EMAC OE SDK Example Projects

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. It is assumed that the procedure in the EMAC OE SDK Configuration Guide has been followed prior to reading this page.

This guide consists of two example files, a C file and a Makefile. They are located within Listing 2 and Listing 3, respectively.

Table 1. Conventions Used

`/path/to/sdk/`	Placeholder indicating the directory to which the sdk will be extracted.
`EMAC-OE-arm-linux-gnueabi-SDK_XX.YY/`	**XX** is The major version **YY** is the minor version **ZZ** is the current revision the major and minor version numbers will match the version of OE for which the SDK was created. The current version is 4.0.

Tools

- GNU `make`. Manages project dependencies.
- EMAC OE SDK. See the EMAC OE SDK Install Page.
- `wput`. Provides FTP capability to the build process.

EMAC SDK Example: Compile the "hello" Project

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

1. The “hello” example project source can be found in the `projects/` subdirectory of the EMAC OE SDK root directory. The full path to the source is as follows:

   ```
   /path/to/sdk/EMAC-OE-arm-linux-gnueabi-SDK_XX.YY/projects/hello/
   ```

2. Cross-compile the program using GNU Make.

   ```
   $ make all
   ```

   `all` is a make target. Make targets have dependencies which are checked to determine whether or not a command needs to be run. In the case of the `all` target, the dependencies are the C source files listed in the `CFILES` Makefile variable. `make` will check the modification times of these C source 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. In reality, the chain of dependencies is 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) .

3. Upload the program to the target machine.

   ```
   $ make upload
   ```

`upload` is another make target. This one uses the development system's `wput` command to send the binary to the target machine through FTP. 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.

4. Connect remotely to the target machine. Run the program as shown in Listing 1 using the remote connection.
Listing 1. Running the Remote Application

```
$ chmod u+x /tmp/hello
$ /tmp/hello
```

    1. Set the root user's permissions to allow execution of the binary. Note that this assumes that you are logged in as the system root user
    2. Run the program without any arguments.

If the file compiles without trouble but has runtime bugs, you can remote debug the application using `gdbserver`. Read the EMAC OE SDK Remote Debugging Guide to learn more.

Example C File

This C file can be used by first time C programmers as an example to begin application programming for EMAC products.
Listing 2. "Hello World" Example Project

hello.c

```
/**
 * @file hello.c
 *
 * Simple Hello World application for EMAC OE.
 *
 * @author EMAC, Inc. <support@emacinc.com>
 */
/*****************************************************************************
 *                                                                           *
 *   This program is free software; you can redistribute it and/or modify  *
 *   it under the terms of the GNU General Public License as published by  *
 *   the Free Software Foundation; either version 2 of the License, or     *
 *   (at your option) any later version.                                   *
 *                                                                           *
 *****************************************************************************/

#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    printf("Hello EMAC OE!\n");
    exit(EXIT_SUCCESS);
}
```

Example Makefile

Listing 3 shows the default `Makefile` file used for the hello example project. This is a necessary component of the EMAC 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.

Listing 3. Example EMAC OE SDK Makefile

Makefile

```
include ../global.properties

TARGET=hello
CFILES=hello.c


OBJS=$(CFILES:.c=.o)
DEPS=$(OBJS:.o=.d)

all: $(TARGET)

$(TARGET): $(OBJS) Makefile
        $(CC) $(VERBOSE) $(OBJS) $(OFLAGS) $(LIBFLAGS) $(SLIBS) -o $@

%.o: %.c
        $(CC) $(VERBOSE) $(CFLAGS) -o $@ -c $<

clean:
        $(RM) *.o *.gdb $(TARGET) $(DEPS)

upload: all
        $(WPUT) $(TARGET) ftp://$(LOGIN):$(PASSWORD)@$(TARGET_IP)/../../tmp/$(TARGET)


-include $(DEPS)
```

Next Steps

After compiling and running an example project, the next step is to create a new project. The EMAC OE SDK New Project Guide details this process from start to finish.

See Also

- EMAC Software Development Kit
  - Install EMAC OE SDK
  - Configure EMAC OE SDK
  - Example Projects
  - New Project
  - Debugging With gdbserver

» emac_oe_getting_started » boot_process » emac_oe_gadget » time » emac_oe_development » esdk » install » configure » linux_start » example

---