

Cache oder DMA?

MCU/DSP-Hybride richtig programmieren

Moderne Embedded-Multimedia-Applikationen erfordern ein verbessertes Zusammenspiel von MCU- und DSP-Funktionen. C-Programmierer, die mit den MCU-Programmiermethoden vertraut sind, sollten trotzdem neue Wege beschreiten, um mit einem intelligenten Management der Befehle und des Datenflusses die Systemleistung erheblich zu verbessern. Dabei fällt der Wahl des richtigen Verfahrens – Cache oder DMA – große Bedeutung zu. DAVID KATZ, RICK GENTILE



DAVID KATZ und
RICK GENTILE,
Blackfin Applikationsgruppe,
Analog Devices, Inc.

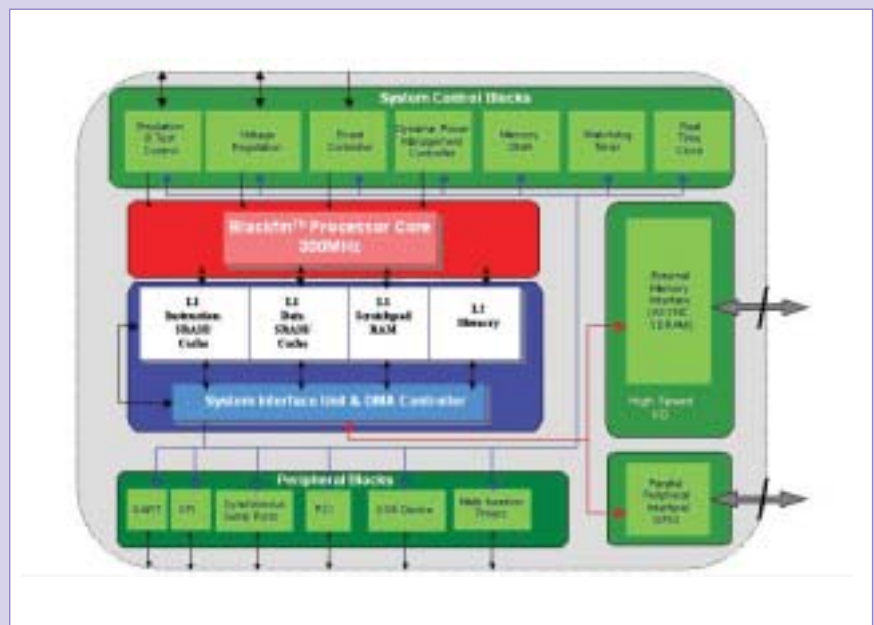


Abb. 1: Blockdiagramm des Embedded-Media-Prozessors ‚Blackfin‘

Die gegenwärtigen Embedded-Multimedia-Applikationen erfordern ein verbessertes Zusammenspiel von Systemsteuerung (typischerweise die Aufgabe eines MCU) und Signalverarbeitung (normalerweise Aufgabe eines DSP). Da neuerdings Embedded-Media-Prozessoren wie der ‚Blackfin‘ verfügbar sind, die sowohl MCU- wie auch DSP-Aufgaben übernehmen können, gehen C-Programmierer, die mit der MCU-Programmierung für die Entwicklung von Applikationen sehr vertraut sind, neue Wege. Mit einem intelligenten Management von Befehlen und Datenfluss kann die Systemleistung erheblich verbessert werden. Die Versuchung für MCU-Programmierer ist groß, ohne Veränderung ihren bisherigen Code zu übernehmen und einfach Befehls- und Daten-

Caches zur Lenkung der Befehle und Daten einzusetzen. Dabei sollte allerdings sorgfältig überlegt werden, welche Möglichkeiten der Hochleistungs-DMA des Mediaprozessors eröffnet. Wenn die Vor- und Nachteile bei Nutzung von Cache und DMA in diesen Anwendungen erkannt werden, versteht man besser, welchen Beitrag die Programmierung zur Systemoptimierung leisten kann.

Speicherarchitektur – Die Notwendigkeit einer geeigneten Speicherverwaltung

Die zurzeit zur Verfügung stehenden Mediaprozessoren besitzen hierarchische Speicherarchitekturen, die versuchen, verschiedene Speicher-

ebenen mit unterschiedlichen Größen und Leistungen auszubalancieren. Typischerweise arbeitet der Speicher, der direkt neben dem Kernprozessor liegt (d. h. der ‚Level 1‘ bzw. ‚L1‘-Speicher), mit der vollen Taktfrequenz und unterstützt dadurch in der Regel Befehlsausführungen in einem einzigen Zyklus. Dabei ist ein L1-Speicher oft unterteilt in Befehls- und Datensegmente, um die Bandbreite des Speicherbusses effizient auszunutzen (zeitgleiches Lesen von Daten und Instruktionen). Üblicherweise kann dieser Speicher entweder als SRAM oder Cache konfiguriert werden. Bei zeitkritischen Anwendungen kann die Anwendung auf das On-chip-SRAM mit einem einzigen Taktzyklus zugreifen. Für Systeme, die größere Programme benötigen, ist meist zusätzlicher On-chip- und Off-chip-Speicher verfügbar – verbunden mit größeren Verzögerungen bei Speicherzugriffen.

Diese Hierarchie hat jedoch ihre Grenzen; die Geschwindigkeit der heutigen Hochleistungsprozessoren würde sich drastisch verringern, wenn sie größere Anwendungsprogramme abzurufen hätten, die nur in langsamere externe Speicher passen. Um die Rechenperformance einer solchen Applikation zu verbessern, hat ein Programmierer verschiedene Möglichkeiten. Zum Beispiel kann er manuell wichtige Codeblöcke in bzw. aus dem internen SRAM bewegen, damit diese schnell ausgeführt werden. Das Hinzufügen zusätzlicher Daten- und Befehls-Cache-Speicher in die Architektur vereinfacht dabei die Verwaltung externer Speicher. Denn durch den Cache wird das manuelle Bewegen von Instruktionen und Daten im Prozessorkern verringert, und zusätzlich muss sich der Entwickler nicht mehr um die gesamten Daten- und Befehlsflüsse im Rechenkern Gedanken machen.

Befehlsspeicher – Cache oder DMA?

Ein kurzer Überblick über den Markt für Embedded-Media-Prozessoren zeigt Core-Prozessor-Geschwindigkeiten von 600 MHz und mehr.

Während diese Leistung die Tür zu vielen neuen Anwendungen öffnen kann, wird die maximale Geschwindigkeit jedoch nur dann erreicht, wenn der Code aus dem internen L1-Speicher stammt. Natürlich würde der ideale Embedded-Prozessor unbegrenzten L1-Speicherplatz besitzen, aber diesen Prozessor gibt es leider nicht. Deshalb müssen Programmierer verschiedenen Alternativen in Betracht ziehen. Man sollte die Vorteile des L1-Speichers, der bereits im Prozessor vorhanden ist, so nutzen, dass man den Speicher und die Datenflüsse für ein spezielles System optimiert. Betrachten wir einige solcher Szenarien.

Der erste und beste Einsatzfall liegt vor, wenn der Code der Zielanwendung komplett in den L1-Befehlsspeicher passt. Für diesen Fall werden

keine speziellen Aktionen benötigt, außer dass der Programmierer den Code direkt in dessen internen Speicherplatz schreibt. Daher sollten Mediaprozessoren, die sowohl MCU- als auch DSP-Funktionalität besitzen, sich besonders durch Codedichte auszeichnen.

Im zweiten Szenario wird ein Caching-Mechanismus benutzt, um Programmieren den Zugang zu größeren, weniger teuren externen Speichern zu ermöglichen. Dabei wird der Code automatisch in den L1-Befehlsspeicher geladen, wenn er benötigt wird. Der Hauptvorteil dieser Vorgehensweise besteht darin, dass der Programmierer nicht mehr die Verschiebungen der Programmteile in den Cache bzw. aus dem Cache verwalten muss. Diese Methode ist dann sinnvoll, wenn der Code mehr oder weniger linear ausgeführt wird. Für nichtlineare Codes kann es passieren, dass die Einträge im Cache zu oft ausgetauscht werden und so wirkliche Leistungssteigerungen behindern.

Der Befehls-Cache verfolgt tatsächlich zwei Aufgaben: zum einen hilft er, Instruktionen vom externen Speicher in einer effektiveren Weise vorzuladen. Ebenso hält der Cache vorhergehenden Code im Speicher. Dies wirkt sich besonders bei Befehlen aus, die häufig ausgeführt werden. Denn wenn der Code schon vorhanden und noch nicht überschrieben worden ist, kann dieser sofort in einem einzigen Core-Zyklus ausgeführt werden.

Die meisten reinen Echtzeitprogrammierer vertrauen eher nicht darauf, dass man mit dem Cache die beste Systemleistung erreicht. Sie argumentieren, dass es einen Leistungsabfall gibt, wenn ein Befehlssatz, der für die Ausführung benötigt wird, nicht im Cache vorliegt. Benutzt man den Vorteil eines Cache-locking-Mechanismus, kann man dieses Problem umgehen. Denn sind die kritischen Instruktionen in den Cache geladen, können die Cacheeinträge geblockt werden, so dass diese Befehle nicht ersetzt werden können. Damit haben Programmierer die Möglichkeit, nur das Notwendige im Cache zu behalten und den Cachingmechanismus zu benutzen, um weniger zeitkritische Instruktionen zu verwalten.

In dem letzten zu betrachtenden Szenario kann der Code mit Hilfe eines DMA-Kanals unabhängig vom Prozessorkern in und aus dem L1-Speicher bewegt werden. Während der Kern auf einem Bereich des Speichers arbeitet, wird der nächste Code mittels DMA-Zugriff in die nächste Sektion geladen. Dieses Schema ist im Allgemeinen bekannt als Overlay-Technik.

Ein höheres Maß an deterministischem Verhalten erzielt der Programmierer, wenn er den Code über DMA-Zugriff in den L1-Befehlsspeicher selbst transferiert, anstatt über den Cache-Speicher zu gehen.

Die vereinfachte Darstellung eines Embedded-Media-Prozessors (Abb. 1) zeigt die mehrfachen Pfade zwischen dem L1-Speicher und größeren externen Speichern. Befehle und Daten, die im

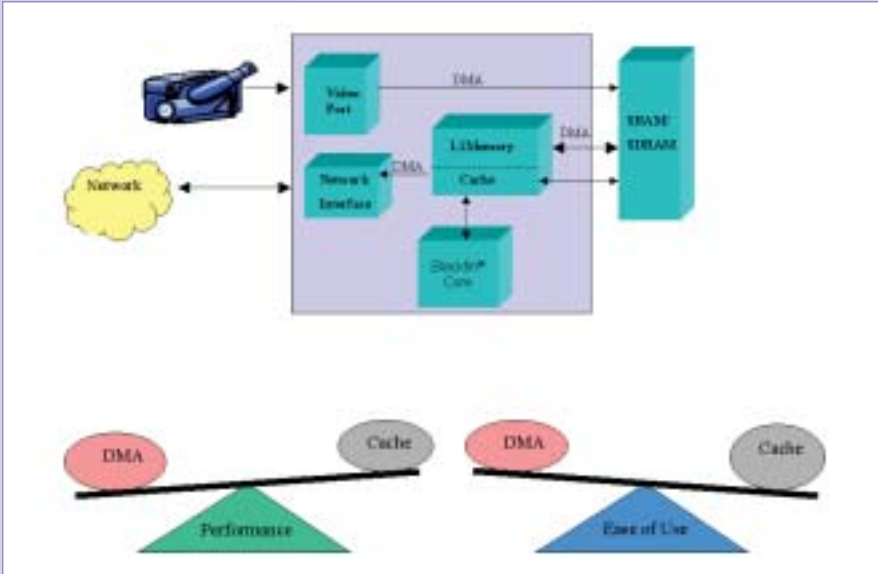


Abb. 2: Die schematische Darstellung eines Embedded-Media-Prozessors zeigt die mehrfachen Pfade zwischen dem L1-Speicher und größeren externen Speichern.

externen Speicher abgelegt sind, können in den Cache-Speicher von L1 übertragen werden. Oder ganze Blöcke lassen sich über den DMA-Controller völlig unabhängig im Hintergrund in den L1-Speicher verschieben. Peripherieeinheiten, die große Datensätze senden oder empfangen, können sie über den DMA-Zugriff direkt in den externen Speicher übertragen, ohne jede Beteiligung des L1-Speichers. Dabei erhöht sich jedoch der Aufwand des Programmierers durch das Ausarbeiten der Overlay-

Strategie und durch den Aufwand für die Konfigurierung der DMA-Kanäle. Dennoch ist die Leistungssteigerung bei einem gut geplanten Ansatz den zusätzlichen Aufwand wert.

Datenspeichermanagement

Die Datenspeicherarchitektur eines Embedded-Media-Prozessors ist für das ganze System genauso wichtig wie der Systemtakt für die

Befehle. Da bei einer Multimediaanwendung oft mehrfache Datentransfers in einer bestimmten Zeitspanne gleichzeitig stattfinden, muss die Busstruktur sowohl Kern- also auch DMA-Zugriffe auf alle Bereiche des internen und externen Speichers unterstützen. Es ist absolut wichtig, dass die Entscheidung der Kanalnutzung zwischen dem DMA-Controller und dem Kern automatisch gehandhabt wird, da sonst die Leistung extrem herabgesetzt wird. Die gegenseitige Beeinflussung von Kern und DMA sollte nur zum Aufsetzen des DMA-Controllers und zum Aktivieren des Datenkanals, z.B. nach einen Interrupt, erforderlich sein. Normalerweise ist das Laden von Daten eine Basisfunktion des Prozessorkerns. Typischerweise ist dies jedoch der am wenigsten effiziente Mechanismus für Datentransfers; dennoch ist es das einfachste Programmiermodell.

Für kleine Datenmengen gibt es meist ein kleinen, schnellen Scratchpad-Speicher im L1-Datenspeicher. Doch bei größeren Datenmengen leidet meist die Zugriffszeit, wenn der Kern die Daten aus dem externen Speicher holen muss. Dabei werden nicht nur mehrere Zyklen benötigt, der Kern wird durch den Übertragungsvorgang auch noch zusätzlich blockiert. In Multimediaanwendungen und anderen datenintensiven Operationen, bei denen große Datenmengen ständig in und aus dem SDRAM transferiert werden, ist dieses eine unhaltbare Situation.

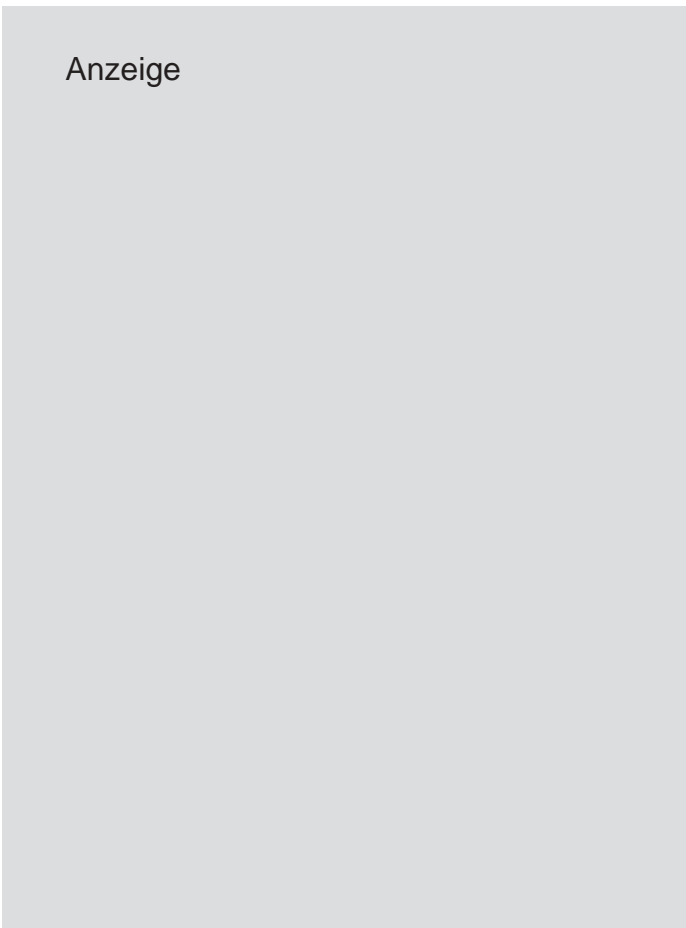
Während im Allgemeinen Datenzugriffe durch den Kern immer wieder benötigt werden, sollten große Bewegungen von Daten mit Hilfe von DMA oder Cache durchgeführt werden, um die volle Rechenleistung zu erhalten.

Der Einsatz von DMA zur Datensteuerung

Um DMA effektiv in einem Multimediasystem zu nutzen, müssen zunächst genügend DMA-Kanäle vorhanden sein, damit die Interfaces des Prozessors voll unterstützt werden.

Das gilt jedenfalls für alle Anwendungen mit mehr als einem Paar DMA-Speicherströmen. Das ist ein wichtiger Gesichtspunkt, denn er berücksichtigt, dass sicherlich unaufbereitete Ströme an Mediadaten über die Hochgeschwindigkeitsperipherie in den externen Speicher fließen, während zur gleichen Zeit Datenblöcke zwischen dem externen Speicher und dem L1-Speicher zur Weiterverarbeitung in der Rechen-einheit ausgetauscht werden.

Auch können DMA-Maschinen, die einen direkten Datentransfer zwischen den Peripherieeinheiten und dem externen Speicher erlauben und kein ‚Stopover‘ im L1-Speicher erfordern, zusätzliche Datenbewegungen in numerisch aufwendigen Algorithmen vermeiden helfen. Wenn die Datenraten und die Leistungsanforderungen steigen, wird es für die Entwickler



äußerst wichtig, dass er die Leistung des Systems beeinflussen bzw. fein abstimmen kann. Zum Beispiel könnte der DMA-Controller optimiert werden, um ein Datenwort bei jedem Takt zu übertragen.

Wenn viele Transfers in die gleiche Richtung ablaufen (z.B. alles vom internen zum externen Speicher), ist es meistens der effizienteste Weg, mit dem DMA-Controller zu arbeiten, da er Leerlaufzeit auf dem DMA-Bus vermeidet. In Fällen jedoch, in denen vielfältige bidirektionale Video- und Audioströme auftreten, wird eine Datenverkehrssteuerung zwingend notwendig. Es gilt nämlich zu verhindern, dass ein Datenstrom den Bus ganz allein für sich beansprucht. Wenn der DMA-Controller beispielsweise jeder Peripherieeinheit, die ein Datenwort zu übertragen hat, den Zugriff auf den DMA-Bus gestatten würde, würde sich der gesamte Durchsatz reduzieren, speziell wenn er auf eine Einheit wie ein SDRAM zugreifen würde. Denn wenn sich die Richtung der Datenübertragungen in beinahe jedem Zyklus ändert, wird die Wartezeit aufgrund des Umschaltens des SDRAM-Busses den Durchsatz signifikant verringern.

Dadurch sind DMA-Controller, die eine kanalabhängige, programmierbare Burst-Länge besitzen, im Vorteil gegenüber solchen mit einer festen Transfergröße. Da jeder DMA-Kanal eine Peripherieeinheit entweder mit dem internen oder dem externen Speicher verbinden kann, ist es auch wichtig, eine solche Einheit automatisch bedienen zu können, wenn sie dringend nach dem Bus verlangt.

Ein weiteres wichtiges Merkmal ist die zweidimensionale DMA-Fähigkeit. Sie bietet einige Vorteile auf der Systemebene. Zum einen erlaubt sie es, Daten intuitiv nacheinander in den Speicher zu übertragen. Kommen zum Beispiel Luminanz- und Chrominanzdaten von einem Bildsensor in Sequenzen an, können sie automatisch in separaten Speicherbuffern abgelegt werden. Die Interleaving/deinterleaving-Funktionalität von zweidimensionalem DMA spart zusätzliche Speicherbus-Transaktionen vor der Verarbeitung von Video- und Bilddaten. Zweidimensionaler DMA erlaubt damit, die Datenbandbreite durch selektive Übertragung zu minimieren, zum Beispiel nur den benötigten Bildausschnitt statt das Gesamtbild zu übertragen.

Ein weiteres Feature des Speichers und wichtiger Aspekt für Programmierer ist der Speicherbau. Typischerweise ist der interne Speicher in Speicherblöcken strukturiert. Dadurch kann ein gleichzeitiger Zugriff vom DMA-Controller und vom Kern in einem einzigen Zyklus durch die Platzierung von Daten in unterschiedlichen Blöcken erreicht werden. Beispielsweise kann der Rechenkern Daten in einem Block bearbeiten, während der DMA einen neuen Speicher in einem zweiten Block füllt.

Gleichzeitiger Zugriff auf den selben Block ist

unter bestimmten Bedingungen ebenfalls möglich. Bei Zugriff auf externen Speicher steht normalerweise nur ein physikalischer Bus zur Verfügung, der oft zwischen synchronen und asynchronen Speichern gemultiplext ist.

Daten-Cache

Die Flexibilität der heutigen DMA-Controller ist ein zweischneidiges Schwert. Wenn eine große C/C++-Applikation zwischen Prozessoren portiert wird, zögert der Programmierer manchmal, DMA-Funktionalitäten in den bereits laufenden Code zu integrieren. Hier kann der Daten-Cache sehr nützlich sein. Denn er ist eine Art Mini-DMA, der es ermöglicht, Daten für die schnellste Verarbeitung in den L1-Speicher zu transportieren. Der Daten-Cache ist reizvoll, weil nur minimaler Aufwand seitens des Programmierers gefordert wird.

Wegen der linearen Cache-Line-Architektur, ist der Einsatz des Daten-Cache am nützlichsten, wenn der Prozessor im externen Speicher an aufeinander folgenden Datenorten arbeitet. Der Grund dafür liegt darin, dass der Cache nicht nur die unmittelbaren Daten, die gerade bearbeitet werden, lädt; stattdessen holt er Daten in einen Bereich ‚angrenzend‘ an die aktuellen Daten. Mit anderen Worten, der Cache-Mechanismus geht davon aus, dass das aktuelle Datenwort Teil eines Blocks benachbarter Daten ist, die verarbeitet werden müssen.

Für Multimediabilder, Audio- und Videostreams ist dieses meist eine begründete Annahme. Da Datenblöcke normalerweise von externen Peripherien stammen, ist mit Daten-Cache-Speichern nicht immer so einfach zu arbeiten wie mit Befehls-Cache-Speichern. Das kommt daher, dass man die die Zusammengehörigkeit der Daten im Cache manuell steuern muss. Bei diesen Cache-Speichern müssen zuerst die alten Daten ungültig gemacht werden, bevor ein Versuch gestartet wird, auf die neuen Daten zuzugreifen.

Fazit

Zusammenfassend kann man sagen, dass es keine einfache Antwort gibt, ob die Wahl auf Cache oder DMA fallen sollte, wenn es um Mechanismen von Code- und Datenbewegungen in einem bestehenden Multimediasystem geht. Sind aber Entwickler einmal mit den Vor- und Nachteilen vertraut, können sie alle Vorteile eines solchen Prozessors nutzen, um die perfekte Optimierung für ihr System zu erreichen.

Beitrag als PDF per ‚more@click‘:

