Freedom Embedded

Balau's technical blog on open hardware, free software and security

U-boot for ARM on QEMU

Posted on 2010/03/10

107

Das U-Boot (http://www.denx.de/wiki/U-Boot/WebHome), the universal bootloader, is a crucial piece of software that runs on embedded platforms: its role is to put in place and boot the linux kernel from a hard drive, a flash memory, network or serial line. Here I explain how to try U-Boot on QEMU (http://wiki.qemu.org/Main_Page), and in particular on the emulation of the VersatilePB (http://infocenter.arm.com/help/topic/com.arm.doc.dui0224i/index.html) platform.

First, install the necessary tools:

- qemu-system-arm: run "`apt-get install qemu`" on Debian, or "`sudo apt-get install qemu-kvm-extras`" on Ubuntu.
- mkimage: install the uboot-mkimage package from the Debian or Ubuntu repository.
- CodeSourcery ARM EABI toolchain (http://www.codesourcery.com/sgpp/lite/arm/portal/subscription?@template=lite) toolchain: download from their website and install.

Grab the U-Boot source code (ftp://ftp.denx.de/pub/u-boot/u-boot-2010.03.tar.bz2) from the U-Boot FTP site (ftp://ftp.denx.de/pub/u-boot/) and decompress it. Go inside the created directory and run:

```
make versatilepb_config ARCH=arm CROSS_COMPILE=arm-none-eabi-
```

This command configures U-Boot to be compiled for the VersatilePB board. Then compile and build with:

```
make all ARCH=arm CROSS_COMPILE=arm-none-eabi-
```

The compilation will create a `u-boot.bin` binary image. To simulate, run:

```
qemu-system-arm -M versatilepb -m 128M -nographic -kernel u-boot.bin
```

The U-Boot prompt should appear:

```
U-Boot 1.1.6 (Mar 3 2010 - 21:46:06)
```

DRAM: 0 kB
Flash: 0 kB
*** Warning – bad CRC, using default environment

In: serial
Out: serial
Err: serial
Versatile #

You can have a list of commands by entering `help`, and then try out various commands (hit "Ctrl-a" and then "x" to exit QEMU). The `bootm` command in particular is used to boot a program that is loaded in memory as a special U-Boot image, that can be created with the tool `mkimage`. This program is usually an operating system kernel, but instead of running a full-blown Linux kernel, we can instead run the simple "Hello world" program described in a previous blog post (https://balau82.wordpress.com/2010/02/28/hello-world-for-bare-metal-arm-using-qemu/). To do so, we create a single binary that contains both the U-Boot program and our "Hello world" program together. The initial address of the "Hello world" program must be changed with respect to the instructions present in the last blog post (https://balau82.wordpress.com/2010/02/28/hello-world-for-bare-metal-arm-using-qemu/), because at `0x10000` (our last initial address) QEMU places the beginning of the U-Boot binary. Since the U-Boot binary is about 100KB, we can place our binary at `0x100000` (that is 1MB) to be safe.

Create `test.c`, `startup.s` and `test.ld` as last time (https://balau82.wordpress.com/2010/02/28/hello-world-for-bare-metal-arm-using-qemu/), but change line 4 of `test.ld` from "`. = 0x10000`" to "`. = 0x100000`". Build the binary with:

```
arm-none-eabi-gcc -c -mcpu=arm926ej-s test.c -o test.o
arm-none-eabi-ld -T test.ld -Map=test.map test.o startup.o -o test.elf
arm-none-eabi-objcopy -O binary test.elf test.bin
```

Now create the U-Boot image `test.uimg` with:

```
mkimage -A arm -C none -O linux -T kernel -d test.bin -a 0x00100000 -e
0x00100000 test.uimg
```

With these options we affirm that the image is for ARM architecture, is not compressed, is meant to be loaded at address `0x100000` and the entry point is at the same address. I use "linux" as operating system and "kernel" as image type because in this way U-Boot cleans the environment before passing the control to our image: this means disabling interrupts, caches and MMU.

Now we can create a single binary simply with:

```
cat u-boot.bin test.uimg > flash.bin
```

This binary can be run instead of the U-Boot binary with:

```
qemu-system-arm -M versatilepb -m 128M -nographic -kernel flash.bin
```

at the U-Boot prompt, we can check that the image is inside the memory: it should be exactly after the u-boot.bin code. To calculate the address, we must take the size of u-boot.bin and sum the initial address where flash.bin is mapped. From the `bash` prompt, the following script prints the command to be written inside U-Boot:

```
printf "bootm 0x%X\n" $(expr $(stat -c%s u-boot.bin) + 65536)
```

in my case it prints "`bootm 0x21C68`". In fact, if I run "`iminfo 0x21C68`" inside U-Boot prompt to check the memory content I get:

```
## Checking Image at 00021c68 ...
Image Name:
Image Type:   ARM U-Boot Standalone Program (uncompressed)
Data Size:    376 Bytes =  0.4 kB
Load Address: 00100000
Entry Point:  00100000
Verifying Checksum ... OK
```

I can then confidently run "`bootm 0x21C68`" (you should substitute your address in this command). This command copies the content of the image, that is actually `test.bin`, into the address `0x100000` as specified in the U-Boot image, and then jumps to the entry point. The emulator should print "Hello world!" as last time, and then run indefinitely (hit "Ctrl-a" and then "x" to exit). This is basically the same procedure that is used to boot a Linux kernel, with some modifications: for example, the Linux kernel accepts some parameters that must be received from U-Boot somehow. I plan to write a post about that in the future.

In a real world example, the binary file we created could be placed inside the parallel Flash memory of an embedded platform, and the boot process can be controlled from the serial port.

Tagged: *ARM*, *boot*, *codesourcery*, *embedded*, *firmware*, *linux*, *qemu*, *serial*, *serial port*, *u-boot*, *uart*, *uboot*
Posted in: *Embedded (https://balau82.wordpress.com/category/software/embedded-software/)*

107 Responses "U-boot for ARM on QEMU" →

Li Zhengke

2010/06/21

After success in excuteing cmd iminfo 0x25258, I did not see the output “hello world” by excuteing cmd bootm 0x25258.The output is below:
VersatilePB # iminfo 0x25258

Checking Image at 00025258 …
Legacy image found
Image Name:
Image Type: ARM U-Boot Standalone Program (uncompressed)
Data Size: 360 Bytes = 0.4 kB
Load Address: 00100000
Entry Point: 00100000
Verifying Checksum … OK
VersatilePB # bootm 0x25258
Booting kernel from Legacy Image at 00025258 …
Image Name:
Image Type: ARM U-Boot Standalone Program (uncompressed)
Data Size: 360 Bytes = 0.4 kB
Load Address: 00100000
Entry Point: 00100000
Loading Standalone Program … OK
OK

U-Boot 2010.03 (Jun 21 2010 – 00:41:30)

DRAM: 0 kB
Unknown FLASH on Bank 1 – Size = 0x00000000 = 0 MB
Flash: 0 kB
*** Warning – bad CRC, using default environment

In: serial
Out: serial
Err: serial
Net: SMC91111-0
VersatilePB #

How can I get the correct output with hello world?

Balau

2010/06/22

Does it print “Hello world!” if you follow the instructions in the previous blog post?

If I can find time, I will try to replicate your problem.

From the U-Boot output, it seems to me that the system has been reset.

Balau

2010/06/22

Maybe I found the solution: the entry point was misplaced. Try to use this linker script file instead:

```
ENTRY(_Reset)
SECTIONS
{
. = 0x100000;
.startup . : { startup.o }
.text : { *(.text) }
.rodata : { *(.rodata) }
.data : { *(.data) }
.bss : { *(.bss) }
. = . + 0x1000; /* 4kB of stack memory */
stack_top = .;
}
```

Li Zhengke

2010/07/02

It does not work!Could you tell me which version of the qemu and u-boot you use?

Li Zhengke

2010/07/02

my qemu version is “QEMU PC emulator version 0.11.0 (qemu-kvm-0.11.0), Copyright (c) 2003-2008 Fabrice Bellard”;and u-boot is u-boot-2010.03 and u-boot-2010.06.I tried on the 2 version of u-boot ,both are not OK.

Balau

2010/07/02

I use u-boot-2010.03 as you do, but I have QEMU PC emulator version 0.12.3 (qemu-kvm-0.12.3), Copyright (c) 2003-2008 Fabrice Bellard

Could you take a look at the test.map file that is generated? It is important that the _Reset label is placed at 0x00100000. My test.map looks like this:

```
Memory Configuration

Name Origin Length Attributes
*default* 0x00000000 0xffffffff

Linker script and memory map

0x00100000 . = 0x100000

.startup 0x00100000 0xf8
startup.o()
.text 0x00100000 0x10 startup.o
0x00100000 _Reset
.data 0x00100010 0x0 startup.o
.bss 0x00100010 0x0 startup.o
.ARM.attributes
0x00100010 0x20 startup.o
.debug_line 0x00100030 0x39 startup.o
.debug_info 0x00100069 0x57 startup.o
.debug_abbrev 0x001000c0 0x14 startup.o
*fill* 0x001000d4 0x4 00
.debug_aranges
0x001000d8 0x20 startup.o
.glue_7 0x001000f8 0x0 startup.o
.glue_7t 0x001000f8 0x0 startup.o
.vfp11_veneer 0x001000f8 0x0 startup.o
.janus_2cc_veneer
0x001000f8 0x0 startup.o
.v4_bx 0x001000f8 0x0 startup.o

.text 0x001000f8 0x6c
*(.text)
.text 0x001000f8 0x6c test.o
0x001000f8 print_uart0
0x0010014c c_entry

.glue_7 0x00100164 0x0
.glue_7 0x00100164 0x0 test.o

.glue_7t 0x00100164 0x0
.glue_7t 0x00100164 0x0 test.o
```

```
.vfp11_veneer 0x00100164 0x0
.vfp11_veneer 0x00100164 0x0 test.o

.janus_2cc_veneer
0x00100164 0x0
.janus_2cc_veneer
0x00100164 0x0 test.o

.v4_bx 0x00100164 0x0
.v4_bx 0x00100164 0x0 test.o

.rodata 0x00100164 0x14
*(.rodata)
.rodata 0x00100164 0x14 test.o
0x00100164 UART0DR

.data 0x00100178 0x0
*(.data)
.data 0x00100178 0x0 test.o

.bss 0x00100178 0x4
*(.bss)
.bss 0x00100178 0x4 test.o
0x00100178 bkp
0x0010117c . = (. + 0x1000)
0x0010117c stack_top = .
LOAD test.o
LOAD startup.o
OUTPUT(test.elf elf32-littlearm)
...
```

Li Zhengke

2010/07/06

This is my test.map.the _Reset label is placed at 0×00100000 indeed.

Memory Configuration

Name Origin Length Attributes
default 0x00000000 0xffffffff

Linker script and memory map

0x00100000 . = 0x100000

```
.startup 0x00100000 0xf0
startup.o()
.text 0x00100000 0x10 startup.o
0x00100000 _Reset
.data 0x00100010 0x0 startup.o
.bss 0x00100010 0x0 startup.o
.ARM.attributes
0x00100010 0x24 startup.o
.debug_line 0x00100034 0x39 startup.o
.debug_info 0x0010006d 0x4e startup.o
.debug_abbrev 0x001000bb 0x14 startup.o
*fill* 0x001000cf 0x1 00
.debug_aranges
0x001000d0 0x20 startup.o

.text 0x001000f0 0x6c
*(.text)
.text 0x001000f0 0x6c test.o
0x001000f0 print_uart0
0x00100144 c_entry

.glue_7 0x0010015c 0x0
.glue_7 0x00000000 0x0 linker stubs

.glue_7t 0x0010015c 0x0
.glue_7t 0x00000000 0x0 linker stubs

.vfp11_veneer 0x0010015c 0x0
.vfp11_veneer 0x00000000 0x0 linker stubs

.v4_bx 0x0010015c 0x0
.v4_bx 0x00000000 0x0 linker stubs

.rodata 0x0010015c 0x14
*(.rodata)
.rodata 0x0010015c 0x14 test.o
0x0010015c UART0DR

.data 0x00100170 0x0
*(.data)
.data 0x00100170 0x0 test.o

.bss 0x00100170 0x0
*(.bss)
.bss 0x00100170 0x0 test.o
0x00101170 . = (. + 0x1000)
0x00101170 stack_top = .
```

LOAD test.o
LOAD startup.o
OUTPUT(test.elf elf32-littlearm)

.comment 0x00000000 0x2b
.comment 0x00000000 0x2b test.o

.ARM.attributes
0x00000000 0x30
.ARM.attributes
0x00000000 0x34 test.o

Balau

2010/07/06

I'm sorry I gave you incorrect information: for this exercise I tried U-Boot 1.1.6 (I was mislead by the compilation date that is displayed before the prompt) that is from 2006. I used your version of U-Boot only to boot Linux in this other post:
https://balau82.wordpress.com/2010/04/12/booting-linux-with-u-boot-on-qemu-arm/

Try to see there if you find something that can help you. When I can, I will try to replicate it with the new u-boot.

Li Zhengke

2010/07/07

Thanks your help.Hoplefully,you will give us more excellent tutorials!

Balau

2010/07/10

Ok the problem was a mkimage option. The correct command is:
"mkimage -A arm -C none -O linux -T kernel -d test.bin -a 0x00100000 -e 0x00100000 test.uimg"

When booting a "standalone" program, the new U-Boot versions have a particular mechanism that run the program inside U-Boot environment. When booting any "kernel" U-Boot resets the environment, for example disabling interrupts, caches and MMUs. In this "clean" environment our program can do whatever it wants, while as a "standalone" program it was constrained by U-Boot settings. Maybe the MMU was active so the program couldn't access the UART memory address, or maybe the start address was fixed as another value different from 0x100000.

Note: the “linux” option in the makefile is only to fool U-Boot. Actually, we don’t run an operating system but a bare metal program.

simon

2010/10/10

Thanks for sharing your experience, really a good guide into uboot and qemu!
Also I have some suggestion on booting “hello world” with uboot, I think we doesn’t need to modify the test.ld, although uboot launched to 0x1000 by qemu, but uboot will copy it self to 0x01000000, so it will be OK set the address of helloworld at 0x10000, and I have pass the test.
————————
qemu-system-arm -M versatilepb -m 128M -nographic -kernel flash.bin
U-Boot 1.1.6 (Oct 10 2010 – 20:38:27)
DRAM: 0 kB
Flash: 0 kB
*** Warning – bad CRC, using default environment
In: serial
Out: serial
Err: serial
Versatile # bootm 0x90000
Booting image at 00090000 …
Image Name:
Image Type: ARM Linux Kernel Image (uncompressed)
Data Size: 400 Bytes = 0.4 kB
Load Address: 00010000
Entry Point: 00010000
OK
Starting kernel …
Hello world!
————————

the only question is we need to ensure the space from 0x10000 to address where helloworld should be larger than the size itself.

Adithya

2011/01/24

I enter this command in the U-boot directory:
make versatilepb_config ARCH=arm
CROSS_COMPILE=/home/aditz/CodeSourcery/Sourcery_G++_Lite/bin/arm-none-eabi-
Output:
Generating include/autoconf.mk
include/common.h:269:29: fatal error: asm/mach-types.h: No such file or directory

compilation terminated.
Generating include/autoconf.mk.dep
include/common.h:269:29: fatal error: asm/mach-types.h: No such file or directory
compilation terminated.
Configuring for versatile board…
Variant:: PB926EJ-S
Can u plz tell me where is this "asm" directory.
And how can i solve this problem

Balau

2011/01/24

The "include/asm" directory is a link to another directory. If you are using U-Boot 2010.03 then the link points to "u-boot-2010.03/include/asm-arm". If you are using U-Boot 2010.09 then the link points to "u-boot-2010.09/arch/arm/include/asm". The link is done during configuration ("make versatilepb_config …" in your case). Your output is strange because in my case the "Generating include/autoconf.mk" is done when I run the "make all …" command. Did you do a "make clean" or "make distclean" without specifying "ARCH" or "CROSS_COMPILE" ? because they tend to generate strange results. Try to do it again from a freshly unzipped source.

Adithya

2011/01/25

Thanks that worked !!!

Adithya

2011/01/25

In the command below:
"make versatilepb_config ARCH=arm CROSS_COMPILE=arm-none-eabi-"
where is the confg file "versatilepb_config" located ?
Can u plz tell me whether u-boot has a config file for realview-pbx-a9 ?

Balau

2011/01/25

The configuration files are header files in "include/configs" directory.
You can also check the various configuration files with the command "grep _config Makefile"
U-Boot currently has no support for that board, I think the people who could someday be able

to develop it are the people in Linaro Kernel Consolidation Working Group, since they are currently working to support platforms with multiple Cortex-A9.

Kris

2011/03/09

Hi Balau,
Thanks a lot for excellent tutorials.
There is an issue while trying to build latest sources of U-boot. Build fails because of couple of undefined symbols.
U-boot source version/date: u-boot-2010.12
QEMU emulator version: 0.14.0
————————————————————————-
kris@kris-laptop:~/bld_uboot/u-boot-2010.12$ make versatilepb_config ARCH=arm CROSS_COMPILE=arm-none-eabi-
Generating include/autoconf.mk
Generating include/autoconf.mk.dep
Configuring for versatile board…
Variant:: PB926EJ-S

kris@kris-laptop:~/bld_uboot/u-boot-2010.12$ make all ARCH=arm CROSS_COMPILE=arm-none-eabi-

[…]

make[1]: Entering directory `/home/kris/bld_uboot/u-boot-2010.12/arch/arm/lib'
arm-none-eabi-gcc -g -Os -fno-common -ffixed-r8 -msoft-float -D__KERNEL__ -DCONFIG_SYS_TEXT_BASE=0x01000000 -I/home/kris/bld_uboot/u-boot-2010.12/include -fno-builtin -ffreestanding -nostdinc -isystem /home/kris/CodeSourcery/Sourcery_G++_Lite/bin/../lib/gcc/arm-none-eabi/4.5.1/include -pipe -DCONFIG_ARM -D__ARM__ -marm -mabi=aapcs-linux -mno-thumb-interwork -march=armv5te -Wall -Wstrict-prototypes -fno-stack-protector \
-o board.o board.c -c
board.c: In function '__dram_init_banksize':
board.c:233:29: error: 'CONFIG_SYS_SDRAM_BASE' undeclared (first use in this function)
board.c:233:29: note: each undeclared identifier is reported only once for each function it appears in
board.c: In function 'board_init_f':
board.c:279:18: error: 'CONFIG_SYS_INIT_SP_ADDR' undeclared (first use in this function)
————————————————————————-

As a quick fix, by defining those two symbols to be a dummy value of zero in the header file, ~/bld_uboot/u-boot-2010.12/arch/arm/include/asm/config.h, and including it board.c source file , I was able to build uboot.bin. But,
qemu-system-arm -M versatilepb -m 128M -nographic -kernel u-boot.bin

didn't show any U-boot prompt, there was no output of any kind, and qemu had to be terminated.

I would appreciate your comments on correct settings for those symbols, and any inputs to create a proper u-boot.bin with above said versions of sources and tools. Thanks.

Kris

Balau

2011/03/10

U-Boot developers recently changed the booting procedure for ARM cores, maybe they haven't cleaned up the versatilepb build.
regarding your two symbols, I think CONFIG_SYS_SDRAM_BASE should work at 0, but CONFIG_SYS_INIT_SP_ADDR surely has the wrong value! This is because CONFIG_SYS_INIT_SP_ADDR is the initial address of the stack pointer, and the stack grows from top to bottom. Try to use 0x8000000, which is 128 MegaBytes. In this way you should place the stack at the end of the available memory.

Kris

2011/03/11

Thanks for your inputs. When CONFIG_SYS_INIT_SP_ADDR was set to 0×8000000, there was a different outcome as seen below.

————————————————————
kris@kris-laptop:~/bld_uboot/u-boot-2010.12$ qemu-system-arm -M versatilepb -m 128M -nographic -kernel u-boot.bin
qemu: fatal: Trying to execute code outside RAM or ROM at 0xff000700

R00=fffcbf70 R01=ffff0000 R02=00000000 R03=01000000
R04=ffff0000 R05=fffcbf70 R06=ffff0000 R07=00000000
R08=08000000 R09=feff0000 R10=0101b1d4 R11=00000000
R12=fffcbfe8 R13=fffcbf68 R14=ff000700 R15=ff000700
PSR=600001d3 -ZC- A svc32
Aborted
————————————————————

While looking for some pointers about "outside RAM or ROM" error case, learnt that "success (of U-boot binary) is dependent on the exact combination of emulated machine and version of QEMU." from the discussion on the following page.

http://www.mail-archive.com/u-boot@lists.denx.de/msg42387.html

Instead of trying with different source versions of U-boot, got the U-boot binary for Versatile PB from the following site

http://arm.com/community/software-enablement/linux.php

As seen in the following log, the downloaded version (U-Boot 2010.09-rc2 (Sep 27 2010 – 07:21:34)) is working file on QEMU emulator version: 0.14.0, that I have.

————————————————

kris@kris-laptop:~/bld_uboot/u-boot-2010.12$ qemu-system-arm -M versatilepb -m 128M -nographic -kernel u-boot_bin_u-boot_versatilepb.axf

U-Boot 2010.09-rc2 (Sep 27 2010 – 07:21:34)
Code cloned from branch 090728_armdevCS of git://linux-arm.org/u-boot-armdev.git
Release AEL-5.0
Remote commit cloned UNKNOWN
Latest commit locally UNKNOWN
git state UNKNOWN
DRAM: 0 Bytes
Unknown FLASH on Bank 1 – Size = 0x00000000 = 0 MB
Flash: 0 Bytes
*** Warning – bad CRC, using default environment
In: serial
Out: serial
Err: serial
Net: SMC91111-0
Hit any key to stop autoboot: 0
Wrong Image Format for bootm command
ERROR: can't get kernel image!
VersatilePB #
————————————————-

Now, Having a working combination of U-boot, QEMU for Versatile PB, I tried to run the "Hello World" program as per the current tutorial. But, not yet successful, because of the error "Unknown image format", as shown in the following log. Even tried with a u-boot.uimg, derived from uboot.bin, instead of test.uimg to creat flash.bin, but the error was same.

I am trying to understand why U-boot was looking for an image at 0x00007fc0 by default.

————————————————

kris@kris-laptop:~/bld_uboot/u-boot-2010.12$ qemu-system-arm -M versatilepb -m 128M -nographic -kernel flash.bin

U-Boot 2010.09-rc2 (Sep 27 2010 – 07:21:34)
[…]
DRAM: 0 Bytes
Unknown FLASH on Bank 1 – Size = 0x00000000 = 0 MB

Flash: 0 Bytes
*** Warning – bad CRC, using default environment
In: serial
Out: serial
Err: serial
Net: SMC91111-0
Hit any key to stop autoboot: 0
Wrong Image Format for bootm command
ERROR: can't get kernel image!

VersatilePB # iminfo
Checking Image at 00007fc0 …
Unknown image format!

VersatilePB # iminfo 0x262f4
Checking Image at 000262f4 …
Unknown image format!
VersatilePB # QEMU: Terminated
————————————————————

Kris

Balau

2011/03/13

I think that U-boot is looking for an image at 0x00007fc0 by default because the "arm.com" version has been compiled with that configuration. Also, I see that you are trying to use a "axf" file, which is an executable file generated from ARM RVDS toolchain, and not a real binary file. I suppose the U-Boot source that arm.com is using to compile is heavily patched because the GCC toolchain for which U-Boot is designed to be built is very different. For these reasons I don't suggest using the pre-compiled images but build one yourself.

In my blog post I'm using the 2010.03 version that you can compile yourself to have full control of the configuration. When you see that the version works, you can go up one version at a time (2010.06, then 2010.09, then 2010.12) and see if it still works. My tutorial is valid for version 2010.03, and if it doesn't work for other versions it means that the U-Boot developers have changed something; unfortunately I have little time to keep myself updated on all the U-Boot development.

Kris

2011/03/15

Hi Balau, thanks for taking time out to suggest on the possible issues with pre-compiled binaries, and those arising out of version incompatibles. I was able to run U-boot on QEMU successfully by sticking to the respective versions mentioned in the tutorial.

By the way, in my experiment with pre-compiled version of U-boot, I'd coverted .axf to .bin the following way, and used it to create flash.bin
kris@kris-laptop:~/bld_uboot/u-boot-2010.12$ arm-none-eabi-objcopy -O binary u-boot_bin_u-boot_versatilepb.axf uboot.bin

Kris

Adithya

2011/04/07

Hello balau,

In the belo statement:

printf "bootm 0x%X\n" $(expr $(stat -c%s u-boot.bin) + 65536)

What does the value 65536 indicate ???

Regards
B. Adithya

Adithya

2011/04/07

Hello balau,

In the below statement:

arm-none-eabi-ld -T test.ld -Map=test.map test.o startup.o -o test.elf

1) What if -Map=test.Map option used? I guess it was not specified during the bare metal arm program.
2) Where is this file test.Map located.

Regards
B. Adithya

Balau

2011/04/07

In the statement:
`printf "bootm 0x%X\n" $(expr $(stat -c%s u-boot.bin) + 65536)`
the value 65536 (`0x10000` hexadecimal) is used as an offset because QEMU, when you pass a binary with the "`-kernel`" option, puts the binary at address `0x10000` that marks the absolute address of the beginning of `u-boot.bin`. The absolute address of the beginning of `test.uimg` is 0x10000 plus the size of `u-boot.bin`, because it is placed exactly after by the "`cat`" command.

The `test.map` file is an optional output of the linking step. It could be useful to understand where the various functions and variables are put, but it is not necessary to generate it, and the output file name can be anything. It should be generated in the same directory where the "`arm-none-eabi-ld`" command is executed, so in this case it should be the same directory as test.ld, test.elf, …

A.Ramachandran

2011/04/22

Thank you for your work. when i try to make using following command
make all ARCH=arm CROSS_COMPILE=arm-none-linux-gnueabi-
i got error
Fatal error: Invalid -march= option: `armv4'
i am using fedora 14, complete report
[root@localhost u-boot-1.1.6]# make all ARCH=arm CROSS_COMPILE=arm-none-linux-gnueabi-
for dir in tools examples post post/cpu ; do make -C $dir _depend ; done
make[1]: Entering directory `/home/Raman/s3c2440/u-boot-1.1.6/tools'
make[1]: Nothing to be done for `_depend'.
make[1]: Leaving directory `/home/Raman/s3c2440/u-boot-1.1.6/tools'
make[1]: Entering directory `/home/Raman/s3c2440/u-boot-1.1.6/examples'
make[1]: Nothing to be done for `_depend'.
make[1]: Leaving directory `/home/Raman/s3c2440/u-boot-1.1.6/examples'
make[1]: Entering directory `/home/Raman/s3c2440/u-boot-1.1.6/post'
make[1]: Nothing to be done for `_depend'.
make[1]: Leaving directory `/home/Raman/s3c2440/u-boot-1.1.6/post'
make[1]: Entering directory `/home/Raman/s3c2440/u-boot-1.1.6/post/cpu'
make[1]: Nothing to be done for `_depend'.
make[1]: Leaving directory `/home/Raman/s3c2440/u-boot-1.1.6/post/cpu'
make -C tools all
make[1]: Entering directory `/home/Raman/s3c2440/u-boot-1.1.6/tools'
make[1]: Leaving directory `/home/Raman/s3c2440/u-boot-1.1.6/tools'
make -C examples all
make[1]: Entering directory `/home/Raman/s3c2440/u-boot-1.1.6/examples'
arm-none-linux-gnueabi-gcc -g -Os -fno-strict-aliasing -fno-common -ffixed-r8 -msoft-float -D__KERNEL__ -DTEXT_BASE=0x01000000 -I/home/Raman/s3c2440/u-boot-1.1.6/include -fno-

builtin -ffreestanding -nostdinc -isystem include -pipe -DCONFIG_ARM -D__ARM__ -march=armv4 -mabi=apcs-gnu -Wall -Wstrict-prototypes -c -o hello_world.o hello_world.c
Assembler messages:
Fatal error: Invalid -march= option: `armv4'
hello_world.c:1:0: warning: target CPU does not support interworking
make[1]: *** [hello_world.o] Error 2
make[1]: Leaving directory `/home/Raman/s3c2440/u-boot-1.1.6/examples'
make: *** [examples] Error 2
[root@localhost u-boot-1.1.6]#
please tell me how to clear this error

william estrada

2011/04/22

A.Ramachandr
I am running Fedora 13 and I use "arm-gp2x-linux-".

Balau

2011/04/23

I tried to run the command that compiles hello_world.c, and it gets compiled with the "CPU does not support interworking" warning but without errors.
It seems that your error does not come from the compiler but from the assembler.
What's your compiler version? mine is the following:

```
$ arm-none-linux-gnueabi-gcc --version
arm-none-linux-gnueabi-gcc (Sourcery G++ Lite 2010q1-202) 4.4.1
...
$ arm-none-linux-gnueabi-as --version
GNU assembler (Sourcery G++ Lite 2010q1-202) 2.19.51.20090709
...
```

A.Ramachandran

2011/04/23

For assembler

[root@localhost u-boot-1.1.6]# arm-none-linux-gnueabi-as –version

GNU assembler (Sourcery G++ Lite 2010.09-50) 2.20.51.20100809
Copyright 2010 Free Software Foundation, Inc.
This program is free software; you may redistribute it under the terms of

the GNU General Public License version 3 or later.
This program has absolutely no warranty.
This assembler was configured for a target of `arm-none-linux-gnueabi'.
[root@localhost u-boot-1.1.6]#

A.Ramachandran

2011/04/25

Compiler version is 4.5.1

arm-none-linux-gnueabi-gcc (Sourcery G++ Lite 2010.09-50) 4.5.1
Copyright (C) 2010 Free Software Foundation, Inc.
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

After redefined the g cc path and compile, i got the following error

arm-none-linux-gnueabi-ld: error: Source object /usr/local/arm/arm-2010.09/bin/../lib/gcc/arm-none-linux-gnueabi/4.5.1/libgcc.a(_bswapsi2.o) has EABI version 5, but target u-boot has EABI version 0
arm-none-linux-gnueabi-ld: failed to merge target specific data of file /usr/local/arm/arm-2010.09/bin/../lib/gcc/arm-none-linux-gnueabi/4.5.1/libgcc.a(_bswapsi2.o)
/usr/local/arm/arm-2010.09/bin/../lib/gcc/arm-none-linux-gnueabi/4.5.1/libgcc.a(_bswapsi2.o):(.ARM.exidx+0x0): undefined reference to `__aeabi_unwind_cpp_pr0'
make: *** [u-boot] Error 1

can you tell me how to resolve it.

Balau

2011/04/26

I found this link: http://www.mail-archive.com/u-boot@lists.denx.de/msg32245.html
It seems that the old version of U-Boot could generate your linker error.
Is there a particular reason for you to use the U-Boot 1.1.6? If you can use a newer version, I suggest downloading something like version 2010.09 from U-Boot FTP archive. If you need to use version 1.1.6, you can add an empty function like the patch that you can find in the previous link. The "unwind" functions implement "exceptions" in case for example of division by zero, so they are not necessary to make U-Boot work, they are called only in special errors.

Stan Wu

2011/06/30

It’s worked for me

```
VersatilePB # bootm 0x250E0
## Booting kernel from Legacy Image at 000250e0 …
Image Name:
Image Type: ARM Linux Kernel Image (uncompressed)
Data Size: 144 Bytes = 0.1 kB
Load Address: 00100000
Entry Point: 00100000
Loading Kernel Image … OK
OK

Starting kernel …

Hello world!
```

Ref

2011/07/01

```
$ mkimage -A arm -C none -O linux -T kernel -d test.bin -a 0×00100000 -e 0×00100000 test.uimg
mkimage: invalid load address 0×00100000
```

— — — — —

test.ld:

```
$ cat test.ld
ENTRY(_Reset)
SECTIONS
{
. = 0x100000;
.startup . : { startup.o }
.text : { *(.text) }
.rodata : { *(.rodata) }
.data : { *(.data) }
.bss : { *(.bss) }
. = . + 0x1000; /* 4kB of stack memory */
stack_top = .;
}
```

Ref

2011/07/01

Nevermind. Worked. As usual, typo.

gangadhar

2011/10/11

hi
how we can findout uboot source am using ltib.i installed uboot,but i unable to find source file uboot

Balau

2011/10/11

I am sorry but I have no experience of LTIB. I have quickly read this manual and it seems u-boot is probably downloaded in `/opt/freescale/pkgs/`

Rajesh G

2011/10/21

Hi … I ve been reading articles in ur blog … And they are very nice !!

Thought i should mention …. had to use the following instead of given !!

make versatilepb_config ARCH=arm
CROSS_COMPILE=~/CodeSourcery/Sourcery_G++_Lite/bin/arm-none-eabi-

Because CodeSourcery installed there by default !!

Balau

2011/10/21

Yeah, in my case when I installed CodeSourcery as a binary executable it modified the .bashrc script to update the PATH variable. If it is not done, then you need to supply the full path to the CROSS_COMPILE as you did.

Thanks for the clarification.

Rajesh G

2011/10/22

Hi .. thanks for the reply … that was fast !!

Plz forgive my noobness … !!
I could get the first two steps (ie) make and make all correctly in u-boot version 2011.09 !!
But it says can't load into RAM or something in the QEMU step (third step) !!
I understand from the comments that various u-boot versions doesn't work !!
But just wanted to ask is that the only reason this is happening to me ??
If i try u-boot 2010.03, will the qemu boot ??

Balau

2011/10/22

At some point U-Boot developers changed the way ARM architectures are managed. But now some old architectures do not work well because U-Boot developers don't have the hardware available, and the Versatile is one of them. So if you use a version of U-Boot before that change (2010.03 or 2010.09), it should work.

jim zhang

2011/10/30

Balau:
I was following the steps in this tutorial, and had problem with following commad:
linuxplayer@ubuntu:~/u-boot-2011.09$ qemu-system-arm -M versatilepb -m 128M -nographic -kernel u-boot.bin
qemu: fatal: Trying to execute code outside RAM or ROM at 0xffff0700

R00=fffcbf70 R01=ffff0000 R02=00000000 R03=0101b9ac
R04=ffff0000 R05=fffcbf70 R06=ffff0000 R07=00000000
R08=008fff78 R09=feff0000 R10=0101b9ac R11=00000000
R12=fffcbfe8 R13=fffcbf60 R14=ffff0700 R15=ffff0700
PSR=600001d3 -ZC- A svc32
Aborted

My question is: where is the kernel image, and how qemu load kernel image?
linuxplayer@ubuntu:~/u-boot-2011.09$ ls
api config.mk drivers MAINTAINERS nand_spl rules.mk u-boot
arch COPYING examples MAKEALL net snapshot.commit u-boot.bin
board CREDITS fs Makefile onenand_ipl spl u-boot.lds
boards.cfg disk include mkconfig post System.map u-boot.map
common doc lib mmc_spl README tools u-boot.srec
linuxplayer@ubuntu:~/u-boot-2011.09$

I am using qemu-0.15.1.
Thank you very much for your help!

Balau

2011/10/31

As I wrote earlier, the problem of new versions of U-Boot is that they don't work anymore with "old" hardware. In my opinion your "problem" is that you simply are using a version of U-Boot that is too new.

About the kernel image question, if you are asking about the Linux kernel, well there's no Linux. Instead, there's a small "Hello world" program described in a previous blog post. If you want to boot Linux you can follow this post.

linux player

2011/11/03

if I am using i.mx35 hardware, what board can I use in u-boot and qemu to emulate it? Will a Realview board work? I am using latest u-boot in my project, and qemu has to match my real project. Your opinion is highly appreciated.

Balau

2011/11/03

Unfortunately QEMU does not support that hardware. You can see the list of ARM hardware that can be emulated by using "`qemu-system-arm -M ?`"
If you want to emulate an ARM11 processor like the one inside the i.mx35, then QEMU can do it with the "`-cpu arm1136`" option, but be aware that the peripherals and the memory map can be completely different.

chandan

2012/02/09

getting error with
make all ARCH=arm CROSS_COMPILE=arm-none-eabi-

Result:

for dir in tools examples/standalone examples/api arch/arm/cpu/arm926ejs /root/uboot/arch/arm/cpu/arm926ejs/ ; do \
make -C $dir _depend ; done
make[1]: ./CodeSourcery/Sourcery_G++_Lite/bin/arm-none-linux-gnueabi-gcc: Command not found

```
make[1]: Entering directory `/root/uboot/tools'
make[1]: Nothing to be done for `_depend'.
make[1]: Leaving directory `/root/uboot/tools'
make[1]: ./CodeSourcery/Sourcery_G++_Lite/bin/arm-none-linux-gnueabi-gcc: Command not found
make[1]: Entering directory `/root/uboot/examples/standalone'
/bin/sh: ./CodeSourcery/Sourcery_G++_Lite/bin/arm-none-linux-gnueabi-gcc: No such file or directory
dirname: missing operand
Try `dirname –help' for more information.
make[1]: Nothing to be done for `_depend'.
make[1]: Leaving directory `/root/uboot/examples/standalone'
make[1]: ./CodeSourcery/Sourcery_G++_Lite/bin/arm-none-linux-gnueabi-gcc: Command not found
make[1]: Entering directory `/root/uboot/examples/api'
/bin/sh: ./CodeSourcery/Sourcery_G++_Lite/bin/arm-none-linux-gnueabi-gcc: No such file or directory
dirname: missing operand
Try `dirname –help' for more information.
make[1]: Nothing to be done for `_depend'.
make[1]: Leaving directory `/root/uboot/examples/api'
make[1]: ./CodeSourcery/Sourcery_G++_Lite/bin/arm-none-linux-gnueabi-gcc: Command not found
make[1]: Entering directory `/root/uboot/arch/arm/cpu/arm926ejs'
make[1]: Nothing to be done for `_depend'.
make[1]: Leaving directory `/root/uboot/arch/arm/cpu/arm926ejs'
make[1]: ./CodeSourcery/Sourcery_G++_Lite/bin/arm-none-linux-gnueabi-gcc: Command not found
make[1]: Entering directory `/root/uboot/arch/arm/cpu/arm926ejs'
make[1]: Nothing to be done for `_depend'.
make[1]: Leaving directory `/root/uboot/arch/arm/cpu/arm926ejs'
make -C tools all
make[1]: ./CodeSourcery/Sourcery_G++_Lite/bin/arm-none-linux-gnueabi-gcc: Command not found
make[1]: Entering directory `/root/uboot/tools'
make[1]: Leaving directory `/root/uboot/tools'
make -C examples/standalone all
make[1]: ./CodeSourcery/Sourcery_G++_Lite/bin/arm-none-linux-gnueabi-gcc: Command not found
make[1]: Entering directory `/root/uboot/examples/standalone'
/bin/sh: ./CodeSourcery/Sourcery_G++_Lite/bin/arm-none-linux-gnueabi-gcc: No such file or directory
dirname: missing operand
Try `dirname –help' for more information.
./CodeSourcery/Sourcery_G++_Lite/bin/arm-none-linux-gnueabi-gcc -g -Os -fno-common -
```

ffixed-r8 -msoft-float -D__KERNEL__ -DCONFIG_SYS_TEXT_BASE=0x01000000 -I/root/uboot/include -fno-builtin -ffreestanding -nostdinc -isystem -pipe -DCONFIG_ARM -D__ARM__ -march=armv5te -Wall -Wstrict-prototypes -o hello_world.o hello_world.c -c
make[1]: ./CodeSourcery/Sourcery_G++_Lite/bin/arm-none-linux-gnueabi-gcc: Command not found
make[1]: *** [hello_world.o] Error 127
make[1]: Leaving directory `/root/uboot/examples/standalone'
make: *** [examples/standalone] Error 2

Balau

2012/02/09

Check that you have in the `PATH` environment variable the directory containing `arm-none-eabi-gcc` or the toolchain you are using; usually when you install CodeSourcery it creates the toolchain executables in directory "`~/CodeSourcery/Sourcery_G++_Lite/bin/`".
It's very strange because even if you call `make` with "`CROSS_COMPILE=arm-none-eabi-`" it tries to compile with `arm-none-linux-gnueabi-gcc` instead.

chandan

2012/02/12

getting error with following command:

make all ARCH=arm CROSS_COMPILE=./CodeSourcery/Sourcery_G++_Lite/bin/arm-none-linux-gnueabi-

Result:

Generating include/autoconf.mk
Generating include/autoconf.mk.dep
./CodeSourcery/Sourcery_G++_Lite/bin/arm-none-linux-gnueabi-gcc -DDO_DEPS_ONLY \
-g -Os -fno-common -ffixed-r8 -msoft-float -D__KERNEL__ -DCONFIG_SYS_TEXT_BASE=0x01000000 -I/root/uboot/include -fno-builtin -ffreestanding -nostdinc -isystem /root/uboot/CodeSourcery/Sourcery_G++_Lite/bin/../lib/gcc/arm-none-linux-gnueabi/4.5.1/include -pipe -DCONFIG_ARM -D__ARM__ -marm -mabi=aapcs-linux -mno-thumb-interwork -march=armv5te -Wall -Wstrict-prototypes -fno-stack-protector \
-o lib/asm-offsets.s lib/asm-offsets.c -c -S
Generating include/generated/generic-asm-offsets.h
tools/scripts/make-asm-offsets lib/asm-offsets.s include/generated/generic-asm-offsets.h
for dir in tools examples/standalone examples/api arch/arm/cpu/arm926ejs /root/uboot/arch/arm/cpu/arm926ejs/ ; do \
make -C $dir _depend ; done
make[1]: ./CodeSourcery/Sourcery_G++_Lite/bin/arm-none-linux-gnueabi-gcc: Command not

```
found
make[1]: Entering directory `/root/uboot/tools'
make[1]: Nothing to be done for `_depend'.
make[1]: Leaving directory `/root/uboot/tools'
make[1]: ./CodeSourcery/Sourcery_G++_Lite/bin/arm-none-linux-gnueabi-gcc: Command not found
make[1]: Entering directory `/root/uboot/examples/standalone'
/bin/sh: ./CodeSourcery/Sourcery_G++_Lite/bin/arm-none-linux-gnueabi-gcc: No such file or directory
dirname: missing operand
Try `dirname –help' for more information.
make[1]: Nothing to be done for `_depend'.
make[1]: Leaving directory `/root/uboot/examples/standalone'
make[1]: ./CodeSourcery/Sourcery_G++_Lite/bin/arm-none-linux-gnueabi-gcc: Command not found
make[1]: Entering directory `/root/uboot/examples/api'
/bin/sh: ./CodeSourcery/Sourcery_G++_Lite/bin/arm-none-linux-gnueabi-gcc: No such file or directory
dirname: missing operand
Try `dirname –help' for more information.
make[1]: Nothing to be done for `_depend'.
make[1]: Leaving directory `/root/uboot/examples/api'
make[1]: ./CodeSourcery/Sourcery_G++_Lite/bin/arm-none-linux-gnueabi-gcc: Command not found
make[1]: Entering directory `/root/uboot/arch/arm/cpu/arm926ejs'
make[1]: Nothing to be done for `_depend'.
make[1]: Leaving directory `/root/uboot/arch/arm/cpu/arm926ejs'
make[1]: ./CodeSourcery/Sourcery_G++_Lite/bin/arm-none-linux-gnueabi-gcc: Command not found
make[1]: Entering directory `/root/uboot/arch/arm/cpu/arm926ejs'
make[1]: Nothing to be done for `_depend'.
make[1]: Leaving directory `/root/uboot/arch/arm/cpu/arm926ejs'
make -C tools all
make[1]: ./CodeSourcery/Sourcery_G++_Lite/bin/arm-none-linux-gnueabi-gcc: Command not found
make[1]: Entering directory `/root/uboot/tools'
gcc -g -Wall -Wstrict-prototypes -O2 -fomit-frame-pointer -idirafter /root/uboot/include -idirafter /root/uboot/include2 -idirafter /root/uboot/include -I /root/uboot/lib/libfdt -I /root/uboot/tools -DCONFIG_SYS_TEXT_BASE=0x01000000 -DUSE_HOSTCC -D__KERNEL_STRICT_NAMES -c -o env_embedded.o /root/uboot/common/env_embedded.c
gcc -Wall -Wstrict-prototypes -O2 -fomit-frame-pointer -idirafter /root/uboot/include -idirafter /root/uboot/include2 -idirafter /root/uboot/include -I /root/uboot/lib/libfdt -I /root/uboot/tools -DCONFIG_SYS_TEXT_BASE=0x01000000 -DUSE_HOSTCC -D__KERNEL_STRICT_NAMES -pedantic -o envcrc.o envcrc.c -c
gcc -Wall -Wstrict-prototypes -O2 -fomit-frame-pointer -idirafter /root/uboot/include -idirafter
```

/root/uboot/include2 -idirafter /root/uboot/include -I /root/uboot/lib/libfdt -I /root/uboot/tools -DCONFIG_SYS_TEXT_BASE=0x01000000 -DUSE_HOSTCC -D__KERNEL_STRICT_NAMES -pedantic -o envcrc crc32.o env_embedded.o envcrc.o sha1.o
make[1]: Leaving directory `/root/uboot/tools'
make -C examples/standalone all
make[1]: ./CodeSourcery/Sourcery_G++_Lite/bin/arm-none-linux-gnueabi-gcc: Command not found
make[1]: Entering directory `/root/uboot/examples/standalone'
/bin/sh: ./CodeSourcery/Sourcery_G++_Lite/bin/arm-none-linux-gnueabi-gcc: No such file or directory
dirname: missing operand
Try `dirname –help' for more information.
./CodeSourcery/Sourcery_G++_Lite/bin/arm-none-linux-gnueabi-gcc -g -Os -fno-common -ffixed-r8 -msoft-float -D__KERNEL__ -DCONFIG_SYS_TEXT_BASE=0x01000000 -I/root/uboot/include -fno-builtin -ffreestanding -nostdinc -isystem -pipe -DCONFIG_ARM -D__ARM__ -march=armv5te -Wall -Wstrict-prototypes -o hello_world.o hello_world.c -c
make[1]: ./CodeSourcery/Sourcery_G++_Lite/bin/arm-none-linux-gnueabi-gcc: Command not found
make[1]: *** [hello_world.o] Error 127
make[1]: Leaving directory `/root/uboot/examples/standalone'
make: *** [examples/standalone] Error 2

Please help

chandan

2012/02/12

so sorry to post again the same query.

chandan

2012/02/12

my ./CodeSourcery/Sourcery_G++_Lite/bin/ directory contains arm-none-linux-gnueabi-gcc, arm-none-linux-gnueabi-gcc-4.5.1 and arm-none-linux-gnueabi-g++ tools.
But the result is showing arm-none-linux-gnueabi-gcc command not found.

Balau

2012/02/12

Actually it's not the same query: now you explicitly wrote that the command has "CROSS_COMPILE=./CodeSourcery/Sourcery_G++_Lite/bin/arm-none-linux-gnueabi-" and that's exactly the problem. During the building of u-boot, "make" changes directory, so the relative path to "arm-none-linux-gnueabi-gcc" changes, and it can't find it anymore.
You have to add the **absolute** path to "arm-none-linux-gnueabi-gcc" inside the PATH environment variable, for example:

```
export PATH="$PATH:/home/chandan/CodeSourcery/Sourcery_G++_Lite/bin/"
```

And then run the building with:

```
make all ARCH=arm CROSS_COMPILE=arm-none-linux-gnueabi-
```

eng trojan

2012/02/14

while trying to simulate the u-boot.bin on qemu i have this error

qemu: fatal: Trying to execute code outside RAM or ROM at 0x33f801f4

R00=00000000 R01=33fb9880 R02=00049868 R03=00000000
R04=00000000 R05=00000000 R06=00000000 R07=00000000
R08=00000000 R09=00000000 R10=00000000 R11=00000000
R12=000100fc R13=00000000 R14=000100fc R15=33f801f4
PSR=800001d3 N— A svc32
Aborted

also this error appears while simulating the flash.bin

Balau

2012/02/14

If you run QEMU adding the following options: "-d in_asm,cpu -D qemu.log -singlestep", it will dump a log file (qemu.log) containing state of the CPU and the instructions during guest execution.
It can be useful to understand where does it crash.
A thing that I notice is that R14 is 0x000100fc. R14 is the link register, which is the register that is used to return from a function. It is an indicator that the function that causes the problem is called from address 0x000100fc.
If you run "arm-none-eabi-objdump -dxS u-boot > u-boot.code" after building u-boot, you can inspect the disassembled code. Keep in mind that QEMU puts u-boot.bin to an offset of 0x10000, and u-boot itself is compiled to run at another offset, 0x01000000, so the call you should seek is probably indicated to be at address 0x010000fc.

chandan

2012/02/19

great tutorial !!!

I got “hello world” printed with the help of this article.

Thank you so much.

gowtham

2012/03/25

hi,
txs for the good tutorials…
i’ve got the problem at the end of the process…. after the printf “bootm 0x%X\n” $(expr $(stat -c%s u-boot.bin) + 65536) statement i get the address as bootm 0x22110… and when i run this in uboot prompt i get this “bad magic number” error… the o/p in the terminal is as foolows:

U-Boot 1.1.6 (Mar 24 2012 – 21:52:16)

DRAM: 0 kB
Flash: 0 kB
*** Warning – bad CRC, using default environment

In: serial
Out: serial
Err: serial
Versatile # bootm 0x22110
Booting image at 00022110 …
Bad Magic Number

Balau

2012/03/25

You could use iminfo and md to search for different memory locations:

VersatilePB # iminfo 0x250E0

Checking Image at 000250e0 …
Legacy image found
Image Name:
Image Type: ARM Linux Kernel Image (uncompressed)
Data Size: 392 Bytes = 0.4 kB
Load Address: 00100000
Entry Point: 00100000

```
Verifying Checksum … OK
VersatilePB # md 0x250E0 16
000250e0: 56190527 65377844 ddde6e4f 88010000 '..VDx7eOn……
000250f0: 00001000 00001000 83a6fc05 00020205 …….
00025100: 00000000 00000000 00000000 00000000 …….
00025110: 00000000 00000000 00000000 00000000 …….
00025120: e59fd004 eb000054 eafffffe 00101188 ….T……..
00025130: 00002341 61656100 A#…aea
```

The Magic Number is the first four bits of the test.uimg file (be aware of the endianess):

```
$ hexdump -C test.uimg
00000000 27 05 19 56 44 78 37 65 4f 6e de dd 00 00 01 88 |'..VDx7eOn……|
00000010 00 10 00 00 00 10 00 00 05 fc a6 83 05 02 02 00 |…….|
00000020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |…….|
*
00000040 04 d0 9f e5 54 00 00 eb fe ff ff ea 88 11 10 00 |….T……..|
00000050 41 23 00 00 00 61 65 61 62 69 00 01 19 00 00 00 |A#…aeabi……|
…
```

And you should find them in the flash.bin image at a different offset (you should subtract 0x10000 from the bootm address):

```
$ hexdump -C flash.bin
…
00015100 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |…….|
*
00015120 04 d0 9f e5 54 00 00 eb fe ff ff ea 88 11 10 00 |….T……..|
00015130 41 23 00 00 00 61 65 61 62 69 00 01 19 00 00 00 |A#…aeabi……|
…
```

Anup

2012/05/04

Hi Balau, I am getting below error when i 'bootm' the address where image is stored. Any idea why??

```
U-Boot 2010.03 (May 02 2012 – 18:20:05)

DRAM: 0 kB
## Unknown FLASH on Bank 1 – Size = 0x00000000 = 0 MB
Flash: 0 kB
*** Warning – bad CRC, using default environment
```

In: serial
Out: serial
Err: serial
Net: SMC91111-0
VersatilePB # bootm 0x25B38
Booting kernel from Legacy Image at 00025b38 …
Image Name:
Image Type: ARM Linux Kernel Image (uncompressed)
Data Size: 136 Bytes = 0.1 kB
Load Address: 00100000
Entry Point: 00100000
Loading Kernel Image … OK
OK

Starting kernel …

arm_sysctl_read: Bad register offset 0x1
arm_sysctl_read: Bad register offset 0x2
arm_sysctl_read: Bad register offset 0x3
arm_sysctl_read: Bad register offset 0x5
arm_sysctl_read: Bad register offset 0x6
arm_sysctl_read: Bad register offset 0x7
arm_sysctl_read: Bad register offset 0x9
arm_sysctl_read: Bad register offset 0xa
arm_sysctl_read: Bad register offset 0xb
arm_sysctl_read: Bad register offset 0xd
arm_sysctl_read: Bad register offset 0xe
arm_sysctl_read: Bad register offset 0xf
goes on till 0xd2 and qemu aborts

Balau

2012/05/04

I never encountered this error.
It seems that the execution tries to access the address of the system controller sequentially byte by byte.
Have you tried launching QEMU with "`-kernel zImage -append `" directly instead of passing the flash binary that contains U-Boot and the kernel?
If it gives the same error, then you have a problem in the kernel compilation. If not, then it's something about U-Boot and the memory map that it expects, which is somehow different than the memory map of the emulated system.

Anup

2012/05/04

Thanks for the reply Balau.

I will try out launching QEMU with zImage. By the way is QEMU version something to do with this?
I am using: QEMU PC emulator version 0.12.3 (qemu-kvm-0.12.3),

Balau

2012/05/05

It could be, my current QEMU version is 1.0.

Rishi Agrawal

2012/06/13

It will be good if with every post you can mention the version of the software you are using as most of the issues are caused by the different versions.

Rishi Agrawal

2012/06/14

Hi Balau, I have become fan of your post .. really … the best part is they are correct to every bit.

I would just like to suggest you that you should post a PREREQUISITE section on the top of every blog which will mention the packages you will be using along with the the exact versions. Like:
UBOOT 1.1.6
Linux Kernel 3.2.0
etc etc

sourav

2012/06/27

I have been following your post and its really useful.Thanks for such good and clear articles.I have few questions regarding u-boot.Does U-boot support versatile express ARM cortex a-15?I didn't see any such file under include/configs in u-boot source.I want to compile u-boot for ARM cortex a-15.Can you please help?

Balau

2012/06/27

It seems to me that the last U-Boot version contains omap5_evm which includes OMAP5430 support.
Other than that I have neither the experience nor the time to help you port U-Boot to the architecture you want.

Terence

2012/07/16

Hey Balau,

Great guides first of all. I'm getting a "Segmentation fault (core dumped)" error after running qemu-system-arm -M versatilepb -m 128M -nographic -kernel u-boot.bin. Any ideas? I'm on Ubuntu 12.04

Balau

2012/07/16

It seems a problem of QEMU. You can also try with different binaries to pass to the "kernel" option, to make sure the behavior is independent from the file you pass.
Then you can submit a bug to https://bugs.launchpad.net/qemu-linaro describing your problem and how to replicate.

chinlin

2012/07/20

Hi Bailau,

Is it possible to set up a video address for qemu, and have sample code to fill color rgba data?

It would be great if you can give some sample.

Thanks

Balau

2012/07/20

I don't know if I understand the question, but I think what you want to do is speak directly to the video controller with bare metal code.
If it's so, the code depends on the peripheral that controls the screen, for example the VersatilePB uses a (slightly modified) PL110 CLCD controller. You should find information on how to use it in the PL110 manual and in the VersatilePB documentation.

I can't give you some sample because I never did something like that. If in the future I do some tests with bare metal code using the LCD controller inside QEMU, then I will surely write a blog post about it. I can't promise I'll do it in the near future.

Chad Colgur

2012/09/20

Thanks for the article. Your link to VersatilePB is dead. I found the following: http://www.arm.com/products/tools/development-boards/versatile/index.php.

Balau

2012/09/23

Thanks for the indication! The link you provided shows only the ARM11 version of the Versatile Platform Baseboard, while for ARM926 there's only the CoreTile + Emulation Baseboard coupling. It seems that ARM dropped support for the VersatilePB that QEMU is emulation. I'm going to link directly to the manual which seems to be still in place.
Thanks again!

shabbir

2012/10/17

Hi Balau,
I am trying to analyse the uboot code of samsung exynos 4210,in which i found enable_mmu in lowlevel_init.S.I want to know what is the need of MMU in uboot level.

Balau

2012/10/17

I really don't know.
I suppose it could be to increase security and robustness during U-Boot execution, or for launching bare metal programs in a constricted environment, acting as an "hypervisor".
You could try asking in an irc channel (for example on freenode), or on U-Boot mailing list.

buiquanghuyen

2012/10/23

Hi Balau ,
I was run uboot but No ethernet found.

“U-Boot 2009.11 (Oct 23 2012 – 03:48:31)

DRAM: 0 kB
Unknown FLASH on Bank 1 – Size = 0x00000000 = 0 MB
Flash: 0 kB
*** Warning – bad CRC, using default environment

In: serial
Out: serial
Err: serial
Net: No ethernet found. ”

Please help my fix problem .Thank You!

Balau

2012/10/23

In my example U-Boot does not use the Ethernet so it’s not a problem, it can boot Linux correctly.
Ethernet is needed when you only have U-Boot on your target and you need to fetch the operating system from the net with something like tftp.
If you need something like that, then the Ethernet controller (NIC) must be supported by U-Boot, and I don’t know if it’s the case for my example with VersatilePB or with whatever architecture you are trying. You could probably find more support from the U-Boot mailing list or freenode.org IRC channels.

buiquanghuyen

2012/10/24

Hi Balau,
I load tftp file uImage and bootm. I don’t known it is not run.
“U-Boot 2010.03 (Oct 23 2012 – 08:42:49)
DRAM: 0 kB
Unknown FLASH on Bank 1 – Size = 0x00000000 = 0 MB
Flash: 0 kB
*** Warning – bad CRC, using default environment

```
In: serial
Out: serial
Err: serial
Net: SMC91111-0
VersatilePB # setenv ipaddr 10.0.2.15
VersatilePB # dhcp
SMC91111: PHY auto-negotiate timed out
SMC91111: MAC 52:54:00:12:34:56
BOOTP broadcast 1
DHCP client bound to address 10.0.2.15
Using SMC91111-0 device
TFTP from server 10.0.2.2; our IP address is 10.0.2.15
Filename '/tftpboot/uImage'.
Load address: 0x7fc0
Loading: ##################################################################
#################################################################
#################################################################
#################################################################
#################################################################
#########################
done
Bytes transferred = 1788440 (1b4a18 hex)
VersatilePB # bootm 0x7fc0
## Booting kernel from Legacy Image at 00007fc0 …
Image Name: Linux-3.5.0
Image Type: ARM Linux Kernel Image (uncompressed)
Data Size: 1788376 Bytes = 1.7 MB
Load Address: 00008000
Entry Point: 00008000
Loading Kernel Image … OK
OK

Starting kernel …

Uncompressing Linux… done, booting the kernel.
```

"

Please see full log bootm at http://canhdongvang.x10.mx/DTVT/balau.png
Thank You !
hqbui.

Balau

2012/10/24

It seems U-Boot does its job, and then the kernel hangs or does not display anything.
Maybe the kernel is booting but it's printing the messages somewhere else. try to add "`console=/dev/ttyAMA0`" to the kernel parameters, or even using the `earlyprintk` parameter.
See also my blog post "Booting Linux with U-Boot on QEMU ARM" if you have not seen it.

balaji

2013/05/09

Hi Balau,

I have one doubt relating to u-boot code execution.

1) I generated u-boot.bin by using arm-none-eabi toolchain with some Text_BASE 0x41e00000.
2) I tried to run the u-boot.bin from the DDR2 with Load_ADDR as 0x40000000 with go command.
3) How u-boot.bin gets executed with the Load_ADDR. What ever the labels present in the u-boot.bin are with the TEXT_BASE addresses.
4) I have seen the disassembly of u-boot where i found the labels are generated with the start of TEXT_BASE.

5) Stand alone application gets hanged when run it with LOAD_ADDR other than with actual TEXT_BASE.

6)But the above case is not happened with u-boot.

Can you clarify me if it is possible. The above are all doing for understanding relocation concept in u-boot.

Thanks
balaji

Balau

2013/05/09

I don't know if I understand correctly, but from what I understood you have two u-boot programs, one (let's call it u-boot-A) is used as the initial boot, and with the "go" command you launch the second (u-boot-B).
Then you have a standalone application, that you launch with u-boot-A using the go command in the same way.
You are seeing that u-boot-B gets executed correctly independently on the address to which it is copied, while your standalone application works only if you copy it on a particular address.

I think probably the reason is because u-boot-B contains position-independent code. When you disassemble, the disassembly is probably decorated with label addresses, but the code itself doesn't contain absolute addresses but only relative addresses to the current program counter.

Then there's relocation, but it's a different thing. I suspect that u-boot-A relocates itself to a high RAM address, then when you run the "go" command u-boot-B starts, then relocates itself at the same high RAM address as u-boot-A, thus overwriting u-boot-A. My hypothesis is that u-boot-B relocates itself to an address that is the same independently the load address that you decided.

You could try to compile your standalone program as position-independent and see if it works. With C code you have the -fPIC option in GCC. With assembly code you need to code it yourself without absolute addresses.

Hope this helps.

balaji

2013/05/10

Your understand is correct. I try with -fpic option.

Thank you very much

balaji

2013/05/14

Hi Balau,

Is there any possibility to see the relative addresses of the labels from u-boot.elf. Even relocation flag is enabled iam only able to see the absolute addresses only. During runtime the addresses are not as absolute address after relocation when i checked with DS-5 Debugger. How absolute addresses are modified with relative addresses. I know it was happened with relative symbol table but who will do the conversion(absolute to relative) linker, compiler or loader responsibility.

For example In disassembly of u-boot i found the following instruction.

bl 3f00200 // Here 3f00200 is the absolute address.

I relocated the code to 0x7000000 and i enabled relocation flag then what would be the transformation of above instruction .

Please help me in this.

Thanks
balaji.

Balau

2013/05/15

I want to stress that I'm not sure that u-boot uses position-independent code, I was suggesting an hypothesis. Maybe u-boot is not completely compiled to be position-independent, and your investigation seems to prove it. In light of this information another hypothesis is that only the initial part of u-boot can be run anywhere, and u-boot relocates itself always at the same address, so that the rest of the execution can contain absolute addresses, and they remain the same because the instructions are placed always on the same addresses.

balaji

2013/05/31

Hi Balau,

During internal development of bootloader, I got MMU precise abort when "MCR p15, #0, r0, c1, c0, #0" is executed. I read the TTB0 register its address is same as my page table address. When i commented above mentioned instruction i am able to go to next instruction. But i need to enable the MMU. It is needed for testing the L2 cache. I didn't enable the caches before enabling the MMU.

Can you tell me is there any possibility to avoid the precise abort.

Thanks
balaji

Balau

2013/06/02

Unfortunately I have no experience with ARM MMU, I am not able to give you pointers with this.

manju

2013/07/04

hello sir ,
how to overcome this error..
:~/Desktop/embedded/uboot/u-boot-2013.01$ qemu-system-arm -M versatilepb -m 128M -nographic -kernel flash.bin

pulseaudio: set_sink_input_volume() failed
pulseaudio: Reason: Invalid argument
pulseaudio: set_sink_input_mute() failed
pulseaudio: Reason: Invalid argument
qemu: fatal: Trying to execute code outside RAM or ROM at 0x08000000

R00=fffcbf60 R01=00000000 R02=00000000 R03=00000000
R04=00000000 R05=fffcbf60 R06=00000000 R07=00000000
R08=fffcbf60 R09=00000000 R10=01000020 R11=00000000
R12=fffcbfe0 R13=fffcbf50 R14=00000710 R15=08000000
PSR=600001d3 -ZC- A svc32
s00=00000000 s01=00000000 d00=0000000000000000
s02=00000000 s03=00000000 d01=0000000000000000
s04=00000000 s05=00000000 d02=0000000000000000
s06=00000000 s07=00000000 d03=0000000000000000
s08=00000000 s09=00000000 d04=0000000000000000
s10=00000000 s11=00000000 d05=0000000000000000
s12=00000000 s13=00000000 d06=0000000000000000
s14=00000000 s15=00000000 d07=0000000000000000
s16=00000000 s17=00000000 d08=0000000000000000
s18=00000000 s19=00000000 d09=0000000000000000
s20=00000000 s21=00000000 d10=0000000000000000
s22=00000000 s23=00000000 d11=0000000000000000
s24=00000000 s25=00000000 d12=0000000000000000
s26=00000000 s27=00000000 d13=0000000000000000
s28=00000000 s29=00000000 d14=0000000000000000
s30=00000000 s31=00000000 d15=0000000000000000
FPSCR: 00000000
Aborted (core dumped)

Balau

2013/07/04

0×08000000 is 128MiB, so it's the first address not mapped as memory.
You could have a problem in your configuration of U-Boot, where the amount of memory is, and for this reason U-Boot is trying to relocate itself to an address that is too big.
Or you could have a problem where the execution jumps somewhere and then continues to execute empty memory until it reaches the end of the memory. In this case I suggest using the "-s -S" options and attaching to QEMU with a debugger.

manju

2013/07/05

sir i'm new to linux ….will u elaborate a little bit about using the "-s -S" options and attaching to QEMU with a debugger.

Balau

2013/07/07

Be aware that what you are trying to do is in my opinion very difficult if you are not familiar to Linux.

Another thing that you could check before debugging is the addresses in file "u-boot.map" that is created during u-boot build.
If the addresses are similar to these, then there's no problem:

```
Memory Configuration

Name Origin Length Attributes
*default* 0x00000000 0xffffffff

Linker script and memory map

0x00000000 . = 0x0
0x00000000 . = ALIGN (0x4)

.text 0x01000000 0x13360
arch/arm/cpu/arm926ejs/start.o(.text)
.text 0x01000000 0x460 arch/arm/cpu/arm926ejs/start.o
0x01000000 _start
0x01000040 _TEXT_BASE
0x01000044 _bss_start_ofs
0x01000048 _bss_end_ofs
0x0100004c _end_ofs
0x01000050 IRQ_STACK_START_IN
0x01000078 relocate_code
```

but if they start from 0x08000000, that's your problem. You should check CONFIG_SYS_TEXT_BASE because that's probably what's wrong with your configuration. I don't know why it could be wrong because I don't know how you built u-boot.

About my previous suggestion:
From "man qemy-system-arm" you can find:

```
...
-S Do not start CPU at startup (you must type 'c' in the monitor).

-gdb dev
Wait for gdb connection on device dev. Typical connections will
likely be TCP-based, but also UDP, pseudo TTY, or even stdio are
reasonable use case. The latter is allowing to start QEMU from
within gdb and establish the connection via a pipe:

(gdb) target remote | exec qemu-system-i386 -gdb stdio ...

-s Shorthand for -gdb tcp::1234, i.e. open a gdbserver on TCP port
1234.
...
```

So you run:

```
$ qemu-system-arm -M versatilepb -m 128M -nographic -kernel flash.bin -s -S
```

This will open up QEMU without doing anything, waiting for a gdb connection.
Then, assuming your toolchain is something like "arm-linux-gnueabi-*" you run in another terminal:

```
$ arm-linux-gnueabi-gdb
```

Then inside gdb, assuming you compiled u-boot in the same directory you launched gdb:

```
(gdb) file u-boot
(gdb) target remote localhost:1234
(gdb) stepi
(gdb) stepi
...
```

There are many gdb tutorials online.

Chetan

2013/09/30

Hi Balau,

I have tried with the u-boot version 2010.03 and 2010.09.
Still I am getting the
qemu:fatal error as trying to execute out of RAM or ROM at address XXXXXXX

Please help me in this

Balau

2013/10/04

The information you give is very generic, in order to understand the cause of a problem it's important to look at the details.
What is the address XXXXXXX? (and why did you even bother to obfuscate it?)
What version of QEMU are you using? How are you running qemu? did you change something with respect to the procedure I posted?
How did you build u-boot? Did you change something with respect to the procedure I posted?

Balaji

2013/10/15

Hi Balau,

Instead of u-boot.bin i want to use the u-boot.elf with the following command for running in qemu to execute u-boot and hello world program.

printf "bootm 0x%X\n" $(expr $(stat -c%s u-boot.bin) + 65536).

Though I tried, i did'nt get the hello world program start address for running with the bootm command in Qemu. Can you tell me if it is possible.

Thanks
balaji

Balau

2013/10/15

The reason why my solution works is because I concatenate the two binaries (u-boot.bin + test.uimg) directly one after the other to create flash.bin. So the size of u-boot.bin is also the offset of test.uimg inside the flash.bin file.
I don't know how QEMU behaves when running ELF programs, and it's probably something that a real hardware cannot do because in order to load an ELF you need some sort of operating system running.
It's also not clear to me how are you executing QEMU and especially how are you providing the hello world program to QEMU.

Balaji

2013/10/16

With the help of dd command i copied u-boot.elf and helloworld.bin into the flash.bin having 1MB space each. To the Qemu i just have given the following command and run it.

qemu-system-arm -M vexpress-a9 -kernel flash.bin -m 256M –serial stdio.

I am able to see the u-boot prompt in qemu. For running helloworld program with the help of bootm command ,i used the following command for knowing helloworld program address in flash.bin.

printf “bootm 0x%X\n” $(expr $(stat -c%s u-boot) + 65536).

What i observed is the above command is not helpful for knowing the address of hello world in flash.bin. Is there any procedure to know the offset of the second file present in flash.bin. Please help me in this.

Balau

2013/10/16

I try to give a hypothesis about what the problem is.

I think QEMU is using the ELF information to load u-boot into memory, and the ELF file does not know that at the end of it there’s the hello world.

QEMU opens flash.bin, it checks the type and it seems to be an ELF file, because at the beginning of flash.bin there is u-boot ELF, and at the beginning of u-boot ELF there are the three characters E L F. So it checks how to load it into memory and finds a table (that you can display with something like “`arm-none-eabi-objdump -p u-boot`“), and uses this table to load u-boot into memory. It then executes from the entry point.

So QEMU doesn’t load the whole flash.bin into the emulated memory: it does not load the debug sections that can be found in an ELF file and it does not load the hello world program because there’s nothing in the ELF file about it.

I don’t think you can do something like that with dd, you should probably touch the linking phase of u-boot and add your program in the linking, as a binary blob (see here for a hint).

Leonardo

2013/11/07

“Newer” U-boot will work with QEMU if first “make” in this post uses “versatileqemu_config” instead of “versatilepb_config” (i.e. make versatileqemu_config ARCH=arm …). Anyway, thanks for another great tutorial.

Raghuram.P

2014/06/11

Hi,
OS : Ubuntu 14.04
qemu-veraion : QEMU emulator version 2.0.0 (Debian 2.0.0+dfsg-2ubuntu1), Copyright (c) 2003-2008 Fabrice Bellard
I created the u-boot.bin by following the steps in

http://www.raspberrypi.org/forums/viewtopic.php?f=66&t=56914

Then created the flash.bin , by following your steps.

I am getting the following error for the command

qemu-system-arm -M versatilepb -m 128M -nographic -kernel flash.bin

pulseaudio: set_sink_input_volume() failed
pulseaudio: Reason: Invalid argument
pulseaudio: set_sink_input_mute() failed
pulseaudio: Reason: Invalid argument
qemu: fatal: Trying to execute code outside RAM or ROM at 0x63696c70

R00=00000000 R01=00000000 R02=00000020 R03=00000000
R04=0003fed8 R05=00000000 R06=00000000 R07=63696c70
R08=07ffff80 R09=00000000 R10=00000000 R11=00000000
R12=07fffff8 R13=07ffff78 R14=0001154c R15=63696c70
PSR=200001d3 –C- A svc32
s00=00000000 s01=00000000 d00=0000000000000000
s02=00000000 s03=00000000 d01=0000000000000000
s04=00000000 s05=00000000 d02=0000000000000000
s06=00000000 s07=00000000 d03=0000000000000000
s08=00000000 s09=00000000 d04=0000000000000000
s10=00000000 s11=00000000 d05=0000000000000000
s12=00000000 s13=00000000 d06=0000000000000000
s14=00000000 s15=00000000 d07=0000000000000000
s16=00000000 s17=00000000 d08=0000000000000000
s18=00000000 s19=00000000 d09=0000000000000000
s20=00000000 s21=00000000 d10=0000000000000000
s22=00000000 s23=00000000 d11=0000000000000000
s24=00000000 s25=00000000 d12=0000000000000000
s26=00000000 s27=00000000 d13=0000000000000000
s28=00000000 s29=00000000 d14=0000000000000000
s30=00000000 s31=00000000 d15=0000000000000000
FPSCR: 00000000
Aborted (core dumped)

Balau

2014/06/11

Raspberry Pi and Versatile PB have different ARM cores, different chip architectures (such as memory map and peripherals), and different boards.
If you compile U-Boot for Raspberry Pi it will most probably not work on anything else.
But you are compiling for Raspberry Pi and emulating a Versatile PB, so it's expected to fail.

Mohammed

2014/11/04

Hell
It's been a while you wrote this but I think the basics still hold true. I cross compiled u-boot's latest release using buildroot toolchain that I generated earlier.

However when I try to invoke qemu using the u-boot umage from it, it complains of kernel not having the correct image:

I invoke it with:
qemu-system-ppc -M mpc8544ds -m 512 -nographic -kernel u-boot.bin

And I get the following error:
Wrong image type 52, expected 2
qemu: could not load kernel 'u-boot.bin'

Could you shed some light about whats wrong here?

Balau

2014/11/04

I searched in QEMU source code and here is the function that prints the error:

https://github.com/qemu/qemu/blob/master/hw/core/loader.c#L479

My guess is that QEMU tries to read an u-boot header from the binary. u-boot binary should be the output of something like the objcopy command on the ELF of u-boot, while the Linux kernel and the ramdisk should be packed with mkimage. When QEMU sees a file with an u-boot header it expects a kernel (image type 2, see http://git.denx.de/cgi-bin/gitweb.cgi?p=u-boot.git;a=blob;f=include/image.h;h=07e9aed16d95190b3cebe2f19dcb29ef15ab63b2;hb=HEAD#l
and prints an error.

I don't know why QEMU thinks that u-boot.bin contains an u-boot header, my only guess is that by chance the binary has at the beginning the magic number 0x27051956, you can check with something like`hexdump -C u-boot.bin|head`.

Mohammed

2014/11/05

Hello (my last post started with 'Hell`. WordPress missed my 'o' for some reason. Apologies for the same nevertheless).

Yes I figured that out too from the same file in Qemu source code that the valid header types go from 0 through to 8.

By the way I am not sure about this statement of yours
I don't know why QEMU thinks that u-boot.bin contains an u-boot header. I mean, the u-boot binary ***should*** be having a header that identifies it as a u-boot image, isn't it?

And when I did dumped the hex contents of the binary, it DID have that magic number ahead of it..
Heres the hex dump
00000000 27 05 19 56 55 2d 42 6f 6f 74 20 32 30 31 34 2e |'..VU-Boot 2014.|

On doing a little search, as per this thread on the u-boot mailing list, this header seems perfectly ok to the u-boot guys. I am a bit confused. I mean when the final binary was packaged, did something go *wrong* somewhere in the multiple stages from compilation to packaging?

Balau

2014/11/05

> *the u-boot binary should be having a header that identifies it as a u-boot image, isn't it?*

Actually it should not; I'll try to explain my understanding:

U-Boot is a boot loader, it's bare metal (talks directly with hardware), and in many cases it's the first software that the processor runs at reset. The processor doesn't care about headers, it usually jumps to a fixed boot address and begin to execute the instructions there. The boot loader should reside at that fixed address, and in many architectures (such as Cortex-A and ARM9) the first address of the boot loader binary contains the first instruction to execute.

The objective of a boot loader is to initialize the hardware to a minimal configuration, load the operating system and run it. The operating system is Linux in our case, and the Linux kernel is a binary image that should be copied somewhere. The boot loader knows about the binary image by reading its header. If I want the boot loader to know about my kernel image I have to prepare the image by packaging it with a header that is specific to the boot loader. That is the header added by `mkimage`.

U-Boot image itself doesn't need an header, the code that reads the header is inside U-Boot image.

QEMU is a different story because QEMU itself contains a kind of boot loader, which understands U-Boot headers, pure binary code and for example also ELF files. So if you give QEMU an image packaged with mkimage it will try to unpack it and load it at the address written in the header.

You could try to run `mkimage -l u-boot.bin` to understand what did the buildroot process create.

9 Trackbacks For This Post

1. *Links 18/3/2010: Steam and Linux; Red Hat's CEO Talks | Boycott Novell* →
   March 19th, 2010 → 03:33
   […] U-boot for ARM on QEMU […]

2. *Compiling Linux kernel for QEMU ARM emulator « Balau* →
   March 27th, 2010 → 19:06
   […] Posts Hello world for bare metal ARM using QEMUU-boot for ARM on QEMUCompiling Linux kernel for QEMU ARM emulatorSimplest bare metal program for ARMSimplest serial port […]

3. *Booting Linux with U-Boot on QEMU ARM « Balau* →
   April 12th, 2010 → 19:55
   […] with QEMU emulation of an ARM Versatile Platform Board, making it run bare metal programs, the U-Boot boot-loader and a Linux kernel complete with a Busybox-based file system. I tried to put everything […]

4. *Blog stats for 2010 « Balau* →
   January 2nd, 2011 → 17:25
   […] U-boot for ARM on QEMU March 2010 14 comments […]

5. *Linux ARM emulation articles* →
   January 10th, 2011 → 15:31
   […] u-boot-for-arm-on-qemu […]

6. *QEMU for ARM, revisited | d-code* →
   February 7th, 2013 → 12:52
   […] Configuration (board_name) for U-Boot is : versatileqemu, as described here : http://comments.gmane.org/gmane.comp.boot-loaders.u-boot/101980. (For info, QEMU places the beginning of the U-Boot binary at 0x10000 : https://balau82.wordpress.com/2010/03/10/u-boot-for-arm-on-qemu/) […]

7. *QEMU 1.5.0 released, a backward compatibility warning | Balau* →
   May 21st, 2013 → 20:51
   […] U-boot for ARM on QEMU […]

8. *Linux Kernel Online and Book Resources collection | Kaiwan's Tech Blog* →
   August 1st, 2013 → 12:04
   […] U-Boot for ARM on QEMU […]

9. *Analyzing C source files dependencies in a program | Balau* →
   November 24th, 2013 → 13:22
   […] example, I tried it on U-Boot compilation for the Versatile PB platform (that I tackled in a past blog post), and this is the resulting […]