Bogdan Cristea

< cristeab@ieee.org >

Revision History	
Revision 0.1	2009-11-21
First release	

Table of Contents

1. Introduction
2. BuildRoot configuration
3. Testing Linux image with QEMU

1. Introduction

Software development for embedded systems is traditionally done directly on hardware, where both the real-time Linux kernel and the accompanying file system are installed. An alternative approach is to use a processor emulator like QEMU in order to start the development process without the underlying hardware. While this approach can be used for the early stages of software development, it cannot replace completely the traditional approach. However, using a processor emulator has also an academic value, allowing students to become familiar with the real-time Linux kernel and with software development techniques for embedded systems.

A good starting point for embedded software development with QEMU is represented by the article "Efficient embedded software development using QEMU" (PDF) by Pradyumna Sampath and Rachana Rao, presented at the 11th Real Time Linux Workshop, 2009. The authors are using a real-time Linux kernel image compiled for Power PC system, but they don’t provide details about how the kernel image, the accompanying file system and the cross compilation toolchain have been built.

A possible solution for compiling an embedded Linux kernel image for a given target system (e.g. i386, Power PC, ARM, MIPS) is represented by the BuildRoot project. This project provides a set of Makefiles and patches allowing also to easily generate a cross-compilation toolchain and the file system needed by the target Linux system.

This HOWTO tries to provide a detailed description of the steps involved in the creation of a cross-compilation toolchain, the real time Linux kernel image and its file system using BuildRoot.

2. BuildRoot configuration

Download first the BuildRoot sources either by using the latest stable release or the latest daily snapshot. I recommend using the latest daily snapshot, since BuildRoot is a rapidly developing project and new features are often added. In the following, **buildroot-20091114.tar.bz2** daily snapshot has been used. An Internet connection is needed since BuildRoot downloads selected packages (e.g. BusyBox, Linux

sources, etc.) from the Internet.

In the following, it is assumed that the Linux terminal is used. Unpack BuildRoot sources in some folder as regular user and change the working directory to that folder. BuildRoot has a configuration interface similar to the configuration interface of the Linux kernel. In order to launch the configuration interface use

```
make menuconfig
```

The configuration menu shown below

```
------------------------------------ Buildroot Configuration -------------------------------------
|  Arrow keys navigate the menu.  <Enter> selects submenus --->.  Highlighted letters are hotkeys.     |
|  Pressing <Y> selectes a feature, while <N> will exclude a feature.  Press <Esc><Esc> to exit, <?>   |
|  for Help, </> for Search.  Legend: [*] feature is selected  [ ] feature is excluded                 |
|                                                                                                     |
| ----------------------------------------------------------------------------------------------- |
| |                  Target Architecture (i386)  --->                                             | |
| |                  Target Architecture Variant (i686)  --->                                     | |
| |                  Target options  --->                                                         | |
| |                  Build options  --->                                                          | |
| |                  Toolchain  --->                                                              | |
| |                  Package Selection for the target  --->                                       | |
| |                  Target filesystem options  --->                                              | |
| |                  Kernel  --->                                                                 | |
| |              ---                                                                              | |
| |                  Load an Alternate Configuration File                                         | |
| |                  Save an Alternate Configuration File                                         | |
| ----------------------------------------------------------------------------------------------- |
|-----------------------------------------------------------------------------------------------|
|                                  <Select>    < Exit >    < Help >                                 |
-------------------------------------------------------------------------------------------------
```

allows to select **Target Architecture** (i386 architecture has been selected) and **Target Architecture Variant** (i686).

The **Target options** allow to configure several parameters of your target (e.g. **System hostname**, **System banner**).

The **Build options** usually do not need to be changed, they contain several commands for package processing (e.g. wget, gzip).

The **Toolchain** menu allows to specify options for the cross-compilation toolchain as **Kernel Header Options**, **uClibc Options**, **Binutils Options**, **GCC Options**, **Gdb Options**, etc. For this example, in **Gdb Options**, **gdbserver** has been selected to be built for the target.

The **Package Selection for the target** menu allows to specify various packages needed on the target system. Normally, at least Busybox package should be selected in order to have a minimal set of Linux commands. Also, **dhcp support** and **sshd** have been selected to provide a network login to the target system using ssh.

The **Target filesystem options** allow to select the filesystem type for the target along with other options. For this example, the ext2 root filesystem has been selected.

The **Kernel** menu allows to specify the **Linux Kernel Version** (2.6.29.6) and the **patch name** (patch-2.6.29.6-rt24.bz2) to be built for the target. One can also provide the **patch site** from where the patch archive can be downloaded. At the time of writing this document, these were the versions of the Linux sources tarball and its RT-Preempt patch as can be found on the OSADL site, "Latest stable" section. Note that **Kernel Header Options** from **Toolchain** menu should contain the same version of the Linux kernel as the one provided in this menu.

After all needed packages have been selected, save the BuildRoot configuration file and start the compilation with

```
make
```

During the compilation process, the real-time Linux kernel needs also to be configured through its configuration interface.

At the end of the compilation process, the Linux image, **bzImage** and its file system, **rootfs.i686.ext2**, should be found in **output/images**. The cross-compilation toolchain binaries should be found in **output/staging/usr/bin** and the related libraries in **output/staging/usr/lib**.

3. Testing Linux image with QEMU

In order to launch QEMU, configured for the previously obtained real time Linux kernel, the following Bash script can be used:

```
KERNEL="bzImage"
DISK="rootfs.i686.ext2"

qemu-system-i386 -kernel $LOCATION/$KERNEL \
-hda $LOCATION/$DISK \
-boot c \
-m 128 \
-append "root=/dev/sda rw" \
-localtime \
-no-reboot \
-name rtlinux \
-net nic -net user \
-redir tcp:2222::22 \
-redir tcp:3333::3333
```

where the LOCATION variable contains the path where the real time Linux kernel image and its file system reside.

The command line parameter **-redir tcp:2222::22** tels QEMU to redirect all requests from host port 2222 to target port 22 (default ssh port on the target). Similarly, the command line parameter **-redir tcp:3333::3333** tells QEMU to redirect requests from host port 3333 to target port 3333 (this should be the port used by gdbserver on the target). Note that QEMU is able to handle correctly those command line parameters only if the specified port on the host isn’t already open. For a detailed description of all command line parameters see QEMU documentation.

Before launching QEMU, a final step is needed to configure the network interface of the target. In order to do so, mount locally the target file system with:

```
sudo mount -o loop rootfs.i686.ext2 /mnt
```

Append at the end of the **interfaces** file, found in **/mnt/etc/network**, the following lines:

```
auto eth0
iface eth0 inet dhcp
```

and unmount the file system

```
sudo umount /mnt
```

Thus, the network interface of the target is configured to take an IP address provided by a DHCP server in a virtual network created by QEMU.

Launching QEMU with the above script should start a new window allowing to see the booting process of the real time Linux kernel. At the end of the booting process (if successful) one should be able to connect to the target from the newly created window, using the **root** account (no password is needed). Verify that the network interface of the target has been correctly configured with:

```
ifconfig
```

The **eth0** network interface should have been configured with an IP address of the form **10.0.2.15**.

Further, in order to be able to connect from the host to the target through ssh, at least one user account with password needs to be setup on the target. For this purpose, type

```
adduser user_name
```

in the target terminal. Thereafter, the following command can be used in a host terminal:

```
ssh -p 2222 user_name@localhost
```

in order to connect through ssh to the target. Thus, one is able to transfer from the host to the target binaries compiled with the cross compilation toolchain. Also, for embedded software development, Eclipse CDT with Remote System Explorer can be used.

The virtualized target system in a powerful host computer offers two ways of development:

- Self-hosted development directly on the target system. This may be somewhat slower than on the host system, (especially, if the target architecture requires emulation), but it may still be very effective.
- Cross-development. This normally provides short compile times but may require more work to find the adequate settings of the environment and compile options.

In any case, the BuildRoot method along with a virtualized target system provides both procedures in such a way that the developer can select the best of the two worlds - even interchangeably - for a particular purpose. However, cross-compilation will be needed anyway, when the final hardware becomes available and the processor is too slow to be used as development system.

So, a virtual machine using a real time Linux kernel has been created allowing software development for embedded systems. By rerunning the configuration interface for BuildRoot, other target architectures can be selected, depending on your needs.