Embedded Cross-Development with Eclipse

Macraigor Systems
Complete JTAG Debug Support

This technical whitepaper describes how to construct a free or low-cost cross-development environment based on the open-source Eclipse IDE and GNU toolsets.

Written by Brian Handley, Sr. Engineer of Macraigor Systems

Macraigor Systems LLC • P.O. Box 471008 Brookline Village, MA 02447 • Ph: (617) 739-8693 • Fax: (617) 739-8694 • www.macraigor.com

The Eclipse development environment has become the de-facto industry-standard environment in which to host embedded development tools. Many of the traditional embedded tools vendors who used to sell their own proprietary development tools and environments have embraced Eclipse and ported their products to run within it to take advantage of the sophisticated, feature-rich framework provided by the Eclipse IDE.

These tools and environments are powerful, but they can still be expensive. For projects on a tight budget, it is now possible to use the freely available, open-source Eclipse IDE along with the open-source GNU tools (binutils, gcc and gdb) to construct a complete cross-development environment at little or no cost.

However, piecing together all the necessary components to build a system such as this is not necessarily easy. Eclipse was not originally built to handle cross-development, or even the C or C++ languages typically used in most embedded projects. Therefore, a significant amount of effort is needed to get Eclipse to perform this task adequately. Beyond this, Eclipse does not currently have any concept of a remote debug connection on its own. If a debug interface such as JTAG or BDM—or even an Ethernet or serial connection to a target-resident debug monitor—is used, Eclipse must be reconfigured to handle this situation. Additionally, the required GNU tools are typically available only in source format, and they must be built for the particular host and target processor being used by a project. Getting these tools to build for a particular host-target combination can be difficult, consuming engineering time that could be better spent on application development.

This article describes how to construct a free or low-cost cross-development environment based on the open-source Eclipse IDE and GNU toolsets. The task can be difficult and time-consuming, and this article also presents the preconfigured Eclipse projects and pre-built GNU tools offered by Macraigor Systems to help speed construction.

Required Components

To build a functional, free, cross-development environment, several components must be obtained and integrated together. The Eclipse development environment is the framework into which the other necessary tools are integrated. Eclipse itself includes an editor, project manager and debugger interface. Since the environment is intended for embedded cross-development, the C and C++ languages must be supported. This requires use of the CDT (http://www.eclipse.org/cdt/) plug-in for Eclipse. The assembler, compiler, linker and other code-generation utilities will be provided by open-source GNU code.

If the goal were to develop native applications in C/C++ using Eclipse, then these tools would suffice. However, for embedded cross-development, a few more pieces are needed. Eclipse with the CDT plug-in has no concept of using a remote debug connection in order to connect to an embedded processor. Zylin AS Consulting (www.zylin.com) offers the open-source Embedded CDT and another plug-in, which together allow the Eclipse debugger to connect to a remote target via any debug connection (see the Zylin Plug-Ins section later in this article). This debug connection is typically a JTAG, BDM, Ethernet or serial connection. In addition, if a JTAG or BDM connection to the target is needed, a method must be provided for the GNU Project Debugger (GDB) to communicate to the target using these interfaces.

The completed development system using a JTAG/BDM target connection is shown in Figure 1.

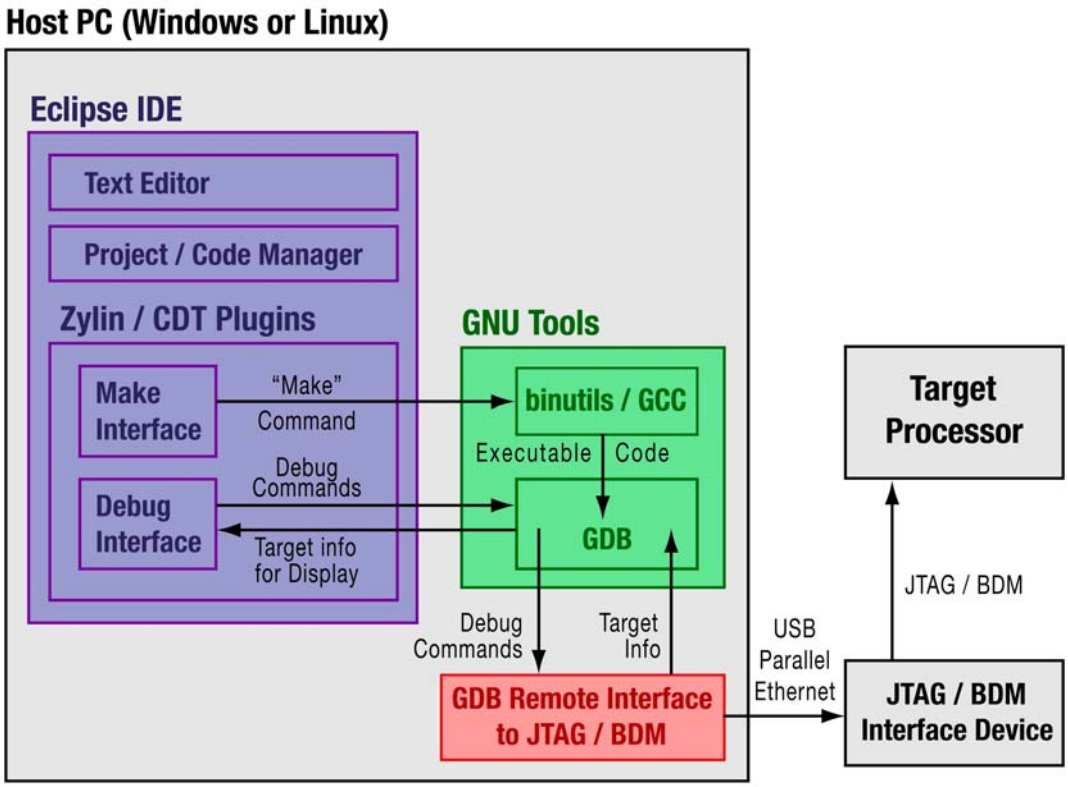

Figure 1 Completed Development System Using a JTAG/BDM Target Connection

A discussion of each of the above required components follows.

Eclipse

According to the official Eclipse Web site (www.eclipse.org), the Eclipse Foundation manages open-source development of “projects [that] are focused on providing a vendor-neutral open development platform and application frameworks for building software.” The Eclipse Foundation has created the Eclipse Platform, which provides a feature-rich integrated development environment with a well defined interface that allows additional features to plug in and work seamlessly with the existing code.

Eclipse has rapidly gained favor among embedded tools companies because it provides a sophisticated IDE into which they can plug their tools and no longer have to worry about building and maintaining their own proprietary environments. In addition, the Eclipse Public License allows the creator of derivative works based on Eclipse to retain their distribution rights, so companies can focus on their core embedded competencies and still profit from their efforts.

This is excellent news for those trying to put together a free development environment. It makes available a commercial-quality IDE that has the backing and support of a large number of both embedded and enterprise-software tools companies. However, as mentioned earlier, Eclipse by itself provides only a framework and some generic tools, such as an editor, code/project manager and debugger interface. In order to construct a cross-development system, several more packages must be obtained and integrated into Eclipse.

CDT

Eclipse was originally developed in and for the Java programming language, and the basic framework is still specific to the Java language. Most embedded cross-development projects still have device drivers, operating system (OS) code and applications written in C, C++ or assembly language. To make Eclipse compatible with, and useable for, C/C++, a sub-project called C/C++ Development Tooling was created to build a plug-in that would add these features to the basic Eclipse framework. This plug-in is available as a free download from Eclipse at http://www.eclipse.org/cdt/downloads.php.

With the Eclipse framework and CDT in place, the environment is capable of supporting and enabling code development in C/C++. However this environment will only work for native application development. For embedded cross-development, there are still some issues that must be addressed, primarily the handling of remote debug connections to a target processor. The Zylin Plug-Ins section discusses this problem further. In addition, an assembler, compiler, linker and loader are still required for the specific target processor that will be used on

the project. The next section discusses using GNU tools that provide these utilities.

GNU Tools

The Free Software Foundation makes available free source code for a wide range of programs and utilities, including a set of tools that together can provide everything necessary to build, link, load and debug an embedded application. The combination of GNU binutils (http://www.gnu.org/software/binutils/), the GNU Compiler Collection (GCC) (http://gcc.gnu.org/) and the GDB (http://sources.redhat.com/gdb/) provide a fairly complete toolset for building and debugging embedded applications. These tools can be used on their own for this purpose. The binutils package provides an assembler, linker, archiver and several other utilities for code development, the GCC provides the C/C++ compiler, and the GDB allows the code to be downloaded to and debugged on the target processor.

The downside of using these tools as they are is that there is no graphical user interface (GUI) and no real integration of the tools. Used by themselves, the tools basically provide a command-line interface. However, Eclipse with the CDT is capable of sufficiently integrating these tools into an environment with a GUI so most of the command-line use of the tools can be avoided.

The other problem with using the GNU tools for code development is that these tools are generally provided only in source form. Although the tools support a huge array of various target processors and just about any host OS and hardware one could imagine, the user usually has to configure and build the tools. This build process can be a time-consuming, frustrating experience, especially for someone who has not done it before. Fortunately, Macraigor Systems provides an alternative to the labor-intensive task of building GNU tools in-house. This is covered later in this article.

Zylin Plug-Ins

An environment consisting of Eclipse, the CDT plug-in and the appropriate GNU tools is close to being a functional embedded cross-development system. As mentioned above, however, Eclipse and the CDT do not support remote target connections to an embedded processor. They assume that debugging is occurring on the host machine. In order to download the embedded code from the host to the target and then connect to a debug agent of some type running on the target hardware, some changes must be made in the way the CDT handles debugging.

To address this problem, Zylin AS Consulting, a Norwegian company, has created and made available Embedded CDT and another small plug-in that

together “understand” and properly handle embedded debugging using the GDB from within Eclipse. These free plug-ins are available at http://www.zylin.com/embeddedcdt.html.

An Eclipse Project called the Device Software Development Platform (DSDP) is now available. This project is specifically aimed at enabling Eclipse to be used for embedded cross-development so that, at some point in the future, the Zylin Embedded CDT modifications may become unnecessary. Further information about the DSDP project can be found at http://www.eclipse.org/dsdp/.

Target Connection

The only missing piece that remains in the integrated cross-development system is some type of debug communication method to connect the host computer to the target processor. Traditionally, this connection is usually made via a serial, Ethernet or JTAG/BDM interface. If a project is using hardware for which a board support package already exists, it may be feasible to simply run a GDB debug agent on the target and connect the GDB to it using a serial or Ethernet connection.

However, for new custom target boards, the interface of choice is usually JTAG or BDM. These types of debug interfaces are built into most of the more popular embedded processors including ARM, MIPS, PowerPC and XScale processors, as well as many others. These interfaces provide a dedicated debug connection directly to the target processor that has several advantages over using a serial or Ethernet connection:

- They are built into the processor and typically only require that the processor is powered and is getting a clock signal in order to work.
- They can be used to write and debug boot code and drivers.
- They do not use any valuable target resources. A serial or Ethernet debug connection usually requires dedicated hardware for the interface in addition to using processor cycles and memory for a driver.

To connect a JTAG or BDM interface device to the target using the Eclipse/GNU environment described above, a debug agent of some sort is required. The GDB has a well defined back-end interface called GDB Remote that has become a common standard for connecting the debugger to an embedded processor via a JTAG or BDM connection. This is usually handled by a proprietary stand-alone utility that runs on the host and provides a TCP/IP port that will accept a GDB connection on the front end and connect to the JTAG/BDM hardware device on the back end. Most vendors of JTAG/BDM interface devices provide a utility for this type of connection with their hardware.

Figure 2 shows the Eclipse Debug Perspective during a debugging session using the system described in this article. The target is a Freescale MPC8280 evaluation board and the debug connection is made via a Macraigor Systems USB JTAG device.

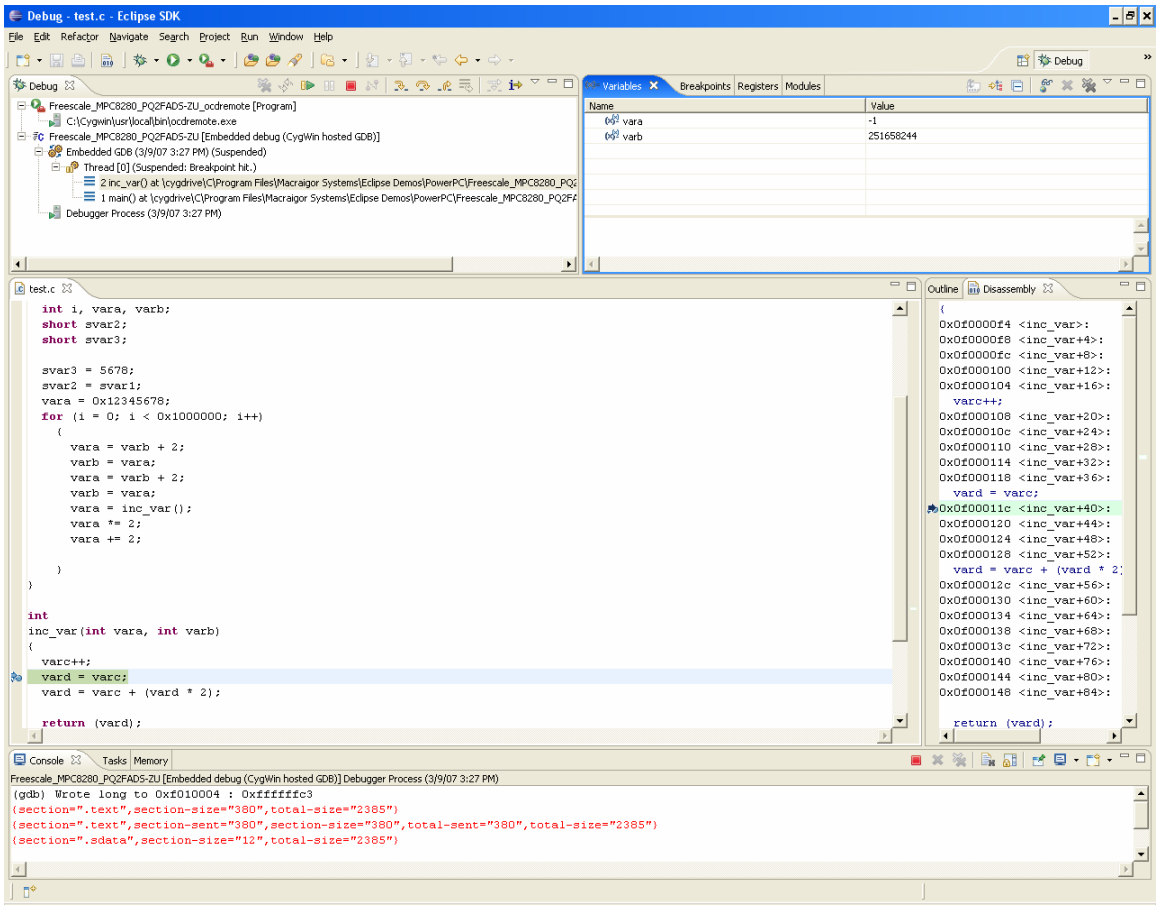

Figure 2 Eclipse Debug Perspective During a Debugging Session

An Easier Way

This article has shown that, using readily available, free, open-source software tools, it is possible to construct a full-featured, integrated environment for embedded cross-development. The process of gathering components, integrating them and—in the case of the GNU tools—building applications from source is likely well within the capabilities of most embedded software engineers. However, this process can still be a time-consuming and difficult task, eating into engineering time that might be better spent writing code for the target hardware.

Macraigor Systems, as a way of promoting and enabling its JTAG/BDM interface devices, has greatly simplified the task of constructing the integrated development environment described in this article. Macraigor Systems has recently made available free downloads of a suite of tools that includes Eclipse, the Zylin Embedded CDT plug-ins, pre-built GNU tool kits for AMDx86, ARM, MIPS, PowerPC, XScale, ColdFire/CPU32, and Freescale DSP target processors and more than 70 example Eclipse projects configured for standard evaluation boards using various embedded processors. Also available is a document containing detailed instructions on downloading, installing and testing the environment with actual target hardware.

The preconfigured Eclipse projects and the pre-built GNU tools, with install programs for Windows OSs and RPM scripts for Linux OSs, allow a user to quickly get the complete environment up and running on actual hardware. More information about these kits and links for downloading the various components is available at http://www.macraigor.com/Eclipse/.

Macraigor Systems also offers a utility called OCD Remote, which provides a connection from the GDB to Macraigor Systems' JTAG/BDM hardware so the Eclipse/GNU environment can be used with any of the company's interface devices.

In conclusion, constructing a free or low-cost cross-development environment based on the open-source Eclipse IDE and GNU toolsets is possible, though time-consuming and very challenging. Macraigor Systems has simplified the process by providing downloads that help facilitate construction and lower the barriers to achieving the sophisticated cross-development environment developers are seeking today.