ECos for Toradex Colibri VF61 Freescale Vybrid CoM - manual

Antmicro Ltd

April 25, 2014

CONTENTS

CHAPTER

ONE

INTRODUCTION

This is a compilation and usage manual for the port of the eCos RTOS (real-time operating system) for the Toradex Colibri VF61 Freescale Vybrid Computer on Module. It gives a brief overview on how to get the port, compile it and run an example program on the module using a Linux host.

The eCos port is targeted for the Cortex-M core of the heterogeneous Vybrid CPU, to provide a robust way to drive a real-time control setup, and is best combined with Linux or similar OS running on the Colibri VF61 Cortex-A core for handling non-critical outside communication, user interfaces etc.

1.1 ECos

ECos is a configurable RTOS intended for use in embedded applications. The documentation for eCos 3.0, which is the most recent version of the system as well as the one ported to Colibri VF61, can be found at http://ecos.sourceware.org/docs-3.0/.

A comprehensive PDF eCos Reference Guide is also available from the eCos website.

1.2 Licence

(based on the eCos licence overview)

ECos is released under a modified version of the well known GNU General Public License (GPL). The eCos license is officially recognised as a GPL-compatible Free Software License. An **exception clause** has been added which limits the circumstances in which the license applies to other code when used in conjunction with eCos. The exception clause is as follows:

> As a special exception, if other files instantiate templates or use macros or inline functions from this file, or you compile this file and link it with other works to produce a work based on this file, this file does not by itself cause the resulting work to be covered by the GNU General Public License. However the source code for this file must still be made available in accordance with section (3) of the GNU General Public License.
>
> This exception does not invalidate any other reasons why a work based on this file might be covered by the GNU General Public License.

The license does not require users to release the source code of any *applications* that are developed with eCos.

1.3 Supported features

This eCos port provides the following software packages specific for Toradex Colibri Vybrid VF61 Vybrid module:

- HAL package
- debug UART driver
- serial port driver
- Flex Timer Module
- GPIO handling

Also, the port has been verified to work with the standard eCos POSIX an µITRON compatibility layers. See *Appendix B: POSIX and µITRON compatibility* for more information on this.

1.4 Version information

Author	Content	Date	Version
Peter Katarzynski	Draft version	2014-03-21	0.1.0
Michael Gielda	Revamp	2014-03-27	0.2.0
Michael Gielda	Prerequisites & compiling sample programs	2014-03-27	0.3.0
Michael Gielda	Running programs	2014-03-28	0.3.1
Michael Gielda	Further updates	2014-04-02	0.3.2
Michael Gielda	Further updates & Appendix B	2014-04-07	0.4.0
Michael Gielda	Cleanup & division into files	2014-04-08	0.5.0
Michael Gielda	Simplified instructions & making tests	2014-04-11	0.5.1
Michael Gielda	Added POSIX compatibility description	2014-04-11	0.5.2

CHAPTER

TWO

BUILDING THE ECOS KERNEL AND APPLICATIONS

This chapter describes how to build an eCos kernel and compile eCos applications for Colibri VF61.

The build process was tested on the Gentoo, Debian, Ubuntu and Mint Linux distributions. The procedures described here should also work on other systems, but if you find any way to improve this manual with respect to tested platforms, please e-mail us at contact@antmicro.com.

Note: The code blocks below, when copy-pasted to a Linux terminal, should all work, provided they were called from the same directory. To avoid confusion, it is best to call them from within a new, empty directory, e.g.:

```
mkdir ~/ecos-from-scratch
cd ~/ecos-from-scratch
```

2.1 Prerequisites

2.1.1 Toolchain

This port of eCos was prepared using a pre-built standard eCos toolchain, which can be obtained e.g. from the GWDG FTP server.

```
wget ftp://ftp.gwdg.de/pub/misc/sources.redhat.com/ecos/gnutools/i386linux/test/\
ecos-gnutools-arm-eabi-20120623.i386linux.tar.bz2
tar xjvf ecos-gnutools-arm-eabi-20120623.i386linux.tar.bz2
```

Alternatively it is possible to compile eCos software using self-built toolchains as described on the eCos website.

To compile eCos and eCos applications, the toolchain's `bin` directory has to be included in the PATH variable. The proper availability of the toolchain can be checked by finding out if `arm-eabi-gcc` is available from the shell.

2.1.2 ecosconfig

The **ecosconfig** tool, available from the eCosCentric website, is used to generate the build tree from the main repository and is a mandatory requirement. **ecosconfig** requires the tcl compiler to work.

Installing tcl on Debian-based distributions

```
sudo apt-get install tcl8.5 # use sudo emerge dev-lang/tcl for Gentoo
```

Now you can download and use **ecosconfig**. You also need to make **ecosconfig** executable after downloading and extracting it from the archive. It is also a good idea to make it available system-wide by moving it to `/usr/local/bin`.

Installing ecosconfig

```
wget http://www.ecoscentric.com/snapshots/ecosconfig-100305.bz2
bunzip2 ecosconfig-100305.bz2
chmod +x ecosconfig-100305
sudo mv ecosconfig-100305 /usr/local/bin/ecosconfig
```

Warning: **ecosconfig** is a 32bit application, thus if you are using a 64bit OS you have to provide 32bit run-time libraries for compatibility. In a Debian-based Linux distributions these could be installed using the command `sudo apt-get install ia32-libs`.

Note: The output of **ecosconfig** are `.ECC` (eCos Configuration) files which are in essence `tcl` scripts storing all the information on what elements will be included in the system image and how they will be configured.

Note: A handbook on ecosconfig exists to help in the manual creation of `.ecc` files. Also, if you want to create custom eCos configuration files, see *Appendix A: custom eCos configuration*.

2.2 Source code

The source of the port can be downloaded by using the following command:

Downloading the Colibri VF61 eCos source

```
git clone https://github.com/mgielda/ecos-colibri-vf61.git
```

2.3 Preparing an .ecc file

The actual configuration of the eCos system is maintained and modified through **ecosconfig**. The following commands will prepare a sample `.ecc` file for a kernel with default settings.

Generating the kernel ecc file from scratch

```
export ECOS_REPOSITORY="$PWD/ecos-colibri-vf61/ecos/packages"
# Create ecos.ecc file based on Colibri VF61 default template
ecosconfig new col_vf61 default
```

You now have a `ecos.ecc` file that holds the default eCos configuration for Colibri VF61. The file can be further edited manually with a text editor and/or **ecosconfig** or graphically using **configtool** (see *Appendix A: custom eCos configuration*), but at this moment it is already enough to compile a sample eCos kernel.

2.4 Building the kernel

The eCos kernel is built in two stages:

- first, a so-called *build tree* is generated from the eCos sources by **ecosconfig**. The build tree is customized for your build as configured in the `.ecc` file used. It is best to generate the build tree in a separate directory (here `build-tree`).
- then, the source files are compiled

Warning: When copy-pasting the following to the terminal, take care not to export the PATH variable multiple times.

Building the eCos kernel

```
export PATH="$PWD/gnutools/arm-eabi/bin:$PATH"
export ECOS_REPOSITORY="$PWD/ecos-colibri-vf61/ecos/packages"

mkdir -p build-tree
rm -rf build-tree/*
cd build-tree

ecosconfig --config=$PWD/../ecos.ecc tree
make
cd ..
```

The resulting kernel files can be found in `build-tree/install/lib`.

2.5 Application

With a compiled kernel files in the `build-tree/install/lib` directory (see *Building the kernel*), a user space eCos application can be compiled and linked to it.

A listing for a short sample application (taken from `ecos-colibri-vf61/ecos/examples/hello.c`) is given below.

hello.c - sample application

```
#include <stdio.h>

int main(void)
{
    printf("Hello, eCos world!\r\n");
    return 0;
}
```

You can compile an eCos program with a procedure similar to the following listing (which you can save for reuse, for example as `make.sh`):

Warning: When copy-pasting the following to the terminal, take care not to export the PATH variable multiple times.

Building a user space application

```
export PATH="$PWD/gnutools/arm-eabi/bin:$PATH"

# Set compiler options
OPT="-Wall -Wpointer-arith -Wstrict-prototypes -Wundef  \
     -Wno-write-strings -mthumb -g -O2 -fdata-sections  \
     -ffunction-sections -fno-exceptions -nostdlib      \
     -mcpu=cortex-m4"

# Set path to eCos kernel
BTPATH="$PWD/build-tree"

# Do compilation and link your application with kernel
arm-eabi-gcc -g -I./ -g -I${BTPATH}/install/include hello.c \
             -L${BTPATH}/install/lib -Ttarget.ld ${OPT}

# Use objcopy to generate a binary
arm-eabi-objcopy -O binary a.out hello.bin
```

2.6 Tests

ECos is shipped with a test suite - in essence a set of simple programs checking various subsystems and interfaces - which can be easily compiled and run on the target. These tests were used in the porting effort, and many (although not all) of them can be used as bases for user programs.

In order to compile tests, use the same procedure as described in *Building the kernel*, but issue `make tests` instead of `make` at the end.

The resulting tests will reside in the `build-tree/install/tests` directory.

CHAPTER

THREE

RUNNING AN ECOS APPLICATION ON COLIBRI VF61

This chapter will explain how to run the eCos application on the Colibri VF61 Cortex-M core from the Cortex-A core in two ways: either from Linux using `MQXBboot` or from U-Boot using TFTP.

Vybrid has three available memory regions:

- OCRAM - 256KB - Default.
- DRAM - 10MB - Available, but needs limiting Linux RAM memory.
- TCML - 32KB - Small. Not recommended.

Note: Out of the 16MB of the DRAM memory in the CPU, 6MB was reserved for enabling the passing of large data blocks between Cortex-A and Cortex-M. This setting can be changed in the `mlt_vybrid_ext_dram.ldi` file located in `ecos/packages/hal/cortexm/vybrid/col_vf61/current/include/pkgconf/`, by modifying the `DRAM LENGTH` and `hal_startup_stack` values (currently `0x9FFFF0`).

3.1 U-Boot over TFTP

Set up TFTP on your host machine and put the binary to be loaded (for example, `hello.bin`) there. Make sure your host machine is connected to the network and you know its IP address.

3.1.1 Configuring U-Boot

Note: The memory restriction is mandatory only if DRAM is used to run eCos. See *Startup memory choice* for details.

Connect the module to the network with an Ethernet cable, power it on. Enter U-Boot and then use the following command sequence:

```
dhcp                          # set module IP address
set memargs mem=240M          # restrict Linux memory space
set serverip xxx.xxx.xxx.xxx # set TFTP server address
save                          # save the configuration
```

3.1.2 Running from U-Boot

The application can then be run over TFTP with the `tftp` command. The two other `mw` commands will set the entry point and turn on the clocks, respectively. OCRAM is recommended as default, but you may refer to *Startup memory choice* for information on what memory to use and how to get eCos to run from it.

OCRAM (default)

```
tftp 0x3f000400 hello.bin
mw.l 0x4006e028 0x1f000411
mw.l 0x4006b08c 0x00015a5a
```

DRAM

```
tftp 0x8f000400 hello.bin
mw.l 0x4006e028 0x0f000411
mw.l 0x4006b08c 0x00015a5a
```

TCML

```
tftp 0x1f800400 hello.bin
mw.l 0x4006e028 0x1f800411
mw.l 0x4006b08c 0x00015a5a
```

3.2 Linux over MQXBoot

3.2.1 Requirements

The following have to be present on the Cortex-A Linux (apart from the eCos binary) to make this method possible:

- `mcc.ko` kernel module
- `mqxboot` binary

3.2.2 Running from MQXBoot

The command to run the binary depends on the memory we want to use. OCRAM is recommended as default, but you may refer to *Startup memory choice* for information on what memory to use and how to get eCos to run from it.

Loading to OCRAM (default)

```
mqxboot hello.bin 0x3f000400 0x1f000411
```

Loading to DRAM

```
mqxboot hello.bin 0x8f000400 0x0f000411
```

Loading to TCML

```
mqxboot hello.bin 0x1f800400 0x1f800411
```

CHAPTER

FOUR

APPENDIX A: CUSTOM ECOS CONFIGURATION

ECos is called a *configurable* system for a reason: it contains a powerful infrastructure for choosing what system components and abstractions are included and how they are configured.

This Appendix will describe briefly how to deal with `.ecc` files for the Colibri VF61.

4.1 Background

The main tool used for building the eCos operating system is **ecosconfig** (see *Prerequisites*). The source tree of eCos, called **eCos repository** (like for example the source code tree provided in this release) is not built directly but instead first trimmed down and configured to suit the needs of a specific user and platform using **ecosconfig**. This static pick-and-build procedure allows the user to exclude these elements of the system which are not necessary, thus reducing the memory footprint. This mechanism also enables easy configuration of system-wide variables and driver specific features.

What exactly can be included, excluded or configured is determined by the contents of `.cdl` files residing side by side with all source files in the eCos repository (usually in the `cdl` directory on the same level as the `src` directory of a given package, like a driver for a particular interface).

4.2 configtool

configtool is a GUI front-end to **ecosconfig** to facilitate the creation of eCos configuration files. It also may be downloaded from eCosCentric.

> **Warning:** **configtool** (just like **ecosconfig**) is a 32bit application, thus if you are using a 64bit OS you have to provide 32bit run-time libraries for compatibility. In a Debian-based Linux distributions these could be installed using the command `sudo apt-get install ia32-libs`.

4.3 Templates

configtool allows the user to build the system however they want using a graphical user interface, provided constraints in `.cdl` files describing the system structure are maintained.

While creating a new `.ecc` file it is easier to also use a predefined template representing common use scenarios, such as **posix** which represents a system which has all the necessary packages to run typical POSIX programs or **redboot** which understandably is used to build a binary of RedBoot, the eCos bootloader.

In order to select a template to base upon, use *build* → *templates*.

Warning: Remember that the templates are just general scenarios, which may contain settings incompatible with the desired ones (baudrates, console mangling, debug console choice, presence of RedBoot ROM monitor). It is necessary to tweak them according to your needs.

4.4 Preparing an .ecc file using configtool

Launch **configtool**.

Select *build* → *repository* specify the path to eCos repository (the `packages` directory). Select the *build* → *template* option and choose the *Toradex Colibri VF61* as your hardware platform with default set of packages. Click *continue* to proceed.

When the default set of packages is used for the platform, the associated `.ecc` file can already be prepared. Save it in a directory accessible by your build script and **remember to point to it in** *the kernel build script*.

Other packages can be added from *build* → *packages*, bear in mind that you may need to alter the chosen packages and options to satisfy some `.cdl` constraints.

4.5 Startup memory choice

There are three memories from which eCos software may be launched in Colibri VF61. This is determined in the `.ecc` file the eCos kernel was based on.

By default the software is prepared to be launched from OnChip RAM (OCRAM). Alternatively DRAM memory may be used for booting. In this approach however the DDR memory block assigned to Linux must be limited to prevent Linux from accessing the memory region already occupied by eCos. This may be achieved by altering the boot arguments from U-Boot as described in the section entitled *Configuring U-Boot*.

The OCRAM and DDR scenarios are recommended; alternatively you may also try to run eCos from TCML, but this method was not tested. Besides, TCML offers a limited amount of memory which may be insufficient for many eCos applications.

To modify the startup memory scenario in the eCos kernel, **configtool** can be used. The appropriate menu option is:

eCos HAL → *Cortex-M Architecture* → *Freescale Vybrid Cortex-M4 Variant* → *Toradex Colibri VF61 Platform* → *Startup type*

The associated parameters are:

- CYG_HAL_STARTUP_PLF (ByVariant / DRAM)
- CYG_HAL_STARTUP_VAR (OCRAM / TCML)

CHAPTER

FIVE

APPENDIX B: POSIX AND µITRON COMPATIBILITY

By default, eCos allows the user to enable POSIX or µITRON compatibility, which may be beneficial for many applications. This Appendix explains briefly how to use them with this eCos port.

For more information refer to the eCos Reference Guide, Chapters XIII and XIV.

5.1 POSIX

POSIX (Portable Operating System Interface), is a well-known family of OS standards. POSIX defines the primitives, nomenclature and API which makes it easier to provide software compliance between operating systems.

This is especially useful with regard to programming heterogeneous devices like the Colibri VF61, where (especially with the POSIX compatibility layer enabled), the programming style for eCos applications dedicated for the Cortex-M4 core can vastly resemble that of Linux programs running on the Cortex-A5 core, lowering the entry barrier for programmers.

To activate this package, use either **ecosconfig** (`ecosconfig add CYGPKG_POSIX`) or **configtool**:

1. Choose *Build* → *Packages*
2. Select *POSIX compatibility layer* and click *Add* >>, then *OK*
3. A new package, `POSIX compatibility layer` should appear in the package list - save your `.ecc` file and exit.

A sample application, verified to work as expected using the port can be found inside the port code, in the subdirectory `compat/posix/current/tests/pthread1.c`.

5.2 µITRON

µITRON is a the name of an Japanese open standard for RTOS, originally undertaken in 1984 under the guidance of Ken Sakamura. eCos supports the µITRON version 3.02 specification, with complete "Standard functionality" (level S), plus many "Extended" (level E) functions.

More about ITRON and µITRON can be read in the following sources:

- introduction to ITRON project
- µITRON3.0 specification
- Dr. Sakamura's book: uITRON 3.0, An Open and Portable Real Time Operating System for Embedded Systems

Since eCos was designed with the µITRON guidelines in mind, it is not strictly necessary to "activate" this compatibility layer, as an eCos application may fulfill the standard anyway. However, eCos provides a package named `CYKPKG_UITRON` setting some additional constraints, and adding it to the kernel is recommended for applications meant to be µITRON compliant. To activate this package, use either **ecosconfig** (`ecosconfig add CYGPKG_UITRON`) or **configtool**:

1. Choose *Build* → *Packages*
2. Select *uITRON compatibility* and click *Add* >>, then *OK*
3. A new package, `uITRON compatibility layer` should appear in the package list - save your `.ecc` file and exit.

A sample application, verified to work as expected using the port can be found inside the port code, in the subdirectory `compat/uitron/current/tests/test3.c`