OpenVMS Technical Journal V14

Virtual Alpha Systems: Quality Control & Testing

Dr. Robert Boers, CEO & Founder Stromasys SA.
(For bio see http://www.stromasys.ch/about-us/board-of-directors)

Virtual Alpha Systems: Quality Control & Testing 1
Introduction 2
Designing and Testing Virtual Systems 2
Introduction of Formal Tools 3
The Current Virtual Alpha System Test Process 4
Organizational Impact 4
Conclusion 5
For More Information 5

Introduction

Since 1994, Stromasys has been developing system emulators, starting with PDP-11 emulation. Now known as "cross-platform virtualization," these system emulators provide a complete software model of a legacy hardware system, running on a modern server. This technology allows the users to keep their original operating system and applications running in an unmodified form, while gaining performance improvements, increased storage capacity, and increased reliability.

When we built the first system emulators, we had no idea that the market would demand for these solutions. Nor, did we realize the efforts required to develop commercial quality emulator products, particularly the importance of a comprehensive product testing strategy. This article gives an insight to the development of testing technologies and procedures that we created to ensure reliable replacement products with superior performance.

Designing and Testing Virtual Systems

In principle, creating a software abstraction layer of a hardware computer component is straightforward. A subsystem, for instance a CPU, is an implementation of sequential logic. The functionality is well defined and can be described in a high-level programming language. In modern CPU design, creating a software model before creating the desired hardware is a common practice, and we repeat this process for all the relevant subsystems.

If the emulator architecture follows the hardware architecture closely, the result is a library of emulated subsystem components, each representing a hardware subsystem in the form of a code module. In the case of the first generation PDP-11 emulator, we designed approximately 130 modules written in C++, representing the various CPU modes, memory management, disk and tape controllers, various serial and parallel I/O options, Ethernet controllers, and so on.

Given the resemblance between the original hardware and the emulated components, the obvious choice for testing the emulator was using the original hardware diagnostics. Having few other tools at our disposal at that time, we pushed the component emulation to a level where the original hardware diagnostics would even show the revision numbers of the hardware components that were emulated. This unnecessary complexity resulted in a slow implementation compounded by running with the virtual system modules in synchronous mode to avoid timing problems. However, in dealing with PDP-11 and slow VAX models, this got us a correctly working emulator with the existing diagnostics.

However, we cannot always use the existing hardware diagnostics. An emulated tape drive, for instance, has a tape positioning time that is close to zero. The hardware diagnostics find that hard to accept, and even on a correct implementation, software might malfunction. Putting NOOP's in drivers to make them work correctly will create havoc in a faster virtual interface, as it would with faster hardware.

In the development cycle of a virtual system, booting an operating system on it is a milestone that indicates that a minimum of virtual hardware is working correctly. Interestingly, most operating systems are not very useful by themselves to debug a virtual system. When we could boot VMS on our MicroVAX emulator, later tests revealed that about 85% of the emulated VAX instructions had errors in some modes. A surprise was that without floating point instructions VMS would not boot. Unfortunately, floating point instructions for VAX are complex to implement correctly on an x86 architecture. Nevertheless, the ability to run the original OS is an important step, since it greatly facilitates the development and execution of custom diagnostic tools.

Before the complexity of our virtual system products drove us to the development of custom diagnostic tools, and from there to a formal test process. We found that implementation of an emulator on more than one host platform helps error discovery. From the same source code, errors typically appear in different places, or only in one implementation, and are easier to track down. For example, in an industrial application on a virtual PDP-11 hosted on Windows NT, using Alpha hardware the menu items were correctly displayed in white text. On an x86 platform, some menu items appeared in red, because the x 86 C++ compilers generated an error that propagated in a rarely used FORTRAN call.

Introduction of Formal Tools

In the development of our CHARON-VAX emulators, we used a tool called AXE, which we obtained from Compaq. AXE was the verification tool for the VAX hardware development groups in Digital. AXE had two purposes at Digital:

- Locate bugs in the hardware during development
- Verify that we meet the requirements of DEC STD 032, the functional requirements for a VAX hardware system.

AXE creates test cases composed of a VAX CPU instruction with operand specifiers appropriate for that instruction. After execution, AXE compares the relevant state with a "known good" result and reports the differences to the user. The individual test cases are generated on the fly, creating billions of unique sequences. These variations cover essentially all aspects of a VAX CPU and memory management unit: opcode, opcode or operand placement with respect to page boundaries, (stack) page protection, translation buffer validity, overlap of string operands, page faulting, etc. AXE also covers testing of exception handling like reserved opcodes, arithmetic traps addressing modes, and invalid translations.

The final AXE test before Digital released VAX hardware was a comparison against a reference database. The green light was given if a test of ten million cases of each user mode instruction (in all access modes), and 5000 cases of each privileged instruction mode were passed without a single error. AXE has been valuable in the development of our virtual VAX systems, and we base the first phase of our virtual Alpha product readiness tests on a similar concept, using proprietary tools. Such testing is very time-consuming. We passed our virtual VAX 3100 AXE readiness test in the year 2002 in a certification sequence that lasted more than three weeks on a 500 MHz Celeron processor.

Both for "hardware" and "virtual hardware" developers, a good working CPU is only one of the many components to integrate and test. An advantage is that many hardware components have standard functions such as SCSI controllers, UARTS, disk drives, Ethernet controllers, and so on. These components are either fully tested by the supplier or can be tested individually while connected to a good system. Virtual system developers do not have that luxury. To create a complete hardware abstraction layer of a new system, we must emulate such components as well, in order to create a portable product. As we learned over the years, this makes the reuse of known good virtual components extremely important. If we make the elements of a virtual component library usable on different virtual systems, saving development time is far from the most important benefit. The repeated integration testing of such components in multiple designs leads to a significant increase in reliability. Provided, that strict code management, re-utilization, and maintenance is enforced. After 10 years, and more than 40 PDP-11, VAX, and Alpha models, the investment pays off in the form of a rich set of reliable virtual components. As an example, a virtual Sparc Workstation prototype reuses 70% of the existing virtual VAX components, allowing fast development and less testing effort.

The Current Virtual Alpha System Test Process

As we pushed the technology further, in particular to develop fast, virtual SMP systems (for instance a virtual 16-CPU CHARON-AXP system), the processes we used to ensure the reliability of a single CPU virtual VAX were insufficient. We had to change the virtualization technology in several fundamental ways:

- High speed virtual CPUs require complex virtualization algorithms to obtain the desired performance, typically involving more than one host CPU per virtual CPU. Just the drive for performance required years of research, as we wanted to reach two goals at the same time: achieve portability of the acceleration code (for reuse with other architectures) and match the clock rate with more modern CPUs.
- It is no longer feasible to run the components of a virtual system in synchronous mode when virtual systems get more complex. The interrupt latency suffers badly and interferes with proper operation. It also prevents effective use of multi-core host systems.
- An important constraint for virtual SMP systems is that their CPUs must clock at exactly the same frequency. This is easy to implement in a hardware system, but requires a very sophisticated synchronization mechanism in the virtual environment. Testing this is implicit, as an operating system like Tru64 shuts down any CPU that has a perceived clock rate that is even slightly different from the primary one.

An advanced virtual SMP system design requires three kinds of tests:

- Functionality tests to ensure that the binary compatibility with its hardware CPU equivalent is uncompromised. In particular, we must test mathematical operations extensively to assure correct overflow and underflow in floating point formats. For virtual Alpha systems, we developed a suite of proprietary tests (similar to AXE) that we calibrated on a hardware Alpha of the same generation as the virtual implementation. These tests include verification that self-modifying code works correctly (there is a myth that emulators would not be able to handle such code). We, typically, run such tests every night when we are working on code optimization of the virtual CPU implementation.
- Performance tests to ensure that the ongoing CPU optimizations have the desired effect. The optimization algorithms are dependent on binary code structures; an automated test procedure submits 100's of binary Alpha code segments representing many different types of user applications. A weighted average is calculated with a check that no type of use is significantly degraded. We aim at a small but overall improvement with every release.
- System integration and Quality control tests, which have become the predominant tests in terms of calendar time before every product release. As we extend our virtual products with new components, for instance, FDDI or Fibre channel support, we must scrutinize the impact on the rest of the system in even more depth than for the original hardware, where the influence between the individual components was limited. Furthermore, our products can run on many different host systems. Proper operation on a representative set of host systems is an essential factor in quality verification.

Organizational Impact

During the development of larger VAX systems in 2004-2005, the time needed to test our virtual systems started to limit our product release throughput, in spite of having a dedicated full-time test engineer in addition to the usual testing process in engineering. Splitting the testing formally in subsets (used during development) and creating a formal "product readiness test" (PRT) improved the situation somewhat. Products that pass a PRT are initially coded in their license as Field test capable. The field test program aims at collecting 20 user months of field test experience, for instance, five users during 4 months, but commercial pressure can reduce this. Field tests are a great way to verify the quality of the documentation. During the field test period, minor product problems are fixed and documentation

is updated. This process is iterative, and a typical product release goes through a PRT three times before its official release as a product. The PRT for the Alpha platform contains more than 200 individual tests, and takes about 2 to 3 weeks. The number of tests is rising as we add functionality to our products.

Engineering groups tend not to be the best judges of the readiness of their products. In light of this, the solution was to create a separate Quality Control group in our organization, also responsible for measuring and publishing formal performance numbers and reviewing product documentation.

Conclusion

We aimed at giving some insight into the process of creating and testing a reliable virtual computer system. Successful hardware emulation depends not only on good design, but also on rigorous testing and quality control procedures. The technologies we have developed and the understanding we have built of the testing processes to create a reliable virtual system are universal, making development of other virtual legacy systems easier. Our newest development center in Shenzhen, China, is now expanding to take these testing technologies outside of our historic focus on DECs PDP-11, VAX, and Alpha systems.

For More Information

To find out more about Stromasys and our CHARON-VAX and CHARON-AXP virtualization software products, please visit www.stromasys.com.