Virtual Development Board

From eLinux.org

If you want to have an Embedded Linux Development Board, and you don't want to pay for it, then you can DIY a Virtual Development Board.

The Virtual Development Board is an emulation board which made from QEMU, actually it's a Virtual Machine.

Contents

- 1 Debug Mode
  - 1.1 QEMU
    - 1.1.1 Download QEMU
    - 1.1.2 Install QEMU
  - 1.2 Bootloader
    - 1.2.1 Prepare Cross Toolchain
    - 1.2.2 Download U-Boot
    - 1.2.3 Cross compile U-Boot
    - 1.2.4 Debug U-Boot
  - 1.3 Linux Kernel
    - 1.3.1 Download Linux Kernel
    - 1.3.2 Cross Compile Linux Kernel
    - 1.3.3 Load Linux Kernel
      - 1.3.3.1 Download and Install Open TFTP Server
      - 1.3.3.2 prepare qemu-ifup & qemu-ifdown
      - 1.3.3.3 tftpboot uImage
    - 1.3.4 run linux kernel
      - 1.3.4.1 prepare the rootfs
      - 1.3.4.2 prepare the nfs
      - 1.3.4.3 run linux kernel
  - 1.4 driver
    - 1.4.1 add a device to QEMU
    - 1.4.2 write the device driver
  - 1.5 GUI
- 2 Run Mode
  - 2.1 QEMU
    - 2.1.1 Support FLASH on QEMU
    - 2.1.2 Fix Compilation Errors
  - 2.2 U-Boot
    - 2.2.1 burn U-Boot into flash
- 3 References
- 4 External links

Debug Mode

QEMU

Download QEMU

There are two ways in which you will usually obtain the source code:

- As a tarball, get the lastest released version qemu-0.14.0.tar.gz (http://download.savannah.gnu.org/releases/qemu/qemu-0.14.0.tar.gz) from the QEMU Download Page (http://wiki.qemu.org/Download). Extract all files:

```
$ tar zxvf qemu-0.14.0.tar.gz
```

- Using the git tool to track the source code.

```
$ git clone git://git.qemu.org/qemu.git
```

Install QEMU

According to the Filesystem Hierarchy Standard (http://www.pathname.com/fhs/pub/fhs-2.3.html#OPTADDONAPPLICATIONSOFTWAREPACKAGES), the directory '/opt' is reserved for the installation of add-on application software packages.

```
$ sudo chmod 777 /opt
```

Install QEMU to '/opt/qemu', now we can go into the source code directory of QEMU

```
$ ./configure --prefix=/opt/qemu --target-list=arm-softmmu,arm-linux-user --enable-debug
$ make -s
$ sudo make install -s
```

Linux determines the executable search path with the $PATH environment variable. The place to add a directory to the $PATH of all users except user root, add the line "PATH=/opt/qemu/bin:$PATH" to the file '/etc/profile' in root authority, and then use the "dot" command let the shell program executed in the current shell:

```
$ . /etc/profile
```

Make sure there is a space between the dot (.) and the script name.

Then you can download arm-test-0.2.tar.gz (http://wiki.qemu.org/download/arm-test-0.2.tar.gz) from the QEMU Download Page (http://wiki.qemu.org/Download), that's the ARM Linux 2.6 test kernel and initrd disk image (thanx to Paul Brook).

```
$ tar zxvf arm-test-0.2.tar.gz
$ cd arm-test
$ qemu-system-arm -kernel zImage.integrator -initrd arm_root.img
```

It will startup a virtual ARM machine, if your mouse is grabbed by it, press "Ctrl-Alt" to exit. That's good, installation is done.

Bootloader

Prepare Cross Toolchain

Open Sourcery G++ Lite Edition for ARM (http://www.codesourcery.com/sgpp/lite/arm/portal/subscription?@template=lite), select GNU/Linux > Packages > Recommended Packages > IA32 GNU/Linux Installer, download the file arm-2010.09-50-arm-none-linux-gnueabi.bin (http://www.codesourcery.com/sgpp/lite/arm/portal/package7853/public/arm-none-linux-gnueabi/arm-2010.09-50-arm-none-linux-gnueabi.bin), install it:

```
$ chmod +x arm-2010.09-50-arm-none-linux-gnueabi.bin
$ ./arm-2010.09-50-arm-none-linux-gnueabi.bin
```

The default installation path is at your home, you can also modify it to be '/opt/CodeSourcery/Sourcery_G++_Lite' in the installation process, in default, it will automatically modify $PATH in the file '.bash_profile' at your home, thus add product to the $PATH for you only.

After the installation is done, you can find the sysroot directories provided with Sourcery G++ in the '/opt/CodeSourcery/Sourcery_G++_Lite/arm-none-linux-gnueabi/libc'. Now you can also write a "hello world" program to test it:

```
$ arm-none-linux-gnueabi-gcc -o hello hello.c
$ qemu-arm -L ~/CodeSourcery/Sourcery_G++_Lite/arm-none-linux-gnueabi/libc hello
```

If you can see "hello world" or something others you let it print on the screen, the cross toolchain is prepared.

Download U-Boot

- released version: download u-boot-2010.09.tar.bz2 (ftp://ftp.denx.de/pub/u-boot/u-boot-2010.09.tar.bz2) from U-Boot Source Code Page (ftp://ftp.denx.de/pub/u-boot/). Extract all files:

```
$ tar jxvf u-boot-2010.09.tar.bz2
```

- current source code:

```
$ git clone git://git.denx.de/u-boot.git
```

Cross compile U-Boot

Go into the U-Boot root directory, then do

```
$ make ARCH=arm CROSS_COMPILE=arm-none-linux-gnueabi- versatilepb_config
$ make ARCH=arm CROSS_COMPILE=arm-none-linux-gnueabi- -s
```

It will generate 'u-boot' and 'u-boot.bin' in the directory.

And we will get the tool "mkimage" under the directory "tools", we will use it to make Linux Kernel uImage, so

```
$ sudo ln -sf $PWD/tools/mkimage /usr/local/bin/mkimage
```

You can start the u-boot in qemu now:

```
$ qemu-system-arm -M versatilepb -nographic -kernel u-boot
```

It will start u-boot in your console, now you can type u-boot command, like "printenv", after the "VersatilePB # " prompt. type "CTRL-a x" to quit.

Debug U-Boot

Add -s and -S options while involking QEMU

- -s shorthand for -gdb tcp::1234
- -S freeze CPU at startup (use 'c' to start execution)

```
$ qemu-system-arm -M versatilepb -nographic -kernel u-boot -s -S
```

To debug u-boot, load the file "u-boot" into gdb (not "u-boot.bin") that is produced by "make" when building u-boot, This file is in ELF format and contain all the symbol information and are not stripped of debugging data until you run "strip" on them, unlike "u-boot.bin". Start ARM gdb in another console window and load "u-boot":

```
$ arm-none-linux-gnueabi-gdb u-boot
(gdb) target remote :1234
(gdb) b do_printenv
Breakpoint 1 at 0x10080f4: file cmd_nvedit.c, line 147.
(gdb) c
Continuing.
```

in the QEMU console window, it will show something like this:

```
U-Boot 2010.06 (Aug 31 2010 - 16:23:16)
DRAM:  0 Bytes
Flash: 64 MiB
*** Warning - bad CRC, using default environment
In:    serial
Out:   serial
Err:   serial
```

```
Net:   SMC91111-0
VersatilePB #
```

type u-boot command "printenv" in the QEMU console window, it will be broken by the ARM gdb:

```
VersatilePB # printenv
```

in the ARM gdb console window, it will show:

```
Breakpoint 1, do_printenv (cmdtp=0x1015520, flag=0, argc=1, argv=0xfddee4)
    at cmd_nvedit.c:147
147             if (argc == 1) {
(gdb)
```

from here we should be able to use all the usual GDB commands, that's good!

Linux Kernel

Download Linux Kernel

- Latest Stable Kernel: linux-2.6.37.4.tar.bz2 (http://www.kernel.org/pub/linux/kernel/v2.6/linux-2.6.37.4.tar.bz2) from Linux Kernel Main Page (http://www.kernel.org/).
- Trace the source code:

```
$ git clone git://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux-2.6.git
```

Cross Compile Linux Kernel

```
$ make ARCH=arm CROSS_COMPILE=arm-none-linux-gnueabi- versatile_defconfig -s
$ make ARCH=arm CROSS_COMPILE=arm-none-linux-gnueabi- uImage -s
```

Then we will get the file "uImage" at the location "arch/arm/boot/".

Link the uImage file to your tftpserver home path, for example '/opt/versatilepb/kernel', like this:

```
$ mkdir -p /opt/versatilepb/kernel
$ ln -sf $PWD/arch/arm/boot/uImage /opt/versatilepb/kernel/
```

Load Linux Kernel

Download and Install Open TFTP Server

Download the tftp server opentftpmtV1.63.tar.gz (http://cdnetworks-kr-1.dl.sourceforge.net/project/tftp-server/tftp%20server%20multithreaded/opentftpmtV1.63.tar.gz) from Open TFTP Server Main Page (http://sourceforge.net/projects/tftp-server/), and install it

```
$ tar zxvf opentftpmtV1.63.tar.gz
$ mv opentftp/ /opt/
$ cd /opt/opentftp/
```

Modify the the file "opentftpd.ini" to specify one directory as home directory, from where files will be served or deposited, for example, we add '/opt/versatilepb/kernel' line under the [HOME] entry in the file. To start tftpserver like this:

```
$ sudo rc.opentftp start
$ sudo rc.opentftp status
```

If displays "Server opentftp is running - Pid : xxxx", that's ok.

We can add the two lines below into the file '/etc/rc.d/rc.local' to start tftpserver at PC startup:

```
/opt/opentftp/rc.opentftp start
/opt/opentftp/rc.opentftp status
```

prepare qemu-ifup & qemu-ifdown

Using TAP network interfaces is the standard way to connect QEMU to a real network.

if there is not device "/dev/net/tun", we should make it :

```
$ sudo mkdir -p /dev/net
$ sudo mknod /dev/net/tun c 10 200
$ sudo /sbin/modprobe tun
```

also we can add them to "/etc/rc.d/rc.local", let it can be made at PC startup.

refer to QEMU/Networking (http://en.wikibooks.org/wiki/QEMU/Networking), make two files: qemu-ifup and qemu-ifdown, but do not need openvpn line, then

```
$ sudo cp qemu-ifup qemu-ifdown /etc/
$ sudo chmod +x qemu-ifup
$ sudo chmod +x qemu-ifdown
```

tftpboot uImage

suppose that IP of the target board is 192.168.1.123, IP of the host PC(server) is 192.168.1.234.

```
$ sudo qemu-system-arm -M versatilepb -nographic -net nic -net tap,ifname=tap0 -kernel $PATH_TO_YOUR_U-BOOT/u-boot
U-Boot 2010.09 (Dec 15 2010 - 18:16:35)
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
VersatilePB # sete ipaddr 192.168.1.123
VersatilePB # sete serverip 192.168.1.234
VersatilePB # sete bootfile uImage
VersatilePB # tftpboot
SMC91111: PHY auto-negotiate timed out
SMC91111: MAC 52:54:00:12:34:56
Using SMC91111-0 device
TFTP from server 192.168.1.234; our IP address is 192.168.1.123
Filename 'uImage'.
Load address: 0x7fc0
Loading: T ##################################T ##############################
         ##########################################
done
Bytes transferred = 1556392 (17bfa8 hex)
VersatilePB # iminfo
## Checking Image at 00007fc0 ...
   Legacy image found
   Image Name:   Linux-2.6.36.2
   Image Type:   ARM Linux Kernel Image (uncompressed)
   Data Size:    1556328 Bytes = 1.5 MiB
   Load Address: 00008000
   Entry Point:  00008000
   Verifying Checksum ... OK
VersatilePB #
```

? how to run qemu-system-arm with tap net in user mode?

run linux kernel

prepare the rootfs

prepare the nfs

run linux kernel

driver

add a device to QEMU

write the device driver

GUI

Run Mode

QEMU

Support FLASH on QEMU

Now, the emulation of Intel flashes is present in Qemu (in *hw/pflash_cfi01.c*) and this emulation is already used in some emulated ARM platforms, but not the **Versatile PB** platform that we use for our object (this platform is nice because it has Ethernet, serial ports, LCD, etc.). So, we must add flash emulation to the

Versatile PB platform. Firstly, we should go into the source code directory of QEMU, and modify the file *hw/versatilepb.c*, assume to support a 64MB flash device, like this:

```
--- qemu-0.12.5/hw/versatilepb.c        2010-07-22 20:39:04.000000000 +0800
+++ qemu_armux/hw/versatilepb.c 2010-09-01 11:59:33.000000000 +0800
@@ -16,6 +16,11 @@
 #include "pci.h"
 #include "usb-ohci.h"
 #include "boards.h"
+#include "flash.h"
+
+#define VERSATILE_FLASH_ADDR           0x34000000
+#define VERSATILE_FLASH_SIZE           (64*1024*1024)
+#define VERSATILE_FLASH_SECT_SIZE      (256*1024)

 /* Primary interrupt controller.  */

@@ -172,7 +177,9 @@
     NICInfo *nd;
     int n;
     int done_smc = 0;
-
+    DriveInfo *dinfo;
+    int hasflash = 0;
+
     if (!cpu_model)
         cpu_model = "arm926";
     env = cpu_init(cpu_model);
@@ -280,13 +287,29 @@
     /*  0x101f2000 UART1.  */
     /*  0x101f3000 UART2.  */
     /* 0x101f4000 SSPI.  */
-
-    versatile_binfo.ram_size = ram_size;
-    versatile_binfo.kernel_filename = kernel_filename;
-    versatile_binfo.kernel_cmdline = kernel_cmdline;
-    versatile_binfo.initrd_filename = initrd_filename;
-    versatile_binfo.board_id = board_id;
-    arm_load_kernel(env, &versatile_binfo);
+
+    dinfo = drive_get(IF_PFLASH, 0, 0);
+    if(dinfo) {
+       if(!pflash_cfi01_register(VERSATILE_FLASH_ADDR,
+                                 qemu_ram_alloc(VERSATILE_FLASH_SIZE),
+                                 dinfo->bdrv,
+                                 VERSATILE_FLASH_SECT_SIZE,
+                                 VERSATILE_FLASH_SIZE / VERSATILE_FLASH_SECT_SIZE,
+                                 4, 0, 0, 0, 0)) {
+               fprintf(stderr, "qemu: error registering flash memory.\n");
+               exit(1);
+       }
+       hasflash = 1;
+    }
+    if(!hasflash) {
+       versatile_binfo.ram_size = ram_size;
+       versatile_binfo.kernel_filename = kernel_filename;
+       versatile_binfo.kernel_cmdline = kernel_cmdline;
+       versatile_binfo.initrd_filename = initrd_filename;
+       versatile_binfo.board_id = board_id;
+       arm_load_kernel(env, &versatile_binfo);
+    } else
+       env->regs[15] = VERSATILE_FLASH_ADDR;
 }

 static void vpb_init(ram_addr_t ram_size,
```

then save the file.

Fix Compilation Errors

```
change: qemu_ram_alloc(NULL, "versatile.rom", VERSATILE_FLASH_SIZE) // add 'NULL' and "versatile.rom"
change: 4, 0, 0, 0, 0, 0) // add extra '0'

    dinfo = drive_get(IF_PFLASH, 0, 0);
    if(dinfo) {
        if(!pflash_cfi01_register(VERSATILE_FLASH_ADDR,
                                  qemu_ram_alloc(NULL, "versatile.rom", VERSATILE_FLASH_SIZE),
                                  dinfo->bdrv,
                                  VERSATILE_FLASH_SECT_SIZE,
                                  VERSATILE_FLASH_SIZE / VERSATILE_FLASH_SECT_SIZE,
                                  4, 0, 0, 0, 0, 0)) {
                fprintf(stderr, "qemu: error registering flash memory.\n");
                exit(1);
        }
        hasflash = 1;
    }
    if(!hasflash) {
```

U-Boot

burn U-Boot into flash

Firstly， we must create a 64MB flash file, then we can burn the 'u-boot.bin' into the flash:

```
  $ dd if=/dev/zero of=flash.img bs=1M count=64
  $ dd if=u-boot.bin of=flash.img conv=notrunc
  $ qemu-system-arm -M versatilepb -nographic -pflash flash.img
```

References

- Using U-Boot and Flash emulation in Qemu (http://thomas.enix.org/Blog-20081002153859-Technologie)
- U-boot for ARM on QEMU (http://balau82.wordpress.com/2010/03/10/u-boot-for-arm-on-qemu/)
- Debugging Linux systems using GDB and QEMU (http://files.meetup.com/1590495/debugging-with-qemu.pdf)

External links

- QEMU (http://www.qemu.org)
- CodeSourcery (http://www.codesourcery.com)
- Das U-Boot (http://www.denx.de/wiki/U-Boot)