The LinuxBIOS Configuration Tool

Ron Minnich

January 16, 2001

1 All you need to know

LinuxBIOS consists of two parts: a small bootstrap, which brings the hardware up; and a Linux kernel. The kernel is built in the usual manner. The bootstrap is built in a build directory, using source components that reside in a source tree. Unlike Linux, and very much like BSD, the bootstrap is intended to be built in a directory that is the *outside* the source tree. We structure it this way so that users can easily load the newest snapshot of LinuxBIOS from the sourceforge.net server, and then build a new LinuxBIOS. The directory in which LinuxBIOS is built is called the *build* directory; the source directory is called the *source* directory, and the Linux kernel directory is called (strangely enough) the *kernel* directory.

The build directory is initially empty. Three files are needed to build a complete LInuxBIOS: a Makefile; a crt0.S (assembly startup) file; and ldscript.ld, a script for ld to link the objects into a bootstrap. Note that the bootstrap and the kernel are not linked via ld; they are totally seperated components. The Makefile contains the list of objects used to build the bootstrap; the location of the source tree; the location of the kernel; and any other commands needed to build a full LinuxBIOS romimage. The crt0.S file contains is just enough code to make the machine usable for the rest of the bootstrap, which is itself written in C. The ldscript.ld file defines address relocations for the bootstrap so it will run correctly from FLASH or Disk On Chip.

The LinuxBIOS configuration tool is used to build these three critical files. The user writes a top-level configuration file and then runs the configuration tool. The tool uses other files in the source tree to set parameters for the generated files, and then writes the configuration files for a given build.

1.1 Configuration file Format

We show an example of a configuration file in Figure 1.

We will describe this file line by line. First, we name the build directory, in this case matsonic. Next, we name the mainboard: matsonic/ms7308e. Next we set a few options, such as SERIAL_CONSOLE (enable serial console logging); UPDATE_MICROCODE (the bootstrap will load new microcode on the broken Coppermine PIIIs if needed); ENABLE_L2_CACHE (the bootstrap will

```
# Sample config file for Matsonic MS7308E with DoC Millennium (as root)

# This will make a target directory of ./matsonic
target matsonic

# Matsonic MS7308E mainboard
mainboard matsonic/ms7308e

# Enable Serial Console for debugging
option SERIAL_CONSOLE

# Enable MicroCode update and L2 Cache init for PII and PIII
option UPDATE_MICROCODE
option CONFIGURE_L2_CACHE

# Use the internal VGA frame buffer device
option HAVE_FRAMEBUFFER

# Path to your kernel (vmlinux)
linux /usr/src/linux

# Kernel command line parameters
commandline root=/dev/nftla1 single
```

Figure 1: A Sample Configuration File for the Matsonic MS7308E

Command	What it does
target	Name the build directory. (Should we remove this? seems pointless.)
mainboard	Name the mainboard to be used for this build. This command will cause the tool to look in the source tree for src/<mainboard name>/Config Commands in that Config file are processed.
option	Set an option in the Makefile
linux	Name the directory in which the kernel image (vmlinux) resides
commandline	Set the command line for Linux

correctly enable L2 cache if needed); and finally, enable the VGA frame buffer. The next command (`linux /usr/src/linux`) defines the path to vmlinux. Finally, the `commandline` command sets the command line for Linux. The minimum commandline currently required is the one which sets the root file system.

This sample configuration file is all that you will usually need. In fact, all the configuration files we have written to date involve small variations on this file.

1.2 The commands you need to know

There are only five commands used in the file above. We describe them in Table 1.2.

```
[root@maxroach /tmp]# cd /tmp
[root@maxroach /tmp]# mkdir matsonic
[root@maxroach /tmp]# cat > matsonic.config
target matsonic

# Matsonic MS7308E mainboard
mainboard matsonic/ms7308e

# Enable Serial Console for debugging
option SERIAL_CONSOLE

# Enable MicroCode update and L2 Cache init for PII and PIII
option UPDATE_MICROCODE
option CONFIGURE_L2_CACHE

# Use the internal VGA frame buffer device
option HAVE_FRAMEBUFFER

# Path to your kernel (vmlinux)
linux /usr/src/linux

# Kernel command line parameters
commandline root=/dev/hda1
[root@maxroach /tmp]# python ~/src/freebios/util/config/NLBConfig.py matsonic.config
~/src/freebios
Will place Makefile, crt0.S, ldscript.ld in  matsonic
Process config file:   /root/src/freebios/src/mainboard/matsonic/ms7308e/Config
Process config file:   /root/src/freebios/src/northsouthbridge/sis/630/Config
Added ram init file:   northsouthbridge/sis/630/raminit.inc
Process config file:   /root/src/freebios/src/cpu/p5/Config
Process config file:   /root/src/freebios/src/cpu/p6/Config
Process config file:   /root/src/freebios/src/lib/Config
Now Process the base files
Makebase is :  /root/src/freebios/util/config/make.base :
Trying to create  matsonic/Makefile
Trying to create  matsonic/ldscript.ld
Trying to create  matsonic/crt0.S
Check for crt0.S param file:  /root/src/freebios/src/include/northsouthbridge/sis/630/param.h
 Adding include to crt0.S for this parameter file:
  #include <northsouthbridge/sis/630/param.h>

[root@maxroach /tmp]#
```

Figure 2: Running the config tool. Your output may vary somewhat.

1.3 Running the configuration tool

To run the configuration tool, you need to make a build directory, then run the tool. The standard BSD config tool will delete and re-create build directories as needed, but we are not that confident. If you want the build directory removed or created, it is up to you.

We show a sample run in Figure 2. We cd to /tmp, make a build directory (matsonic), make a build file (matsonic.config), and run the config tool using Python. The tool will show the additional config files it processes. Error handling right now is not very good, but there are some errors that it will show.

That is about all you should need to know to build most Linux kernels. In the next section we will cover more detail, for those who need to know more.

Resource Name	Path
inboard	TOP/src/mainboard/<vendor>/<partname>
cpu	TOP/src/cpu/<partname>. NOTE: no vendor in this path. We did this because so much of this code is generic. Was this right?
northbridge	TOP/src/northbridge/<vendor>/<partname>
northsouthbridge	TOP/src/northsouthbridge/<vendor>/<partname> NOTE: sets path for northbridge and southbridge, since this command is for integrated chipsets (i.e. SiS 630)
southbridge	TOP/src/southbridge/<vendor>/<partname>
superio	TOP/src/superio/<vendor>/<partname>
pcibridge	TOP/src/pcibridge/<vendor>/<partname>

Table 1: How resources are named

2 More than you need to know

The config tool is written in Python, and processes files written in a little language. Most of the commands in the little language set values of variables in the program, or add elements to lists. Some of the commands result in other files being read in and processed. In most cases these are other configuration files. Commands that name hardware (such as the `mainboard` command) in most cases search for a configuration file in the path TOP/src/<name>/Config, where TOP represents the top of the source tree. For an example, see Table 1.2.

2.1 Commands that name resource directories.

Resource directories contain C source code for LinuxBIOS. The naming of resource directories in most cases is <vendor id>/<hardware>. We show the resource directory commands in Table 1.

The commands and their resulting path are shown in Table 1. Each of these commands (except for superio, currently) will check for a Config file in the directory and process that file. The Config file can contain any command.

The configuration tool keeps track of three directories: the resource directory that it is currently processing (think of this as the current working directory); the TOP of the freebios source tree; and the directory that holds the build (i.e. the target directory). All of these variables come into use when resource directory Config files are processed. For example, if the Config directory for northbridge specifies the file northbridge.c, then an object rule for northbridge.o will be added to the Makefile, with source being found in the resource directory. All source names are made relative to TOP.

2.2 Commands that modify the Makefile

There are several commands that add rules and options to the Makefile. They are shown below.

Rule Name	Result
object <name>.o	Add the object <name>.o to the Makefile, depending on TOP/current-directory/<name>.c. Add a rule for building the object using cc.
makerule <name> : [<deps>] ; [<rule>]	Add a rule for <name>, depending on <deps> (optional), with actions <rule> (optional).
addaction <rule> <action>	Add an action to an existing rule
adddepend <rule> <dependency>	Add a dependency to an existing rule
makedefine <variable>=<value>	Add a makerule to the Makefile. Note: if you just want to add a -Doption or -Uoption to the Makefile, use the option and nooption commands instead.
option <name>	add -D<name> to the Makefile CPUFLAGS
nooption	add -U<name> to the Makefile CPUFLAGS
commandline <linux-command-line>	Set the commandline for Linux. This can include spaces, e.g. commandline root=/dev/hda1 console=tty0 single
docipl <path>	Turn on Millenium Disk On Chip Makefile options. This changes variables for ldscript.ld, sets option for crt0.S, and sets the USE_DOC option in CPUFLAGS.
linux <path>	Set the path for Linux, assumed to be in <path>/vmlinux.

Table 2: Commands that modify the Makefile

Command	Action
setdata <value>	Set the base of data. Default is 0x4000.
setbss <value>	Set the base of bss. Default is 0x5000.
setstack <value>	Set the base of the stack. Default is 0x90000.
setlinuxbiosbase <value>	Set the linuxbios code base. Default is 0xf0000.

Table 3: The ldscript.ld commands

2.3 Commands that modify ldscript.ld

There are four commands that modify ldscript.ld. These commands determine the base address of the data, bss, stack, and LinuxBIOS code segments (not x86 segment registers, but ELF-style segments). These commands are currently used primarily by the docipl command.

2.4 Commands that affect crt0.S

The only commands that affect crt0.S are option commands. These set options that affect how crt0.S operates when it starts. The options that affect crt0.S are shown below. Note that except for RAMTEST, these options affect other

Option	Meaning
RAMTEST	Run a simple RAM diagnostic after initializing SDRAM
USE_DOC_MIL	This option is usually set by the docipl command, and has two main effects. It turns off the SDRAM init in crt0.S (since the docipl does it). It also modifies the range of memory used by the SDRAM test code (since we don't want SDRAM test code overwriting LinuxBIOS code in SDRAM)
SERIAL_CONSOLE	Enables serial console output in crt0.S

Table 4: Options that affect crt0.S