Alpha Runs x86 Code with FX!32

Digital's Emulation Strategy Could Help Boost Alpha/NT System Sales

by Jim Turley

Digital is giving its Alpha NT systems the ability to execute standard Win32 application programs written for x86 systems. The new technology, dubbed Digital FX!32, is still in development but is expected to begin shipping with all Alpha Windows NT systems in 2Q96, and it will be available free of charge to all existing Alpha customers.

FX!32 uses a combination of run-time emulation and background binary translation to decipher x86 binaries and convert them to native Alpha code. Digital claims translated applications can run as fast on a midrange Alpha chip as they would on a high-end Pentium system. The company has not yet achieved its performance targets, however, nor has it demonstrated that FX!32 delivers software compatibility.

The strategy can be viewed either as proof of Alpha's performance superiority over x86 processors or as proof of the hopelessness of bucking the x86 software tidal wave. Either way, FX!32 is an interesting technical feat and a bold strategic move on Digital's part. The ability to run "shrink-wrapped" software should help boost Digital's NT system sales by allowing its customers to run common PC productivity applications. With acceptable x86 performance, the company could make its Alpha/NT platform more attractive than alternative RISC-based NT systems to buyers concerned about the availability of commercial software.

Emulation, Translation Intertwined

Digital discovered early on that it was impossible to perform a straightforward single-pass translation of x86 object code. Offering translated binaries for popular applications would have been the easiest solution for its customers, but this technique presented serious legal and technical hurdles.

For example, it is impossible to examine an x86 application and locate all the branch targets or even determine which portions are code and which are data. The problem lies with the x86's variable-length instruction encoding and its irregular use of extension words, prefixes, suffixes, and segment-override bytes. The tendency of compilers to intersperse static data among code strings also complicates attempts at automated disassembly.

Without actually executing the program, automated methods of extracting the code stream are prone to error. Products such as FlashPort from Echo Logic or Hunter Systems' Xdos sidestep this problem by relying on hand-tuned translator "hint" files developed for individual applications.

Digital hoped to avoid any kind of user intervention or manual tweaking and make FX!32 totally automatic. In the end, the company relies on the applications themselves to assist FX!32 in separating code from data.

Applications Get Off to Slow Start

The FX!32 translator requires at least two passes through each application to complete a partial translation. Rather than try to pick out the executable portions of a program, FX!32 begins by emulating x86 instructions at run time.

As Figure 1 shows, the FX!32 software consists of three component parts: the runtime emulator, the background translator, and the transparency server. The first time an x86 application is executed, the transparency server determines that the file is an untranslated x86 program and hands it off to the emulator.

The program runs the first time in emulation. No permanent x86-to-Alpha translation is performed during this phase. Instead, x86 instructions are emulated one by one. All operating system calls are handled natively. Because FX!32 works only with x86 applications written for the 32-bit Win32 API, trapping NT system calls is a simple matter. This approach is somewhat different from the method used by Insignia Solutions' SoftWindows or Apple's 68K-to-PowerPC emulator *(see* **080704.PDF***)*, which look at certain sequences of instructions.

The FX!32 emulator keeps track of which portions of the application it has emulated. The results are kept in a log file, which Digital calls the execution profile. This profile enables the later translation stage.

The first time an x86 application is run on an Alpha system, its performance will likely be disappointing because it is completely emulated; no optimization is performed and

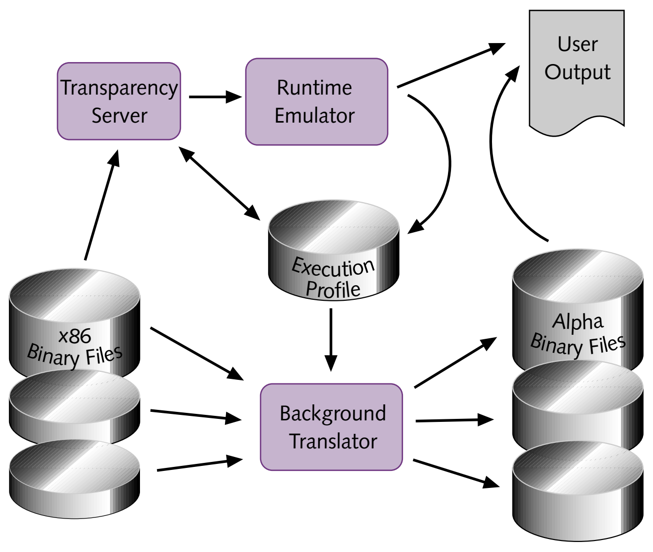

Figure 1. Digital's FX!32 emulation technology consists of a runtime-emulation stage and a later background-translation stage. The transparency server manages the executable images.

Price and Availability

The Digital FX!32 emulation/translation software will be available free of charge when it is released around mid-1996. Distribution media have not been determined. For more information, contact Digital Semiconductor (Hudson, Mass.) at 800.322.2712 x32 or 508.628.4750 x32; fax 508.626.0547.

performance may be only a small fraction of full native speed. Compute-intensive applications are particularly vulnerable; conversely, software that relies heavily on NT system calls may run reasonably well. Unfortunately, Digital is not prepared to release any performance indicators until FX!32 testing is further along.

In effect, the first pass through an application is merely a training run, intended only to help FX!32 determine which portions of the application are executable and where the branch targets are. It is not until the user exits the application for the first time that the real translation begins.

Later Passes Speed Translated Performance

Using the execution profile generated during the first run, FX!32 begins translating the application in earnest. This translation is always performed as a background process after the application has quit. Commercial x86 applications frequently include several executable (EXE) and library (DLL or LIB) files; each of these files is translated into a corresponding Alpha object file. From the point of view of the operating system, each translated file is stored as a dynamic link library (DLL) file on disk.

Only code that was actually executed during the first run will be translated. The exception to this rule is simple branches. In its current version, when the background translator encounters a two-way branch, it translates both forks of the branch even if one fork was never explored in the emulation run. However, multiway switch constructs often remain partially untranslated. Digital expects this artifact to be eliminated in the final release of FX!32.

After all previously executed portions of the software are translated and stored as DLLs, the application is ready to run again. This time, FX!32 examines the application's execution log and, seeing that certain files that make up the application have already been translated, it loads and runs the translated DLLs in place of the original x86 object files whenever possible. Assuming the second run of the application exercises exactly the same features as the first run, the application is now sufficiently translated into Alpha code to run noticeably faster. In this instance, no run-time emulation would be required; the application would run entirely from translated Alpha DLLs.

If the application veers into previously unexplored sections of code, the FX!32 server will again invoke the emulator to work through untranslated portions at run time while updating the application's execution profile. After the user exits the application, the background translation process starts afresh.

Portions of the application that are never exercised—such as little-used features the user never exploits—may never be translated. Different users might generate different binary translations of the same application because of their different usage habits.

Generally, once an x86 file has been translated the first time, it is never retranslated or updated. However, if the application's usage changes substantially—such as when a new program feature is exercised for the first time—the FX!32 server may determine that the application needs to be optimized and retranslated. A frequent consequence of retranslation is that multiway switch constructs are optimized and translated more completely.

The decision to retranslate is based on the growth of the application's execution profile. If the profile grows beyond a certain threshold after the file's initial translation, the server initiates a retranslation. Although this threshold is set to a nominal value when FX!32 is delivered, users can adjust the threshold to suit their tastes. Adjusting the threshold up or down will decrease or increase, respectively, the amount of optimization and retranslation FX!32 performs.

Only 32-Bit Windows Applications Work

All calls to the operating system are trapped and executed natively, which provides the best possible performance. Like emulated Macintosh code, the more time the application spends in system calls, the faster it runs.

FX!32 works only with Win32 applications, that is, with code written for the 32-bit programming model used by Windows NT and Windows 95. FX!32 does not translate legacy 16-bit programs written for Windows 3.x or DOS.

However, Windows NT itself includes emulation capability for many 16-bit applications. The current release of NT 3.51 for RISC processors ships with an equivalent to SoftWindows, allowing those systems to run 16-bit, standard-mode x86 applications through emulation. Future releases of NT will add support for enhanced-mode 16-bit code as well.

Between NT and FX!32, Alpha users will have access—one way or another—to most Windows applications. Because SoftWindows is an emulator, however, the performance of 16-bit software will not be up to Digital's hopes for FX!32's native translation of 32-bit applications.

Digital does not consider this to be a major drawback, and will continue to rely on Microsoft and Insignia to provide compatibility with 16-bit code. This is probably a sound strategy for Digital's market; we believe a number of factors are pushing software developers to adopt the Win32 API, steadily reducing the need to support 16-bit applications as time goes on. Intel's Pentium Pro, for instance, delivers notoriously poor performance on 16-bit code, and Microsoft itself no longer releases Win16 applications.

Modest Memory, Disk Overhead

Because translation is carried out on a file-by-file basis, FX!32 is able to take advantage of shared code libraries. That is, when multiple applications share a single library file, as is often the case with Microsoft products, FX!32 will translate the shared library once, and the translated version will then be shared among all the appropriate applications.

Obviously, the translated DLLs require additional disk space. The x86 instruction set yields very good code density—for all the same reasons that make it difficult to disassemble—while Alpha's fixed-length 32-bit instruction words result in fairly bulky binaries. Digital estimates the translated binaries may be about twice the size of their x86 originals. The exact ratio depends on a number of factors, including the ratio of code to data (data does not get translated) and the percentage of code actually exercised and converted. Note that this is in addition to the original x86 files, which are not deleted.

The memory requirements of FX!32 are relatively modest. The transparency server, which determines which applications have been translated and which haven't, uses only 140K of RAM. The run-time emulator requires about 2M of RAM, which is fairly small, considering the task at hand. SoftWindows running on a Power Macintosh, for example, requires several megabytes of RAM, although it must emulate the entire PC/Windows environment as well. When the background translator is running, it requires approximately 1.5M of memory plus enough for both the original and translated versions of the file being translated. Considering the amount of RAM Digital ships with its Alpha NT systems, FX!32 makes modest demands on the system.

Performance Still a Big Question Mark

Unfortunately, Digital has not released performance numbers for any translated applications. The company hired National Software Testing Labs (NSTL) to conduct application benchmark testing, but results are not available, casting some suspicion on the current condition of the product.

Digital believes FX!32 might someday achieve 70% of native Alpha performance on translated applications—a laudable goal, but a difficult one to gauge, since native Alpha versions do not exist for virtually any Windows applications (the whole reason for FX!32's existence). Digital cautions the 70% goal has not been achieved in the current implementation of FX!32 and offers no timeline for reaching that particular milestone.

Correlating Alpha performance to Pentium performance involves a number of variables, but a specific example can be helpful. Judging from SPEC95 results, a 21164-300 delivers 7.75 SPECint95 (base) versus 4.76 for a Pentium-166, or 63% of the 21164's performance. Thus, 70% of native Alpha performance would be slightly faster than native Pentium-166 performance. If FX!32 were to meet its performance goal, then in this instance Alpha would execute Pentium applications faster than today's fastest Pentium.

Granted, the Alpha chip is running at nearly double the clock rate of the Pentium and has a huge L2 cache. And these numbers ignore the fact that the Alpha processor, much less the entire system, costs twice as much as the Pentium. The role of FX!32 is not to displace PCs but to make Digital's Alpha machines more attractive than competing RISC-based NT systems.

It's clear that Digital won't lure away customers who are interested primarily in PC applications—that is not the company's intent. Instead, FX!32 offers a booster to the company's NT line by removing the emotional obstacle of PC compatibility. It may now be feasible to satisfy, for example, the engineer who requires access to native Alpha applications as well as to a few productivity applications that are available only for x86 processors.

So far, only Apple has pulled off a transparent binary conversion from one architecture to another. In Apple's case, the wide disparity in performance between PowerPC and the 68K made it easier to mask the inefficiencies of a run-time emulator. Pentium is a closer match for Alpha, at least in integer performance, making it tougher to run in emulation mode and still get acceptable speed. By allowing FX!32 to take its time and work its binary translation in the background, Digital has given its product a potential performance advantage over the Apple or Insignia approaches. The cost is more disk space and a certain amount of delayed gratification when running an application for the first time.

Adding FX!32 to its NT systems will not help Digital penetrate the mainstream PC market, nor will it boost the company's Unix system sales. FX!32 will appeal only to the portion of the market that is considering Windows NT. In that segment, Digital competes with systems based on PowerPC, Pentium, and Pentium Pro processors, with MIPS support fading fast. By offering better floating-point performance than the x86 systems and better x86 compatibility than the other RISC systems, Digital noses ahead on the features checklist. FX!32 won't make buyers choose NT over Unix, but it may help them choose Alpha over PowerPC.

Digital's biggest competition is coming from Pentium Pro systems. Their integer performance is as good as Alpha's, the selection of vendors is greater, prices are much lower, and x86 compatibility is a given. Finally, Pentium Pro's much-maligned 16-bit performance is not an issue here because FX!32 doesn't translate 16-bit applications.

FX!32 is an interesting technical exercise, but so far, a completely unquantified one. In its current state, FX!32 does not meet its goal of running at 70% of native Alpha speed. For some applications, such as e-mail and word processing, reaching top speed may not be a big issue. It is important that Digital demonstrate that FX!32 can work transparently with any Win32 application without side effects, bugs, or special hand waving. If the company can clear that hurdle, FX!32 will have reached its most important goal: allowing users to simply install and run shrink-wrapped PC applications on their Alpha systems. M