Wie erfüllt man die Leistungsanforderungen von Embedded-Anwendungen?

Ian Ferguson **Designer müssen bei den meisten Embedded-Anwendungen Systeme mit immer höherem Leistungsniveau in immer kürzeren Designzyklen entwickeln. Die Internet-Explosion in der Kommunikationstechnik zwingt Systemlieferanten, die schnell wachsende Gruppen von Anwendern zu unterstützen, die hohe Datenströme über das Netz übertragen wollen.**

Die Benutzerschnittstellen von Konsumelektronikgeräten werden komplexer, weil die Systeme heute Video-, Audio- und Grafikdaten verarbeiten und die Möglichkeit bieten, spezielle Informationen vom Host über das Netz anzufordern.

Auf den Prozessor bezogen, kann man die Effekte dieser kontinuierlichen Leistungssteigerung aus unterschiedlichen Perspektiven betrachten:

Zunächst einmal sollte man die Leistung in ausgewogener Weise steigern. Erhöht man einfach die Pipeline-Taktrate unabhängig vom restlichen System, dann wird die Hochgeschwindigkeits-CPU einen Großteil ihrer Zeit mit dem Warten auf Daten vergeuden. Soll die CPU so nahe wie möglich an ihrer theoretischen Leistungslimit arbeitet, so muss man sich eingehend mit der Speicherhierarchie des Systems befassen.

Die Effizienz der Codeverarbeitung pro Taktzyklus lässt sich steigern. RISC-Technologien wie die MIPS-Befehlssatz-Architektur werden laufend verbessert, um einen Beitrag zur Lösung spezieller Systemprobleme zu liefern.

Darüber hinaus sind viele neue Funktionsbereiche des Systems zu beachten. Für CPU-Wahl und architektonische Entscheidungen sind neben den Leistungs-Benchmarks wie *SpecINT* und *Dhrystone* auch andere Kriterien entscheidend. Aspekte wie die Effizienz des Exception-Handling-Modells einer CPU oder die Unterstützung komplexer Multitasking-Umgebungen durch die CPU gewinnen zunehmend an Bedeutung. Mit welchen Techniken kann der CPU-Entwickler diese Aspekte verbessern? Zwei Themengebiete sind zu berücksichtigen, wenn man die CPU-Leistung in der Nähe des theoretischen Maximalwerts halten will: die Verbesserung der Effizienz der Pipeline selbst und Maßnahmen zur kontinuierlichen Versorgung der Pipeline mit Daten. Viele Hochleistungs-CPU-Cores sind superskalar aufgebaut – d. h. ihr Kern kann pro Taktzyklus mehr als eine Instruktion ausgeben. So enthält z.B. die vor kurzem von IDT vorgestellte RISCore64600 Architektur mehrere Funktionseinheiten; mit jedem Taktzyklus kann sie mehrere Instruktionen für diese Einheiten ausgeben. Insbesondere kann das System diese Instruktionen in einer anderen als der vom Programm definierten Reihenfolge ausführen: Es kann Instruktionen überspringen, die darauf warten, dass eine Funktionseinheit verfügbar wird, oder dass eine bestimmte Berechnung fertiggestellt wird, deren Ergebnis für diese bestimmte Instruktion benötigt wird (diese Situation nennt man "Data Dependency"). Darüber hinaus enthalten die Funktionseinheiten umfangreichen Look-ahead Buffer, so dass sich Instruktionen aus der Programm-Queue abarbeiten lassen, damit nachfolgende Instruktionen ausgegeben werden können. Zu jedem Zeitpunkt können bis zu 32 Instruktionen „unterwegs" sein, d.h. sie können sich irgendwo in den Phasen der Decodierung, Pufferung, Berechnung oder des Zurückschreibens in der Pipeline befinden.

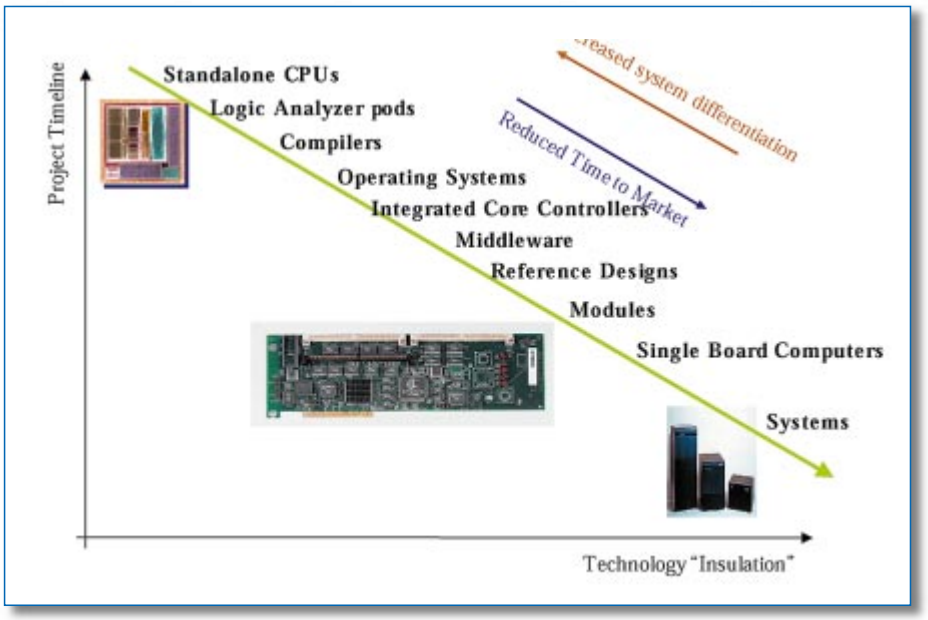

Bild 1: Entscheidungsfindung: Zukaufen oder selbst entwickeln?

Der Cache machts aus

Damit eine Hochleistungs-CPU ohne Unterbrechung Daten erhält, sind hoch-

effiziente Speicherhierarchien erforderlich. Größere Primär-Caches mit einer breiteren Set-Assoziativität, schnellere Level-2-Caches und der zunehmende Einsatz von 64-bit-Bussen sind da ziemlich nahe liegend. IDT hat kürzlich eine Reihe von Embedded-Systemen mit unterschiedlichen Kombinationen von Pipeline-Clocks, Primär- und Sekundär-Caches modelliert. In der Untersuchung wurde ein konsistenter Befehlssatz vorausgesetzt und die Frequenz des Systembusses in allen verschiedenen Szenarien bei 100 MHz gehalten. Als Referenz-Plattform diente eine Kombination aus einer 200-MHz-CPU und einem 100-MHz-Systembus. Dabei zeigte sich, dass eine Geschwindigkeitssteigerung des CPU-Pipeline-Clocks um den Faktor 5 weniger Leistungsgewinn brachte als eine bescheidene Steigerung der CPU-Pipeline-Frequenz um 50% in Kombination mit doppelt so großen integrierten Primär- und externen Sekundär-Caches.
Darüber hinaus kann man auch zusätzliche Instruktionen in die Prozessorarchitektur einbinden, damit der Programmierer die Bandbreite besser nutzen kann. Mit der„Prefetch"-Instruktion kann der Programmierer z.B. dem Core mitteilen, dass ein bestimmter Datenblock demnächst in der CPU zur Verarbeitung benötigt wird. So kann die CPU diese Daten schon laden, bevor sie benötigt werden, was die Hit-Rate des Primär-Caches verbessert. Ein Anwendungsbeispiel dafür wäre die Verarbeitung von ATM-Zellen. Der Programmierer kennt das Format des einlaufenden Datenstroms und kann daher herkömmliche I/O-Referenzen als cachiert anstatt nichtcachiert behandeln. Mit diesen Verfahren sollte sich die durchschnittliche Zahl von Instruktionen pro Clock-Zyklus (IPC) steigern lassen, die über einen bestimmten Anwendungscodebereich ausgegeben wird. Bestimmte Software-Algorithmen für Embedded-Systeme wie Modem-Code, Video/Audio-Kompression und Voice-over-IP kann man auch durch die Einbindung spezieller Instruktionen in den CPU-Kern optimieren. Die Einbindung von Multiply-Accumulate und Multiply-Subtract Instruktionen in zahlreichen Architekturen ist eine direkte Folge des immer häufigeren Wunsches nach einer Implementierung von DSP-Algorithmen in Universal-RISC-Prozessoren. Viele RISC-Architekturen wie PowerPC, MIPS und Hitachi's SH werben heute mit umfangreicheren "Media Processing" Erweiterungen. Ganz nebenbei - das "R" in RISC, das für einen reduzierten Befehlssatz steht, ist auch nicht mehr, was es einmal war: Viele RISC-Architekturen enthalten heute mehr Instruktionen als CISC-Befehlssätze.

Die Verbesserung von Aspekten wie Task Switching erfordert auch Weiterentwicklungen im Inneren des CPU-Cores. Dazu muss der CPU-Entwickler den Overhead bei der Abarbeitung externer Interrupts vor der Rückkehr zum Hauptprogramm verringern. Bei der Entwicklung des RISCore64600 untersuchte man die prinzipiellen Schritte bei der Verarbeitung einer Exception.
Sie umfassen:

- Prioritäten-Bildung für die aktiven Interrupts
- Adressberechnung für die Interrupt-Serviceroutine
- Abspeichern bestehender Registerstände zur Freigabe von Speicherplatz für die Serviceroutine
- Ausführen der Serviceroutine, möglicherweise Zurückschreiben von Ergebnissen in den Hauptspeicher
- Rückkehr von der Exception mit Wiederherstellung der ursprünglichen Registerstände

Fast Exception

Beim RISCore64600 kann der Programmierer einen Interrupt und einen Systemaufruf als „Fast Exception" Modell verarbeiten. Zur Vermeidung von Adressberechnungsschritten programmiert man diese Aufrufe mit speziellen Interrupt-Vektoren. Weil sie eine höhere Priorität als alle anderen Exception-Bedingungen besitzen, kann man so auch die Prioritätenbildung aller aktiven Interrupts umgehen. Außerdem muss man nun nicht mehr die Registerstände sichern, da diesen Exceptions acht dedizierte Register zugeordnet sind. Bei manchen Anwendungen, wie etwa bei der Berechnung von CRC-Zwischen-Prüfsummen eines Datenstroms, entfällt auch das Zurückschreiben der Daten in den Hauptspeicher, weil der Kontext dieser acht Spezialregister zwischen den Exceptions gesichert wird.
Kurz gesagt, zunehmende Leistungsanforderungen im Embedded-Markt führten zur Entwicklung von CPU-Pipelines mit höheren Frequenzen, breiteren Systembussen und leistungsfähigeren internen Prozessorarchitekturen, wobei man sich an den realen Anforderungen von Embedded-Systemen orientierte. Die besondere Herausforderung für den Systementwickler besteht darin, diese komplexen Komponenten zu integrieren, während die Zykluszeiten für Projekte immer kürzer werden. Die Halbleiterhersteller müssen dazu Werkzeuge liefern, mit denen der Kunde dieses Ziel erreichen kann. Viele Kunden wollen dabei keine Schaltungen entwickeln, die bei mehr als 100 MHz arbeiten und mit dem Prozessor verschaltet sind. Auch sind nur wenige bereit, Assemblercode zu schreiben, um das Leistungspotenzial der Komponenten ausreizen zu können. IDT ermöglicht es solchen Kunden, sich von der Low-Level Technologie zu distanzieren oder abzukoppeln.
Bild 1 zeigt, wie sich das erreichen lässt. Der gewünschte Grad an Abkopplung bestimmt den Punkt auf der Kurve, an dem der Kunde ansetzt. Manche nutzen verfügbare Produkte mit PCI-Schnittstelle auf Baugruppenebene und können sich damit auf die Integration von Peripherieschnittstellen und die Entwicklung proprietärer Software konzentrieren, wo die wichtigste Wertschöpfung erzielt wird. Andere benötigen Zugriff auf den CPU-Systembus und nutzen ein CPU-Modul, das die Höchstfrequenzaufgaben im Schaltungsdesign übernimmt.
Nachdem sich die Entwicklung von RISC-Architekturen über etliche Jahre an den Anforderungen der Workstations orientierte, konzentriert sie sich heute fast ausschließlich auf Embedded-Anwendungen. Alle konkurrierenden technologischen Lager entwickeln ihre Architekturen auf die eine oder andere Weise weiter; Ziel ist dabei immer eine Verbesserung der Effizienz, mit der Prozessoren Software für Embedded-Anwendungen ausführen können. Zu den Verbesserungen zählen Code-Kompression zur Verringerung des Speicherbedarfs, Befehlserweiterungen zur Verarbeitung komplexer Medien-Datenströme und Weiterentwicklungen bei den CPU-Pipeline-Strukturen. Prozessorhersteller müssen auf diesem Markt zwei Herausforderungen meistern: Zum Ersten muss man sich darauf konzentrieren, die zentralen Systemprobleme zu lösen, mit denen Embedded-Anwendungsentwickler kämpfen müssen. Und zweitens müssen die Kunden in der Lage sein, den Großteil des Bauteil-Leistungspotenzials nutzen zu können – von der Hochsprach-Programmierung bis zu einer relativ einfachen Hardware-Verschaltung, um die Systementwicklungszyklen noch weiter verkürzen zu können. (la)

552 IDT

Ian Ferguson *ist Technical Marketing Manager bei Integrated Device Technology, UK*