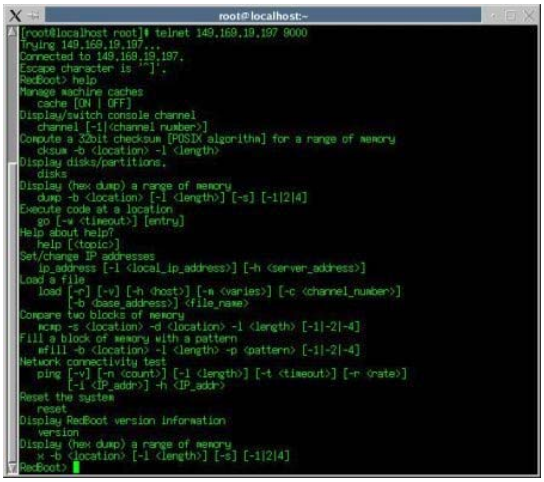

Figure.3

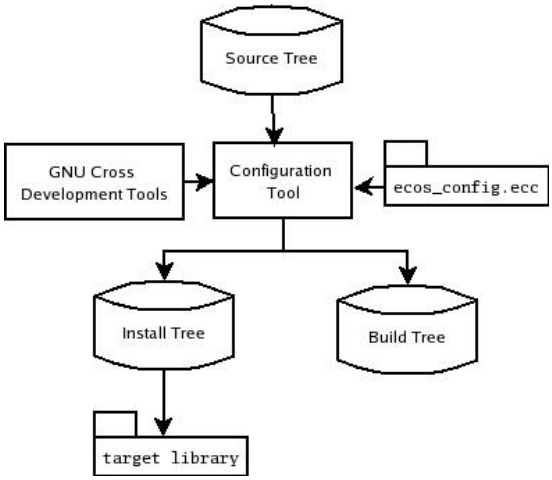

Figure.4

At this point the console i.e. RedBoot ROM monitor is only accessible from telnet prompt since only one channel can be used at one time. To switch the channel issue the command 'channel x' where x is the channel no. for the device i.e. 'channel -1' which is the default channel used, keyboard connected to the system in our case.

4. Building eCos

In this stage we generate the eCos library file according to our configuration settings. Figure.4 outlines the build process.

We repeat the same procedure for configuring and building eCos as outlined in section 3.3. except we select *default* from *Build -> Templates* packages drop-down list. Next we save our configuration by selecting *File -> Save As* from the menu, input the filename.ecc and click *OK*. Say we name the file '*ecosConf.ecc*'.

Configuration tool creates the same directory structure as in section 3.3. of this paper. Then we follow the build step outlined in section 3.3. of this paper and the output binary files are created in

'*$ECOS/ecosConf_install/bin*', header files in '*$ECOS/ecosConf_install/include*' and library files in '*$ECOS/ecosConf_install/lib*' directories.

5. Writing an application on the host system

We are now ready to start building our sample application and incorporate our eCos library built in the last section of this paper. Create a file '*sample.c*':

```
#----------START_OF_FILE----------
/*
Sample eCos application that prints a string on STDOUT.
Author: Weqaar Janjua <janjua@asu.edu>
*/
#include <stdio.h>
int main ( void) {
    printf ("Hello embedded world!\n");
    return 0;
}
#----------END_OF_FILE----------
```

Next step is to write a Makefile:

```
#----------START_OF_FILE----------
# Makefile for sample.c
#Author: Weqaar Janjua
# <janjua@asu.edu>

ECOS = /opt/ecos

CC = $(ECOS)/gnutools/i386-elf/bin/i386-elf-gcc

ECOSDIR = $(ECOS)/ecosConf_install

CFLAGS = -g -Wall -I$(ECOSDIR)/include -ffunction-sections -fdata-sections

LDFLAGS = -nostartfiles -L$(ECOSDIR)/lib -L$(ECOS)/gnutools/i386-elf/lib/gcc-lib/i386-elf/3.2.1
 -L$(ECOS)/gnutools/i386-elf/i386-elf/lib

LIBS = -Ttarget.ld

LD = $(ECOS)/gnutools/i386-elf/bin/i386-
```

```
elf-ld

all: sample

sample.o: sample.c
	$(CC) -c -o $*.o $(CFLAGS) $<

sample: sample.o
	$(LD) $(LDFLAGS) -o $@ $@.o \
	$(LIBS)

clean:
	rm -f sample.asm
	rm -f sample.o *.ihx *.lnk *.lst *.map \
	*.rel *.rst *.sym
#----------END_OF_FILE-----------------------
```

The author assumes the reader is familiar with writing makefiles, if this is not the case please refer to *make* utility documentation.

Once we are done writing the make file, our next step is to execute '*make*' which creates a binary named '*sample*'.

5.1. Loading the application on the target system and executing it

Now upload the file '*sample*' to the target system using the serial port. In our default configuration for eCos we left the serial port baud rate set to 38400bps, we need to make sure both the ends are set to the same baud rate.

Execute a terminal emulation software on the host system, we use '*minicom*', and set it to 38400bps 8N1 . On the target system's RedBoot prompt execute '*load -v -m yMODEM*' i.e. '*RedBoot> load -v -m -yMODEM*', since we use the ymodem protocol to transfer files between the systems. On the host system, using minicom go to options and select *send file*, select '*sample*' from the menu and send using ymodem protocol. The target system will display a message after file transfer is complete. Now execute the application by typing *go* on the prompt i.e. '*RedBoot> go*'. The target system should display the string “*Hello*

embedded world!" on the console, of course without quotes, Eureka!

6. Shortcomings in eCos and areas of future work

eCos kernel currently does not have wireless extensions built into it and because of that there are no wireless packages available to it. This is an area open to research and development.

7. Conclusion

If you go through eCos configuration tool you will find out the fine level of granularity provided by eCos configuration tool. No other operating systems can provide such level of fine-tuned configuration as compared to eCos. eCos is an excellent substitute for generic ROM monitors and in some cases firmware. Many commercial products like routers and switches use of-the-shelf Operating Systems that have higher resource utilization and they don't provide such finer level of granularity in configuration.

REFERENCES:

[1] http://ecos.sourceware.org

[2]http://ecos.sourceware.org/redboot

[3]http://ecos.sourceware.org/hardware.html

[4]http://sources.redhat.com/ecos/

[5]http://ecos.sourceware.org/docs-latest/ref/hal-architecture-variant-and-platform.html

[6]http://ecos.sourceware.org/docs-latest/user-guide/ecos-key-features.html

[7]http://ecos.sourceware.org/docs-latest/ref/kernel-overview.html

[8]http://www.ecoscentric.com