Efficient embedded software development using QEMU

Pradyumna Sampath
ABB Corporate Research
Bhoruka Tech Park, 5th Floor, Block 1,
Whitefield Road, Mahadevapura,
Bangalore, Karnataka, India
pradyumna.sampath@in.abb.com

Rachana Rao
ABB Corporate Research
Bhoruka Tech Park, 5th Floor, Block 1,
Whitefield Road, Mahadevapura,
Bangalore, Karnataka, India
rachana.rao@in.abb.com

Abstract

Software based processor emulators have been around for a long time and predominantly used by developers of Operating Systems. With virtualization technology becoming more common and gaining in-processor support, these emulators have expanded to support several processor architectures and a richer feature set. Consequently, popular operating systems such as Linux have been adapted to run within virtual environments seamlessly. As part of the software development process, this virtualization technology has huge potential to increase product quality and developer efficiency.

This paper looks at employing virtualization for in-house real-time application software development through QEMU, a popular open source emulator. It also details the process of establishing such an environment and looks at some of the advantages of such a development model for embedded systems.

1 Introduction

Industrial Automation and Power technologies are characterized by extremely complex software systems which are in many cases very tightly coupled with mechanical and electrical sub-systems. These systems tend to be safety and mission critical. Development of such systems is considerably time consuming. However, they are also charecterized by long if not very long product lifecycles sometimes in the order of fifteen years. The scope and complexity of software components in these industrial products has gone up significantly over the years. Continuous remodeling and development of software for the enduring industrial automation systems, places considerable challenges of efficiency on the development environment and processes followed. This paper attempts to look at acceleration of embedded software development by decoupling it from the underlying hardware.

Traditionally, embedded software development is inherently dependent on hardware availibility. Hardware development includes design, simulation and testing of hardware architecture, logic, circuit schematics and finally the PCB. This is a very time-consuming process, often iterative and unpredictable due to dependency on factors like component availability and vendor support. Software developers usually cannot afford to wait for the entire hardware design to be completed and the board to arrive on their desks before starting on application development. This would increase the product development time substantially which is unacceptable. Application development as well as verification(testing) should be well underway by the time the hardware

is finalized and delivered to the software developers. This is where system emulation and virtual environments step in and save the day. With the various advanced hardware emulators and virtual machines available today, embedded software developers can get a head start on their applications.

The ablility to develop applications for a hardware without the physical device itself is a definitely a big advantage. However, there are other advantages provided by the emulator too. It can make a safe and secure environment for the testing of new and untried applications. Since modern emulators now support various kinds of hardware as well as different operating systems, the application can also be tested for scalability and reliability and not just functionality. This makes virutalized environments very powerful and feasible testing platforms. Emulators also prove valuable in cases of kernel and device driver development, where a small mistake can crash the entire operating system. Developing and debugging drivers on an emulator, makes it similar to user-space applications, which at the worst can lead to the emulator crash.

2 Virtualization and Emulators

Since the term virtualization[1] was coined, over four decades ago, it has been defined many times and in many different ways. Virtualization is bascially abstraction of one or more computer resources to achieve the behaviour of the desired system. The desired system here could be a high end grid computer or a testing platform for a different architecture etc. In the beginning, the term emulation was usually used for virtualization using hardware. However, recently it has also become common to use the term emulation in the software context. Emulation[2] refers to the replication of functions of a system by another, so that the emulator acts similar to the emulated system.

There are various types of virtualizaton[3][4], depending on the system being virtualized and levels of abstraction used to achieve it. Hardware emulation refers to the virtualization of the required hardware on the host machine. It usually suffers from being slow, however the major advantage offered is the ability to simultaenously develop software through simulation, as the hardware is being developed. QEMU and Bochs are examples of hardware emulators. Native or Full virtualization uses a hypervisor which acts as a mediator between the operating system and the underlying hardware. This method is faster than emulation, however the criteria is that the underlying hardware must be supported by operating system. KVM and VMware are examples of full virtualization solutions. Paravirtualization uses a hypervisor for shared access to the underlying hardware. This method offers the best performance. However, since the virtualization code is integrated into the operating system itself, the guest operating systems have to be modified for the hypervisor. Xen and User Mode Linux are examples of paravitualization.

3 Our Setup

3.1 Hardware

The target for emulation is a custom board made by ABB. This board is based on the Freescale Lite5200 evaluation board[5]. The microprocessor on board is the Freescale MPC5200B[6], a PowerPC embedded processor based on the 400MHz MPC603e series e300 core. It is provided with 64 megabytes of RAM (DDR) and 32 megabytes of flash memory.

3.2 Software

The target board is a part of time critical control systems and hence there are real time constraints on the application response times[7]. To accomplish this, the target runs Linux kernel patched with the latest rt-preempt patch[8]. As of now, the version running on the hardware is Linux 2.6.29.4-rt17. However, since the real time performance is not a criteria as of now, we have reverted to the non rt version of the Linux kernel for this emulation. Hence, for the present time our requirements are limited to the latest mainline kernel and also support for ethernet ports. Ethernet is needed by the application to be run on the board emulator as well as for remote debugging of the application.

Traditionally, development of embedded software has been done on Windows host, cross-compiled and deployed on the target. In order to reduce the learning curve for embedded developers, it is important that the host platform for this emulation be retained as Windows.

4 Choices

A few virtualization methods and solutions were evaluated keeping the above needs in mind. Below is the list of the options considered.

4.1 Kernel Virtual Machine (KVM)

KVM[9] is a full virtualization solution for Linux on x86 hardware. It comes in the form on a loadable kernel module which provides the virtualization infrastructure as well as the processor specfic implementations. KVM has been a part of the mainline kernel since 2.6.20 and is a very active open source project with a strong developer community. The PowerPC support however is still work in progress. Also there is no support from Windows as the host machine.

4.2 VMware Server

VMware server[10] is a virtualization platfrom which allows one to create and run virtual machines for different operating systems on a Linux or Windows x86 machine. It is partly open source. However, it does not support PowerPC emulation.

4.3 Xen

The Xen[11] virtual machine monitor is a paravirtualization solution for x86 and PowerPC architectures. The Xen hypervisor is the most privileged layer, above which a wide range of guest operating systems can run concurrently. The guest operating systems need to be modified to be able to run on Xen. Also the PowerPC support does not extend to MPC603e core processors.

4.4 PearPC

PearPC[12] is an architecture independent PowerPC platform emulator. This is an outdated project with no current developments ongoing.

4.5 QEMU

Qemu[13] is an open source machine emulator written by Fabrice Bellard. It has two modes of operation. In the User Mode emulation, it can execute binaries compiled for different architectures on a x86-Linux host. It can achieve near native performance in this mode. In the System Mode emulation, operating systems and applications complied for different architectures can be run on the host system with a different Architecture-OS combination.

4.6 Why QEMU

Of the above options, a decision was made to use QEMU for the emulation of the ABB custom board. It was based on the following considerations.

- QEMU supports MPC603e core processor emulation
- QEMU supports both x86-Windows and x86-Linux as host machines
- QEMU is an active open source project with a good developer community and support

5 Programming Embedded systems using QEMU

An effective IDE which provides for efficient application development consists of two main components

5.1 QEMU

For the reasons mentioned above, QEMU is the system emulator that was selected to emulate the ABB custom board for which applications need to be developed. The applications can be run and debugged on the QEMU emulator just as they would on the real board. The compilers and linkers used to build the source code also remain the same, i.e. the application is compiled on the host machine for the Lite5200 target using cross-compilers (ELDK[14] on Linux and CodeSourcery Lite[15] on Windows) as usual. It is also possible to (remotely) debug the application running on QEMU from the host machine using GDB.

5.2 Eclipse

Eclipse is widely adapted as a development platform for embedded systems for many good reasons[16]. One of the main reasons is that it is an open framework with a plugin architecture that enables easy development and integration of a variety of software tools for embedded development. This is also the reason why it supports usage of a number of third party tools and interoperability between them, which is a especially pertinent to embedded development. Most of the issues faced by embedded software developers like cross-platform development and debugging, application portablility etc have been addressed and solved to a large extent by the vast open source community surrounding this project. A number of commercial embedded IDE vendors also employ Eclipse as their foundation to build upon[17].

Eclipse Ganymede[18] is used along with C/C++ Development Tools (CDT), Remote System Explorer (RSE) and Remote CDT as the development platform on a Windows host. The CDT perspective is used for application coding, compilation and linking. It provides a means of building the application for the Lite5200 platform thru the use of CodeSourcery cross tool chains for PowerPC. The RSE perspective

is used for establishing a SSH connection with the QEMU. The RSE provides a console to the QEMU and also enables one to browse through the remote QEMU filesystem. The SSH connection made by the RSE is also used by the Remote CDT debugger to remotely run and debug applications on the QEMU emulator.

6 QEMU on PowerPC

6.1 The right QEMU-PowerPC

Unfortunately, support for PowerPC platform in QEMU is unstable. The QEMU emulator version 0.9.1 works for the latest kernel versions on PowerPC platform [19] The latest QEMU releases (up to 0.10.5) support Linux kernels only up to the version.2.6.26. Presently we have a 2.6.26 kernel running on PowerPC QEMU on Windows host that is primarily used for application development and debugging. We also have the 2.6.31 kernel running with the older QEMU on Linux host that is used as a test machine.

6.2 Kernel image for PowerPC

The Linux kernel image for QEMU-PowerPC is built in the usual manner, however arriving at a working kernel configuration and compiling a working system image requires effort. We use the standard Debian configuration(Lenny) with a few minor changes on Linux host. The DietPC[20] project has been a valuable resource in getting the QEMU PowerPC up on Windows host.

6.3 Getting QEMU-PowerPC up

The emulator is invoked using the following command

```
qemu-system-ppc -L . -M g3beige -m 128 \
-localtime -no-reboot -name P-Robo \
-hdc "PRobo.img" -boot c\
-net nic -net user \
-redir tcp:22::22 tcp:2345::2345
```

Where qemu-system-ppc is QEMU PowerPC emulator.

- -L . points to the directory containing the openbios firmware.
- -M g3beige sets the emulation to beige G3 powermac which is compatible with PowerPC 603e cores.
- -m 128 sets the virtual RAM size to 128M.
- -localtime sets the RTC to local time.
- -no-reboot sets up QEMU to exit instead of reboot.
- -name P-Robo sets the name of the guest to P-Robo.
- -hdc "PRobo.img" directs QEMU to use PRobo.img as the ide hard disk image.
- -boot c sets up the QEMU to boot from hard disk.
- -net nice creates a NIC to and connects it to a VLAN.
- -net user connects a user mode network stack to the VLAN.
- -redir tcp:22::22 redirects SSH connections from host to guest.
- -redir tcp:2345::2345 redirects default port for gdbserver connections from host to guest.

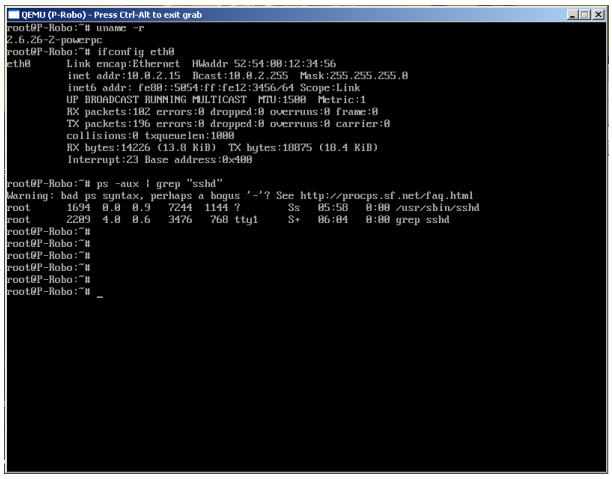

FIGURE 1: *QEMU PowerPC on x86-Windows host*

6.4 Eclipse

6.5 CDT and CodeSourcery

Eclipse Ganymede (with Eclipse CDT 5.0.1) is used as the IDE for developing and building the application. The CodeSourcery Lite cross tool chain for PowerPC is used to compile for mpc5200 target on Windows host. The cross compiler and linker are specified under the C/C++ build section in the project properties along with appropriate build flags.

6.6 SSH connection to QEMU

RSE (version 3.0.0 from DSDP) is used to create a SSH connection to the QEMU emulator. This can be done by connecting to the local host via SSH, since the SSH port of the host has been redirected to the SSH port of the emulator. The procedure to connect to the QEMU via SSH on root login is depicted by the snapshot below.

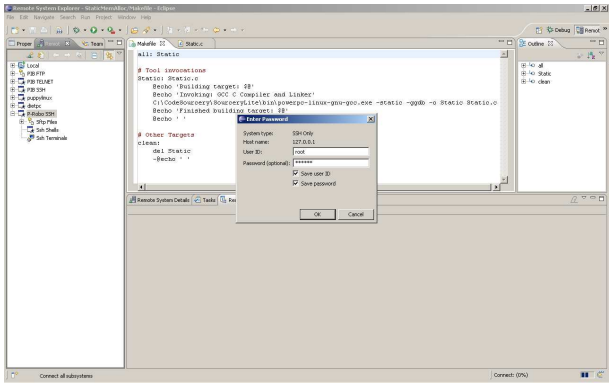

FIGURE 2: *Creating a SSH connection from host to QEMU in Eclipse*

Once this connection had been established a remote shell can be launched on the QEMU using the "Launch Shell" option under "SSH Shells" as shown below. This shell can be used to browse the emulator filesystem and run commands on the emulator.

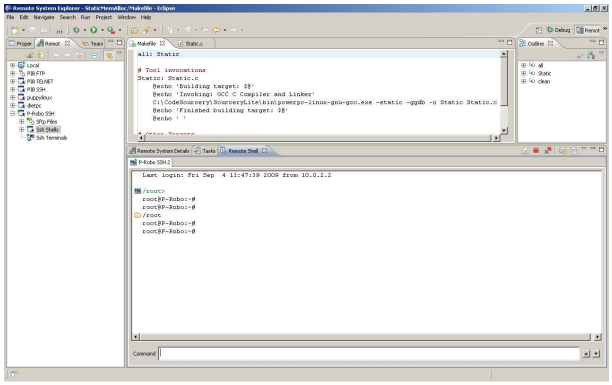

FIGURE 3: *QEMU (remote) shell on Eclipse*

6.7 Application Execution on QEMU

Once the application has been built for the target, it can be downloaded and run on the target by choosing the appropriate run configuration. The "C/C++ Remote Application" configuration has be selected and settings like "Connection", "Remote Absolute File Path" etc has to be set as shown below.

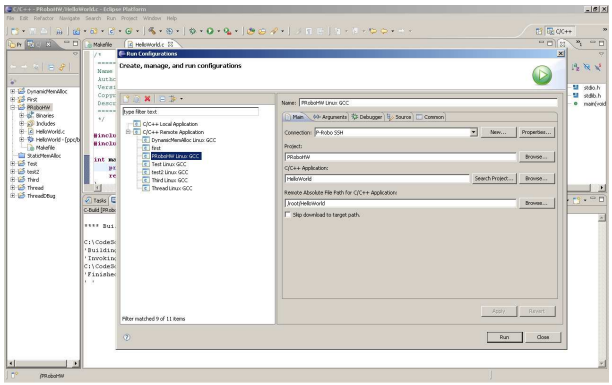

FIGURE 4: *Setting up Eclipse to run the application remotely*

On running the application, the output is displayed on a remote console on Eclipse as shown.

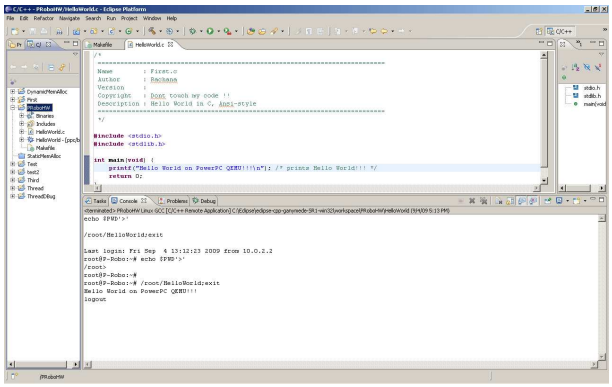

FIGURE 5: *Remote QEMU console on application*

6.8 Remote Debugging on QEMU

Once the application is running on the remote emulated board, the next step is to debug it. This is done by setting the debug configuration to "C/C++ Remote Application" and directing it to the right cross-tool gdb executable

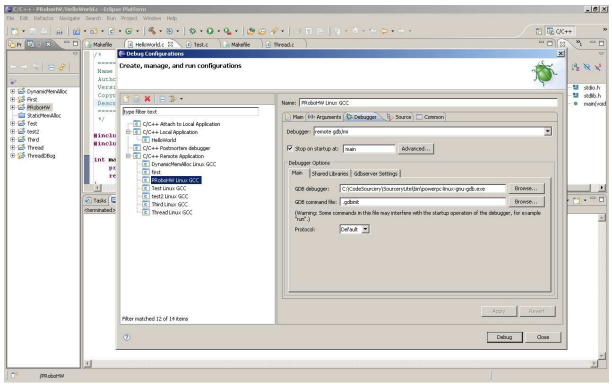

FIGURE 6: *Setting up Eclipse to debug the application remotely*

Debugging the application on the remote emulated host is done in a similar manner as with native applications. The snapshot below shows a very simple hello world program being debugged on the PowerPC QEMU.

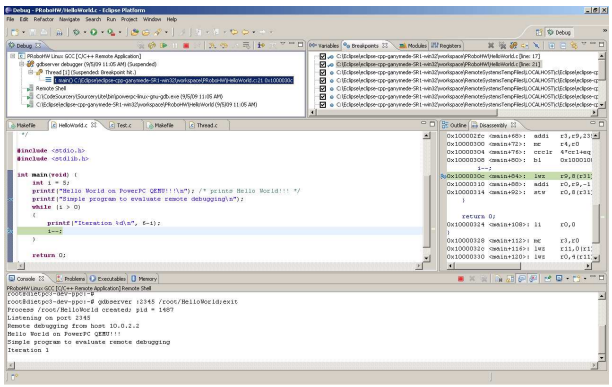

FIGURE 7: *Debugging application remotely on QEMU with Eclipse*

7 Conclusions

Using the above mentioned method, it is thus possible to create a development environment for embedded applications, that frees the developer team from the constraints imposed by hardware. It is also possible to further develop this into an secure and efficient testing platform. Virtualization enables this by providing the ability to simulate components like load generators, performance monitors and other resources beyond the scope of a usual test setup. QEMU proves to be a viable tool today to shorten development time for complex embedded system projects. When integrated with Eclipse it makes a fairly complete software development environment reducing the time to market for embedded products. Having a complete development and test emulation in place without having to depend on the underlying hardware, is a great advantage to embedded software developers attempting to optimize the product development in this fast changing market

8 Future Scope

QEMU for x86 based embedded platforms today is stable and effective. However, it has some distance to go when it comes to extensibility to allow users to easily customize peripherals and also support multicore and latest CPU architectures other than x86.

The potential that simulation and emulation technologies offer in different stages of embedded software development is truly exciting. Be it easy and quick prototyping of research ideas for concept validation or to simulate large and distributed systems in a virtual environment, these technologies save valuable time and effort. One interesting use of these technologies is in cross-platform development without the need for cross-compliers and linkers[21]. Cross compiling the BSP and user-space applications for another platform, is the most tedious and error prone stage of cross-platform development that requires repeated setting up of the environment variables and tool upgradation. Creating a virtualized native development environment for the applications is an efficient way to overcome these obstacles.

9 Glossary

QEMU	Quick EMUlator
IDE	Integrated Development Environmnet
CDT	C/C++ Development Tools
RSE	Remote System Explorer
ELDK	Embedded Linux Development Kit
SSH	Secure SHell
GDB	GNU Debugger

References

[1] *http://en.wikipedia.org/wiki/Virtualization*

[2] *http://en.wikipedia.org/wiki/Emulator*

[3] *Virtual Linux, An overview of virtualization methods, architectures, and implementations by M Tim Jones, IBM developerWorks*

[4] *An Introduction to Virtualization by Amit Singh, Kernelthread.com*

[5] *http://www.freescale.com/files/microcontrollers/doc/prod_brief/MPC5200LITEPB.pdf?fsrch=1*

[6] *http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=MPC5200B&fsrch=1*

[7] *Evaluation of Linux rt-preempt for embedded industrial devices for Automation and Power technologies - A case study by Morten Mossige, Pradyumna Sampath, Rachana Rao*

[8] *http://rt.wiki.kernel.org*

[9] *http://www.linux-kvm.org/page/Main_Page*

[10] *http://www.vmware.com/products/server/*

[11] *http://www.xen.org/*

[12] *http://pearpc.sourceforge.net/*

[13] *http://www.qemu.org/*

[14] *http://www.denx.de/wiki/DULG/ELDK*

[15] *http://www.codesourcery.com/sgpp/lite/power*

[16] *Eclipse: Under The hood, Overview of the Eclipse open-source IDE framework by Robert Day, Embedded.com*

[17] *Standardizing on a common embedded Eclipse IDE by Martin Whitbread, Embedded.com*

[18] *http://www.eclipse.org/ganymede/*

[19] *Rob Landley's blog thing for 2009 - http://www.landley.net/notes.html#25-06-2009*

[20] *DIET-PC(DIskless Embedded Technology Personal Computer) project by Paul A. Whittaker - http://www.dietpc.org/*

[21] *Firmware Linux - Embedded Linux build system by Rob Landley - http://www.landley.net/code/firmware/*