

## Search

Search for:  Go

- [Home](#)
- [Archives](#)

// you're reading...

## Embedded Systems

# Embedded System Development: QEMU + BuildRoot + Linux ARM

By [Gabriel Gonzalez](#) · February 21, 2011 · [Post a comment](#)

Filed Under [computer engineering](#), [embedded systems](#), [Linux](#)

```
Initializing random number generator... done.
Starting network...
ifup: interface lo already configured

Welcome to Buildroot
buildroot login: root
# uname -a
Linux buildroot 2.6.36.1 #1 Sun Feb 20 02:39:40 CET 2011 armv5tejl GNU/Linux
# cat /proc/cpu
/proc/cpu/      /proc/cpuinfo
# cat /proc/cpuinfo
Processor       : ARM926EJ-S rev 5 (v5l)
BogoMIPS       : 285.08
Features        : swp half thumb fastmult vfp edsp java
CPU implementer : 0x41
CPU architecture: 5TEJ
CPU variant     : 0x0
CPU part        : 0x926
CPU revision    : 5

Hardware       : ARM-Versatile PB
Revision       : 0000
Serial         : 0000000000000000
#
```

## QEMU + Linux + BuildRoot

Here are some basic steps to get a virtual ARM Development board which will allow to practice our embedded systems skills or test code / firmware or whatever we want to do but without the need of having a physical device.

The following takes place in a Linux machine running on Virtual Box (512MB RAM), which I first thought would've taken much more time to compile the whole stuff but it only took ~2h.

For the whole procedure you are gonna need some development packages such as gcc, g++, bison, flex, gettext, texinfo, zlib, ncurses, uclibc and maybe others, just install them when the process complains.

### First Step: Setting up the Environment

We need to make use of tun/tap device to create a local lan in order to let the embedded linux find the NFS mount point.

- Setting up tap0:

```
$ sudo tuncctl -u youruser  
  
Set 'tap0' persistent owned by 10101  
  
$ sudo ifconfig tap0 192.168.10.1 up
```

- Setting up NFS mount point:

Since we want this set-up for development I assume we want to add new binaries or modify configuration on-the-fly so I have created `~/root-nfs` where the file system for our embedded device will reside. Update `/etc/exports` appropriately:

```
$ echo "/home/someuser/root-nfs" 192.168.10.0/192.168.10.255(rw, sync,no_root_squash)" &gt; /etc/exports
```

Remember the `no_root_squash` method or files only accessible by root will no be exported through NFS.

## Second Step: Configuring QEMU

Grab the source from [www.qemu.org](http://www.qemu.org) and compile it, or if you prefer just used some prebuilt binaries. It shouldn't make any difference.

**QEMU** not only emulate CPUs but also a bunch of development boards with its attached hardware so you can run real software that interacts with the ethernet or whatever other peripheral available.

For our purpose we are going to use **versatilepb** which is an **ARM** based development board which you can [physically have it](#) but thanks to **QEMU** we can use a virtual one.

The only thing you need be sure when configuring/compiling/installing **QEMU** is that it supports **ARM** as target and the **versatilepb** machine is available.

## Third Step: Getting BuildRoot

Buildroot is a great framework for crosscompiling, targeting mainly embedded systems. It is a bunch of Makefiles and configuration files which automatizes the toolchain generation for the target platform, building bootloader, kernel, libraries, binaries and preparing the appropriate images for booting a device.

The configuration is pretty similar to the Linux Kernel so if you are used to deal with `menuconfig` this will not show any difficult to you.

After decompressing buildroot we have to perform some minor configuration to produce the binaries we need for our target device.

Just type:

```
$ make menuconfig
```

And change the following entries:

- Target Architecture = arm

- Target Architecture Variant = generic\_arm
- Target ABI = EABI
- Target Options -> Generic Serial Port Config -> serial port to run getty on (ttyAMA0)
- Package Selection, make sure Busybox is selected
- Target FileSystem Options -> check tar the root filesystem
- Kernel -> set deconfig name to "versatile"
- Kernel -> Kernel binary format to zImage

Afterwards do:

```
$ make linux26-menuconfig
```

Accept the default configurations and change anything you feel like changing

Note: I had a problem with the provided fakeroot version so I downloaded the fakeroot\_1.11 from the [debian repository](#) and copied it to buildroot/dl/ (which is the place where downloaded packages are stored) and changed the version to 1.11 in buildroot/packages/fakeroot/fakeroot.mk and then make again.

Just type:

```
$ make
```

And let the whole Makefiles produce the desired images!

They will be placed under buildroot/output/images, there you will find the zImage and the root.tar containing a whole filesystem with everything need to boot a Linux Operating System.

### Final Step: Booting QEMU

Just before booting we should uncompress the root fs to the exported NFS directory as root, otherwise the special files under /dev will not be properly created which will stop our system working properly.

Now we just need to tell qemu the following parameters to run our system:

- -M versatilepb # our target platform
- -kernel buildroot/output/image/zImage # our generated kernel image targeting ARM
- -net nic # so qemu emulates a NIC interfaces as part of the versatilepb platform
- -net tap,ifname=tap0,script=no # tell qemu to link the emulated NIC to the preconfigured tap device
- -append "console=ttyAMA0 root=/dev/nfs rw  
nfsroot=192.168.10.1:/home/someuser/root-nfs ip=192.168.10.2" # to instruct the booted linux kernel to search for the root-fs in the specified NFS server using the provided ip as its own
- -nographic # to output everything to stdout

And all together:

```
qemu-system-arm -M versatilepb -kernel buildroot/output/image/zImage -net nic -net tap,ifname=tap0,script=no append
```

Now you should see the booting process of Linux and then the busybox login, as seen in the image above; just use root and no password to log in.

## Need Help?

**Click Here**



<http://www.intelligentrd.com>

**Embedded Systems**

**Complex Software  
Architectures**


**Reverse Engineering**

**Smart Phones /  
Tablets Applications**

## Tags

[Active Directory](#) [ARM](#) [CAN](#) [CANBus](#) [CANOpen](#) [computer engineering](#) [computer networks](#) [Computer Security](#) [CORBA](#) [covert channel](#) [distributed systems](#) [embedded systems](#) [javascript](#) [jquery](#) [Linux](#) [MySQL](#) [NSIS](#) [open source tools](#) [Python](#) [reverse engineering](#) [RMI](#) [Software Engineering](#) [SQL](#) [SQUID](#) [Twitter](#) [vulnerability engineering](#) [windows](#)

© 2016 Practical Software Engineering. All Rights Reserved.

Powered by [WordPress](#). Designed by  WooTHEMES