Using XMOS Makefiles

IN THIS DOCUMENT

- Projects, Applications and Modules
- The Application Makefile
- The Project Makefile
- The module_build_info file

Projects created by the XMOS Development Environment have their build controlled by Makefiles. These Makefiles execute the build using the program `xmake` which is a port of Gnu Make[1]. The build is executable either from within the XDE or from the command line by calling `xmake` directly.

You do not need to understand the Gnu Makefile language to develop applications using the XMOS tools. The **common XMOS Makefile** provides support for projects, applications and modules. You need only specify the required properties of the build in **Project Makefiles** and **Application Makefiles**.

1 Projects, Applications and Modules

An application is made up of source code unique to the application and, optionally, source code from modules of common code or binary libraries. When developing an application, the working area is described in terms of *workspaces*, *projects*, *applications* and *modules*.

Workspace
A *workspace* is a container for several projects.

Projects
A *project* is a directory possibly containing several applications and modules plus other files relating to a particular project. A project may contain the code for a particular board or reference design or be a software component containing modules for other projects to use.

Applications
An *application* is a directory containing source files and a Makefile that builds into a single executable (`.xe`) file. By convention application directories start with the prefix `app_`. These applications appear at the top level in the project explorer in the XDE.

Modules
A *module* is a directory containing source files and/or binary libraries. The source does not build to anything by itself but can be used by applications. by

[1] http://www.gnu.org/software/make/

Publication Date: 2012/3/20 Document Number: X6348A

convention module directories start with the prefix `module_`. These modules appear at the top level in the project explorer in the XDE.

1.1 Example Structure

An example workspace structure is shown below.

```
sw_avb/
    app_avb_demo1/
    app_avb_demo2/
    module_avb1/
    module_avb2/
    doc/
sc_xtcp/
    module_xtcp/
    module_zeroconf/
sc_ethernet/
    module_ethernet/
```

There are three projects within this workspace: `sw_avb`, `sc_xtcp` and `sc_ethernet`. The `sw_avb` project contains two applications, each of which builds to a separate binary. These applications can use source from the modules within the projects and can use modules from their own project (`module_avb1` and `module_avb2`) and from other projects (`module_xtcp`, `module_zeroconf` and `module_ethernet`).

Alternatively, a workspace may be structured in the following way:

```
app_avb_demo1/
app_avb_demo2/
module_avb1/
module_avb2/
doc/
module_xtcp/
module_zeroconf/
module_ethernet/
```

In this case, all applications and modules are at the top level of the workspace.

2 The Application Makefile

Every application directory should contain a file named `Makefile` that includes the common XMOS Makefile. The common Makefile controls the build, by default including all source files within the application directory and its sub-directories. The application Makefile supports the following variable assignments.

`XCC_FLAGS`*[_config]*
: Specifies the flags passed to xcc during the build. This option sets the flags for the particular build configuration *config*. If no suffix is given, it sets the flags for the default build configuration.

`XCC_C_FLAGS`*[_config]*
: If set, these flags are passed to xcc instead of `XCC_FLAGS` for all `.c` files. This option sets the flags for the particular build configuration

config. If no suffix is given, it sets the flags for the default build configuration.

`XCC_ASM_FLAGS`*[_config]*
: If set, these flags are passed to xcc instead of `XCC_FLAGS` for all `.s` or `.S` files. This option sets the flags for the particular build configuration *config*. If no suffix is given, it sets the flags for the default build configuration.

`XCC_MAP_FLAGS`*[_config]*
: If set, these flags are passed to xcc for the final link stage instead of `XCC_FLAGS`. This option sets the flags for the particular build configuration *config*. If no suffix is given, it sets the flags for the default build configuration.

`XCC_FLAGS_`*filename*
: Overides the flags passed to xcc for the filename specified. This option overides the flags for all build configurations.

`VERBOSE` If set to 1, enables verbose output from the make system.

`SOURCE_DIRS` Specifies the list of directories, relative to the application directory, that have their contents compiled. By default all directories are included.

`INCLUDE_DIRS`
: Specifies the directories to look for include files during the build. By default all directories are included.

`LIB_DIRS` Specifies the directories to look for libraries to link into the application during the build. By default all directories are included.

`EXCLUDE_FILES`
: Specifies a space-separated list of source file names (not including their path) that are not compiled into the application.

`USED_MODULES`
: Specifies a space-separated list of module directories that are compiled into the application. The module directories should always be given without their full path irrespective of which project they come from, for example:

```
USED_MODULES = module_xtcp module_ethernet
```

`MODULE_LIBRARIES`
: This option specifies a list of preferred libraries to use from modules that specify more than one. See X4954 for details.

3 The Project Makefile

As well as each application having its own Makefile, the project should have a Makefile at the top-level. This Makefile controls building the applications within the project. It has one variable assignment within it to do this:

`BUILD_SUBDIRS`
: Specifies a space-separated list of application directories to build.

4 The module_build_info file

Each module directory should contain a file named `module_build_info`. This file informs an application how to build the files within the module if the application includes the module in its build. It can optionally contain several of the following variable assignments.

`DEPENDENT_MODULES`
: Specifies the dependencies of the module. When an application includes a module it will also include all its dependencies.

`MODULE_XCC_FLAGS`
: Specifies the options to pass to xcc when compiling source files from within the current module. The definition can reference the `XCC_FLAGS` variable from the application Makefile, for example:

```
MODULE_XCC_FLAGS = $(XCC_FLAGS) -O3
```

`MODULE_XCC_XC_FLAGS`
: If set, these flags are passed to xcc instead of `MODULE_XCC_FLAGS` for all `.xc` files within the module.

`MODULE_XCC_C_FLAGS`
: If set, these flags are passed to xcc instead of `MODULE_XCC_FLAGS` for all `.c` files within the module.

`MODULE_XCC_ASM_FLAGS`
: If set, these flags are passed to xcc instead of `MODULE_XCC_FLAGS` for all `.s` or `.S` files within the module.

`OPTIONAL_HEADERS`
: Specifies a particular header file to be an optional configuration header. This header file does not exist in the module but is provided by the application using the module. The build system will pass the a special macro `__filename_h_exists__` to xcc if the application has provided this file. This allows the module to provide default configuration values if the file is not provided.