Embedded Software Engineering Kongress

ESE Kongress 2013

Vortragsskript:

Multicore im Embedded System – Wie geht das?

Welche besonderen Aufgaben kommen auf die Entwickler zu?

Renate Schultes, MicroConsult GmbH

Multicore ist mittlerweile auch im Embedded Bereich in aller Munde – aber ist es auch in den Köpfen der Entwickler angekommen? Welche besonderen Herausforderungen müssen bewältigt werden, um ein Embedded Multicore-System nach den vorgegebenen Qualitätsmerkmalen „richtig“ zum Laufen zu bringen?

Grundlagen

Grundsätzlich können zwei verschiedene Arten von Multicore Anwendungen unterschieden werden:

- Separated Application Domain
- Single Application Domain

Separated Application Domain

In jedem Core läuft eine eigenständige Anwendung, es gibt keine Kommunikation oder Interaktion zwischen den Anwendungen.

Das scheint auf den ersten Blick die einfachere Variante zu sein. Sauber getrennte Applikationen, ohne Schnittstellen, ohne Synchronisationsaufwand. Aber Vorsicht: Hier muss in der Designphase darauf geachtet werden, dass die vorhandenen Ressourcen auf die Applikationen verteilt werden. Eine gemeinsame Nutzung von Speicher und Peripheriemodulen würde sofort dazu führen, dass Zugriffsschutz, Zugriffsrechte und der entsprechende Verwaltungsaufwand benötigt werden.

Und wie sieht es mit Speicher – Programm- und Datenspeicher – aus? Verfügt der Baustein über nur einen Programmspeicher für alle Cores (Uniform Memory Access – UMA), kann es zu verlängerten Zugriffszeiten kommen, wenn sich die Cores bei gleichzeitigem Instruction-Fetch gegenseitig behindern.

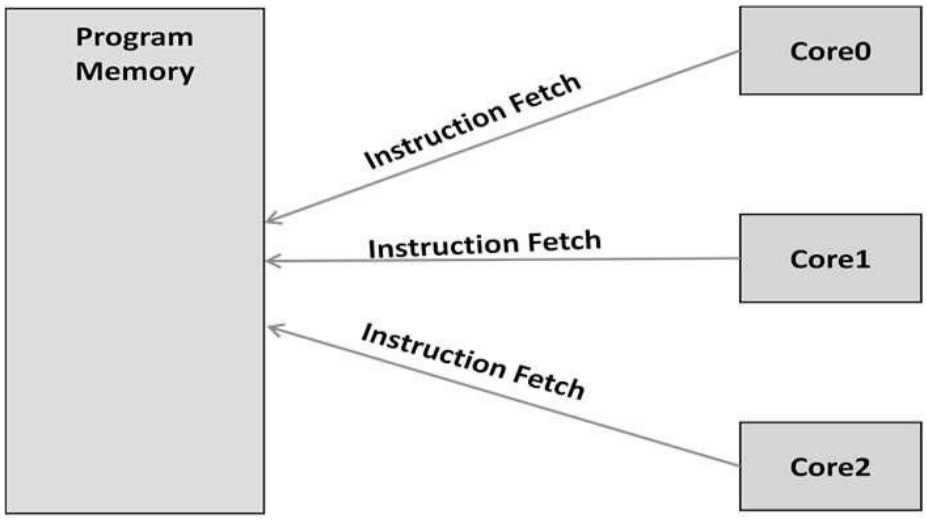

Bild 1: Uniform Memory Access – UMA

Enthalten die einzelnen Cores Instruction Cache (Cache, ICache), sollte dieser unbedingt genutzt werden, um zusätzliche Wartezeiten durch direkte Speicherzugriffe zu vermeiden. Der Instruction Cache sollte allerdings nur für Programmteile genutzt werden, die häufig, bzw. ständig in Gebrauch sind. Interrupt Service Routinen gehören nicht dazu. Sie werden nur benötigt, wenn der Interrupt Request ausgelöst wird, und das passiert im Normalfall mit mehr Zeitabstand. Gibt es neben dem Instruction Cache im Core auch noch „privates“ RAM für Instruction Fetches (Tightly Coupled Memory), bietet dieser sich an für die Nutzung durch Interrupt Service Routinen.

Für Datenspeicher gilt ähnliches: Gibt es privaten Datenspeicher in der unmittelbaren Core-Umgebung (Tightly Coupled Memory), sollte dieser hauptsächlich genutzt werden. Gibt es gemeinsam genutzten Datenspeicher, der nur über einen Buszugriff erreichbar ist, führt dies zu längeren Zugriffszeiten, wenn mehrere Cores gleichzeitig zugreifen wollen. Deshalb sollte im Design wieder sorgfältig darauf geachtet werden, wie die Aufteilung des Gesamtspeichers auf die Cores erfolgt.

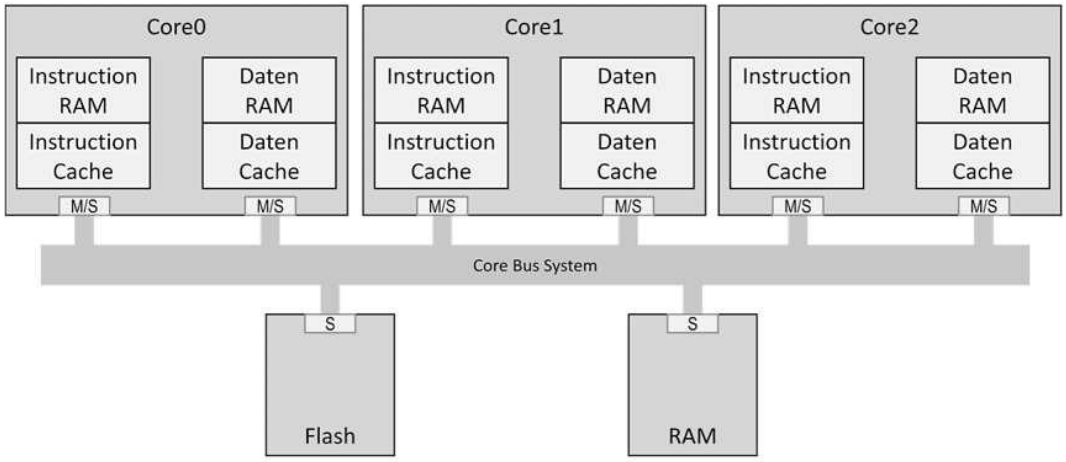

Bild 2: Multi-Core System mit Cache und lokalen Programm- und Datenspeichern

Stehen mehrere getrennte Programmspeicher mit getrennten Busschnittstellen zur Verfügung (Non-Uniform Memory Access – NUMA), muss darauf geachtet werden, dass die Programmteile der unterschiedlichen Applikationen auf die verschiedenen Speicherbereiche verteilt werden.

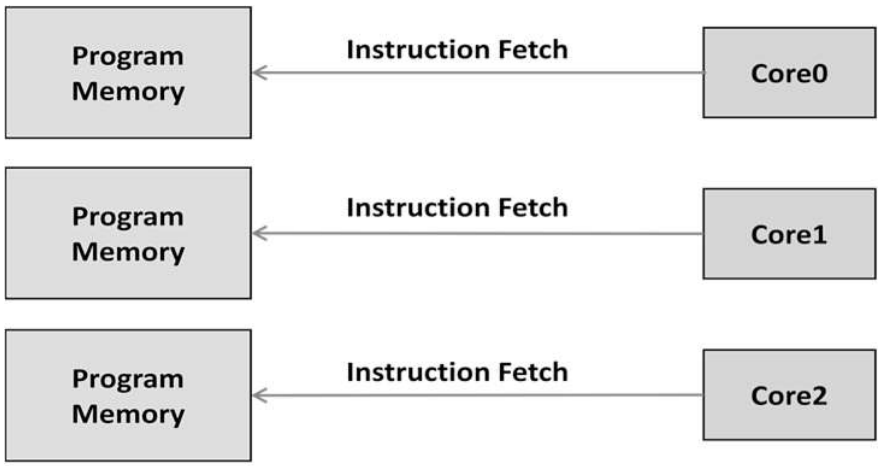

Bild 3: Non-Uniform Memory Access – NUMA

Moderne Mikrocontroller haben anstelle des klassischen Bussystems (Adressbus, Datenbus, Steuerleitungen) heute eine sogenannte Crossbar. Dabei ist jedes Bus Masterinterface mit jedem Bus Slaveinterface direkt verbunden. Haben verschiedene Speichermodule getrennte Bus Slaveinterfaces, können verschiedene Cores gleichzeitig auf verschiedene Speicherbereiche zugreifen. Das beschleunigt die Programmabarbeitung erheblich.

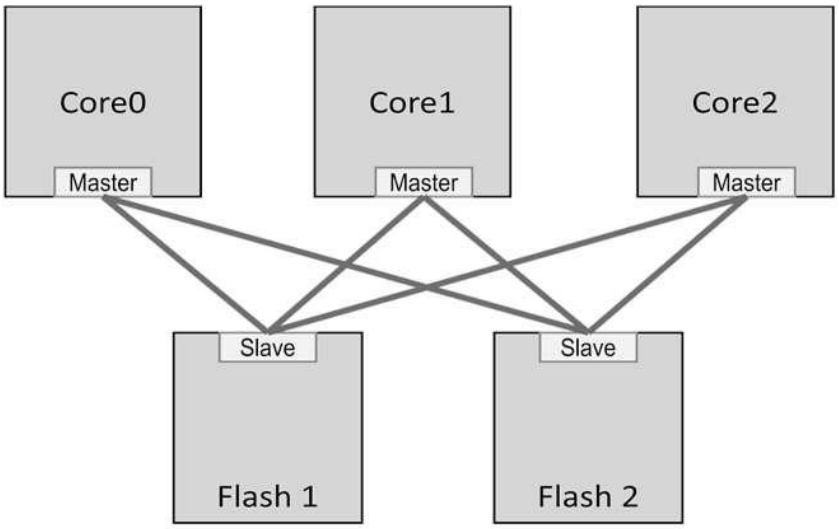

Bild 4: Multi-Core System mit Crossbar und getrennten Programmspeichern

Die saubere Trennung von Programm und Daten für die verschiedenen Cores ist also nicht nur die Basis für die klare Trennung auf logischer Ebene, sondern auch für die Performance eines „Separated Application Domain“ Systems.

Single Application Domain

Eine Anwendung läuft verteilt auf mehreren Cores. Kommunikation und Synchronisation zwischen den Cores ist erforderlich. Das wird wohl der typische Anwendungsfall für Multicore Systeme sein. Und hier kommen die größeren Herausforderungen auf die Entwickler zu.

Zugriffsrechte, Zugriffsschutz, Synchronisation und Nachrichtenaustausch spielen hier, neben den zuvor schon genannten Punkten (Zugriff auf gemeinsame Ressourcen wie Programm- und Datenspeicher), eine zentrale Rolle. je größer die Anzahl der Cores, desto höher ist der Kommunikationsaufwand (er steigt überproportional).

Zugriffsrechte und Zugriffsschutz

Wie in einem Multitasking System ergeben sich auch beim Multicore System Fragen, die in der Designphase geklärt werden müssen:

Welcher Core darf wann auf gemeinsame Ressourcen wie Speicher und Peripheriemodule zugreifen, und welche Art von Zugriff (lesend/schreibend) ist erlaubt?

Welcher Core ist für die Initialisierung des Gesamtsystems zuständig?

Wie wird das System vor nachträglichen Änderungen durch andere Cores geschützt?

Mittlerweile stellen auch Mikrocontroller in Hardware eingebaute Mechanismen für die Verwaltung und Überwachung von Zugriffen durch verschiedene Prozesse bzw. Cores zur Verfügung.

- Eine Memory Protection Unit MPU stellt zum Beispiel sicher, das nur ausgewählte Prozesse auf bestimmte Speicherbereiche Zugriff haben, bzw. welche Art von Zugriff (read/write, read only oder write only) durchgeführt werden.

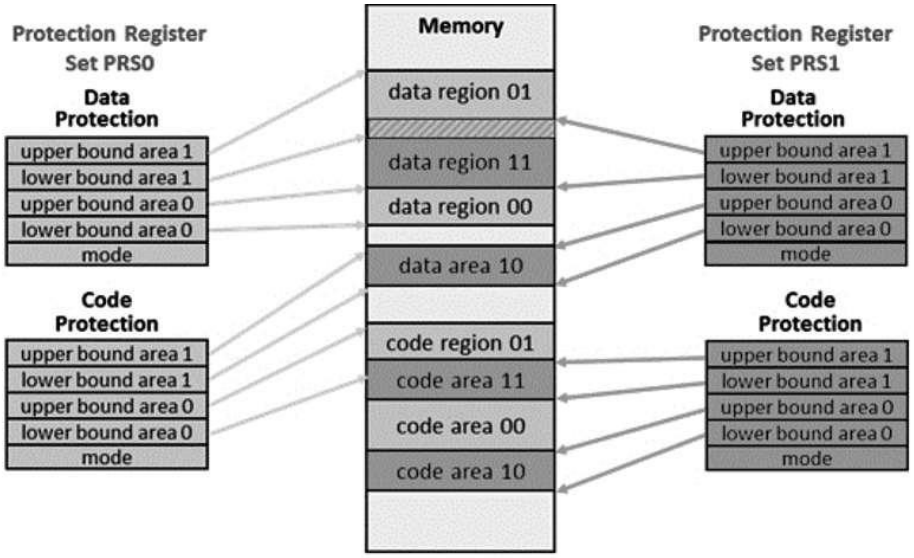

Bild 5: Memory Protection Unit –MPU

- Supervisor und User Einstellungen im Core können eingesetzt werden, um Zugriffsrechte auf spezielle Register (z.B. Basiskonfiguration eines Cores, Takteinstellungen des gesamten Systems, Bus- und Speicherkonfigurationen, usw.) einzuschränken.

- Spezielle Schutzmechanismen, zum Beispiel in die Buszugriffslogik eingebaut, schützt Register eines Cores bzw. Basiskonfigurationsregister vor dem Zugriff durch den „falschen“ Core.

  Solche Schutzmechanismen sind typisch nach Reset inaktiv, das heißt, sie müssen per Software aktiviert werden.

  Auch das muss in der Designphase berücksichtigt werden, da die Laufzeit der Initialisierungsphase eines Systems verlängert wird, wenn all die möglichen Schutzmechanismen erst aktiviert werden müssen.

Kommunikation

Die Kommunikation beruht auf versenden von Nachrichten zu Empfängern. Nachrichtenformen können wie im Multitasking System unter anderem

- Funktionsaufrufe
- Signale
- Datenpakete

sein. Werden Signale und Datenpakete verwendet, stellen sich auch hier die üblichen Fragen wie in einem Multitasking System:

- Wie wird sichergestellt, dass eine Nachricht zur Verfügung steht, wenn sie gebraucht wird?
- Wie wird sichergestellt, dass die Nachricht rechtzeitig abgeholt wird, bevor sie veraltet ist?

Bei sehr komplexen Prozessen ist es oft notwendig, einen gemeinsamen Zeitbegriff zu realisieren.

- Mit Hilfe einer „logischen Uhr“ (monoton steigender Zeitwert) können Ereignisse mit einem Zeitstempel versehen werden. Der Zeitstempel des auslösenden Ereignisses muss immer kleiner sein, als der des ausgelösten Ereignisses.

- In vielen Mikrocontrollern steht zum Beispiel für Multitasking Systeme eine Hardware-Echtzeituhr zur Verfügung. Diese kann ebenfalls genutzt werden, um Nachrichten mit einem Zeitstempel zu versehen.

Steht gemeinsam genutzter Datenspeicher für alle Cores zur Verfügung, kann die Kommunikation stark vereinfacht werden. Der auf einem Core laufende Prozess schreibt zum Beispiel einen Wert in eine gemeinsam genutzte Variable. Diese Variable wird von einem Prozess, der auf einem anderen Core läuft gelesen und ausgewertet.

Grundsätzlich sollten die Multicore Systeme so geplant werden, dass eine maximale Entkopplung erreicht wird.

Synchronisation

Die zeitliche Abfolge von Ereignissen muss im Design bestimmt, und zur Laufzeit eingehalten werden. Aktionen können in einer bestimmten Reihenfolge oder auch gleichzeitig auftreten.

Zum Beispiel können die Echtzeituhren der unterschiedlichen Cores verglichen, und wenn notwendig angepasst werden.

Mit Hilfe von Software Posted Interrupts kann ein Core an einen, oder wenn Broadcast möglich ist an mehrere andere Cores gleichzeitig, ein Event schicken, das zum Beispiel anzeigt, ob eine bestimmte Aufgabe gemeinsam gestartet werden soll.

Toolunterstützung

Um die Anforderungen bei der Implementierung von Multicore Systemen in den Griff zu bekommen, benötigen die Entwickler eine anpassungsfähige Tool Umgebung. Eine gute Tool Umgebung muss sowohl die Separated Application Domain, als auch die Single Application Domain unterstützen.

- Separated Application Domain
  Separate ELF-File Erzeugung individuell lokatierbar für jeden Core.

- Single Application Domain
  Ein gemeinsames ELF-File, Trennung von lokalen Daten und Programmteilen, veröffentlichen von gemeinsam genutzten Daten und Programmteilen zum spätestmöglichen Zeitpunkt.

  Idealerweise erfolgt die Verbindung zwischen den parallelen Abläufen erst im Linkerlauf.

  Hier kann auch die Aufteilung der Speicherbereiche erfolgen, wenn getrennte Speichermodule zur Verfügung stehen.

Aufwand und Nutzen

Multicore Applikationen bedeuten – wie zuvor beschrieben – einen Mehraufwand in der Designphase, in der Implementierungsphase und in der Testphase. Die verwendeten Mechanismen für Zugriffsschutz, Nachrichtenaustausch und Synchronisation müssen implementiert und getestet werden. Zwei Cores bedeuten nicht, dass die doppelte Arbeit in der gleichen Zeit erledigt werden kann. Wie in der Zusammenarbeit eines Teams, gibt es auch hier einen gewissen Reibungsverlust, der durch die vorher aufgeführten Mechanismen entsteht. Lohnt sich der Einsatz von Multicore Systemen trotz allem?

Die Anzahl der benötigten Bausteine wird reduziert und das führt zu einer Kostenersparnis. Da viele Applikationen immer aufwendiger werden, ist der Einsatz von Multicore Systemen nicht aufzuhalten (Leistungssteigerung bei gleichzeitiger Kostenersparnis). Die steigende Nachfrage und die Vielzahl der auf dem Markt verfügbaren Bausteine sprechen für sich.

Zusammenfassung

Die nächsten Jahre werden in Hinblick auf Multicore im Embedded Umfeld sicher spannend. Immer mehr Singlecore – Multitasking Applikationen werden umgestellt auf Multicore (und Multitasking). Entscheidend wird sein, wie die Entwickler die vorhandenen Tools und Betriebssysteme einsetzen.

Ohne sich vorher Gedanken zu machen, welches Ziel erreicht werden soll, wird der Schritt von Singlecore nach Multicore nicht funktionieren.

Fazit

Vor der Implementierung kommt das Design, das ist ja nichts Neues. Auch dass die Tool-Umgebung eine wichtige Rolle im gesamten Entwicklungsprozess spielt, ist nicht wirklich neu. Aber die Handhabung von globalen Elementen (Variablen, Funktionen), die richtige Nutzung der vorhandenen Ressourcen für Multicore Applikationen und die Synchronisation von Prozessen und Cores wird immer komplexer. Deshalb ist es besonders wichtig, sich mit den Fähigkeiten des eingesetzten Tools auseinanderzusetzen – um diese Fähigkeiten als Hilfsmittel zur Vereinfachung im ersten Entwicklungsschritt, und zur leichteren Änderbarkeit und Erweiterbarkeit vorhandener Systeme auch nutzen zu können.

Autorin

Renate Schultes - kaum jemand kennt die Welt der Mikrocontroller so gut wie sie. Als Trainerin, Beraterin und Autorin von Fachbüchern und Publikationen zu 8, 16 und 32-Bit Architekturen hat sie schon vielen Entwicklern den Weg in die praktische Anwendung erleichtert. Sie hat eine große „Fangemeinde“ an Entwicklern, die sich darauf freuen, wenn sie wieder mit ihr gemeinsam in die Welt einer neuen Prozessorfamilie eintauchen dürfen.