OpenVMS Technical Journal V11

A Starlet[1] is Born: New Options for VAX and Alpha Hardware Replacement

Camiel Vanderhoeven, Hardware Illusionist

[1] Starlet was the code name for the program that developed the VMS operating system. See the OpenVMS Wikipedia entry http://en.wikipedia.org/wiki/OpenVMS) for additional information.

Introduction

Those who are looking for options to replace aging VAX and Alpha hardware should be aware of the arrival of a new player in the field. Migration Specialties International, a respected OpenVMS consulting firm that is well known for its legacy hardware replacement options, RPG compiler and other migration aids, has teamed up with a number of partners to deliver its own suite of software-based VAX and Alpha hardware emulators.

For this suite of emulators, we've defined an underlying architecture that will allow us to add different emulated systems and options to the suite with an unprecedented degree of flexibility.

This article, written by the lead architect, will focus on the internal architecture designed to support these new emulators.

Goals

We will create a software platform that can virtualize a variety of Alpha and VAX hardware. We want to be able to emulate enough different systems to provide viable alternatives to any existing hardware configuration, including multi-CPU systems.

Our emulators will support OpenVMS (VAX and Alpha emulators) and Digital UNIX/Tru64 UNIX (Alpha emulators only) as operating systems running on top of the virtual hardware.

Our emulators will be hosted on Integrity servers running OpenVMS and Proliant servers running Windows. We will consider supporting a Linux version of the product at a later stage.

When run on OpenVMS/Integrity as the host platform, our VAX and Alpha emulators running OpenVMS will offer the same high-availability features that real VAX and Alpha systems running OpenVMS have to offer.

In the future, we will explore the possibility of coupling our emulators with hardware bus support to enable the use of the emulator with custom hardware interfaces. It would be conceivable to see one of our VAX emulators with an attached Q-Bus or XMI card cage used for replacement of factory automation systems.

We are targeting a production release for a first Alpha emulator in early 2010.

Emulator Design

This section provides a high-level overview of the emulator and shows how the various bits and pieces fit together.

Component hierarchy

The easiest way to think of an emulator is to think of it as a piece of hardware because that is what it acts like to the operating system and other software running on top of it.
Like the real hardware, the emulator consists of modules (components) that interact with each other. Most of the components correspond directly to physical hardware components.
Components have a parent-child relationship to each other. Child components are usually connected to their parent through a virtual bus. For example, disk components are children of a disk controller component, and PCI device components are children of a PCI controller.

Emulator component

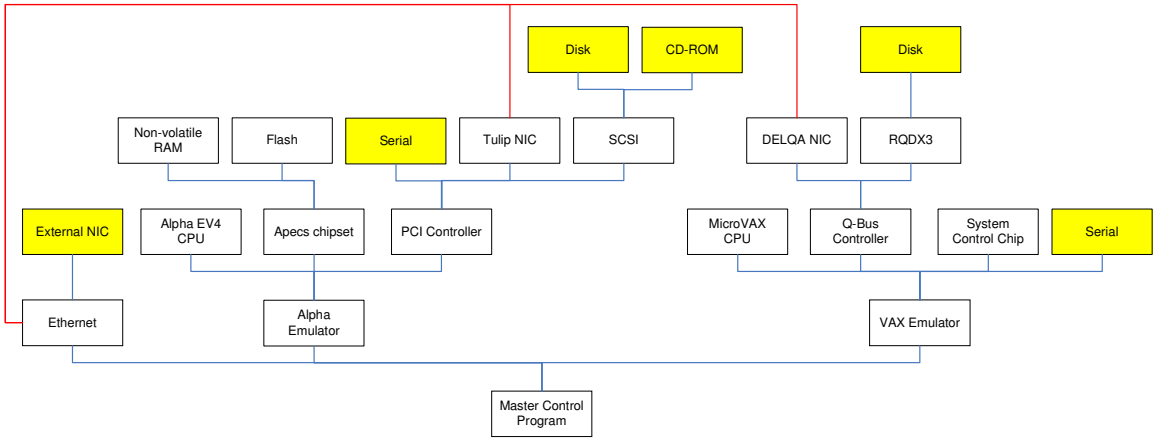
This abstraction poses a problem at the top-level of the emulated system. Most systems have a top-level bus that has no real controller to act as its parent. For example, in the AlphaServer ES40, the top-level bus consists of the D-chips that connect the CPU's to the C-chips (system chipset), the P-Chips (PCI controllers), and main memory. Therefore, the decision was made to create both a VAX emulator component and an Alpha emulator component. These emulator components act as the controller for the top-level bus. For ease of implementation, these components also include main memory.

The easiest way to think of an emulator is to think of it as a piece of hardware

Master Control Program (MCP)

Finally, different emulators and components like networks need to be tied together. This is accomplished through the master control program component. (Are there any fans of either Burroughs B5000 mainframes or the movie *Tron* out there?)

The following (simplified) image shows the components used to emulate an AlphaServer 400 and a MicroVAX, interconnected through an Ethernet network that is also connected to the outside world through one of the host system's network interfaces. The yellow components are those that interact with the outside world. The red line indicates the "extra" parent-child relationship between the network interface cards and the network top-level component.



Virtualization layer

From the beginning, the emulator was written to be very flexible. We first created a framework to be used for writing different emulators. All forthcoming VAX and Alpha emulators will use this common framework. Into this framework, we incorporated all of the functions that all or most emulators will be likely to need, such as:

- Functions for configuring the emulator: instantiation, configuration, and connecting together of all emulator components;
- Functions for controlling the emulator: structured, sequenced discovery, initialization, starting and stopping of all emulator components;
- Emulated Ethernet connectivity between emulators;
- Common interfaces to the outside world for networking, hard-disk emulation and I/O components: for example, communications ports provide the ability to communicate through a telnet session or a physical serial port. This way, this functionality can be shared by any emulated communications port without requiring additional effort, simplifying the emulation environment and providing a more consistent user experience;
- Hiding differences between different host systems from the emulator components, so the same emulator will run on both OpenVMS and Windows;
- Support for making use of multi-CPU or multi-core host systems by threading;
- Emulator licensing and protection.

In short, these are all the emulator functions that are not directly related to the bits, bytes and registers of the emulated hardware. We've named this framework the "Virtualization Layer" because it creates a complete virtual environment for the individual emulators.

Hardware Platform Abstraction Layer

As we want our emulators to run on both Windows on Proliant servers and OpenVMS on Integrity servers, we were confronted with the fact that Windows and VMS behave differently. To avoid having to write platform-specific code for each emulator component, we implemented an abstraction layer as part of the virtualization layer that hides these differences from the rest of our code. These differences are mainly in the following areas:

- Threading and locking. On VMS, we use the Pthreads library; on Windows, we use the Windows API.
- Physical device access. On VMS, we use QIO's; on Windows, we use various API's.
- Timekeeping.

Putting all platform-dependent code in one place helps us to keep our code base clean.

Use of Object-Oriented Programming (OOP)

The emulator makes extensive use of OOP, particularly of the features offered by the C++ language. While C and C++ are reviled by some for their perceived cryptic nature (although there is no rule that says C or C++ code has to be cryptic), they are commonly considered to be the languages of choice for low-level, portable programming found in operating systems, device drivers, and emulators. C and C++ give programmers a level of control over the bits and bytes of their code few other high-level languages offer, and C and C++ compilers that produce blazingly fast code are available for virtually any platform.

Classes

All components are implemented as classes. That means that a class has been designed for each different kind of emulated component. For instance, if a RQDX3 controller needs to be emulated, a RQDX3 class will be written. Once the class exists, the emulator can create as many instances of that class as required. For example, to emulate a VAX with three RQDX3 controllers, three instances of the RQDX3 class would be generated.

Inheritance

The RQDX3 controller needs to be able to interface with the Q-bus controller and vice versa. The mechanisms involved are the same for all Q-bus devices; the way the RQDX3 communicates with the Q-bus controller is no different than the way a DELQA network interface communicates with it. Because of this shared behaviour, all Q-Bus components share a common base class, the Q-Bus Device base class. This way, the Q-Bus controller can address each of its child components as Q-Bus Devices, rather than as individual types of interface. This takes full advantage of the power of inheritance, a defining feature of OOP.

C and C++ give programmers a level of control over the bits and bytes of their code few other high-level languages have to offer.

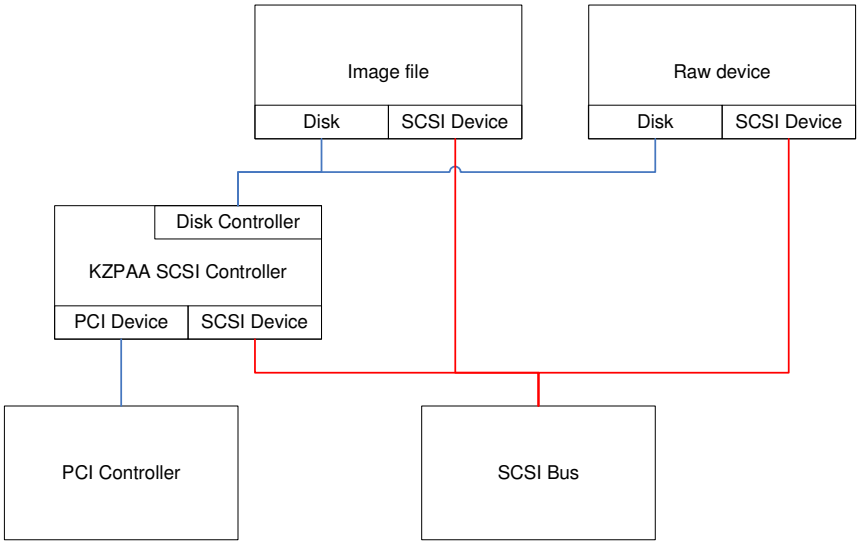
Component Base Class

The Q-Bus Device class in turn has the Component class as its base class. The Component class is part of the framework, and provides for such basic emulator-wide functions as naming components, creating parent-child relationships, initializing, stopping and starting the emulator.

Multiple Inheritance

It gets trickier though. Besides being a Q-Bus device, the RQDX3 is also a disk controller. As not all disk controllers are Q-Bus devices (for example, the KZPAA SCSI disk controller is a PCI device), the disk controller base class can't have the Q-Bus device class as its parent. So, the RQDX3 needs to have both the Q-Bus device class and the disk controller class as its base classes. This is called multiple-inheritance.

In the case of the KZPAA SCSI controller, it is even more complicated; it inherits from PCI device, Disk controller, and SCSI device classes. The following diagram illustrates this:



Conclusion

Like computer hardware or operating systems, successful, scalable and adaptable hardware emulation requires an underlying, well-defined architecture. This architecture is the foundation for the entire product. We have spent considerable effort to define a flexible architecture for our emulators and, hopefully, we've shown you some of its interesting properties in this article.

About the author

Based in the Netherlands, Camiel Vanderhoeven is a relative newcomer to OpenVMS. Introduced to the operating system as old as he is in 2003, it has rapidly become the favorite on his list. So much so, that he set out to write the only open-source AlphaServer emulator capable of running OpenVMS on a laptop, the ES40 Emulator. Camiel delivered presentations about the ES40 Emulator at the 2008 VMS bootcamp, and at Community Connect Europe. In January 2009, Camiel started his own consulting business around VMS, Camicom SSC, in which capacity he is now developing a series of new VAX and AlphaServer emulators for Migration Specialties (www.MigrationSpecialties.com).

Camiel is married to Martha, and they are expecting their first child. Rumors that they are planning to name their child Victoria Madeleine Samantha or Victor Mark Samuel are neither denied nor confirmed.

For more information

For more information and updates about the upcoming multi-platform VAX and Alpha emulator discussed in this article, visit http://www.migrationspecialties.com/VAXAlphaEmulator.html. For more information about the author, visit http://www.camicom.com. For more information about the open-source ES40 emulator or to download its source code, visit http://www.es40.org.

