Alchemy Build System

Yves-Marie Morgan

Paris Embedded

13/04/2016

Why Another Build System ?

- Makefiles can be hard to write and maintain when projects are growing.
- Autotools / CMake require learning a new language and some expertise to be used correctly in big projects.
- Many developers do not have the correct knowledge / expertise to write makefiles that integrate well in complex system.
- Need to easily add a new component in an existing system (from people that are not build/integrator experts).

Alchemy

- Mix between Android build system and buildroot.
- Packages or modules are described in makefiles by mostly filling some variables.
- Can be used to generate a full linux system (root fs).
- Can be used to simply build a single application (or library) for linux/windows/macos/ios/android (the native part only) or even for micro-controllers without os.
- Can generate sdk from a full system (or partial system) to be used in a bigger system.
- Can be integrated in other build system by using or generating an alchemy sdk describing prebuilt dependencies.

Concepts taken from Android

- The workspace is searched for module definition (in atom.mk files).
- Modules can be built directly from source code to produce libraries or executables.
- Build and link dependencies are automatically handled (import/export of include directories and compilation flags).

Concepts taken from Buildroot

- The system to be built is configured with a kconfig based interface. However some target settings are set directly in product specific makefiles (to allow a kind of inheritance between products).
- Modules can be built from other build system (autotools / cmake /qmake / custom)
- Source code can be in archives and allow applying patches. However Alchemy does not propose downloading archives or source code, this is left to other tools.

Anatomy of an atom.mk

```
LOCAL_PATH := $(call my-dir)

include $(CLEAR_VARS)

LOCAL_MODULE := libpng
LOCAL_CATEGORY_PATH := libs/graphics
LOCAL_DESCRIPTION := PNG reference library

LOCAL_LIBRARIES := zlib
LOCAL_EXPORT_LDLIBS := -lpng

LOCAL_AUTOTOOLS_VERSION := 1.6.21
LOCAL_AUTOTOOLS_ARCHIVE := $(LOCAL_MODULE)-$(LOCAL_AUTOTOOLS_VERSION).tar.gz
LOCAL_AUTOTOOLS_SUBDIR := $(LOCAL_MODULE)-$(LOCAL_AUTOTOOLS_VERSION)

include $(BUILD_AUTOTOOLS)
```

Anatomy of an atom.mk

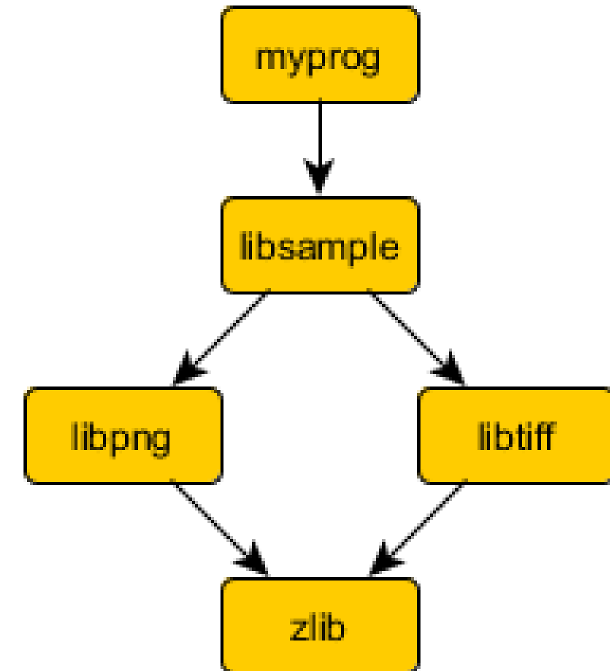
```
LOCAL_PATH := $(call my-dir)

include $(CLEAR_VARS)
LOCAL_MODULE := libsample
LOCAL_CONFIG_FILES := libsample/Config.in
$(call load-config)
LOCAL_DESCRIPTION := Sample library
LOCAL_SRC_FILES := libsample/libsample.c
LOCAL_EXPORT_C_INCLUDES := $(LOCAL_PATH)/libsample
LOCAL_LIBRARIES := libpng libtiff
include $(BUILD_SHARED_LIBRARY)

include $(CLEAR_VARS)
LOCAL_MODULE := myprog
LOCAL_DESCRIPTION := Sample program
LOCAL_SRC_FILES := myprog/myprog.c
LOCAL_LIBRARIES := libsample
include $(BUILD_EXECUTABLE)
```

Dependencies

- Each module declares the list of modules on which it depends (libraries in general).
- Each module declares headers/flags that it want to exports to other modules.
- Alchemy (using make rules) order the build and construct the complete command line required to build any module by importing headers/flags from other modules.

Dependencies Example

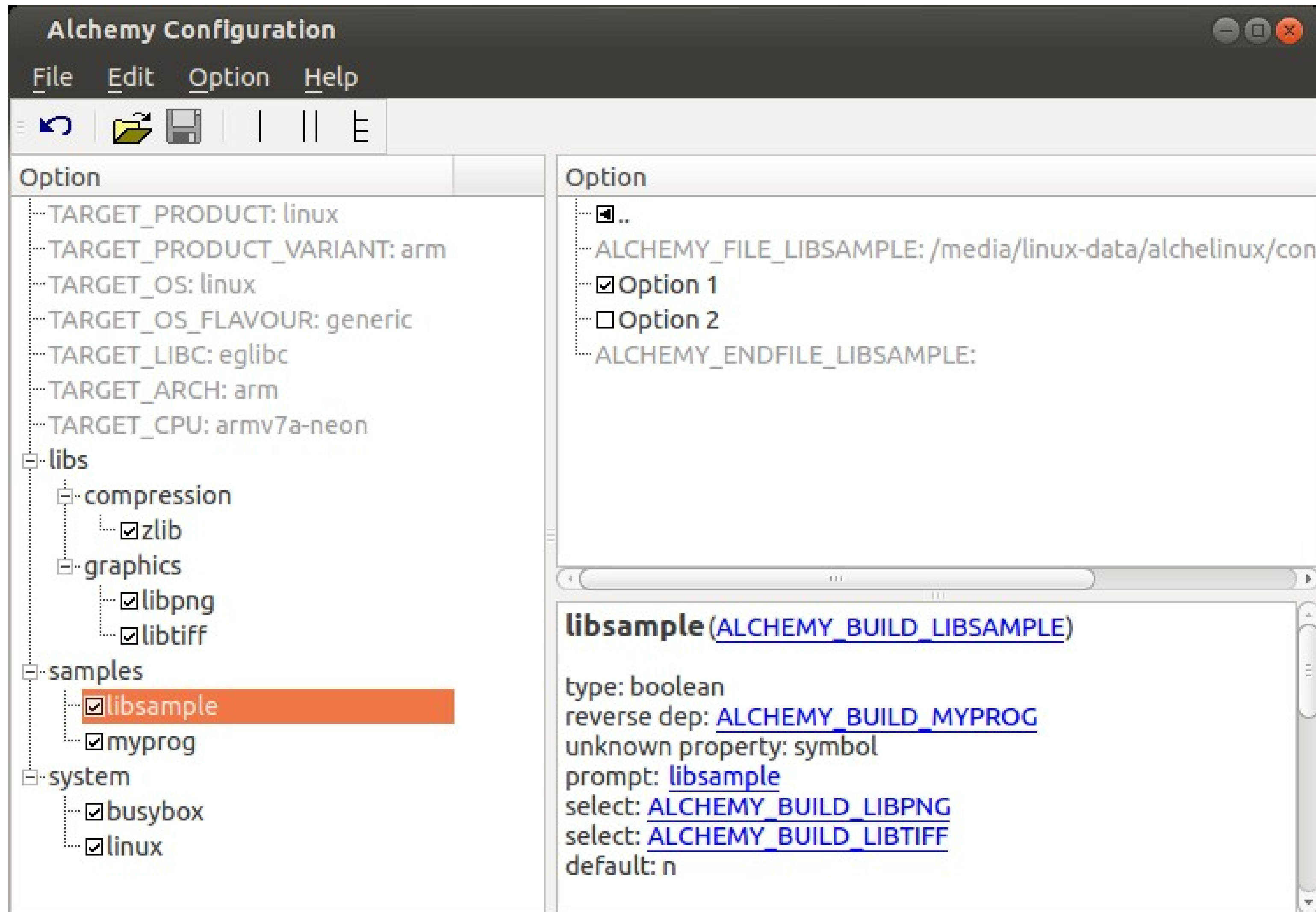
- Libsample will be compiled with headers and flags exported from libpng and libtiff
- Libsample will be linked with -lpng and -ltiff (exported by those autottols libraries)
- Myprog will be compiled with headers and flags exported from libsample (as well as libpng and libtiff in case public headers use them)
- Myprog will be link with libsample.so automatically (no need for libsample to explicitly export it)



Configuration

- Modules to be built are selected in a kconfig interface. Each module can define a description and a category for the display.
- The kconfig input is automatically generated from the atom.mk (so there is a single location for expressing dependencies)
- Any module can specifies additional custom configuration options that will be added in the kconfig interface.
- Module custom configuration will be saved in a separate file to ease maintenance and sharing of configuration.

Configuration



Resources

- Alchemy:
  - https://github.com/Parrot-Developers/alchemy
- Linux for qemu arm sample
  - https://github.com/ymorgan/alchemy-sample-linux
- Parrot Drone ARSDK using alchemy (for ios/android and linux/darwin native)
  - https://github.com/Parrot-Developers/arsdk_manifests
  - https://github.com/Parrot-Developers/Docs/blob/master/Installation/INSTALL.md
  - http://developer.parrot.com/docs/bebop/#go-deeper