Compilation Flags

Author:

Armando Emmanuel Correa Amorelli

Zapopan, Jalisco, May 7, 2023

Table of Contents

Introduction

Scope

The following activity seeks to select the appropriate compilation flags generating different results that are adapted to specific platforms and project specifications. Compilation flags are one of several ways that the resulting binary, or executable, can be configured to only be compatible with a particular architecture.

Objective

Compare build results by identifying the impact of building with different flags.

U-Boot README

Questionnaire

- What is the purpose of the project's mailing list?
  The project's mailing list serves as a means of communication for U-Boot developers and users. It is used to discuss issues, share solutions, announce new releases, and more.
- What content is in the API directory?
  The API directory contains the application programming interface (API) functions for use with U-Boot. These functions are designed to be used by other applications and programming tools.
- What content is in the boot-loader directory?
  The boot-loader directory contains the source code of the U-Boot core, which is responsible for loading the operating system onto the target device.
- What content is in the libraries directory?
  The libraries directory contains commonly used libraries in U-Boot, which can be used by other system modules.
- What content is in the licenses directory?
  The licenses directory contains open source licenses used in U-Boot.
- What are the typical functions used in the process of board initialization?
  The board initialization process typically includes initialization of memory, configuration of input/output (I/O) pins, initialization of peripherals (such as the Ethernet controller), and clock configuration.
- What is the purpose of the low-level initialization stage?
  The low-level initialization stage is the first stage of U-Boot, which runs before the system's RAM is initialized. Its purpose is to prepare the hardware so that the U-Boot code can be loaded and executed.
- What is the purpose of the board initialization stage?
  The board initialization stage is the final stage of U-Boot, which runs after the operating system has been loaded. Its purpose is to configure the hardware and necessary device drivers so that the operating system can function properly.

U-Boot Compilation

AM335x Architecture

Setting default configuration for am335x

For the set up of the default environment configuration, we need to execute the make command, specifying the compilation architecture, compiler and default configuration. In this specific case it would be as follow

```
make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi am335x_evm_defconfig
```

All that line can be simplified to a single command followed by the default configuration if we export the cross compiler option and create an alias with the following commands

```
export CROSSCC=arm-linux-gnueabi
```

```
alias armake='make ARCH=arm CROSS_COMPILE=$CROSSCC-'
```

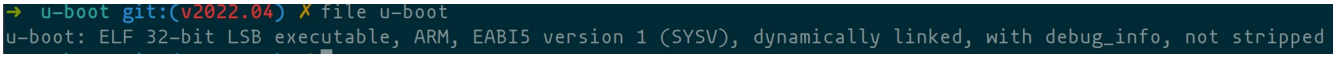
Getting as result the following

Cross compilation

Once the default configuration it's set we can continue with compilation. Because this is a binary that would be running in a different architecture we need to specify the compiler and the architecture when building with make. For this we would use the previous `armake` alias that we created to simplify the process.

Output file

To verify the correct build was performed, first, we need to look for the *u-boot* binary located in the project's root directory. Once we find it we can use the `file` command, and we should get a similar output to the one shown in the following image.



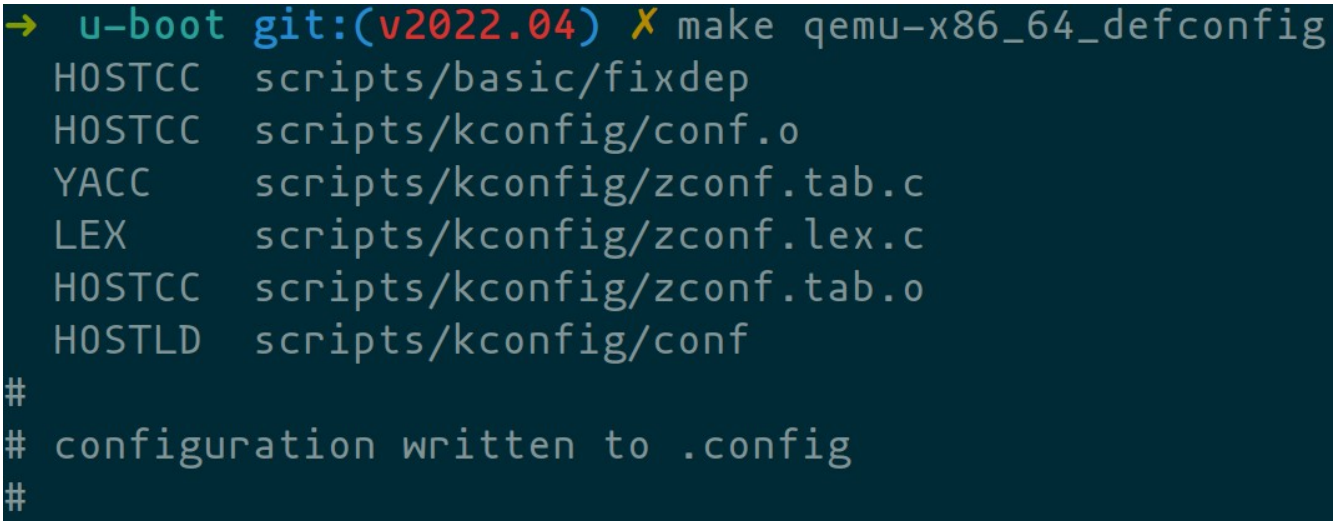
X86_64 Architecture

Setting default configuration for QEMU X86_64

For the set up of the default environment configuration, we just need to execute the make command followed by the default configuration. Because, we are building it from a x86_64 architecture.

```
make qemu-x86_64_defconfig
```

Getting as result the following



Cross compilation

Once the default configuration it's set we can continue with compilation. Because this is a binary that would be running in a same architecture we don't need to specify the compiler and the architecture when building with make. For this we would use the `make` command.

Output file

To verify the correct build was performed, first we need to look for the *u-boot* binary which should be located at the root directory of the project. Once we found it we can use the `file` command, and we should get a similar output to the one shown in the following image.

Comparison between output files

	AM335X	X86-64
Endianes	LSB	LSB
Architecture	ARM	x86-64
File format	ELF	ELF
Word size	32 bit	64 bit
Application Binary Interface	EABI5	SYSV (ABI)

Looking at the previous table we can see, there are several differences between the two files. The second file is compiled for the x86-64 architecture, while the first is compiled for the ARM architecture. Additionally, the second files indicates that format is 32-bit rather than 64-bit. However, both files are little-endian.

U-Boot ARM configuration options

U-Boot provides a wide range of configuration options related to the ARM architecture, which can be used to customize the behavior of the boot-loader and adapt it to specific hardware platforms. Some of the most important options include:

- ARCH: This option selects the target architecture for the build, which in this case would be arm.
- CROSS_COMPILE: This option specifies the prefix used for the cross-compiler tool-chain to be used for the build, such as *arm-linux-gnueabi-* or *aarch64-linux-gnu-*.
- CPU: This option specifies the specific ARM CPU variant to be targeted, such as cortex-a9, cortex-a53, cortex-m7, etc.
- TUNE: This option specifies the instruction tuning for the target CPU, such as cortex-a9 or cortex-a15.
- SOC: This option specifies the specific System-on-Chip (SoC) to be targeted, such as omap3, imx6, rk3399, etc.
- BOARD: This option specifies the specific board configuration to be used, such as beaglebone, raspberrypi, jetson-tx2, etc.
- CONFIG_SYS_EXTRA_OPTIONS`: This option allows for additional architecture-specific configuration options to be set.

These options can be set either manually in the U-Boot configuration file (.config) or via a menu-based configuration tool (make menuconfig). The specific options available may vary depending on the version of U-Boot and the specific configuration being used.

Conclusions

Through this activity I gain insight into how powerful the U-Boot project is in conjunction with the make files and all the different compilation flags provided. With the later, and with the correct tool-chain, you can build a complete boot-loader for all the supported architectures and even tweak it to the project's specifics.

Bibliography

[1]“The U-Boot Documentation — Das U-Boot unknown version documentation,” *u-boot.readthedocs.io*. https://u-boot.readthedocs.io/en/v2022.04/index.html (accessed May 07, 2023).

[2]“u-boot/u-boot,” *GitHub*, May 07, 2023. https://github.com/u-boot/u-boot (accessed May 07, 2023).

[3]“file(1) - Linux manual page,” *man7.org*. https://man7.org/linux/man-pages/man1/file.1.html (accessed May 07, 2023).