Linux Development with Eclipse

Manual

REV. 1.2

EMAC, inc.
EQUIPMENT MONITOR AND CONTROL
2390 EMAC Way, Carbondale, IL 62902
World Wide Web: http://www.emacinc.com
(618) 529-4525 FAX: (618) 457-0110

Table of Contents

Disclaimer

EMAC Inc. does not assume any liability arising out of the application or use of any of its products or designs. Products designed or distributed by EMAC Inc. are not intended for, or authorized to be used in, applications such as life support systems or for any other use in which the failure of the product could potentially result in personal injury, death or property damage.

If EMAC Inc. products are used in any of the aforementioned unintended or unauthorized applications, Purchaser shall indemnify and hold EMAC Inc. and its employees and officers harmless against all claims, costs, damages, expenses, and attorney fees that may directly or indirectly arise out of any claim of personal injury, death or property damage associated with such unintended or unauthorized use, even if it is alleged that EMAC Inc. was negligent in the design or manufacture of the product.

EMAC Inc. reserves the right to make changes to any products with the intent to improve overall quality, without further notification.

1 Introduction and Installation

Eclipse is a full-featured Open Source IDE that can organize the development of software for use on EMAC products such as the Server In a Box (SIB), Single Board Server (SBS), iPac, and System on Module (SoM) product lines. Eclipse is a Java based IDE which can be used to develop applications in many languages and environments.

This document details the installation of the EMAC Eclipse Distribution, compiling and running applications for the EMAC SIB/SBS and SoM, and remote debugging using GDB and GDBServer using both the Linux shell and Eclipse. This document applies specifically to Eclipse version 3.2 with Sun Java 2 version 1.5.0 running on a Debian GNU/Linux system, but is not Debian specific and should describe operation on other versions of Linux as well. Operation should be similar with more recent versions of Java as well. Some familiarity with the Linux environment and file system are assumed.

1.1 Installing Java

Because Eclipse depends on Java, ensure that a suitable version of Java has been installed before using Eclipse. To maintain compatibility with Eclipse and various plugins, EMAC recommends installing Java 2 version 1.5 or higher. EMAC has tested up to version 1.5.0_10.

Minimally, a Java Runtime Environment (JRE) is required for Eclipse. However, a Java Software Development Kit (JDK) may be required depending on your intended use of Eclipse. See the Eclipse page at http://www.eclipse.org/ and the Sun Java page at http://sun.java.com/ to determine which Java package best suits your needs.

Before downloading and installing a new version of Java, check to see if a suitable version is already present on the machine. The easiest way to do this is by issuing the command `java -version` at the shell. Please note that the GNU interpreter for Java bytecode (GIJ) was not fully compatible with Eclipse at the time of testing.

There are several ways to install a new version of Java. You may install a new version directly from Sun Microsystems at http://sun.java.com/. Also, a redistributable version of the JRE is located on the EMAC distribution CD under Development_Kits/EMAC_Open_Tools/Linux, which is known to work with the EMAC Eclipse Distribution. Download the package required for your platform, unpack it, and place a link to the executables in your path if necessary. Refer to the Java installation instructions available on the Java website and in the downloaded package for more information on installing Java.

Finally, test the Java installation with the `java -version` command to ensure that it has been installed properly.

1.2 Installing Eclipse

Following a successful Java installation, the Eclipse IDE must be installed on your machine. EMAC recommends that you install version 3.2, the most recent stable version as of this testing. This will prevent problems due to untested versions and ensure that the operation of Eclipse matches the instructions in this document. Versions earlier than Eclipse 3.2 are not compatible with the EMAC

SDKs. Follow the steps below to install the EMAC Eclipse Distribution. See Appendix A for instructions regarding manual installation and configuration of Eclipse and Eclipse plugins.

The EMAC Eclipse 3.2 Distribution is available on the distribution CD under Development_Kits/EMAC_Open_Tools/Linux, or you may download version 3.2 (or the latest tested version) of the EMAC Eclipse Distribution from the EMAC FTP site at ftp://ftp.emacinc.com/PCSBC/Development_Kits/EMAC_Open_Tools/Linux/. The EMAC Eclipse Distribution also includes all of the recommended plugins for developing software using EMAC's SDKs. These include the C/C++ Development Tools (CDT), Astyle Eclipse Code Formatter for CDT, Remote Systems Explorer, and the Target Management Terminal SDK.

The EMAC Eclipse Distribution installation process is as follows:

1. Download the EMAC Eclipse Distribution archive file from the CD or FTP site.
2. Uncompress and extract the archive to any location that you would like to install it into. This can be a system wide installation, such as in `/usr/local`, or a local installation somewhere in your home directory for example.
3. Upon extracting the file, there will be a single directory titled `eclipse`. Navigate to this directory and read the file `README-EMAC` for further information about the package as a whole.
4. If you wish to use the Target Management Terminal Plugin, the RXTX Java libraries must be installed into your Java virtual machine (JVM). EMAC provides a custom script to automate this process, or you can follow the official RXTX installation instructions described in the file `INSTALL` under `rxtx-2.1.7-bins-r2`. (Note that the installation script has only been tested on Java 1.5, although it should be valid on other versions). All of the necessary files are located in the `rxtx-2.1-7-bins-r2` directory within the `eclipse` folder. Execute the `rxtx-EMAC-install.sh` script as root from within the `rxtx-2.1-7-bins-r2` directory to install RXTX into your JVM.
5. The Eclipse executable is `eclipse`, and is located under the `eclipse` directory. EMAC also provides a startup script to start Eclipse with a limit of 512MB of virtual memory rather than the default 256MB. Type `./start_eclipse` to execute this script and start Eclipse. Refer to the Eclipse documentation for more startup options.
6. Place a link to the Eclipse executable somewhere in your path to allow for easier use. Depending on your configuration, you may want to modify the `start_eclipse` script to match the needs of your environment.

1.3 Initial Configuration

After installing Eclipse and running it, you will be prompted for a workspace location. Eclipse uses this location to store projects and save information about the state of your projects from previous sessions. Because Eclipse stores this data in the actual workspace, the workspace location must be a valid Eclipse workspace or a path for Eclipse to create a new workspace. Eclipse will not recognize a folder without this data as a valid project. Choose a workspace location and continue.

Once Eclipse loads, the Eclipse welcome screen will be shown as in Figure 1. Navigate through the various options in the top row to become more familiar with Eclipse and its operation.

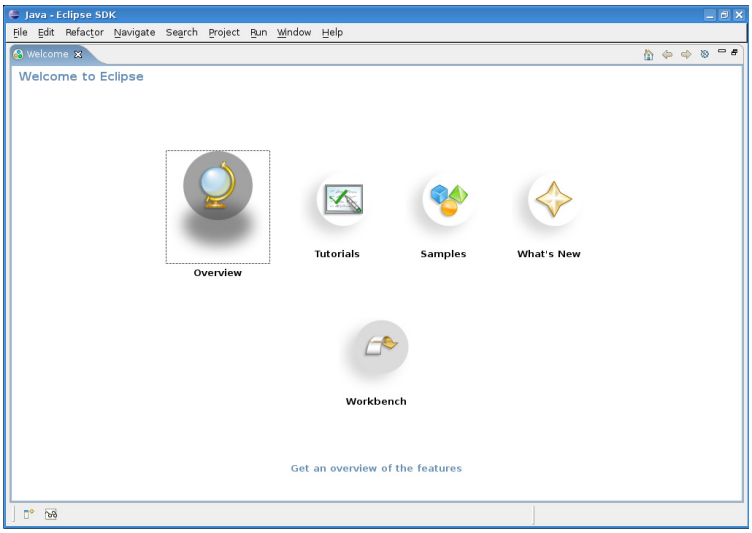

Figure 1: Welcome Screen

The *Workbench* link on the welcome screen will open the Eclipse Java development perspective, shown in Figure 2. The *Java* tab (1) in the upper right hand corner indicates the current perspective. The far left window (2) is used to navigate the current workspace and project, the center window (3) will display any open files, and the far right window (4) will display an outline of the current file or executable. Program output, errors, and other information are displayed in the window at the bottom center of the screen (5).

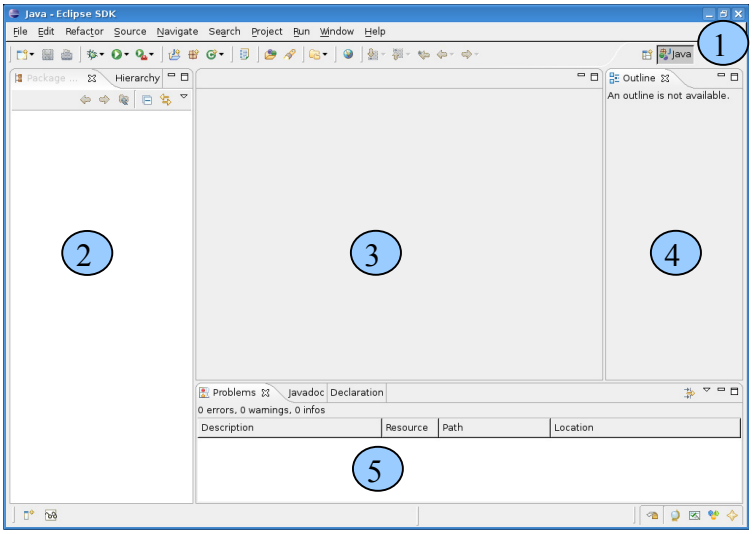

Figure 2: Eclipse Workbench Components

1.4 Installing the CDT Plugin

In order to develop applications in C or C++ through Eclipse, you must first install the C/C++ Development Tools Plugin (CDT). If you installed the Eclipse SDK directly from EMAC, the CDT Plugin is already included. The process of manually installing the Eclipse CDT Plugin is covered in Appendix A. The process for installing the CDT Plugin can also be applied to most other plugins in a

similar fashion. For this reason, users who are new to Eclipse might find it beneficial to read through Appendix A as well.

To open the Eclipse CDT perspective, navigate to the Workbench and attempt to change the Eclipse perspective to C/C++ by clicking the icon next to the *Java* tab in the upper right hand corner and choosing *Other* as shown in Figure 3.

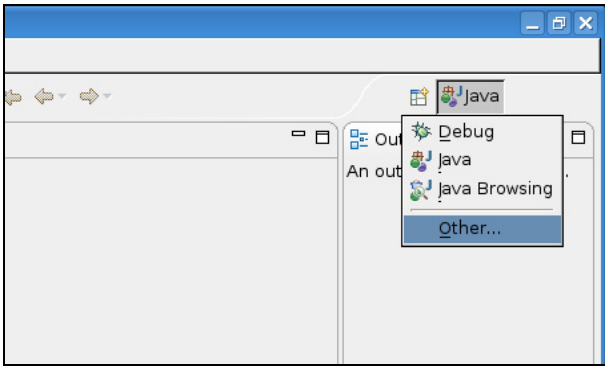

Figure 3: Eclipse Perspective Selection

This will bring up the *Open Perspective* screen. Choose *C/C++* and click *OK*. Eclipse should now switch to the C/C++ perspective, changing the windows accordingly.

1.5 GCC Versions

In order to develop applications through Eclipse, you will first need a compiler. Different versions and configurations of the GNU Compiler Collection (GCC) are required depending on the platform that you are developing for. EMAC provides software development kits (SDKs) containing these cross-compilers and other tools through the EMAC FTP site as well as on the CD-ROM included with your product.

If you are developing for the EMAC SoM-5282, look on the CD under SoM/SoM-5282M /Tools. Run the install script `m68k-elf-tools-20031003` (or newer version) to complete the installation. Following installation, the executables should be located in the directory `/usr/local/bin`. Next, extract the *SoM5282EM-SDK* from the *Tools* directory directly into your Eclipse workspace.

For the EMAC SIB/SBS, the SDK is located on the CD under EMAC_Linux/SIB /EMAC_Linux_3/Eclipse_SIB-SDK. Extract *SIB-SDK-3.0* (or a newer version) directly into your Eclipse workspace.

For other supported EMAC products, check the product specific documentation to determine the location of the toolchain used with your device.

Although they handle libraries differently, the required libraries for the SoM, SIB/SBS, and other products are built into the respective toolchain to ensure that the correct versions of these libraries are used. For the SIB/SBS, these are located under the `SIB-SDK-3.0/gcc-4.1.1-i486-D/sysroot` directory. For the SoM, the installer places these libraries in the `/usr/local/m68k-elf` directory on the host development machine. To ensure compatibility, it is important that the correct libraries are compiled against when developing software.

1.6 Importing and Creating Projects

Having installed the CDT Plugin and the necessary cross GCC version, you are ready to develop projects in C or C++. EMAC provides sample Eclipse projects that will help to familiarize you with the Eclipse development process. These projects are located under the provided Software Development Kit (SDK) for your device as described in section 1.5.

Each EMAC SDK is a valid Eclipse project, and may be imported directly into the Eclipse workspace. First, switch to the C/C++ perspective if necessary. Right click in the white area of the *C/C++ Projects* window in the far left of the screen and select *Import*. This will bring up a new window to specify the import type and location. Expand the *General* list and select *Existing Projects into Workspace* as shown in Figure 4.

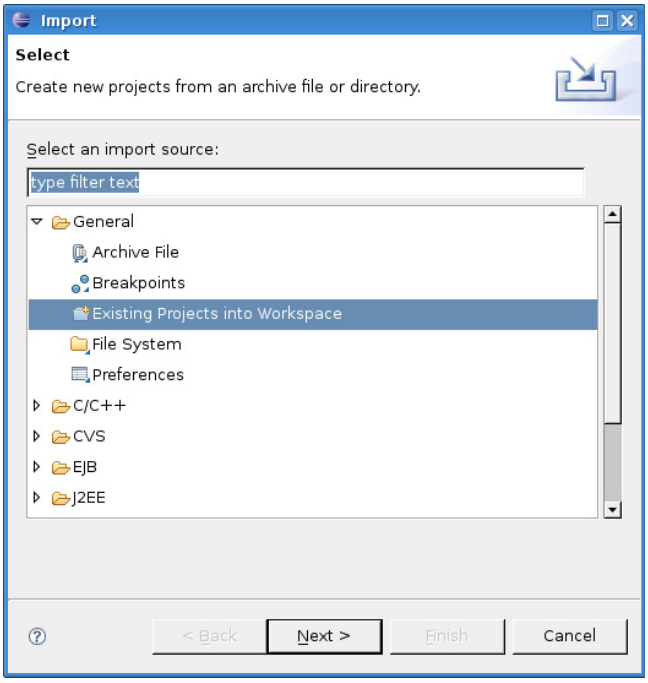

Figure 4: Importing A Project

After clicking *Next*, you will be prompted for the root directory of the project to import. Select the appropriate SDK directory and ensure that it is selected under the *Projects* section. To preserve symbolic links, make sure that the *Copy projects into workspace* option is ***not*** checked before proceeding. An example of these settings is shown in Figure 5.

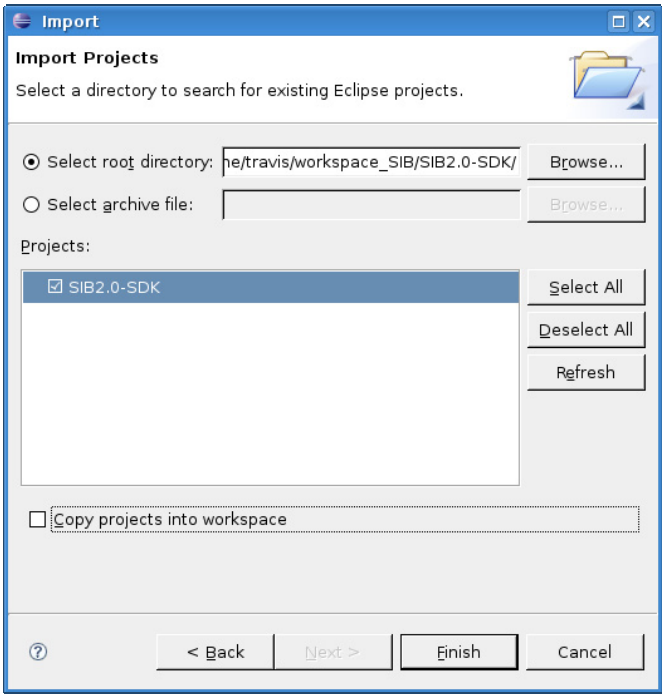

Figure 5: Selecting an SDK to Import

Finally, click on *Finish,* and allow Eclipse to import the project.

When you are ready to develop your own applications, you may create a new project in the existing workspace by right clicking on the white space in the *C/C++ Projects* window and selecting *New.* Selecting *Project* will open the project wizard and allow you to customize the project. Create a new C or C++ *Standard Make Project.*

Once you have successfully imported an SDK, browse the example projects located in the *projects* directory. Double click on a file to open it with the Eclipse editor. Notice the syntax highlighting and outline that Eclipse provides automatically.

1.7 Installing the Target Management Plugin

The Target Management (TM) Eclipse Project provides a plugin called the Remote System Explorer (RSE) and the TM Terminal SDK. These extensions allow for integration of remote systems management and communication into Eclipse and the CDT.

Provided is an Eclipse extension that allows for connections via SSH, Telnet, FTP, and serial terminals along with integrated remote launching and remote debugging support and much more all from within Eclipse. The RSE allows for seamless SSH connections supporting copy and paste operations between the file system on the target board and the local machine as well. The basic RSE plugins and Terminal SDK are provided with the EMAC Eclipse Distribution and require no further installation steps other than installing the RXTX libraries as described in section 1.2.

Instructions in Appendix A detail the steps of installing theses plugins manually if you are not using the EMAC Eclipse SDK.

Browse the RSE documentation to familiarize yourself with its operation. This should be accessible through the *Help* menu under *Help Contents*. The *RSE User Guide* link is located in the left pane of the help window.

1.8 Using the Astyle Eclipse Code Formatter

The Astyle Eclipse Code Formatter Plugin for CDT is also included with the EMAC Eclipse SDK. This plugin provides a method for improving code standardization and readability by automatically formatting code based on several styles or custom settings.

In order to use the Astyle Code Formatter, it must first be selected through the CDT preferences settings. To change these settings, select *Preferences...* from the *Window* menu. A new window should be initiated as shown in Figure 6. Select *C/C++* and then *Code Formatter*. The *Astyle Plugin* should be available as an option on the drop down list. Be sure to press *Apply* after making changes.

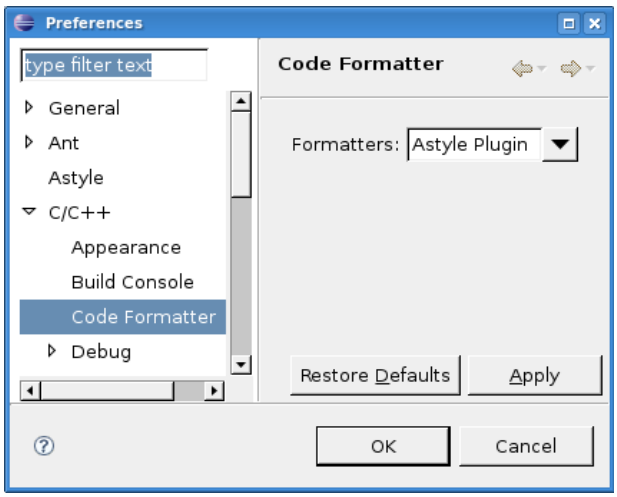

Figure 6: Code Formatter Selection

After selecting the Astyle Code Formatter Plugin, the settings for the code style to use can be customized. Select *Astyle* on the *Preferences* screen, and choose a code formatting style. Additionally, it is possible to customize a style using a template. More information regarding this is available from the Astyle Eclipse project page, which is listed among the links in section 5. Figure 7 shows an example of the Astyle configuration page.

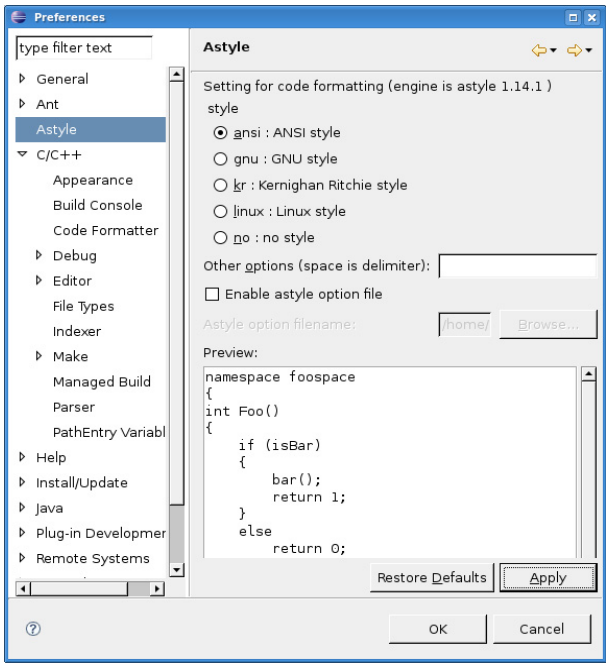

Figure 7: Astyle Plugin Configuration

2 Application Development

Developing applications through Eclipse is efficient and simple once you become familiar with the process. Compiling and building applications through Eclipse relies on makefiles and the `make` command, which simplify the use of different cross-compilers and toolchains. This section will detail the process of compiling and running an application through Eclipse.

Before continuing, ensure that both the Make package and GCC have been installed on the host development machine. For more information on installing these packages, see the links in section 5.

2.1 Makefiles

The projects in the EMAC SDK provide makefiles with targets for compiling, cleaning, and uploading a given application. These makefiles can be easily modified for a new application. To locate a makefile, navigate to one of the example projects within the appropriate SDK. These are generally located in the *projects* directory. Simply double click on the file titled *Makefile* (note the capital *M*) to open it with the Eclipse editor.

Notice that the correct GCC and compiler flags have been written into the makefile. Towards the end of the makefile, you will notice several headings followed by indented lines. Each of these headings is called a target, and performs a specific function.

The `all` target will compile all of the specified source code for the given project using the compiler and compiler flags specified in the makefile. The `clean` target will delete all files created from the `all` target, if they exist. The `upload` target will transfer the specified files to the device as specified by the

`TARGET_IP`, `LOGIN`, and `PASSWORD` entries in the makefile. Editing the `wput` commands will change the location of the files on the target board.

From the shell on the host development machine, issuing the command `make all`, `make clean`, or `make upload` will execute all of the commands in the respective target. For example, issuing the command `make all` followed by `make upload` would first compile the specified source files and then upload the files to the target board.

For more information on makefiles, see the link in section 5.

2.2 Creating and Using Make Targets

Eclipse requires that a *Make Target* be created for each target in the makefile in order to execute the various commands. To do this, first check that your *Make Targets* view is visible by using the *Window* menu and selecting *Show View* followed by *Make Targets*. The *Make Targets* view will be located on the far right of the screen, as shown in Figure 8.

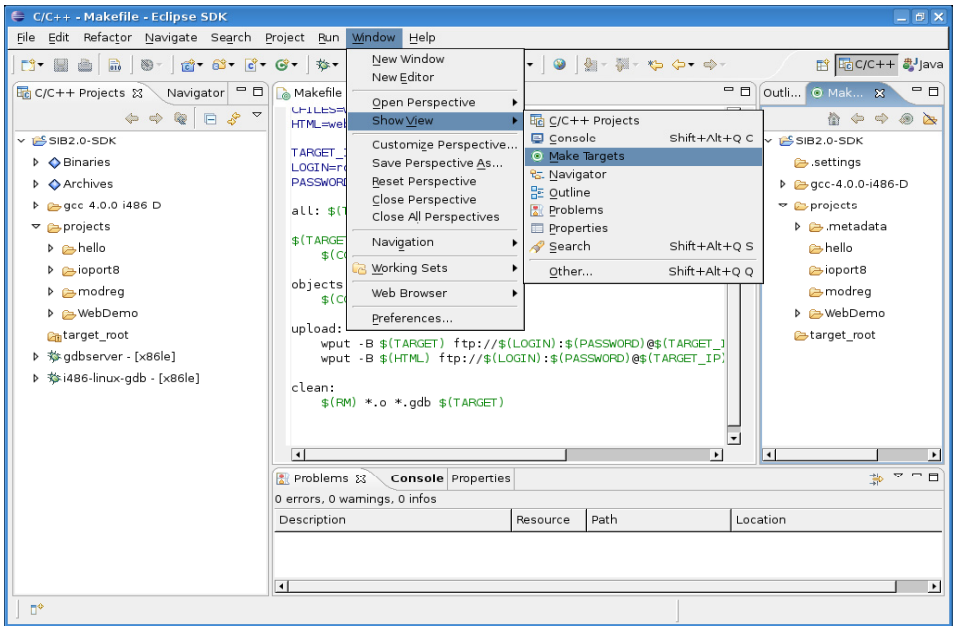

Figure 8: Make Targets View

To create a new *Make Target*, navigate to the project using the *Make Targets* view. Right click on the project and select *Add Make Target*. This will bring up a new window that will specify the name for the target and the command to call. An example is shown in Figure 9.

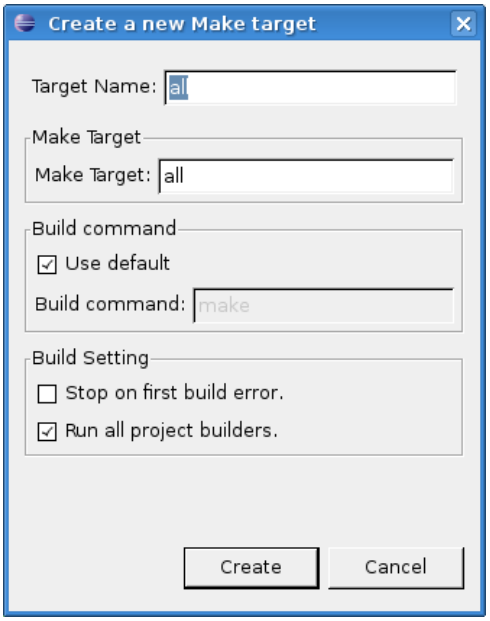

Figure 9: Creating a New Make Target

Target Name specifies the Eclipse name for the target. There are few constraints on this name, but it is easiest if it matches the actual target in the makefile. *Make Target* specifies the target to call in the makefile. For example, the window above will create a *Make Target* called *all* that will call the command `make all` when executed.

After creating the Eclipse *Make Targets*, they will become visible under the project in the *Make Targets* view. For instance, after creating targets for `make all`, `make upload`, and `make clean`, Eclipse will display them all as shown in Figure 10.

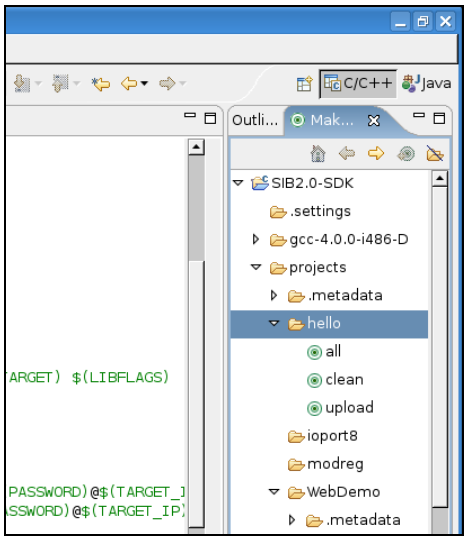

Figure 10: Executing Make Targets

Following the creation of an Eclipse *Make Target,* it may be executed by simply double clicking the target in the *Make Targets* view. For example, double clicking *all* in Figure 10 will compile all of the

source files specified in the makefile for the `hello` project. The *Console* tab in the bottom center window will show the output of all commands that are executed.

2.3 Executing Applications

Eclipse can also be used to execute applications on the host development machine. This can be useful when it is practical to test an application or algorithm on the host machine before uploading it to the target board for testing, or when the remote target board is not available for testing. Also, this is valuable when developing companion software to be used in cooperation with the target board.

Eclipse can only execute applications that have been compiled using the compiler for the environment on the host development machine. In other words, Eclipse will not be able to run applications that have been compiled with `m68k-elf-gcc` or other incompatible compilers.

With some applications written for a specific target board, running applications through Eclipse may be as simple as creating a copy of the original makefile and changing the compiler to the host GCC. In cases where the program is more platform or device dependent, the code may need to be broken down and modified considerably before it will compile and execute on the host development machine. In these instances, it will be more advantageous to test applications for the target board after physically uploading them and executing the file, using the remote debugging process described in section 3 to work out any problems with the application.

To create and run applications for use on the host development machine only, simply create a makefile that calls the GCC version on the host system with the appropriate flags. Follow the same process of creating and executing *Make Targets* and makefiles as described in sections 2.1 and 2.2. Also, ensure that GCC has been installed on your system first. For more information on installing GCC, see the links in section 5.

After compiling an application using the Eclipse *Make Targets*, you may run it using the *Run* menu. Select *Run...* and specify how the program should be executed through the configuration screen that Eclipse will bring up. A sample view is shown in Figure 11.

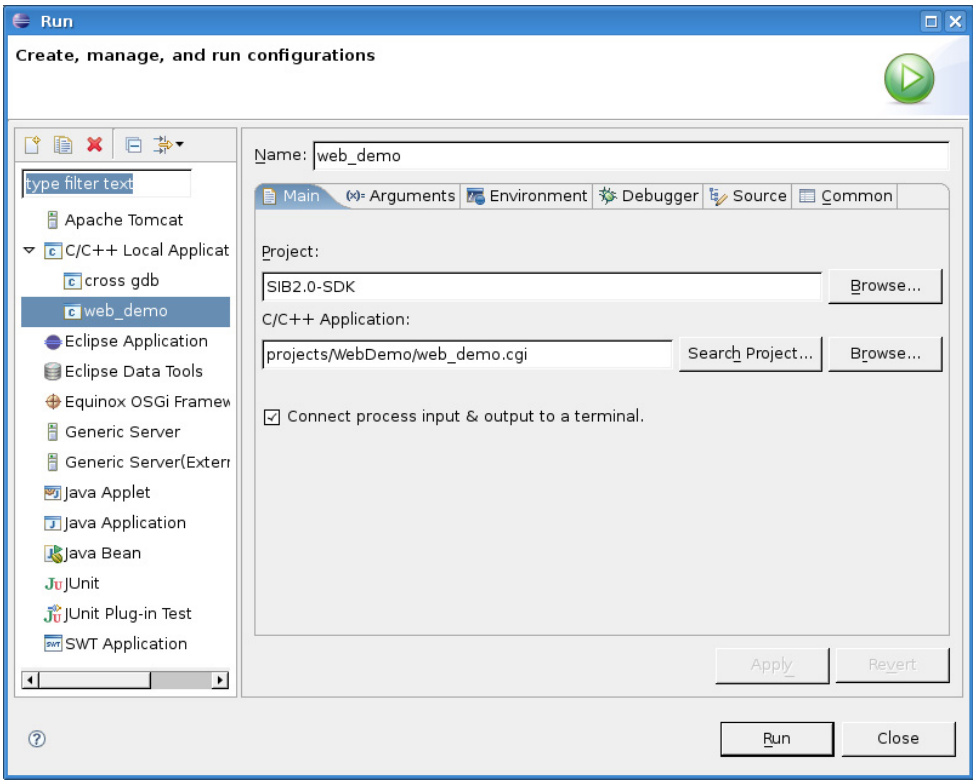

Figure 11: Run Configuration

Once run, all program input and output is facilitated through the *Console* tab in the bottom center window of the Eclipse workbench.

The RSE plugin allows for remote launching and execution of programs on the target board. This includes uploading the executable, executing it, and piping the program input and output through the Console window in Eclipse. This functionality is only available for boards with an SSH connection such as the SIB/SBS.

To use remote launching, right click on a compiled executable and select *Run...*. Select *C/C++ Remote Application* and *New Configuration*. Create a new connection and specify the proper settings on all of the tabs. Be sure that the remote path for the executable is set correctly as well. Program output and input are facilitated through the Console in Eclipse.

(***Note:*** EMAC has found that the upload function in this version of the RSE remote launch is not always successful at changing the permissions of uploaded files. If remote launch fails, manually set the executable permissions of the file through a terminal connection to the target board and try again. Further attempts to upload to the same location should then be successful.)

3 Remote Debugging

When developing applications for an embedded environment, it is important to be able to debug programs from a remote machine, as this is generally difficult or impossible to do on the embedded device itself. The combination of the GNU Debugger (GDB) and its companion server (GDBServer)

allow for this to be done quite easily and efficiently. Furthermore, Eclipse integrates remote debugging using this software combination, allowing for an easy to use graphical debugging session.

3.1 Compiling for Debugging

In order to facilitate remote debugging, only the executable must be present on the target board. On the host machine, however, the source code for the application must be present as well as the executable.

The executable on the host machine must have been compiled with debugging symbols included in order for GDB to work properly. This is done with the use of the `-g` option to GCC. Although it is possible to compile debugging symbols into the remote executable as well, this may cause large applications to become too big for the embedded device, and will not provide any benefits. The debugging symbols option should be the only compiler flag that differs when compiling the executables for the target board and the host machine.

It is also important to ensure that no optimization is preformed during assembly, as this will make debugging the code nearly impossible. This may be specified with the `-O0` (capital "O" followed by "zero") option to GCC, but should be implied by default as long as no other optimizations are specified.

Compile and upload the files using the Eclipse *Make Targets* created in section 2.2. It may be necessary to modify the Makefile to reflect the required compiler flags. It is also possible to edit the Makefile to create two executables, one with debugging symbols and one without.

3.2 Establishing a Connection

Before continuing to set up the required software for the debugging session, establish and test a connection with the remote target board. Ensuring that a reliable connection has been secured between the host development machine and the target board will make troubleshooting any problems within the debugging session easier.

This connection can be established via an Ethernet or serial port connection. Ethernet connections are generally established via SSH, Telnet, FTP, and similar protocols. Serial connections require setting up a terminal on the target board and connecting to this terminal from the host development machine. Refer to the documentation for your specific board for information about how to set up a serial terminal.

For the SIB/SBS and other SSH equipped products, Ethernet connections should be established through the SSH server. This will ensure that all traffic is encrypted. There is also an optional SSH module available for the SoM-5282. Most SoM products do not currently support SSH connections, requiring Ethernet connections to be made using Telnet and FTP.

In the past, connections to the target board were generally managed externally to the IDE, forcing the developer to switch between programs in order to communicate with the device and develop applications simultaneously. Now, however, solutions to this problem are available through Eclipse with the use of the Target Management / RSE plugin.

If you have not already created a new user on your target board, you will need to do this before connecting to the SIB/SBS through the Target Management Plugin because the Eclipse terminal will not

display the menu system on the SIB properly. Connect to the SIB using SSH and use the menu configuration system to create a new user. You should then be able to log in through SSH with the new user name (using a shell command like `ssh new_user@192.168.2.101`). Leave this connection open for now to allow access to the shell on the board. It is necessary to have an SSH client installed on your machine to enable this connection. More information about SSH can be found in the links in section 5.

An SSH key to be used for the newly created user can also be generated. The following instructions will help you to create a new key to use with the SIB/SBS. To create key pairs for the SoM-5282 with optional SSH or other SSH equipped boards, the commands may be slightly different. See the documentation that came with your board for more information about how to do this. To create a new key on the SIB/SBS, generate a key pair on the host development machine with the command `ssh-keygen -t dsa`. Using the system defaults, this should create the files `~/.ssh/id_dsa` and `~/.ssh/id_dsa.pub`. The `.pub` file contains the public key, and must be securely copied to the target board. A command similar to `scp ~/.ssh/id_dsa.pub root@192.168.0.101:/home/new_user/` will then copy the file to the home directory of the newly created user (replace the IP address and directories with the appropriate values).

The final step in using the new key is to copy it into the keys file. On the SIB/SBS, if the directory `~/.ssh` and the file `~/.ssh/authorized_keys` do not exist, create them. Next, add the key to the authorized keys file with the command `cat ~/id_dsa.pub >> ~/.ssh/authorized_keys` on the SIB/SBS. Close the SSH session with the SIB/SBS, and attempt to log in as the newly created user. The password (if any) used in the creation of the public key will be prompted for before login. Delete the `id_dsa.pub` file in the new user's home directory on the SIB/SBS.

You are now ready to open a connection with the board through SSH using the Target Management Plugin. Begin by launching the RSE perspective using the *Window* menu. Select *Open Perspective* followed by *Other*. Select *Remote System Explorer* in the new window and click on *OK*.

When the new perspective opens, you should see the *Remote Systems* tab to the left of the screen. In this window, create a new *SSH Only* connection as shown in Figure 12. This will bring up a new screen that will allow you to configure the connection. Ensure that the host name or IP address of the target board is entered correctly in the *Host Name* field.

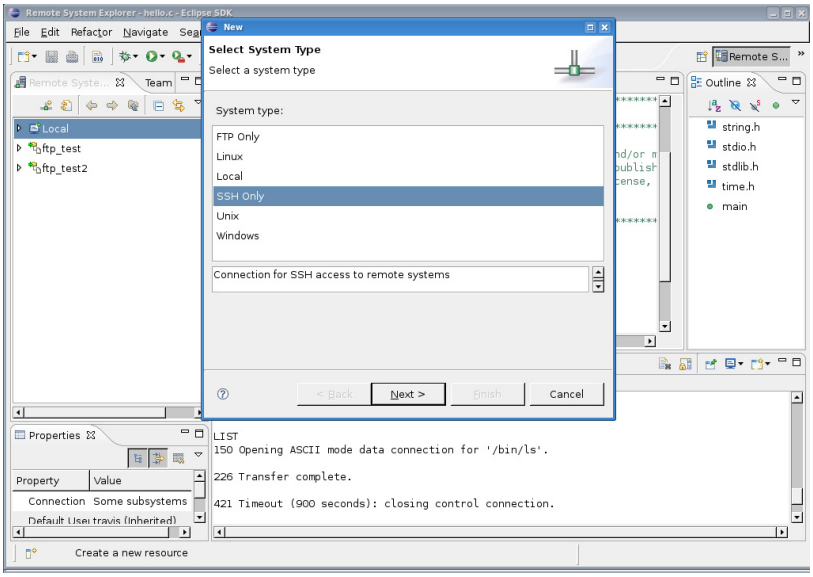

Figure 12: Creating a New Connection

The newly created connection should now appear in the *Remote Systems* window on the left of the screen. Monitor this connection with Eclipse by right clicking on the connection and selecting *Show in table.* Also, right click on the connection and select *Monitor*. You should see the connection added to the windows at the bottom center of the screen. Finally, right click on the connection and select *Connect*.

The system should then prompt for your user name and password. Enter the user name and password for the user created earlier in this section. The resulting configuration should be similar to the following screen shot. You will also want to open the local connection to allow browsing of the local file system and shells. This connection should be opened automatically. Figure 13 shows the completed setup.

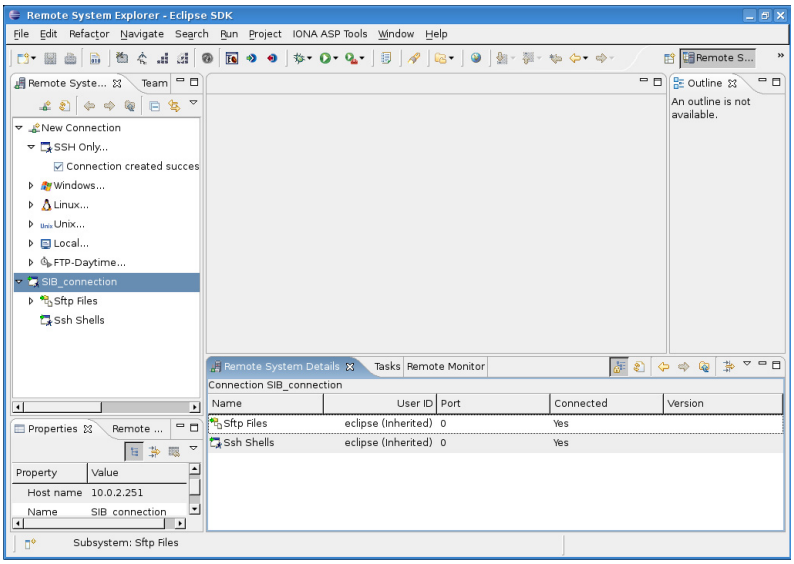

Figure 13: Remote System Explorer

You may use the *Sftp* files link to browse, edit, copy, and paste files to and from the file system on the target board as if they were located on the development host. From the other perspectives, connections

may be monitored and established by activating the various *Remote System Explorer* windows in the current perspective through the *Window* menu under *Show View*.

To open an SSH shell with the target board, right click on *Ssh Shells* and select *Launch Shell*. This will bring up a new view that will allow you to execute commands on the target board. Local shells on the development host may also be invoked in the same way through the Local connection.

For boards that are running an FTP server, it is possible to browse the remote file system and edit files remotely using FTP through the RSE plugin as well. To do this, follow the above process to set up a connection, but select FTP rather than SSH as the connection type. It is not possible to open a remote shell using FTP, however.

Terminal connections can also be established using the Target Management Terminal SDK, which is also included with the EMAC Eclipse Distribution. This supports connections through serial terminals, SSH, and Telnet.

To establish a terminal connection, select *Window -> Show View -> Other...* and select *Terminal* from the list that is brought up. The Terminal view will be opened in the lower window. Next, click on the *Settings* tab within the Terminal view and fill in the appropriate information for the connection. After clicking *OK*, Eclipse will attempt to establish a connection. The terminal will then be connected to the Terminal view in Eclipse, allowing full access to the remote system. An example of this is shown in Figure 14.

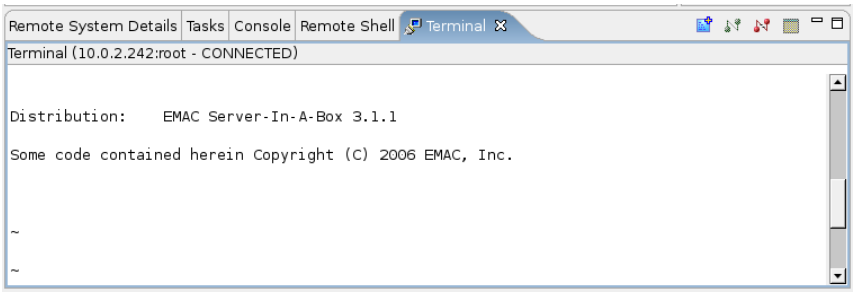

Figure 14: Terminal Connection

3.3 GDBServer and GDB Versions

Once a connection has been successfully established with the target board, the required programs for debugging on the target board and local development host must be located and installed. The required GDBServer programs are included with the EMAC Linux distributions for the SoM-5282 and the SIB/SBS. The appropriate version of GDB must be installed on the host machine, however.

For the SIB, GDB is located in the same location as GCC in the SIB-SDK-3.0 directory. To find it, look for `i486-linux-gdb` in `SIB-SDK-3.0/gcc-4.1.1-i486-D/bin`. Place a link to this executable somewhere in your path for easier use.

For the SoM-5282, EMAC provides a different configuration of GDB than the version included with the m68k-elf-tools toolchain to allow for compatibility with Eclipse. This can be located on both the EMAC FTP site and the CD-ROM. Refer to the instructions in the `README.htm` file within the SoM-5282EM-

SDK to build this new version of GDB. Replace the version of `m68k-elf-gdb` installed from the m68k-elf-tools toolchain with this version. It should be located in `/usr/local/bin` on the host development machine.

For other EMAC products, refer to the product specific documentation for information on how to install the toolchain for the respective device.

3.4 Using GDBServer

After compiling and uploading the required programs and locating and installing the appropriate version of GDB for your target board, GDBServer must be invoked on the target board. This will allow the target board to accept incoming connections from GDB on the development machine.

This can be done through a TCP connection or a dedicated serial connection from the host machine to the target board. In either case, first navigate to the directory of the executable on the target board and ensure that execute permissions have been enabled for the program (using the `chmod` command). GDBServer should be run from this directory.

The general form of the command to initiate GDBServer is `gdbserver COMM PROGRAM [ARGS]`
where `COMM` specifies the connection and `PROGRAM [ARGS]` specifies the program to debug along with any command line arguments that the program should be passed.

When debugging over a TCP connection, `COMM` takes the form of `HOST:PORT` to describe the IP address and port to listen for connections from. For example, if the development host is at IP address 192.168.0.100, and you want to debug the `web_demo.cgi` program from the EMAC project examples, you could enter a command like `gdbserver 192.168.0.100:2828 web_demo.cgi` to listen for incoming connections on port 2828. This port number can be anything that you like, as long as it is not a reserved port number. Leaving the `HOST` portion of the command blank, as in `gdbserver :2828 web_demo.cgi` will allow GDBServer to accept connections from any machine with access to it from the network.

Debugging over a serial connection requires that the `COMM` portion of the GDBServer command denote the serial port to listen for connections on. For example, if the host development machine and the target board have a serial connection on port `/dev/ttyS1` on the target board, the command `gdbserver /dev/ttyS1 web_demo.cgi` will allow GDBServer to listen for incoming connections over this serial line.

3.5 Shell Operation

After compiling the executable with debugging symbols for the host machine and starting GDBServer on the target board, you are ready to start debugging using GDB. Testing GDB and establishing a connection using the shell on the host machine is a good way to test your GDB and GDBServer setup before using Eclipse.

From the shell on the development machine, navigate to the directory containing the executable to be debugged as well as all required source code. From here, issue the command for the appropriate version of GDB.

For example, if the executable to be debugged were `web_demo.cgi.gdb`, issue the command `m68k-elf-gdb web_demo.cgi.gdb` or `i486-linux-gdb web_demo.cgi.gdb` depending on the platform you are developing for. This assumes that the GDB executable is located in your path as described in section 3.3.

GDB should start and return a prompt for commands. Next, initiate a connection to the remote machine for debugging by issuing a `target` command. This command will differ according to the type of connection (serial or TCP).

To connect over a TCP connection, you must know the IP address of the target board as well as what port GDBServer is listening for connections on. Following the earlier examples and assuming the IP address of the target board is 192.168.0.101 listening on port 2828, issue the following command at the GDB prompt: `target remote 192.168.0.101:2828` to establish a connection with the target board.

When debugging over a serial connection, the command becomes even simpler. For example, if the target board is connected on the host development machine port `/dev/ttyS1` and GDBServer is listening for connections on the respective serial port on the target board, issue the command `target remote /dev/ttyS1` at the GDB prompt to establish a connection.

After the GDB prompt returns, the shell on the target board should report that a connection has been successfully established. To test your connection, first add a breakpoint in the main function of the program with the command `b main` at the GDB prompt. To execute the code up to the breakpoint, continue by typing `c` at the GDB prompt. The program should advance and stop. Set other breakpoints in the program and step through the application. You may use the `list` command to show the line numbers of surrounding code that you wish to set a breakpoint at.

The output of the program will be displayed on the shell of the target board through the output of GDBServer. Any required input for the program will also need to be entered on the target board shell within the running GDBServer session.

3.6 Remote Debugging with Eclipse

The next step after successfully debugging an application through the shell is to use Eclipse to set up a remote connection with GDBServer. This will allow for a much more user-friendly interface and make debugging more efficient. There are currently two main ways to do this: manually establishing a connection, and using the RSE Plugin to automate the process (for boards with SSH capabilities).

3.6.1 Manually Establishing a Connection

This method requires that the user start GDBServer on the target board and connect using GDB through Eclipse. The GDBServer command will be exactly the same on the target board as it was when debugging through the shell. After compiling and uploading the necessary applications, start GDBServer on the target board according to section 3.3.

On the Eclipse workbench, right click on the application that you are trying to debug. Make sure that this file has been compiled with debugging symbols. From the menu that is brought up, select *Debug As* followed by *Debug...* as shown in Figure 15. This will bring up a new window to allow you to configure the way the program is debugged. These features can also be accessed from the *Run* menu.

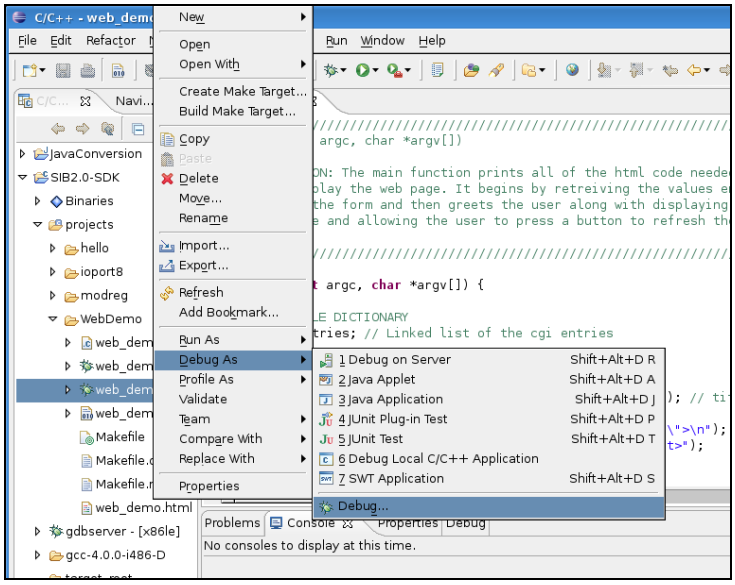

Figure 15: Debugging an Application

On the new window, select *C/C++ Local Application* on the left and name the configuration on the right side of the screen so that your changes are saved for future sessions. Select the project and application to be debugged if necessary. This is illustrated in Figure 16.

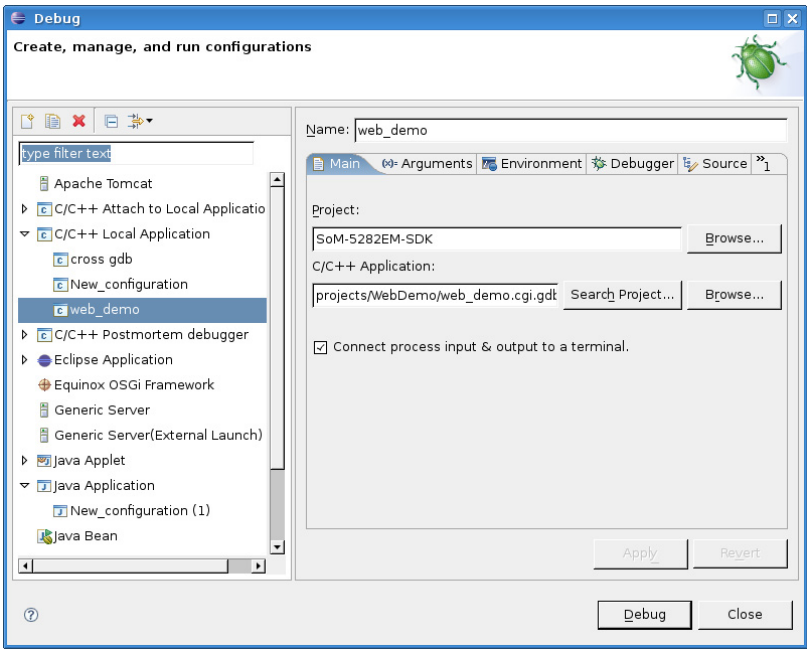

Figure 16: Debugging Configuration

The *Debugger* tab on this window is where most of the configuration will take place. Select this tab and choose *gdbserver Debugger* as the debugger. Next, on the *Main* tab under *Debugger Options*, choose the appropriate *GDB Debugger* for your platform as described in section 3.3. Also, set the *GDB command set* to *Standard (Linux)* and the *Protocol* to *mi*. The EMAC versions of GDB are not configured to use a *GDB command file* by default, so this setting is not necessary. The final setup should be similar to Figure 17.

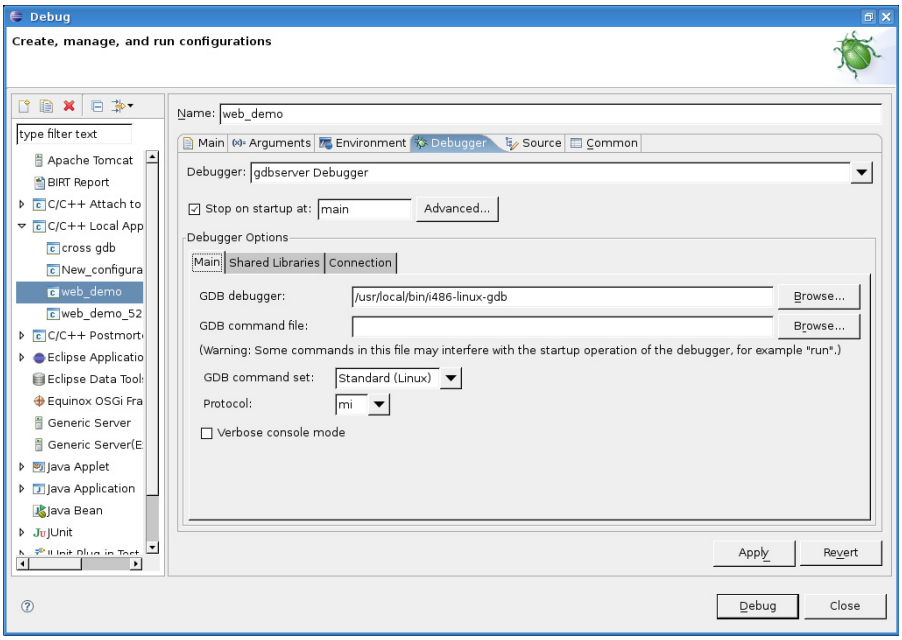

Figure 17: Debugging Configuration (2)

The *Shared Libraries* tab can be useful in telling GDB exactly where to search for shared libraries that are utilized by an application. If GDB cannot find the required libraries, it will print errors while executing, and the location of the libraries will need to be specified directly through this screen.

On the *Connection* tab, the connection type and parameters for Eclipse to locate the target board must be specified. For example, to specify a TCP connection to IP address 192.168.0.101 on port 2828, use the settings shown in Figure 18.

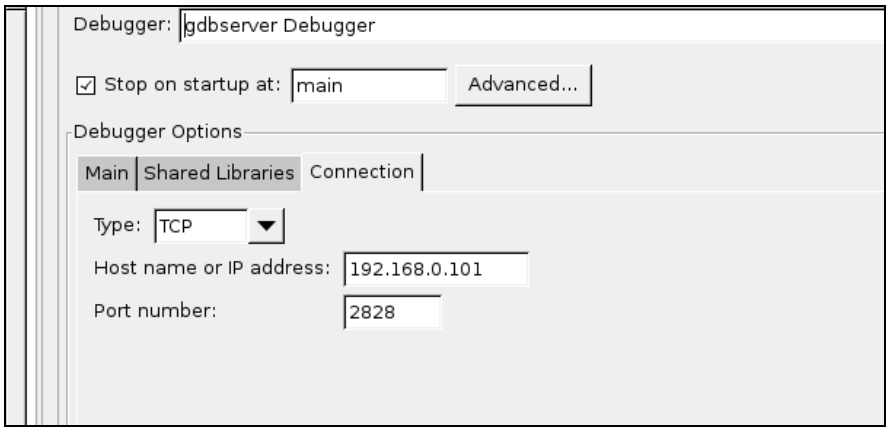

Figure 18: Remote Debugging Configuration for TCP

Connection over a serial line requires setting the device and line speed as in Figure 19 according to the settings of the serial connection to the target board.

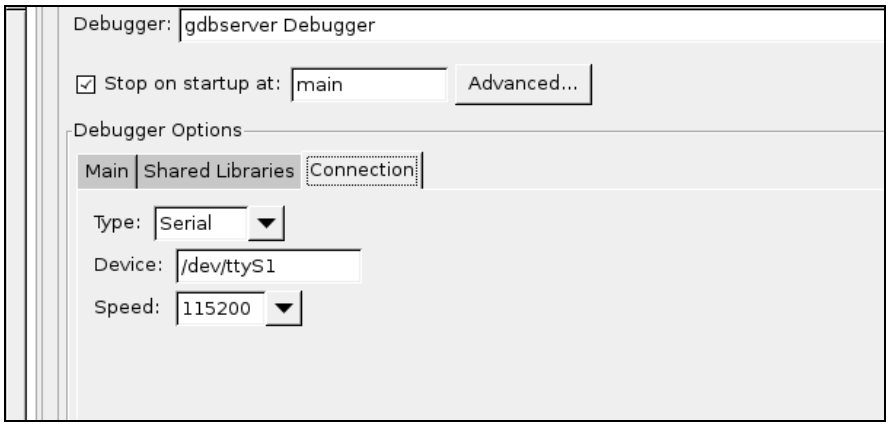

Figure 19: Remote Debugging Configuration for Serial Connection

Use the *Source* tab to specify the search path for the program source files. Check to ensure that this is set so that the directory containing the source files for the program is located in the search path as shown in Figure 20. If not, add this location to the path.

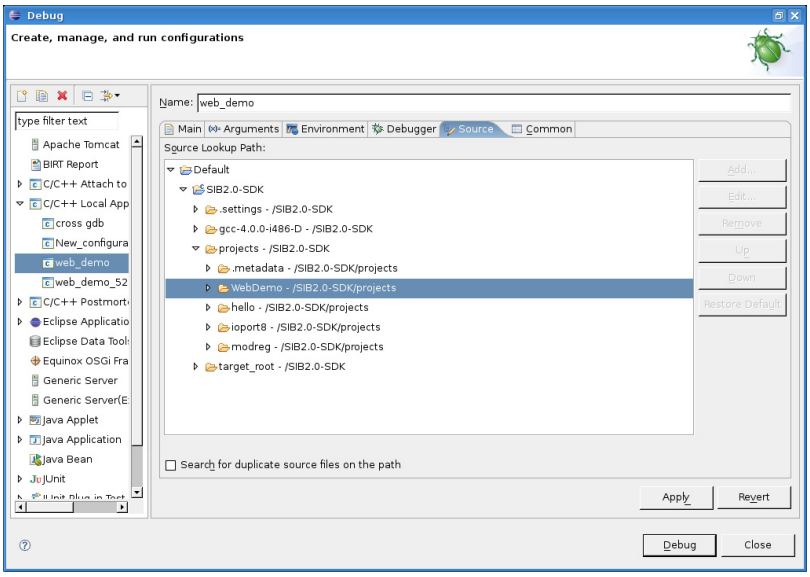

Figure 20: Setting the Search Path for Source Files

When you are finished with the configuration and have started GDBServer on the target board, press *Apply*. Press *Debug* to initiate the connection and begin debugging.

3.6.2 Using the RSE Remote Debugging Tool

The RSE plugin can be used to automate the remote debugging and testing process. Essentially, this consists of automatically uploading the executable to the target board, opening a GDBServer connection for that application, connecting to the server, and piping the program input and output to the Console within Eclipse. This is only possible with boards running SSH.

The process for automated remote debugging is similar to the process discussed before, except that a *Remote C/C++ Application* configuration should be used rather than a local configuration. As before, choose an executable to debug and right click on it. Select *Debug As -> Debug...* . On the window that comes up, choose *Remote C/C++ Application* and enter the information for the connection to the target board. From this point forward, operation is as described in section 3.6.1, except that output and input for the program are facilitated through the Eclipse Console.

3.6.3 The Eclipse Debugging Perspective

After establishing a connection with the program on the target board, Eclipse will automatically open the *Debug* perspective, locate the source code, and wait for commands. This is shown in the screenshot below. The GDBServer process running on the shell of the target board should report that a connection has been established if a connection was established manually (not using the RSE Remote Launch).

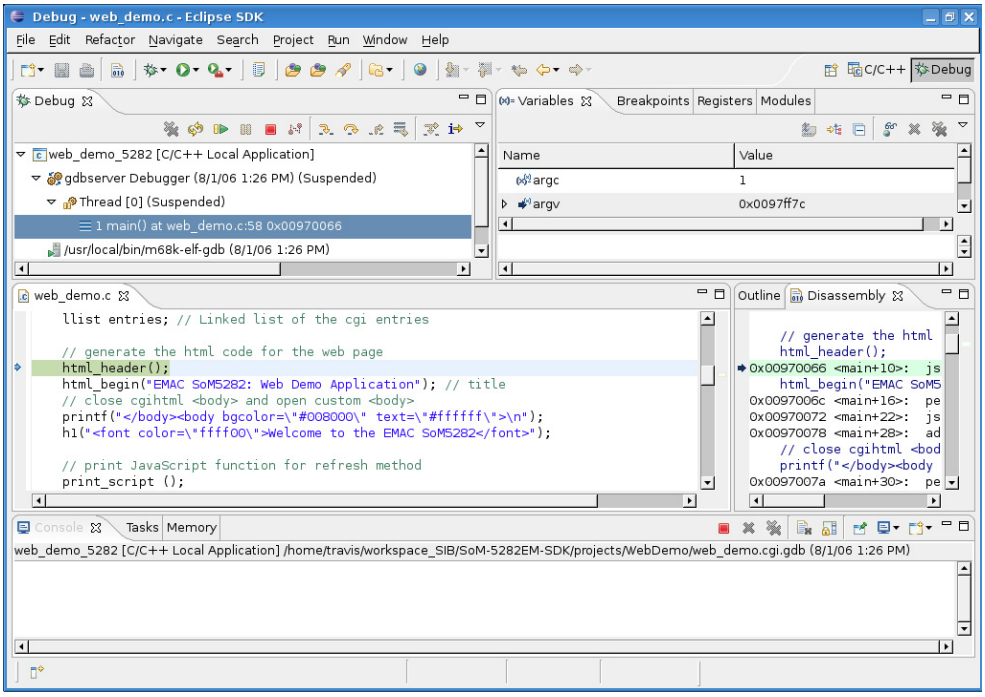

Figure 21: Eclipse Debugging Perspective

In Figure 21 above, the current location in the program is highlighted in green in the source code window. Directly to the right of the source code, the current location in the assembly code is also highlighted under the *Disassembly* tab. Any output from the debugger is displayed in the Console window at the bottom of the screen. Note that program I/O will still take place on the shell of the target board if you established the connection manually. Using the RSE Remote Debugging tool automatically pipes the program I/O to the Eclipse Console. For manual connections, the program can be accessed from any of the connection types described in section 3.2, including the TM Terminal SDK SSH, Terminal, or Telnet connections within Eclipse.

Use the various buttons under the *Debug* tab in the upper left of the screen to manage your debugging session. You can use these to step through the program, resume and pause execution, and toggle the stepping mode, along with other options.

To add a breakpoint, simply double click in the gray margin to the left of the source code on the line where you would like it to be placed. Breakpoints, variables, modules, and registers are monitored in the window in the upper right of the screen. After adding a breakpoint, it will be marked with a breakpoint pin to the left of the line on the source code as well as an entry under the *Breakpoints* tab in the upper right window.

Step through the application and familiarize yourself with the operation of the Eclipse Debugging perspective. This will also allow you to test your configuration by ensuring that everything is working properly before attempting to develop your own applications. For more information on GDB and debugging within Eclipse, see the links in section 5.

4 Kernel Modules and RTAI

The EMAC SIB-SDK also provides the ability to build modules and RTAI applications that must be compiled against a specific kernel. RTAI applications are particularly difficult to cross-compile, as they must be built against the exact same kernel that they will be loaded into and must be compiled with the same compiler and configuration as the kernel as well. The EMAC SIB-SDK greatly simplifies this process.

Due to size restrictions and maintainability, EMAC provides kernel sources and RTAI packages separately from the SDK. Contact EMAC Support for information on where to locate the kernel for your device. To determine which kernel you need, use the uname -r command. Download the corresponding kernel source package, and extract it into the `gcc-4.1.1-i486-D/sysroot` directory. If this is an RTAI kernel, RTAI headers will be added to your SDK as well. The kernel source will be placed in the `sysroot/usr/src/<kernel version>` directory within the SDK. Note that modules that are built against any other kernel than the one running on the target board will not work.

For an example of how to build kernel modules within the SDK, look at the rtai_sine project. In order to build this example, change the KERN variable in the Makefile to the version that you are building against to allow make to find the correct source. This must be an RTAI enabled kernel. When writing your own kernel modules, be sure to include the header files from the include directory for your kernel, rather than those from `sysroot/usr/include/linux`. The files in `sysroot/usr/include/linux` are the static kernel headers that glibc was built against. Modules must be built against the running kernel rather than the glibc headers.

Compile the rtai_sine example just like any of the other projects. Make will automatically build everything that is needed using the correct kernel version based on the information given in the makefile. During compilation, you will most likely see several errors, including several about an unsupported GCC version and undefined modules. The unsupported GCC message comes from the RTAI headers, and can safely be ignored. The undefined modules messages are caused by the fact that the modules are not currently loaded by the kernel on the development machine, and will disappear as soon as the module is loaded onto the target board.

The easiest way to test this example is to use the upload make target to upload the entire example directory to the target board. Then, execute `./run` from within this directory on the target board. This will load all of the appropriate modules and execute the example. You should then see three scrolling columns calculating a counter, sine, and cosine values. If this does not work correctly, the `dmesg` command can provide valuable clues into what went wrong.

Use the `rtai_sine` example as a template for developing your own kernel modules and RTAI applications.

5 Further Information

The combination of Eclipse and the EMAC development tools should allow you to easily and efficiently develop, debug, and test custom applications for use on and with EMAC products such as the SoM-5282M and the SIB. For more information about the software mentioned in this document, start with the links below.

Eclipse:	www.eclipse.org
Eclipse Wiki:	wiki.eclipse.org
Java:	java.sun.com
GCC:	gcc.gnu.org
GNU Make:	www.gnu.org/software/make
Makefiles:	www.wlug.org.nz/MakefileHowto
Open SSH:	www.openssh.com
Target Management:	www.eclipse.org/dsdp/tm/
GDB:	www.gnu.org/software/gdb
GDBServer:	www.linuxmanpages.com/man1/gcc.1.php
Remote Debugging:	linuxdevices.com/articles/AT6046208714.html
RXTX:	www.rxtx.org/
Eclipse CDT:	www.eclipse.org/cdt/
Astyle Eclipse:	astyleclipse.sourceforge.net/
EMAC Software:	ftp://ftp.emacinc.com/PCSBC/Development_Kits/EMAC_Open_Tools/ ftp://ftp.emacinc.com/Controllers/Development_Kits/EMAC_Open_Tools/
EMAC Documents:	ftp://ftp.emacinc.com/PCSBC/Development_Kits/EMAC_Open_Tools/Manuals/ ftp://ftp.emacinc.com/Controllers/Development_Kits/EMAC_Open_Tools/

If you have any questions regarding the information described in this document contact EMAC support (support@emacinc.com) with the subject heading “Eclipse Development Support.”

Appendix A

A.1 Installing Eclipse Manually

The following instructions cover the installation of Eclipse and the necessary plugins. These steps are not required for the EMAC Eclipse Distribution.

Eclipse is also available directly from Eclipse at http://download.eclipse.org/eclipse/downloads/. After downloading the package, unpack the contents of the gzipped tar file to the desired location, such as `/usr/local`.

Note that some Linux distributions are releasing Eclipse packages as well. If you decide to use one of these packages, be aware that the installation procedure will differ slightly from this document. Also, ensure that the correct version of Eclipse is installed.

After installation is complete, navigate to the top level of the Eclipse directory. The Eclipse executable is named `eclipse`. Executing this file will start the Eclipse IDE. For ease of use, place a link to this file somewhere in your path or add the Eclipse directory to your path. See the Eclipse documentation in the package and online for more information about command line arguments and other options.

A.2 Installing the CDT Plugin

The utility to locate and install new features in Eclipse is located under the *Help* menu. Select *Software Updates* and then *Find and Install*, as shown in Figure A-1.

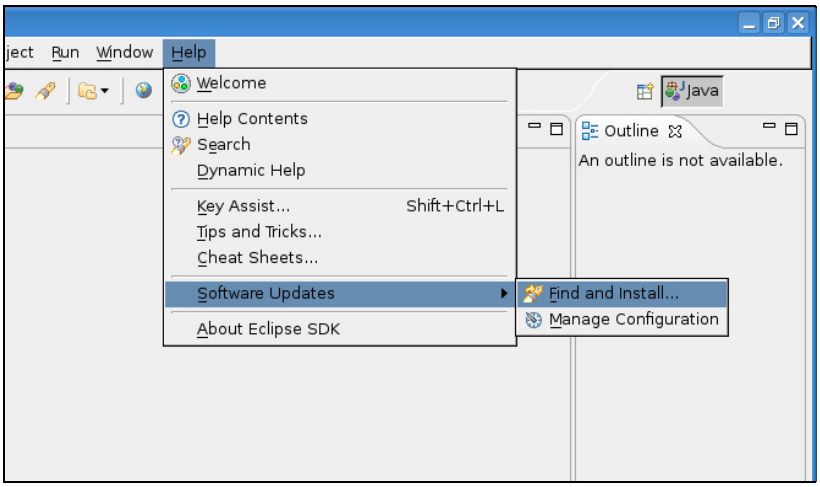

Figure A-1: Installing Plugins

This will bring up a new window, which prompts for the type of features to search for. Select *Search for new features to install* and click on *Next.*

The next screen is important because it determines which sites to search for features. Select the *Callisto Discovery Site*, and click on *Finish.* Eclipse will begin to search for updates, and may prompt you to select the mirror to download from. Select a mirror and wait for searching to complete.

Once Eclipse has determined which features are available for your version, it will list them on a new screen. Expand the *Callisto Discovery Site* list, check the box next to *C/C++ Development Tools,* and click *Next*. You will be prompted to accept a license agreement for the new features. Following this screen, check to ensure that the download location is correct, and click *Finish*. Answer any necessary questions during installation. After the tools have downloaded and installed, you will be prompted to restart Eclipse for the changes to take effect.

A.3 Installing the RSE Plugins

Installing the RSE Plugins follows the same process as the CDT Plugin. At this time, TM/RSE provides the "Terminal SDK" separately from RSE. From the *Help* menu, go to *Software Updates* and select *Find and Install.* Select *Search for new features to install* and click *Next*. Create a *New Remote Site* named "RSE" with a URL of "http://download.eclipse.org/dsdp/tm/updates/". Check the box next to the RSE site and select *Finish*. When presented with the available packages, choose those that apply to your environment and needs, and proceed with the installation.