

As Easy As Eclipse?

Tutorial: Using a completely open source tool chain for ARM
software development

From Richard Barry,
August 2007

Introduction

Love it or hate it there is no denying the momentum of the Eclipse Workbench within the embedded development community. Many hardware and software vendors are turning away from proprietary development environments in favour of tailoring the Eclipse Workbench to meet their customer's needs.

As the maintainer of FreeRTOS.org – an open source mini real time kernel – I am continuously being asked to add Eclipse to the list of already supported development environments. I have resisted this, until now, primarily because my early experimentation with Eclipse showed that its file management could not cope with the established FreeRTOS.org source code structure. I am informed that ongoing developments have rectified these limitations, so is the time now right to embrace Eclipse for open source development?

It is difficult to form an opinion on the answer to this question without first gaining practical experience of using Eclipse on an actual project. To this end, and following some internet research, I converted an existing FreeRTOS.org based embedded WEB server implementation to be both built and debugged from within the Eclipse Workbench. Presented in this article are the results of this undertaking, along with links to information and resources that I found valuable along the way. It is hoped that the information will assist readers in quickly replicating the development environment and in doing so, assessing the open source tools for themselves.

All the tools, project files and embedded WEB server source code can be downloaded for free, allowing readers to experiment with the setup for just the purchase cost of the hardware. To keep this cost to a minimum I selected to use the *LM3S6965 Ethernet Evaluation Kit* from Luminary Micro. This kit is not only low cost (USD \$79), but contains all the circuitry required for flash programming and debugging, so does not require an external JTAG interface. Also, the evaluation board is powered through the USB port so an external power supply is not required. The ARM Cortex-M3 core and on chip ROM and RAM provide plenty of resources for the WEB server implementation.

Tool Chain

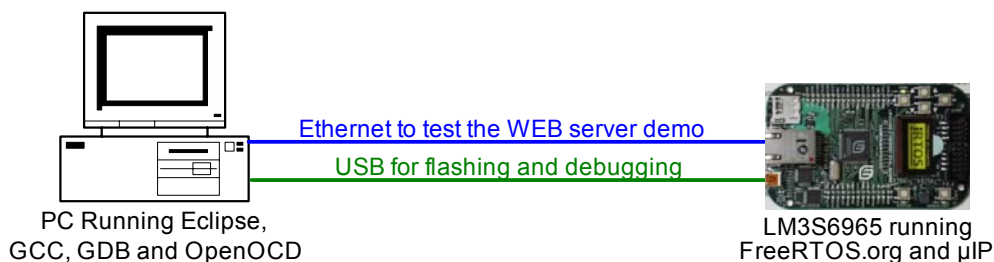


Figure 1 The development setup

The open source development and run time environments used to build and run the WEB server demo are depicted by Figure 1. Full information on the tasks created and pages served by the demo can be found on the FreeRTOS.org WEB site (actual page link provided below).

The next section describes the components needed to build the demo, and the section following that, the components included in the demo application itself. After this I have provided step by step instructions on how the demo can be built and debugged. The article then concludes with a subjective assessment of the development environment.

Links:

http://www.luminarymicro.com/products/lm3s6965_ethernet_evaluation_kit.html [evaluation board]
<http://www.FreeRTOS.org/portlm3sx965.html> [existing FreeRTOS.org WEB server demo]

Required Components

This section describes the tools that are required to build the embedded WEB server example. The links provided are those that I found to be the most convenient.

Eclipse

Eclipse is an open source software framework, originally created by IBM and now managed by the Eclipse Foundation. Its extensible nature has led to a diverse community of Eclipse users. Of interest to embedded developers is the CDT (C/C++ Development Tools) project which adds C and C++ project structure and syntax recognition to the Eclipse Workbench – turning Eclipse into a familiar looking embedded development environment. The Yagarto WEB site provides a download that includes both the Eclipse Workbench and a special cross development version of the CDT as a convenient single executable installation package. A link to this download can be found on the Yagarto home page. Note that the use of the Cortex-M3 processor means that, at the time of writing, the other tools available from the same page are not yet suitable for the demo application.

Eclipse is primarily written in Java so requires the Java Runtime Environment (JRE) to execute. Most computers will already have the JRE installed. To check if this is the case for your computer, open a command prompt and type “java -version”. If the command is not recognised then you will need to download and install the JRE from Sun Microsystems prior to installing Eclipse. I successfully used Java version 1.6.0_01.

Links:

<http://java.sun.com/javase/downloads/index.jsp> [JRE installation - install first]

<http://www.yagarto.de> [Eclipse and CDT installation]

GCC Compiler and GDB Debugger

The Eclipse Workbench provides a visual interface that is used for software development in a number of languages. It provides a graphical interface to the compilation and debug tools, but does not include the compilation tools themselves – these must be obtained separately.

At the time of writing CodeSourcery are the only providers of pre-built GCC tool chains that include support for the Cortex-M3 core, and are therefore suitable for the demo application. The “ARM EABI Sorcery G++ Light Edition” version is available as a single convenient Windows installation package.

Make sure that the directory in which the GCC binaries are installed is included in your PATH environment variable. To check this is the case, open a command prompt and type “arm-none-eabi-gcc.exe --version”. If the command cannot be found then your PATH environment variable needs to be updated.

Links:

http://www.codesourcery.com/gnu_toolchains/arm/download.html [ARM EABI GCC tools]

The Demo Application

The demo application zip file includes an OpenOCD executable, µIP source code, the FreeRTOS.org source code, the demo source files themselves, and a makefile allowing the whole project to be built. These components are described in the next section.

Links:

<http://www.FreeRTOS.org/CortexEclipseExample.html>

Demo Application Components

This section describes the components included in the demo application zip file.

FreeRTOS.org

FreeRTOS.org is a popular open source mini real time scheduler that has been ported to many 8, 16 and 32bit architectures. At the time of writing the FreeRTOS.org source code is downloaded more than 1000 times a week.

Two commercial variants are also available: OpenRTOS is a commercially licensed and supported version of FreeRTOS.org. SafeRTOS is a version that has been certified by TÜV SÜD as having been developed in accordance with the requirements for use in IEC 61508 SIL 3 projects. The confidence gained by the rigorous certification effort has also directly benefited FreeRTOS.org and OpenRTOS users.

Links:

I have included the FreeRTOS.org source code in the demo application zip file so nothing need be downloaded from the following links:

<http://www.FreeRTOS.org>

<http://www.SafeRTOS.com>

<http://www.HighIntegritySystems.com>

OpenOCD

OpenOCD (Open On Chip Debugger) is a software utility that interfaces GDB with the LM3S6965 Evaluation Board. In effect, it converts commands received via a TCP/IP connection from GDB to USB commands that are sent to and understood by the LM3S6965 debug system. OpenOCD can also be used to programme the microcontroller flash memory.

OpenOCD was written by Dominic Rath, and extended for use with the Cortex-M3 core by Magnus Lundin.

Links:

I have included a pre-built OpenOCD executable in the demo application zip file so nothing need be downloaded from the following link.

<http://openocd.berlios.de/web/>

µIP – Embedded TCP/IP stack

µIP is a TCP/IP implementation written by Adam Dunkels for use in memory constrained embedded systems. It is easy to use and easy to configure.

µIP saves on RAM usage by sacrificing data throughput. It does this by only permitting one packet to be outstanding on the network at any one time – delays in the acknowledgement of this packet being the cause of the low throughput. This limitation is most noticeable when communicating with desktop computers¹ and less so when communicating with other µIP nodes. The data throughput is adequate for most human interaction embedded systems making µIP a good choice in most cases. lwIP can be used in place of µIP where a higher throughput is required.

¹ Desktop TCP/IP implementation generally attempt to minimise network traffic by deliberately delaying prior to acknowledging any received packets. This is done to see if any further packets arrive within a short space of time, permitting the multiple received packets to be acknowledged together.

Links:

I have included the µP source code in the demo application zip file so nothing need be downloaded from the following links.

<http://www.sics.se/~adam/uiip/>

<http://savannah.nongnu.org/projects/lwip/>

Building, Downloading and Executing the WEB Server Demo

Preliminaries

Using the links provided above you should now have the Eclipse and GCC tools installed, and obtained the demo application zip file.

This section takes you through the steps to build, download and debug the example using the Eclipse Workbench – with the numbered items indicating the sequence of steps required.

Opening the Demo Project

1. Unzip the demo application files into a convenient location, which for the purposes of this article I shall assume is `c:\demo`. Take care when extracting the files that the directory structure is maintained. After extracting the files `c:\demo` should contain two further directories called `.metadata` and `RTOSDemo`.

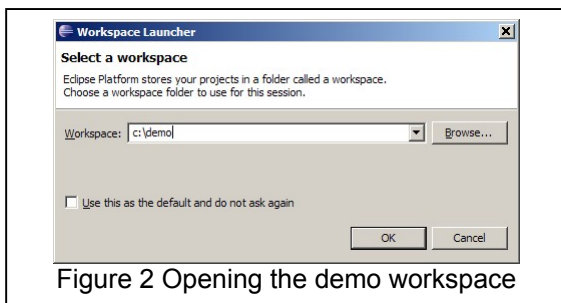


Figure 2 Opening the demo workspace

2. Start Eclipse - a shortcut will probably have been placed on your desktop.
3. You will be prompted to select a Workspace – this is the location in which the project information is stored. Enter `c:\demo` (or whichever directory the files were unzipped into) as per Figure 2. The Eclipse Workbench will then open.

Perspectives

An Eclipse 'perspective' is a collection of views and editors – effectively defining how data is presented.

The demo project uses two perspectives - the C/C++ Perspective and the Debug Perspective. The two buttons in the top right corner of the main window can be used to switch between the two (Figure 3).

The C/C++ perspective is used for project management and editing, and the Debug perspective for controlling debug sessions. These two perspectives present information in a format that should be familiar to users of commercial development environments.

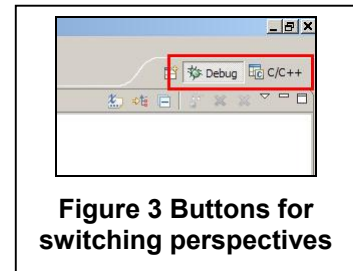


Figure 3 Buttons for switching perspectives

Project Configuration

Eclipse Workbench is in effect a graphical user interface to external tools. This is why I have avoided calling it an IDE (Integrated Development Environment). Project configuration in this case is the definition of how Eclipse will interact with a collection of external (and therefore not integrated) components. We have already seen that Perspectives define how the data output by these tools is displayed.

The flexibility offered by the Eclipse environment can make knowing where to start when creating a new project complex for a novice user. The volume of information available on the internet can also seem overwhelming. Again I found the information on the Yagarto WEB site a particularly useful guide to the project creation and configuration process.

The project included in the demo application zip file is already configured to build using the CodeSourcery GCC compiler and Make utility, interact with the LM3S6965 evaluation board using the CodeSourcery GDB debugger and OpenOCD, and to display information using the perspectives already described.

Links:

<http://www.yagarto.de/howto/yagarto2/index.html> [Information on creating a new embedded project within the Eclipse Workbench]

Building the Demo Application

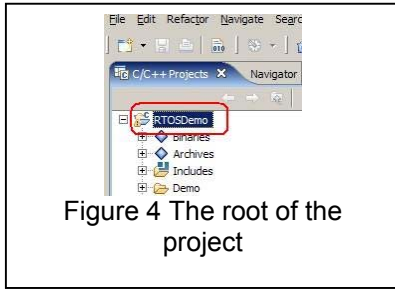


Figure 4 The root of the project

4. Open the C/C++ perspective.
5. The IP address of the embedded WEB server must be compatible with the network in which it is to be placed. The IP address is set by the constants `uiIP_ADDR0` to `uiIP_ADDR3` in the file `Demo\CORTEX_LM3S6965_GCC\webserver\uiIP_Task.c`.
6. Right click on the RTOSDemo project in the C/C++ Project window (Figure 4), then select "Build Project" from the resultant pop up menu.

The build output is sent to the Console pane (Figure 5) at the bottom of the C/C++ perspective. Build errors and warnings can be viewed by selecting the Problems tab – although the project should build without either.

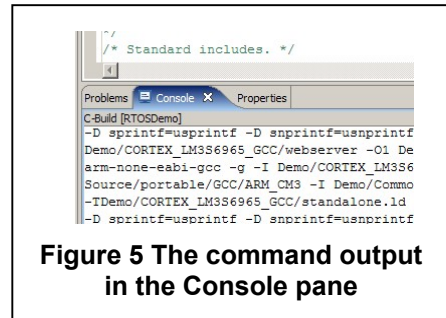


Figure 5 The command output in the Console pane

Programming the Microcontroller Flash

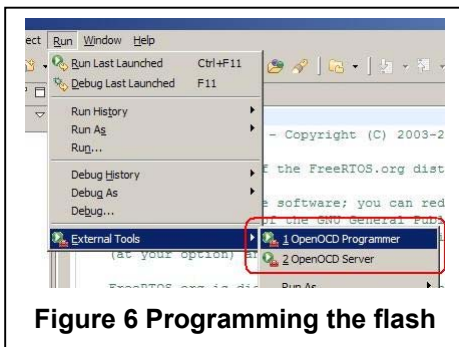


Figure 6 Programming the flash

7. Connect the LM3S6965 evaluation board to your computer using the USB cable provided in the kit. If this is the first time the board has been connected then you will be prompted to install several USB drivers – each of which can be found on the CD that accompanies the evaluation board.
8. Select "OpenOCD Programmer" from the "Run | External Tools" menu (Figure 6). This calls the OpenOCD executable with the correct configuration to program the microcontroller flash – again the output of the command can be viewed in the Console pane.

Starting a Debug Session

9. Start OpenOCD by selecting "OpenOCD Server" from the "Run | External Tools" menu. This again calls the OpenOCD executable, but this time telling it to start listening for a connection from GDB.
10. Switch to the Debug perspective (Figure 3).
11. Click the little bug speed button (Figure 7) to start a GDB session, causing GDB to connect to the evaluation board through the OpenOCD server, then break at the start of `main()`.

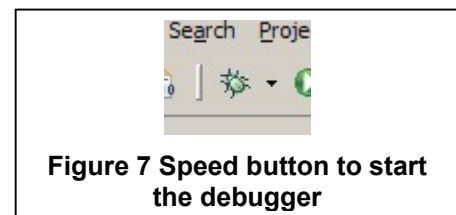


Figure 7 Speed button to start the debugger

Controlling a Debug Session

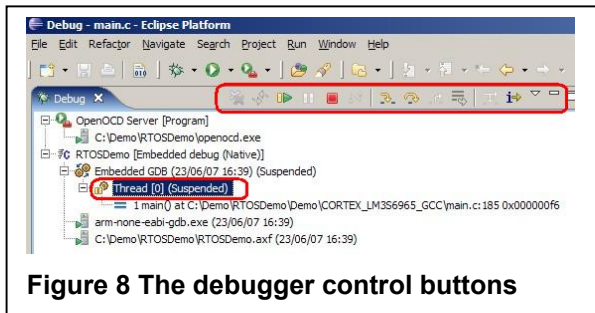


Figure 8 The debugger control buttons

At this point Eclipse has been used to start two external tools, OpenOCD and GDB. The two active tools can be seen in, and controlled from, the Debug window. Selecting one of the tools within this window has two effects: First, the output of the tool can be viewed in the Console pane, and second, context specific control buttons appear within the Debug window itself. Figure 8 shows the GDB session thread as the selected tool and the GDB specific buttons being displayed. These buttons are those that would normally be available to control a debugger: Run, Stop, Step over, Step into, Step out of, etc.

Continuing and Pausing Execution

To start the demo running, select the GDB program in the Debug window as per Figure 8, then click the 'Go' green arrow button. To pause the demo again, select the Halt button. Refer to the link provided in the introduction for information on the demo application functionality.

Setting Break Points

Break points can be set from within the editor pane itself - simply double click the grey area to the left of the line on which the break point is to be placed.

Viewing and Editing Variables

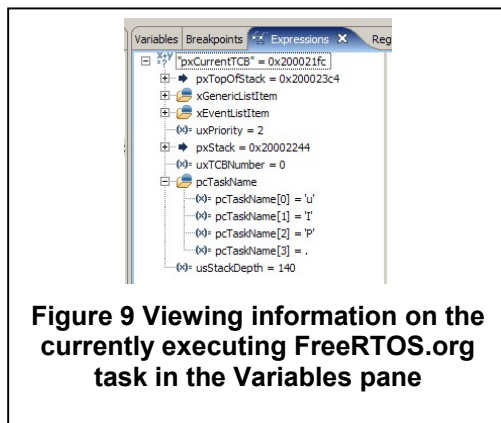


Figure 9 Viewing information on the currently executing FreeRTOS.org task in the Variables pane

The Variables pane can be used to 'watch' variables in the expected manner. For example, the demo application creates more than 20 real time tasks, to see which task is executing at any particular time, add the pxCurrentTCB variable to the watch list (Figure 9).

Terminating a Debug Session

To terminate the debug session you simply select the GDB application in the Debug window (Figure 8) then click the red Terminate button. Similarly, to terminate the OpenOCD server (which must be done prior to a new binary being programmed into the LM3S6965 flash), first select the OpenOCD tool in the Debug window and then click the Terminate button.

Conclusions

I have presented my first attempt at converting an existing FreeRTOS.org project to be built, downloaded and debugged using the Eclipse workbench and OpenOCD. How would I rate the result?

To answer this we need a point of reference, a commercial IDE to which Eclipse can be compared. CrossWorks is a reasonably priced IDE that also uses the GCC compilation tools. It is therefore a suitable comparison candidate.

















Category	Opinion	
	Eclipse	CrossWorks
Installation		
Configuration		
Ease of use		
Support		
Robustness		
Performance		
Documentation		
Overall experience		

Table 1 Eclipse / CrossWorks comparison

Installation

CrossWorks installs all the required IDE and compilation tools from a single installation executable.

Setting up an Eclipse tool chain required the installation of components from several sources. The installation process itself was straight forward, but only once I had identified the best (or at least my preferred) installation packages. Now I have identified these packages future upgrades should not pose any problems.

Configuration

CrossWorks is designed to be, and only be, an embedded development environment. As such the configuration required is minimal and limited to ensuring that the correct chip or board support installation package is obtained.

Eclipse is a generic tool, and must first be configured to behave as an embedded C development system. As per installation – the configuration process turned out to be straight forward – but only following the location of good and concise instructions. As a novice user I found the volume of information available required an investment in time in order to identify the information that was relevant to me.

Ease of Use

Project Definition

CrossWorks provides Wizards to facilitate the creation of new projects. The build setup is created automatically following your selection of the target device. Different build configurations (for example debug, release, run from ROM, run from RAM, etc.) can be selected during project creation, a very powerful feature although subsequent editing of these configurations can be confusing at first (for those of us that never read the documentation that is).

The Eclipse based demo project is configured as a 'standard make' project. Using Eclipse in this manner does not remove the requirement for the generation of an appropriate makefile and linker script. It just means that the project can be built using a button from within the Eclipse environment rather than by typing the appropriate commands within a command shell. Building from within Eclipse does however mean that errors and warnings generated by the build process are presented in a user friendly, easily navigated, way.

The alternative method of using Eclipse is to use 'managed make' in place of a 'standard make'. Managed make project creation is closer to the CrossWorks process – with a wizard guiding you through the steps of creating multiple configurations. However, when creating such a project I ran into difficulty in defining which files within the directory tree I wanted to be included in the build – so for the time being preceded with standard make projects only.

Files Management

My only real problem with Eclipse/CDT is with the way it manages the files included within a build project – even when using the standard make method. The Eclipse project location must be at the root of the project directory tree. This fact makes its use impractical for (probably) all the professional projects I have worked on – primarily because it severely limits how the files included in your project can be structured. In particular it makes sharing common code among multiple projects difficult without having multiple copies of the common file (undesirable).

For example, the primary FreeRTOS.org download includes ports and demos for all the officially supported FreeRTOS.org ports. It is organised as follows:

```
FreeRTOS
|
|--Demo
|   |
|   |--Common      The demo application files that are used by all the ports.
|   |--Dir x       The demo application build files for demo x
|   |--Dir y       The demo application build files for demo y
|
|--Source          The FreeRTOS.org kernel code common to all ports
|   |
|   |--Portable    Processor specific code.
```

So the demo project makefile, for example, the LPC2106 is found in the `FreeRTOS/Demo/ARM7_LPC2106_GCC` directory. This includes some source files that are common to all the demos (not just the LPC2106 demo) from a directory that is higher than the location of the makefile, the `FreeRTOS/Demo/Common` directory. It also includes the FreeRTOS.org kernel source file from a directory that is again higher than the directory that contains the makefile, the `FreeRTOS/Source` directory.

The current CDT version cannot use this directory organisation. It can only locate source files (for example when debugging or when clicking on a build error) that are below it in the directory tree and therefore the Eclipse project must be in the FreeRTOS root directory. This is not acceptable for a source tree that is to be built with many different compilation tools and shares common files amongst many different projects.

Having said all that, and as I eluded at the top of this article, I have been told that this barrier to usage will be removed in version 4 of the CDT. Version 4 permits a project to create links to source files that are outside of the directory tree, facilitating the use of maintainable project structures. I look forward to testing the embedded development version in anticipation that it will provide an effective interface for FreeRTOS.org users.

Navigating to files within the Workbench can also be laborious as its 'navigation' pane does not offer any options other than to view the files in the directories in which they appear on the hard disk. Contrast this to the CrossWorks project interface, which permits me to select files from anywhere on my hard disk and permits their presentation in logical categories.

Support

It is a trait of popular open source projects that there are numerous enthusiastic people willing to offer fast and practical advice – and this was my experience in this case. Active forums, news groups or mailing lists exist for all the components used to create and build the WEB server demo.

Robustness

CrossWorks has proved to be a very reliable tool. The IDE and debugging interfaces are tightly coupled, fast and reliable.

Overall I also found the open source tools reliable and usable, but there were some exceptions as detailed below.

Eclipse

The only, but very significant, problem I encountered with the Eclipse Workbench itself occurred when attempting to abort the execution of an external tool that had been launched through the Eclipse interface. This occurred on a couple of occasions – once when attempting to shut down OpenOCD before it had fully started, and once when attempting to abort a build that was using the cs-make utility (CodeSourcery make). On both of these occasions the external tool neither completed its task nor shut down, but instead entered a zombie state. Once this state had occurred attempting to shut down Eclipse resulted in a major system crash.

OpenOCD

On occasion, probably 10% of the time, OpenOCD failed to stop the target at main() when commencing a debug session. This was easily rectified by simply stopping the debugger and starting the debug session again – a very quick process.

OpenOCD also output several warning messages to the Console when used for both flash programming and debugging. These did not seem to be detrimental to its function, and through the excellent support received from Dominic Rath and Magnus Lundin I was able to ascertain that these warnings were benign.

It should be noted that at the time of writing the Cortex-M3 extensions to OpenOCD were not mature. Since this time they have been incorporated into the main OpenOCD product but remain less mature than the ARM7 equivalent.

Care must be taken that any firewall software installed does not block the internal TCP/IP connections between GDB and OpenOCD. The first time I started OpenOCD I received a warning from the Windows firewall. When this warning had been cleared I needed to close then reopen OpenOCD in order to get a connection.

Performance

Running on a 1.7GHz Pentium with 1GByte of RAM I found the performance of the Eclipse Workbench to be more than adequate.

OpenOCD both started up and programmed the microcontroller flash quickly.

Documentation

Well there is certainly enough of it – although the context specific help specific to the CDT configuration was lacking in some places.

Overall Experience

My experience of setting up and using this tool chain has been positive. The only concern being the project file management – a blight that will soon (hopefully) be consigned to history.

As an engineer I liked the control, feedback and freedom that integrating the various tool components provided – but this comes at the cost of a learning curve that will not suit everyone. The time invested in learning the tools is definitely greater than that required for an equivalent commercial product.

Once put together, the tool chain was definitely capable of being used to edit, compile, debug and maintain a project such as the presented embedded WEB server. It is however a long way from being as tightly integrated and slick as tools such as CrossWorks. With an enthusiastic and growing user base Eclipse will undoubtedly continue to improve.

END