Using the EMAC OE SDK Example Projects with Eclipse

The EMAC OE SDK is distributed with a set of example projects intended to demonstrate how to use the EMAC OE toolchain and libraries. This guide demonstrates the process of compiling one of the example projects and running it on the target machine using the Eclipse IDE. A basic familiarity with Eclipse is assumed for this guide. For a quick intro, see the Eclipse First Steps Guide.

This guide uses the `hello` EMAC OE SDK example project. It consists of a C file and `Makefile`.

Tools Required

- GNU `make`
- EMAC OE SDK
- `wput`

EMAC SDK Example: Compile and Run the hello Project

Setup

1. Modify `global.properties` according to the SDK Remote Upload Setup to provide `make` with the correct user, password, and IP address for the `upload` target.
2. Create a remote terminal connection. This is necessary to use the Remote System Explorer's SSH Terminal feature in the following procedure.

Procedure

This procedure provides an overview of how to compile and run C applications for EMAC products in Eclipse. It assumes familiarity with the C programming language and is not intended as a general guide on learning to program.

1. Click *Window → Open Perspective → Other...* To bring up a dialog window with a list of Perspectives to choose from.
2. Choose *C/C++* and click *Ok*.
3. Select the *Make Targets* View.
4. Expand *EMAC-OE-arm-linux-gnueabi-SDK_4.0 → projects → hello*.
5. Cross-compile the program:
   Double-click the *all* Make Target. The result is shown in Figure 1 below.

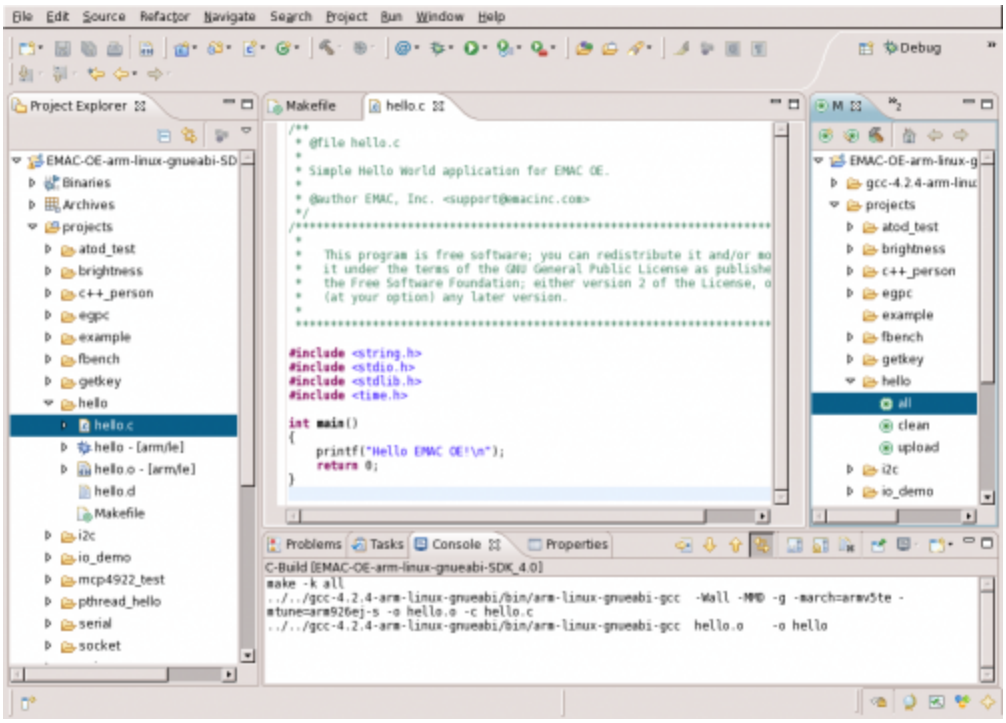

Figure 1: Make all

`all` is a `make` target. Make targets have dependencies which they check to determine whether or not a command needs to be run. In the case of `all`, the dependencies are the C source files listed in the `CFILES` Makefile variable. `make` will check the modification times of these *.c files against that of the currently existing executable (if there is one) to see if they have been modified since the last time the executable was compiled. If they have, `make` reruns the toolchain commands necessary to produce a new executable. The chain of dependencies is actually more complex since it also involves the intermediate object files produced by the compiler before the linking stage. For a more in-depth explanation see the GNU `make' Manual (http://www.gnu.org/software/make/manual/make.html) .

6. Upload the program to the target machine:
   Double-click the *upload* Make Target. The result is shown in Figure 2 below.

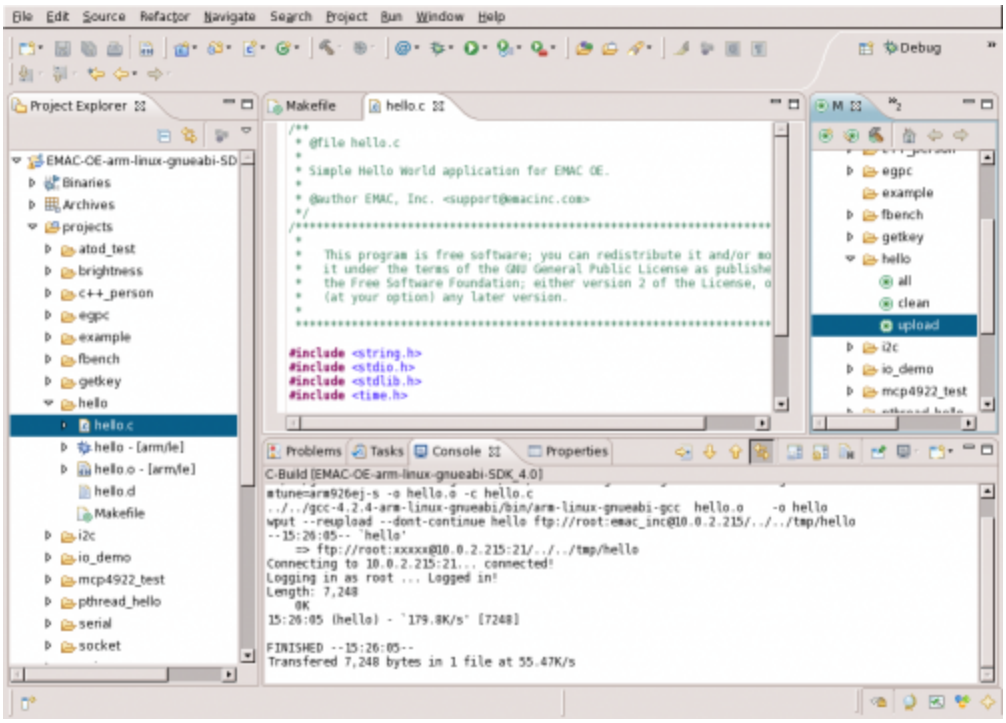

Figure 2: Make upload

`upload` is another `make` target. This one uses the development system's `wput` command to send the target binary to the target machine. This is accomplished using variables stored in the `global.properties` file, which is included in the Makefile using the `include` keyword. The `global.properties` file also contains variables which `make` passes to the compiler to ensure that the executable produced is compatible with the target CPU architecture.

7. Connect to the target machine.
8. Run the program as shown in Figure 3 using the remote terminal created in the Remote Terminal Setup Guide.

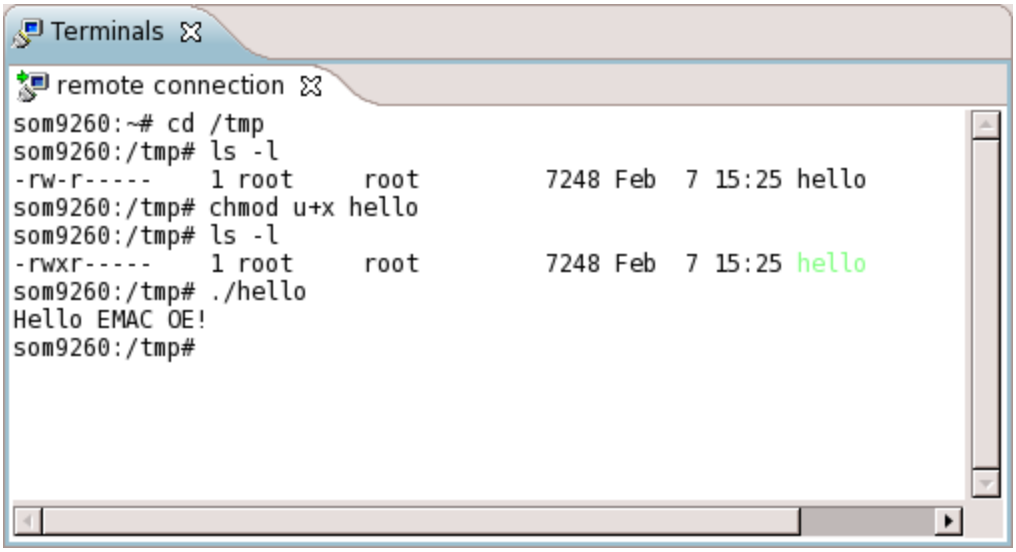

Figure 3: Running the Remote Application

The following is a brief description of each command seen in the SSH Terminal window above:

1. `cd /tmp`: Change the current working directory to `/tmp/`.
2. `ls -l`: List the current working directory's contents with file permissions shown. Notice that root does not have execute permissions for the file `hello`.
3. `chmod u+x hello`: Give executable permissions to the owner of the `hello` file. Note that this assumes

that the same user is used to log in through the SSH Terminal as was specified in `global.properties` according to Step 1 of the Setup for this guide–in this example, the user is `root`.
4. `ls -l`: List the current working directory's contents with file permissions shown to be sure that root now has execute permission. The new `x` indicates that `root` does now have execute permissions.
5. `./hello`: Execute the binary. The output shown is a simple message printed to the screen, "Hello EMAC OE!".

If you want to make modifications to the Example C File and recompile it, follow the same procedure as above to test it again. With a few iterations of this the process will become familiar and easy.

Example C File

This C file can be used by first time C programmers as an example to begin application programming for EMAC products. It is included in the EMAC OE SDK Example Project Guide.

Example Makefile

The EMAC OE SDK Example Project Guide shows the default `Makefile` file used for the `hello` example project. This is a necessary component of the EMAC OE SDK which directs GNU 'Make' in resolving source code dependencies before calling the cross-compiler to create a binary for the target platform. It also provides a convenient `upload` target which utilizes the development system's `wput` command to send the compiled binary to the target system.

If you intend to write your own `Makefile` from scratch, EMAC recommends paying close attention to the include paths to learn more about the SDK cross compilation tools.

Next Steps

The next step is to create a new Makefile-based Eclipse project from scratch. This process is similar to the example above, but requires a few extra steps. See the New Project Guide for help on this.

See Also

- Eclipse IDE
  - Install
  - Development System Configuration
  - First Time Using Eclipse
  - Import EMAC OE SDK
  - Eclipse Terminal View
  - Using the EMAC OE SDK Examples Projects
  - Create New EMAC OE SDK Projects
  - Using the EMAC OE SDK Eclipse Plugin
  - Remote System Explorer Configuration
    - RSE Setup
    - RSE SFTP Setup
    - Remote Shell/Terminal Setup
  - Execute Remote Applications
  - Debug Remote Applications

» debug » existing_build » eclipse » install » configure » firststeps » import » terminal » linux_start » example

---