Sustainable Speedup of a Legacy Semiconductor Manufacturing Parametric Test System

CM Heong[1], Emelia Khamis[2], CM Thong[3]
[1,2]*Member, IEEE*

DEDICATION

We would like to thank Mr. Chan Chee Kian for his valuable assistance in the early part of this project. Mr. Chan left to pursue other interests.

***Abstract*—the sustainable speedup of a manufacturing process without compromising product quality and without substantive investment is particularly relevant during difficult economic conditions.**

The average time a tester spends testing a device can be divided into two categories: mandated program test time and others. Others include time taken by the CPU to process the operating system and application program, and the time taken for communications between the two embedded controllers.

Speedup gained by optimizing and changing the test program may have a direct impact on product quality and need to be qualified very carefully by a full Quality team comprising Equipment, Process, Product and Test engineers.

For older obsolescent systems there is an opportunity to speed the "other" category.

Current advances in CPU speed, and especially virtualization technology allows a novel approach to speeding up obsolescent embedded controllers

The approach is to exploit the new VM instructions to model an entire embedded controller in software, so that the proprietary kernel, operating system, application program and most importantly the test program can run without modification. While emulation results in a slower system, CPU speed gains since 1990 more than compensated for the overhead of running an emulator. This resulted in a speedup of as much as 20% in test time.

The insights gained from modeling the controllers resulted in non-emulated speed gains of up to 50%.

***Index Terms*— parametric tester, QEMU, speedup, virtual machine.**

I. INTRODUCTION

In these challenging economic conditions semiconductor manufacturing cost-efficiency is a plus and extra capacity seems superfluous. Yet, if extra capacity can be obtained without significant investment or operating cost, this results in greater cost-efficiency: fewer machines and people can be used for the same output, in less time.

Our manufacturing facility has over 100. 1980-era semiconductor parametric testers doing 100% testing on a variety of discrete devices including diodes, TVRs, MOSFETs and bipolar transistors.

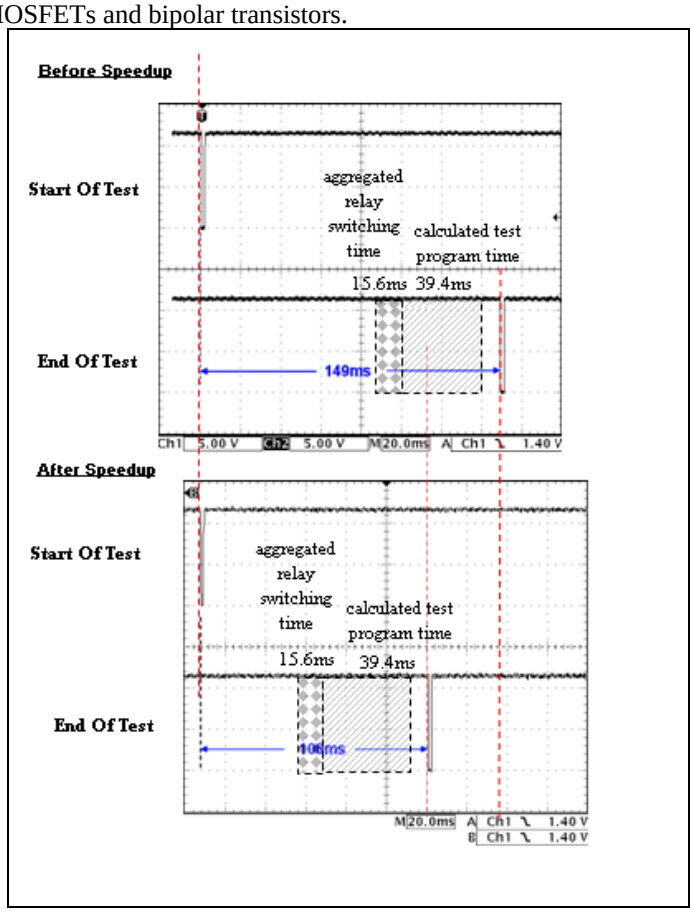

Fig. 1 Tester-Handler cycle. By Emelia Khamis.

Test time per unit varies from 80ms-300ms, resulting in a throughput of 12 to 30 thousand units per hour (Kuph).

An analysis of the tester's operation shows that the time required to test a device consists of test time, processing time and time spent waiting for downstream equipment. Units are fed manually in bags of 2500-15,000.

Test program time is the time required for the parametric test suite itself (called a program). This is some 40-250ms per device. Processing time is the time required by the computer to set up the test, and to read, calculate and transmit the results. This is variable for different devices, but has been estimated to be around 85ms. Delay time is the time spent waiting for downstream equipment, usually the test handler, and this can be 0-80ms. For

example the test handler can take some 80ms to move each device into position. It can overlap its motion with some 80ms of test time, so if each device took exactly 80ms to test, the handler will operate at optimum speed of 22.5Kuph. Any test time more than 80ms will slow down the overall speed.

Test program time can be reduced by carefully optimizing the individual tests in the test program. This, however is a multi-departmental effort involving Equipment, Test Engineering, Process and Quality Assurance groups. Handler Delay time is easier to modify, and can be done by a single group but can involve expensive mechanical modifications.

The low-hanging fruit is clearly the CPU processing time of 85ms. Reduction of this processing time will greatly improve the overall Production Line speed.

We show how Virtual Machine technology can be used to replace or upgrade legacy equipment controllers. This is done purely by upgrading the host controller to a 64-bit PC, and without modifying the original equipment software.

II. Tester Architecture

Our legacy Tester comprises a host PC controller connected to two Multibus I local controllers, each capable of testing one device. The connection is via high-speed parallel interface card. This divides our speedup problem neatly into host CPU, interface card channel and local CPU. By modeling each section we can then observe the effect of local speed-up each section on overall speed.

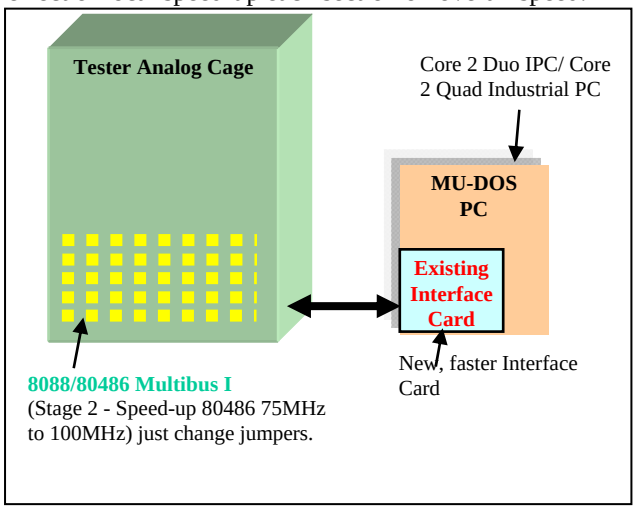

Fig. 2 Tester and interface card (existing system). Graphic by SK Teo, used with permission.

III. The Problem

Our legacy Tester ran a PL/I application on top of PC-based realtime legacy OS. Over many years, the host PCs became obsolete and increasingly expensive to maintain. For example, it is no longer easy to acquire advanced PCs with the legacy ISA Bus. The main problems are with the network cards: the legacy OS supported just 5 NICs, and all are obsolete. The original host is a DEC Pentium I 166MHz MMX PC, and the last compatible PC hosts were the 10-year old Dell Pentium III Optiplex GX1.

IV. The Solution

Virtual Machine approach seems like the ideal solution: it offers the original OS and application program the same emulated environment regardless of the actual machine used. In this case, we use Linux and QEMU to emulate an entire PC in software. There is a price to be paid: performance. But if the measured loss in performance can be made up by a much faster host PC, then the virtual machine approach is worth implementing just for maintenance reasons. Also there is no reason that given enough speed in the host machine, why the emulated PC performance should not **exceed** that of the original system.

V. Virtual Machine

A Virtual Machine is defined as “an efficient, isolated duplicate of a real machine” [Ref 1]. A computer program is used to emulate and replace a real computer. A single computer can then run a few operating systems and their applications at the same time. It is currently very popular in server farms and is used to consolidate operating costs by having one powerful computer replace several servers.

Our prototype replaced a legacy Pentium I 166MHz MMX PC semiconductor parametric tester with a 2.4GHz Intel Core 2 Duo for a 27% increase in throughput for a very reasonable investment of USD1K.

The key is the efficiency of the emulation, and the speed penalty we pay for it; from Fig 2, we see that a 70-20% penalty is possible.

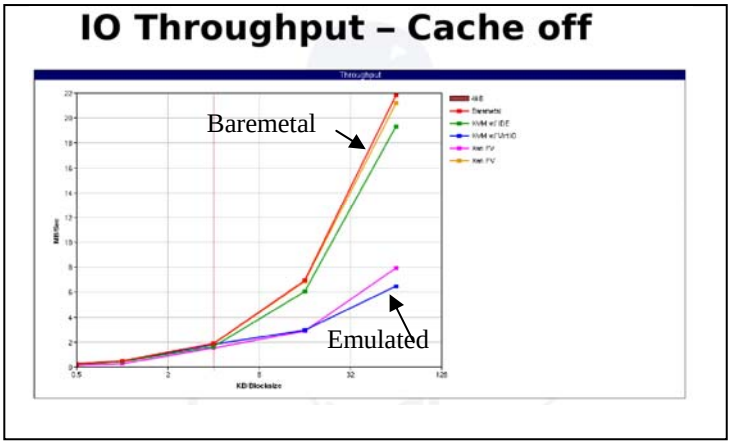

Fig. 3 Chart of virtualization speed [Ref 7].

A Pentium MMX Whetstone performance is 105, Pentium III 159, and Core 2 Duo 178 (Ref 4). Very roughly a Core 2 Duo is 69% faster than the Pentium MMX and 12% faster than the Pentium III. This makes possible the use of VM techniques for embedded control, particularly for legacy systems where we can make up for emulation overhead using the latest-model CPUs.

There are roughly 3 classes of virtual machines: pure emulation, OS emulation, and virtualization. OS emulation

is impractical for embedded systems, so that leaves pure emulation and virtualization if our host CPU is the same family as the target CPU.

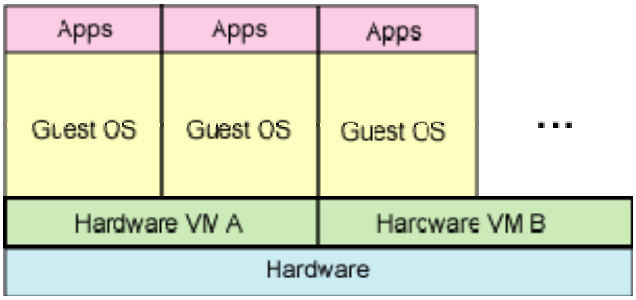

Fig. 4 Pure emulation. Embedded controllers 'Hardware VM' are emulated using a program. [Ref 5]

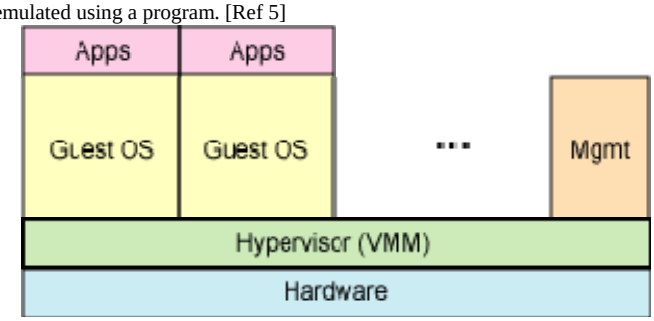

Fig. 5 Virtualization. Note the use of a hypervisor program. [Ref 5]

We go for pure emulation: it is a more general solution and more likely to work first time (cycle-exact emulation, arbitrary guest CPU). If speed is an issue we can then revisit virtualization (but also restricting our target CPU).

A classic pure emulator like Bochs [Ref 2] fits the bill but we need to leave the door open for virtualization, especially for the very special case of the PC. Virtualization gains rely on the fact that some guest code can be directly executed by the host without emulation. This is made possible by new virtualization instructions on some 64-bit CPUs from AMD and Intel.

In addition, system resources required by the embedded processor, like mass-storage drives, graphics cards, network cards, etc need to be emulated. Such a "system" emulator can often support more than one guest (each of which may run a different embedded OS). Low-level coordination of the host resources are done using a "hypervisor" which can be Type 1 (a microkernel that runs directly on hardware before OS is loaded) like Xen, or Type 2 (runs from a host OS which then runs the guest as a regular user application).

On a low budget, this means Open Source which reduces to Xen (Type 1) and Qemu (Type 2).

Xen requires changes to the target OS, which seems to be an additional step, and recurring effort for different embedded systems.

That leaves Qemu.

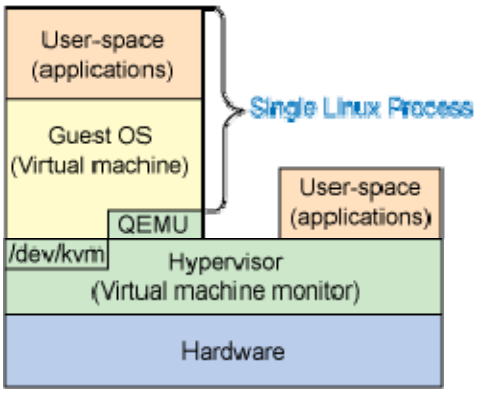

Fig. 6 System emulation using Qemu in combination with virtualization, using KVM, for the PC-based controller. [Ref 5]

VI. QEMU

Qemu is a project by Fabrice Ballard [Ref 6]. It is a system emulator, and is a relative newcomer, has no company sponsor, and lacks paravirtualized drivers [Ref 7]. In spite of this Qemu is the most portable (it runs on 9 different hosts), and amazingly emulates some 13 systems, with support for 200 devices [Ref 9].

Qemu's system emulation is crucial: indeed it often seems any open source virtual machine project of consequence uses qemu: KVM, Xen HVM and xVM VirtualBox. Qemu seems to be the proverbial "Last Mile" where no matter what the upstream advances we still need to squeeze into the system emulation straightjacket [Ref 7].

So, Qemu for pure emulation with the possibility for speed increases via dynamic translation and virtualization. The former via Kqemu (Fabrice Ballard's dynamic binary translator), and the latter using Qumranet's KVM.

VI.I. QEMU, KQEMU AND KVM PERFORMANCE AS EMBEDDED CONTROL SYSTEM EMULATOR

We find that "KQemu acceleration" is not necessarily faster than pure emulation of an embedded controller, and is actually marginally **slower.** KVM failed to load our embedded legacy OS. We believe this is because KVM's virtualization is incomplete. In particular some 16-bit CPU instructions have not been implemented yet. This is consistent with Qemu and KVM directions: they tend to implement features for Linux, Windows (both 32-bit OS) first, since the bulk of the implementers and users use these systems. Since both are ongoing projects and incomplete a successful virtualization for a legacy embedded system is quite accidental.

Thus our subsequent use of Qemu is restricted only to the "pure emulation" features, without using the dynamic binary translation (Kqemu) features or the 64-bit virtualization (KVM) features.

The resulting performance for the pure emulation of a legacy guest embedded system is still surprisingly good. In particular, Linux's scheduler seems better than the legacy OS: overall throughput was higher. It was advantageous to move some tasks into separate (as much a

4) virtual machines and leave the legacy OS with just one main task.

VI.II. QEMU ISSUES

When running the legacy OS image, floppy disk driver does not work properly; we had data corruption issues.

Keyboard I/O of emulated legacy OS is too slow, more work, maybe required for production ready system.

The preferred emulated network card is Novell's NE2000, for some reason seems more stable.

QEMU also proved be a useful debugging tool when we encountered tester-server LAN issues. It was used to emulate the server for problem isolation.

VII. LINUX

Qemu runs on both Linux and Windows, so there is a choice of host OS to use. We chose Linux, for cost reasons, and also since Qemu solutions tend to emerge for Linux first. Also, Linux offers more flexibility as an embedded system: we can chose "headless" (ie. no monitor or keyboard) configuration, tamper-proof (running from DVD-ROM), low power (running from USB flash drive) and lastly communications: Linux is easier to integrate into a heterogeneous (mixed Novell, Apple and Microsoft) LAN.

VIII. IMPLEMENTATION

VIII.I. CUSTOM HARDWARE EMULATION

Embedded systems often contain custom hardware which is not usually emulated in the standard releases of Qemu. In our case this is a custom high-speed interface card communications channel to the Tester local CPU, a Multibus II 80486. This is an ISA card which can handle **two** Testers.

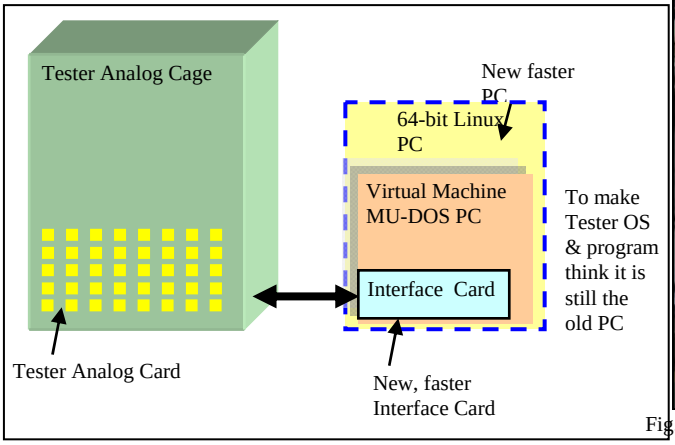

Fig. 7 Tester and Virtual Machine architecture (Speedup system). Graphic by SK Teo, used with permission.

This was done by first building a Linux device driver for the interface card, allowing Linux to communicate with the card. The interface device is relatively simple, we elected to use Qemu to trap accesses to the interface card IO addresses and "Pass-Through" to the guest OS. It can also be used to produce an I/O log can be used as diagnostic tools known for problems like system lockups. This was surprisingly easy with only some 600 lines of C code in total.

This required the use of a specialized Industrial PC (IPC), one that has a Core 2 Duo CPU and still retains the ISA bus (PICMG 1.0 or PICMG 1.2), and represents the major part of the declared cost of USD1K.

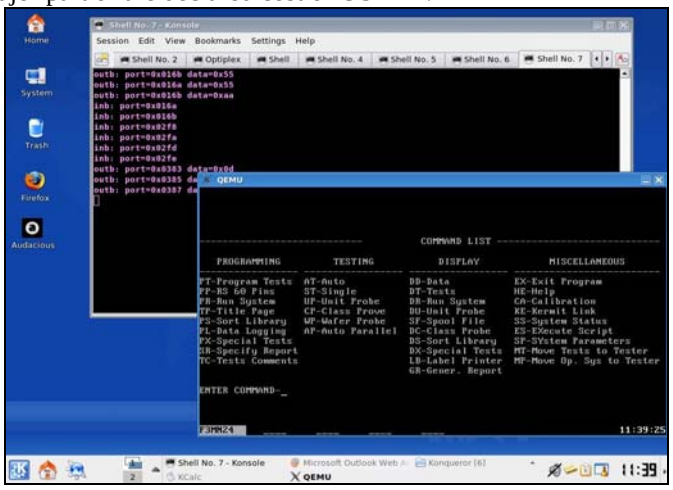

Fig. 8 Screenshot of Linux with Qemu and Emulated embedded controller. Note I/O log in background window. Photo by Emelia Khamis.

VIII.II. INTERFACE CHANNEL UPGRADE

The full solution required us to build a PCI version of the interface card. This will allow us to use desktops rather than the more expensive IPC. Alternatively, faster IPCs with PICMG versions above 1.2 can be used. It also lets the host control more than 2 Tester. This is especially useful when an extra 'QA' test suite is required.

Fig. 9 Photo of desktop with adapter card, with the ISA-bus interface card piggy-backed. Note interface card cable on the bottom left corner. Photo by Emelia Khamis.

This was tested using a PCI-to-ISA adapter card; the necessary software was completed. The final version of the PCI card is being designed, completion expected in 2010.

IX. Results

Despite many months of testing no functional difference was observed in the behavior of the Tester and Test Handler.

Under worst-case conditions for the upgrade, with the Tester running just one single parametric test, the virtual machine is 10% faster than the original Pentium MMX 166-based host. This is roughly equivalent to running the embedded system software directly on a Pentium III 500MHz host with 100MHz FSB.

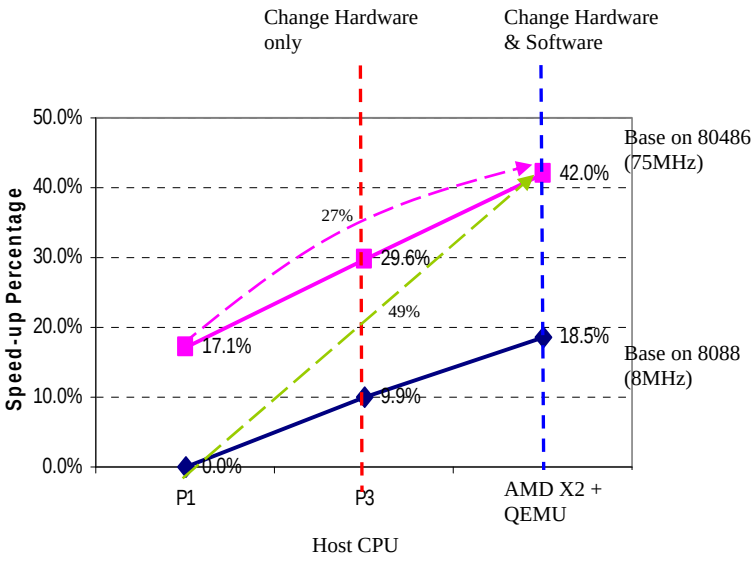

Fig. 10 Rate of Speedup chart. Chart by SK Teo.

X. Further Results

Qemu allowed us to build an emulator for the legacy Tester as well. Because Qemu merely "pass-through" Interface IO cycles from the Linux driver to the guest embedded system, it was straightforward to intercept and "record" the Interface communications. A very rudimentary Tester emulator can then be constructed by writing a program which implements a "state-machine" to "play-back" the communications. Luckily the legacy Tester contains a "script" execution mode where a fixed series of commands can be executed from a file. By combining a fixed Tester script with the correct "recording" the system can be modeled very quickly in software. This let us observe a new bottleneck in the Tester local Multibus I CPU. By upgrading this CPU from 75MHz to 100MHz, the overall speed can be increased to 18.5%. Further modelling show that the bottleneck has returned to the emulated host PC. This was borne out by running the embedded system directly on the IPC, which now returned speeds of between 130-150%

XI. Business Decision

A business decision was taken to lock in the speedup at 30-50%. This meant running **without** the Qemu virtual machine, and running the embedded system directly on the new Core 2 Duo. Workarounds were devised for incompatibilities, in particular for the new unsupported network cards, and for memory contention issues. The thinking is, we install the speedup hardware early, and when Qemu (in particular KVM) caught up in emulation speed, we can revert to emulated mode by dual-booting into Linux. The project is currently in the final stages of qualification for Production.

XII. Memory Contention

The legacy OS was designed back in the late 80's when there was a memory allocation "barrier" at 640KB, with the remaining 360KB allocated to system ROM, add-on cards local memory, etc. The legacy OS actually probed the add-on card region for useable RAM, and will actually load itself into the newfound region in segments of 16KB. This worked quite well when the legacy OS was writing all the device drivers, but new mainboards come with late-model VGA and NIC chips built-in, which tend to have much more add-on RAM used up in the 360KB IO region. Since the BIOS settings in these devices are rarely set to minimize memory use, new mainboards tend to make the legacy OS unstable, especially when DMA is used. The workarounds involve moving memory buffers and controlling memory usage and is very configuration-specific: for every setup we use, the workarounds have to be manually tweaked.

XIII. Emulation is the Way Forward

Further work is in progress: Qemu/KVM is being closely monitored.

There is also an in-house effort to implement the 16-bit virtualization that is missing from KVM. We believe that at virtualization speed of 80% the balance will tip in favor of virtualization.

References

[1] Gerald J. Popek and Robert P. Goldberg, "Formal Requirements for Virtualizable Third Generation Architectures", Communications of the ACM 17

[2] Kevin Lawton, "Running multiple operating systems concurrently on an IA32 PC using virtualization techniques"
http://www.floobydust.com/virtualization/lawton_1999.txt

[3] Avi Kivity, "KVM, One Year On by Avi Kivity" KVM Forum 2007 http://www.linux-kvm.org/wiki/images/6/61/KvmForum2007%24kf2007-keynote.pdf

[4] Roy Longbottom. PC CPU Performance Comparisons Benchmark Results http://freespace.virgin.net/roy.longbottom/cpuspeed.htm#anchorPara04

[5] M. Tim Jones. "Discover the Linux Kernel Virtual Machine"
http://www.ibm.com/developerworks/linux/library/l-linux-

kvm/

[6] Fabrice Bellard. “QEMU, a Fast and Portable Dynamic Translator”. Proceedings of the annual conference on USENIX Annual Technical Conference

[7] Ryan Harper and Karl Rister

“KVM Limits Arbitrary or Architectural?” KVM Forum 2008 http://www.linux-kvm.org/wiki/images/b/be/KvmForum2008$kdf2008_6.pdf

[8] Avi Kivity, Yaniv Kamay, Dor Laor, Uri Lublin, Anthony Liguori. “kvm: the Linux Virtual Machine Monitor” Proceedings of the Linux Symposium 2007 http://ols.108.redhat.com/2007/Reprints/kivity-Reprint.pdf

[9] Anthony Liguori. “Building a Better Userspace” KVM Forum 2008 http://www.linux-kvm.org/wiki/images/6/65/KvmForum2008$kdf2008_4.pdf

Heong Chee Meng (M'95-SM'04) is a Senior Staff Engineer in ON Semiconductor Inc, Seremban Malaysia. His research interests include embedded computing, Linux and electronics. He has a degree in engineering majoring in electronics from the University of Malaya, Kuala Lumpur

Emelia Khamis (M'08) was born in Muar, Johor in Malaysia. She works in ON Semiconductor Inc, Seremban Malaysia as an Electronics Engineer. Her research interests include Electronics and Linux. She has a B.Eng from the Univeristy of Malaya, Kuala Lumpur.