MICRO CONSULT

07/2015

Trends bei Multicore-Mikrocontroller-Architekturen

Multicore-Mikrocontroller gewinnen in Embedded-Systemen immer mehr an Bedeutung. Grund dafür sind unter anderem die ständig steigenden Anforderungen der Automobil-Industrie und der hochdynamischen Consumer-Branche. Was bieten diese Systeme heute und in naher Zukunft, um den Hunger nach mehr Rechenleistung bei gleichzeitiger Energie-Diät zu stillen?

Einiges an Trends ist bei diesen hochintegrierten Bausteinen auszumachen. Zunächst betrifft es die allgemeine Struktur des Mikrocontroller-Designs, bei denen zwei grundlegende Architekturen zur Auswahl stehen, je nachdem ob man auf generische Designs setzt oder aber auf applikationsspezifische. Nicht immer hat man aber nach aktuellem Status der Multicore-Mikrocontroller auch die entsprechende Wahl.

Optimierungen der Performance stehen nach einer ersten Welle an Multicore-Mikrocontrollern ebenfalls an. Die Hersteller haben an dieser Stelle aus den ersten Designs gelernt bzw. das Feedback ihrer Kunden berücksichtigt. Auch die gesteigerte Nachfrage nach Designs im Umfeld von funktionaler Sicherheit sowie Security lassen mehr oder weniger getrennte Core-Domänen aufkommen. Komplexere Kommunikationsschnittstellen halten Einzug auf Multicore, und um den Stromverbrauch trotzdem gering zu halten, kommen komplexere Stromsparmodi zum Einsatz.

Bewegung ist ebenfalls auf der Seite der Toolhersteller zu sehen, die entsprechende Unterstützung dieser Systeme anbieten, was sowohl den Einstieg als auch den Umstieg von bestehenden Singlecore-Systemen erleichtern soll. Auch die Debug-Schnittstellen müssen dem Multicore-Trend folgen und angepasst werden.

Applikationsspezifische Multicore-Mikrocontroller-Designs

Zur Auswahl stehen zwei grundsätzliche Arten von Multicore-Mikrocontrollern, nämlich heterogene und homogene Architekturen.

Heterogene oder gemischte Architekturen enthalten verschiedene CPU-Typen, die für spezielle Applikationsanforderungen designed sind. Ein Beispiel sei mit TI 66AK2E05 genannt, der mit vier ARM-A15 und einem TI-DSP Core ausgestattet ist.

Anwendungen finden sich in der Unterhaltungselektronik, Luftfahrt-Industrie und Telekommunikation. Der Vorteil dieser Architekturen ist die Anpassung und Auslegung auf spezifische Applikationen, was einen höchst effizienten Einsatz erlaubt.

Homogene Architekturen enthalten gleichartige CPU-Typen, die für spezielle, typischerweise sehr rechenintensive Applikationen angeboten werden. Beispiele hierfür sind z.B. Infineon AURIX TC279 mit drei TriCoreTM-Cores (Hauptanwendung: Automobil-Motor- und Getriebemanagement) oder Freescale i.MX6 mit mehreren ARM® CortexTM-A9 Cores (Dual Core, Quad Core).

Sie kommen z.B. in Multimedia Anwendungen für die Unterhaltungselektronik zum Einsatz.

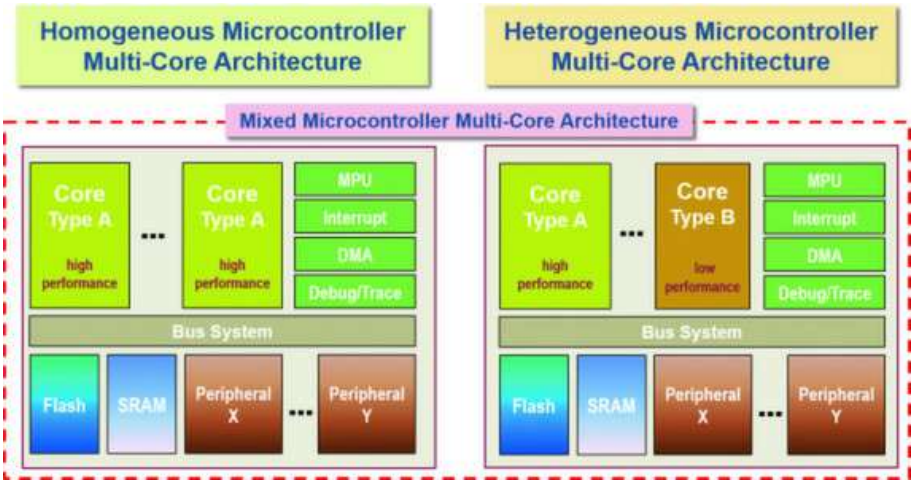

Bild 1: Welche Architektur verwendet wird, hängt von den Auswahlmöglichkeiten und den Applikationsanforderungen ab

Performance-Trends bei Multicore-Mikrocontrollern

Eine Erhöhung der Bandbreite von Speichern wird durch verschiedene Arten der Implementierung ermöglicht. Busbreiten mit doppelter bzw. vierfacher Datenbreite sowie Block-Übertragungen (Bus Burst Accesses) für Lese- und Schreibzugriffe reduzieren die Häufigkeit von Buszugriffen für systemglobale Speicher und somit den Performance Overhead, der ansonsten zum Tragen käme. Außerdem müssen Ansätze gewählt werden, die es erlauben, effizient auf geteilte Ressourcen zugreifen zu können. Eine Busmatrix erlaubt mehreren Bus-Master-Blöcken, gleichzeitig auf verschiedene Bus-Slave-Bausteine zuzugreifen.

Beispielsweise können eine CPU gleichzeitig auf Programm-Flash und Daten-SRAM zugreifen und eine weitere CPU auf andere nicht belegte Peripherie oder ein DMA Controller Transfers durchführen. Ein verzögerter Zugriff tritt nur dann auf, wenn zwei Master gleichzeitig denselben Slave-Baustein adressieren. In diesem Fall werden die Speicherzugriffe entsprechend der Bus-Master-Priorität über einen Arbiter bearbeitet, wie es bei herkömmlichen Bussystemen der Fall wäre.

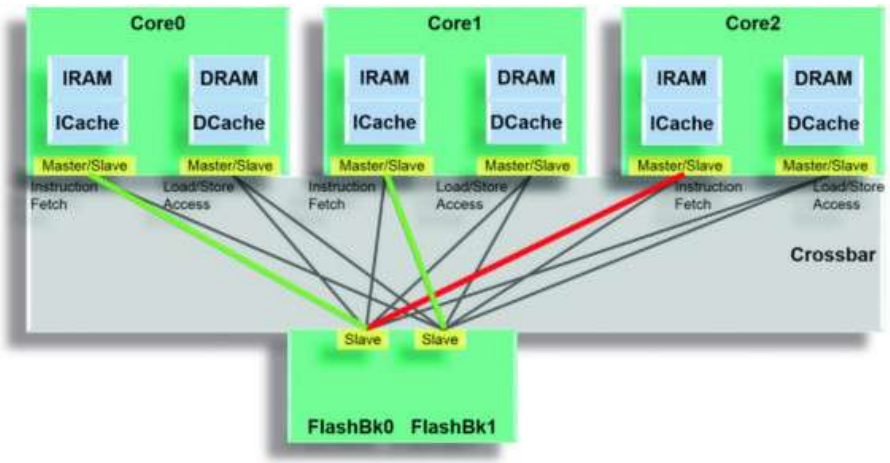

Bild 2: Bus-Matrix ermöglicht den gleichzeitigen Zugriff mehrerer Master auf unterschiedliche Slaves

CPUs neuerer Generation erlauben die parallele Befehlsabarbeitung mit hoher Taktung (Superscalar Architecture). Diese Art der Befehlsabarbeitung (z. B. parallele Ausführung eines Arithmetik/Logik- und eines Load/Store-Befehles) erfordert, dass die CPU mit genügend hoher Geschwindigkeit auf die Befehlsspeicher zugreifen kann.

Da Flash-Speicher (mit Programm und konstanten Dateninhalten) nicht mit der Geschwindigkeitszunahme der CPUs standhalten, müssen in der CPU-Pipeline Wartezyklen (Wait States) bei den Flash-Lesezugriffen eingefügt werden, was zu Wartezyklen bei der CPU-Befehlsverarbeitung führt. Dieses Geschwindigkeitsproblem lässt sich mit der Implementierung schneller Level 1 Speicher wie SRAM (Tightly Coupled Memories) und Cache-Speichern beheben.

Werden zur Speicherzugriffsbeschleunigung Caches eingesetzt, können Datenkonsistenz-Probleme bei gemeinsam verwendeten globalen Speichern auftreten, wenn die Daten im Betrieb beschrieben werden sollen. Eine CPU ändert die Daten bei einem Speicherschreibzugriff nicht im Originalspeicher, sondern zunächst nur in der „Kopie“ im Cache-Speicher. In diesem Fall liefern Lesezugriffe anderer CPUs den ursprünglichen Wert aus dem Originalspeicher und nicht den geänderten Wert. Dieses Cache-Kohärenzproblem löst z. B. ARM® durch den Einsatz eines in Hardware implementierten Cache-Snoopers. Dieser aktualisiert die Inhalte von Caches anderer Cores nach einem Schreibzugriff auf einen Cached-Memory. Damit werden systemweit transparente Dateninhalte für globale, durch Cache-Einsatz beschleunigte Speicher garantiert.

Greifen mehrere Cores auf globale (nicht durch Cache-Speicher beschleunigte) Datenspeicher-Arrays oder Strukturen (SRAMs) zu, müssen die Zugriffe, die die Datenbusbreite übersteigen, durch Semaphore geschützt werden. Hierfür gibt es zwei unterschiedliche Implementierungen über atomare Befehlssequenzen (Atomic Instructions) oder Hardware-Semaphore.

Bei ersteren verbietet das Bussystem für sogenannte Read/Modify/Write-Back-Befehle die Unterbrechung des Befehlsablaufes, z.B. durch Interrupt oder gleichzeitigen Zugriff einer weiteren CPU. Damit lassen sich in Multicore-Systemen die Zugriffe auf Semaphore-Variablen (realisiert in Software), die zur Zugriffssteuerung auf globale Ressourcen eingesetzt werden, synchronisieren. Vorteil der SW-Semaphore ist, dass „beliebig“ viele Semaphore-Variablen verwendet werden können.

Hardware-Semaphore sind spezifische zugriffsgesteuerte Speicherzellen, die für die Zugriffssteuerung auf Ressourcen (z.B. gleichzeitiger Speicherzugriff durch verschiedene CPUs) durch parallel arbeitende Prozesse vorhanden sind. Eine Einschränkung in der Applikation kann sich durch die Anzahl der implementierten HW-Semaphore ergeben.

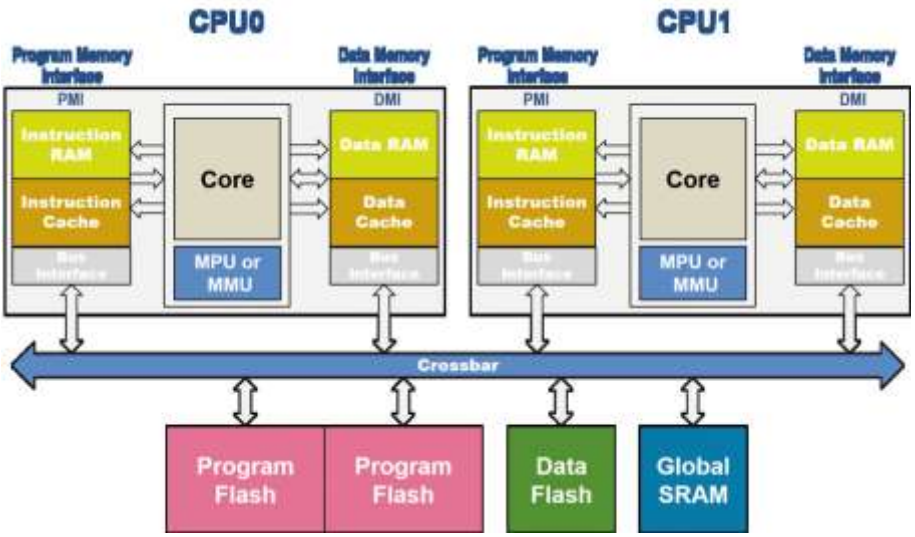

Bild 3: Lokale schnelle Speicher und globale geteilte Speicher erhöhen den Aufwand in der Software-Architektur

Implementierung von Sicherheitsfunktionalität: Safety und Security

Je nach möglicher Gefährdung von Personen durch das Gerät, die Maschine, das Fahrzeug etc. werden Sicherheitsnormen in den verschiedenen Industriezweigen definiert. Je höher die Gefährdung, desto mehr Überwachungsmechanismen eines Systems werden gefordert, damit eine funktionale Sicherheit des Systems/der Maschine gewährleistet werden kann. Als Beispiele seien hier die SIL- und ASIL- (Automotive SIL) Spezifikationen genannt.

In die neuen Multicore-Mikrocontroller-Architekturen werden daher eine Reihe von Funktionsblöcken eingebaut, für die Überwachung von und entsprechende Reaktion auf sicherheitsrelevante Ereignisse (Schutz von Speicherinhalten durch Fehlererkennungs- und Fehlerkorrekturmechanismen, Spannungsüberwachung für Über- und Unterspannungserkennung, Frequenzüberwachung für Über- oder Unterschreitung der Betriebsfrequenz, Temperaturüberwachung des Siliziums, Schutzmechanismen zur Erkennung von Übertragungsfehlern, Überwachung der CPU-Befehlsverarbeitung, Softwareablaufsicherung durch Watchdog Timer, Speicherzugriffsschutz-Einheiten, Differenzierung zwischen sicherheitsrelevanter und nicht sicherheitsrelevanter Software auf Basis von Status Bits).

Alle Überwachungsfunktionen liefern erkannte Fehler an eine eigene Fehlersammel- und Korrektureinheit (z.B. Fault Collection and Control Unit FCCU, resp. Safety Management Unit SMU). In diesen Einheiten kann für jeden Fehlertyp die entsprechende Fehlerreaktion bestimmt werden. Typischerweise stehen der Aufruf einer Interrupt-, bzw. einer Trap-Funktion oder das Auslösen eines internen Resets (Neustart der Software) zur Verfügung. Ebenfalls möglich ist die Signalisierung nach extern über einen spezifischen Signalisierungs-Port/Pin.

Der Vorteil dieser Safety-Implementierung liegt in der Möglichkeit der softwareunabhängigen Fehlerreaktion (hardwaregesteuerte Fehlerantwort - Generierung eines Resets bzw. externen Signals) auch bei fehlerhafter Taktung des Systems. Diese Safety-Methoden dienen zum Schutz vor Gefahren, zur Verhinderung von Schäden ausgelöst durch fehlerhaft arbeitende Systeme und zur Minimierung der Risiken.

Abhängig von der Art eines Systems werden unterschiedliche Anforderungen an die Sicherheit (Vertraulichkeit, Verfügbarkeit und Datenintegrität) gestellt. Jährlich entstehen z.B. der Automobilindustrie Schäden in Millionenhöhe durch Veränderungen der Steuerungssoftware im Automobil (sogenanntes Software Tuning z.B. zur Steigerung der Motorleistung). Gleiche Anforderungen kann man in der Welt des IoT (Internet of Things) erkennen. Wie können unsere Konten vor nicht autorisiertem Zugriff über das Internet geschützt werden?

In die neuen Multicore-Mikrocontroller werden eigene Security-Cores eingefügt. Diese sind durch eine Firewall gegen unerlaubten Zugriff geschützt. In diesen Security-Architekturen sind Ver- und Entschlüsselungsmodule enthalten (z.B. AES-Module – Advanced Encryption Standard Module), Random Number Generator RNG und zugriffsgeschützte Speicher (gesichertes RAM zum Verwahren von Sicherheitsschlüsseln und geschützter Programmteile). Durch die Kapselung bzw. strikte Trennung dieser Sicherheitsdomäne vom Rest des Systems lassen sich Angriffe von außen besser abwehren.

Komplexere Kommunikations-Interfaces

Gesteigerte Anforderungen an die Kommunikationsschnittstellen beinhalten die Kommunikation großer Datenmengen, hohe Übertragungsraten und sichere Kommunikation (Generierung von Sicherungscodes beim Sender sowie hardwaregesteuerte Prüfung beim Empfänger).

Je nach Anforderung an die Kommunikation können hierfür Kommunikationsmodule wie Ethernet oder FlexRay implementiert sein. Ethernet bietet hierbei Datenraten von 10Mb/100Mb sowie Sicherung auf der Übertragungsstrecke durch 32-Bit CRC-Checksummen. FlexRay unterstützt bis zu 10 Mb Datenrate, Identifier und Daten-Checksummen, zweikanalige gesicherte Datenübertragung mit zeitgesteuerter Kommunikation (deterministische und arbitrierungsfreie Datenübertragung, Time Triggered Protocol TTP). Anwendungsfälle sind in diesem Zusammenhang z.B. X-by-Wire Steuerungen im Automobil.

Aufwertungen gibt es auch auf etablierten Schnittstellen wie SPI. Mehrere Slaves mit unterschiedlichen Anforderungen an Kommunikationsframes und Geschwindigkeiten können an ein solches SPI Interface angeschlossen werden. Die Umprogrammierung für die jeweilige Slave-Schnittstelle kann DMA-gesteuert erfolgen und entlastet somit die Cores. Der Anwender kann hierzu neben den Sendedaten auch die Einstellungsdaten in einem Speicher ablegen/vorbereiten. Ist die Schnittstelle frei, kann der DMA die nächsten/neuen Einstellungsinformationen des SPI-Protokolls laden und anschließend die Kommunikationsdaten übertragen. Hierbei wird keine weitere CPU-Last generiert (hardwaregesteuerte Kommunikation).

Power-Management in Multicore-Mikrocontrollern

Die Anforderungen an die Stromaufnahme werden immer höher, damit in batteriebetriebenen Systemen eine möglichst lange Betriebsdauer gewährleistet werden kann und die Gesamt-Stromaufnahme die vorgegebene maximale Verlustleistung im System nicht übersteigt. Unterschiedliche Methoden zur Minimierung bzw. Reduktion der Stromaufnahme können angewendet werden. Die Applikationssoftware kann in verschiedene Partitionen aufgeteilt werden. Jede Partition wird einem anderen Core zur Verarbeitung zugewiesen. Wird dabei ein Core temporär nicht benötigt, kann diese CPU in den IDLE-Mode versetzt werden (dazu wird der CPU-Takt vorübergehend abgeschaltet). Bei Bedarf kann ein anderer Core bzw. ein Interrupt-Ereignis den Takt wieder einschalten, damit die CPU die Befehlsausführung fortsetzen kann.

Des Weiteren können die verschiedenen Cores mit unterschiedlicher Geschwindigkeit betrieben werden und somit Strom sparen. Dazu sind z.B. einige heterogene Multicore-Architekturen mit einem kleinen leistungssparenden und einem leistungsstarken (high-performance) Core versehen. ARM® hat dies in der big.LITTLE-Implementierung umgesetzt, zusätzlich zu der Möglichkeit, Versorgungsspannung und Taktfrequenz dynamisch an die jeweiligen Erfordernisse anzupassen. Ein Cortex-A15 (big) + Cortex-A7 (LITTLE) sind als Oktal-Core in der Samsung-Handyplattform Exynos 5 Octa eingesetzt. Im heterogeneous Multiprocessing (HMP) Mode teilt der Scheduler die Tasks entweder einem big- oder einem LITTLE-Core entsprechend dem aktuellen Workload zu und verringert damit die systemweite Stromaufnahme.

Multicore-Toolsupport für erweiterte Debug-, Test- und Simulationstools

Multicore-Debugger und spezielle Timing-Toolsuites lassen sich für die Analyse der Multicore-Programmausführung für die funktionale Verifikation und Fehleranalyse, Performance-Analyse und Leistungsverteilung zwischen den Cores sowie Intra- und Intercore-Timing-Analyse einsetzen.

Entsprechende Debugger ermöglichen einen synchronen Start/Stop der Cores über sogenannte Run Control Groups. Bei Erreichen eines einfachen/komplexen Break-Kriteriums können mehrere bzw. alle Cores inklusive Teile der On-Chip-Peripherie gleichzeitig gestoppt bzw. im Anschluss wieder gestartet werden.

Traceaufzeichnungen können in Multicore-Systemen beispielsweise zur Analyse des Programmablaufs/Scheduling in den Cores, der Intercore-Kommunikation und Synchronisation durch Interrupts dienen. Dazu werden bei einigen Multicore-Architekturen spezielle Debug-Devices und spezielle Trace-Interfaces bzw. zusätzlicher Debug-/Tracespeicher benötigt. Multicore-Systeme generieren mehr potentielle Tracedaten als Singlecore-Bausteine. Für die Pufferung der Tracedaten gibt es zwei Implementierungsansätze basierend auf größeren on-Chip Tracespeichern bzw. on-Chip High-Speed Streaming Interfaces (Gigabit-Kommunikation) mit Tracedaten-Zwischenspeicher off-Chip in der Debugger Hardware. Unterschieden wird bei den Traceaufzeichnungen zwischen Code- und Data-Trace.

Trotz des Einsatzes von Gigabit-Interfaces bzw. eines mehrere Megabyte großen Emulationsspeichers können typischerweise bei einer Code-Traceaufzeichnung nicht alle internen Core-Informationen aufgezeichnet werden. Bei einer Multicore-Implementierung lassen sich unter Umständen nur die Befehlsabläufe von zwei CPUs aufzeichnen. Für die System-Testbarkeit sollte daher die Software-Architektur Einschränkungen (z.B. limitierte Tracebandbreite) bei der Aufteilung der Threads/Tasks auf die Cores berücksichtigten.

Für eine effiziente Traceauswertung bieten die Debugger die Möglichkeit der Definition von komplexen Multicore-Trace-Trigger-Signalen. Damit lässt sich die auszuwertende Trace-Datenmenge reduzieren bzw. eingrenzen.

Traceaufzeichnungen können in der Folge auch für die Analyse der Software-Architektur verwendet werden. Beispielsweise lassen sich Code-Coverage-Analysen durchführen (Statement, Branch, Path, Condition Coverage) oder Performance-Analysen (Execution Times der einzelnen Cores, Lastverteilung der CPUs (Load Balance), Blockierzeiten der CPUs (Blocking Times, z.B. bei Shared Ressources)).

Moderne Debug- und Trace-Analysetools sind ferner imstande, schon während laufender Traceaufzeichnung eine Analyse der Tracedaten durchzuführen. Damit lässt sich erheblich Zeit bei der Systemanalyse einsparen.

Fazit

Multicore-Systeme versprechen höhere Rechenleistung bei besserer Energie- und Kosteneffizienz. Geteilte Ressourcen und neue Aufgabenstellungen hinsichtlich Intercore-Kommunikation und -Synchronisation sowie Austausch von Daten ziehen Mehraufwand im Software-Design nach sich. Um diese Systeme gezielt und bestmöglich nutzen zu können, ist außerdem das Verwenden von chipspezifischen Features unumgänglich. Generische Ansätze über homogene Architekturen versprechen Portabilität und breitere Tool-Unterstützung; heterogene Implementierungen stehen für optimierte applikationsspezifische Lösungen. Sofern es diesbezüglich überhaupt ein Rennen gibt, ist es noch nicht entschieden.

Quellen:

Multicore im Mikrocontroller, Kursunterlage MicroConsult GmbH
Embedded-Architektur, Kursunterlage MicroConsult GmbH
AURIX, TriCore, ARM, Kursunterlagen MicroConsult GmbH

Autoren:

Dipl.-Ing. (TU) **Marcus Gößler** schloss das Studium der Elektrotechnik an der Technischen Universität Graz ab. Seine berufliche Laufbahn begann als Field Application Engineer für analoge und digitale Produkte im Bereich Luft-und Raumfahrt. Weitere Applikationsfelder umfassten Audio/Video, portable Systeme und Infotainment im Automobil. Er leitete Applikationsorganisationen in Zentral- und Osteuropa und zeichnete verantwortlich für große Halbleiterhersteller im Vertriebskanal und Marketing. Bei MicroConsult ist er heute als Trainer und Coach im Bereich Embedded Systems tätig, mit Schwerpunkten auf sicherheitsrelevanten Anwendungen und Multicore-Bausteinen.

Dipl. Ing. **Ingo Pohle** ist Mitgründer und Geschäftsführer der MicroConsult. Er ist ein international anerkannter Spezialist für Embedded-Lösungen mit reichem Erfahrungsschatz rund um den Einsatz von Mikrocontrollern, Bussystemen und RTOS.

MicroConsult GmbH - Experience Embedded:

MicroConsult ist Ihr Partner für Embedded Systems Engineering - professionelle Beratung, Projektunterstützung und Schulungen.

www.microconsult.de