Embedded Software Engineering Kongress

ESE Kongress 2016

Vortragsskript:

Ressourcen-Management für die Multicore-Mikrocontroller-Auswahl

Ingo Pohle, MicroConsult GmbH

Die Anforderungen an Mikrocontroller-gesteuerte Systeme steigen von Jahr zu Jahr. Dies hängt damit zusammen, dass sie z.B. mehr Komfort, erweiterte Funktionalität und höhere Sicherheit für den Anwender bringen sollen. Die Rechenkerne, die die erweiterten und neuen Aufgaben bearbeiten, benötigen also immer mehr Rechenleistung.

Gleichzeitig ist man mit Problem konfrontiert, dass diese Erweiterungen den Stromverbrauch in den Steuerungen erhöhen. Dies trifft insbesondere bei batteriebetriebenen Geräten zu. In der Entwicklung von Mobiltelefonen ist sprichwörtlich das Ende der Fahnenstange erreicht. Die für den Betrieb benötigten Akkus können nicht immer mehr Strom liefern, ohne dass dies einen Einfluss auf Größe und Gewicht hat.

Mehr MIPs pro Watt
Bei Notebooks hat man den Weg beschritten, die Steigerung der Rechenleistung durch Multicore-CPUs bei moderater Taktung zu bewerkstelligen. Dieser Lösungsansatz wurde gewählt, damit der Stromverbrauch nicht zu stark durch die extrem hohe Taktung in Singlecore Bausteinen steigt. Bei der Entwicklung der Mikrocontroller haben die Multicore-Architekturen mit ein paar Jahren Verzögerung Einzug gehalten. Hier gibt es die gleichen Forderungen: mehr Rechenleistung ohne signifikante Steigerung der Stromaufnahme.

Die Automobilindustrie als Vorreiter beim Einsatz der Multicore-Mikrocontroller
Bei der Entwicklung von Steuerungssystemen in der Automobiltechnik sind zwei treibende Aspekte zu nennen:

- Einhaltung gesetzlicher Forderungen bezüglich Verbrauch von Treibstoff bzw. Schadstoffausstoß
- Forderung nach höherer Leistungsfähigkeit bezüglich der Kommunikations- und Entertainment-Systeme im Auto und ganz besonders das autonome Fahren

Die Bausteinhersteller von Mikrocontrollern bieten verschiedenste neue Multicore-Architekturen an:

- Mehrere Rechenkerne vom gleichen CPU-Typ
  - homogene Multicore-Mikrocontroller
- Verschiedene spezialisierte Rechenkerne
  - heterogene Multicore-Mikrocontroller
- Mehrere gleichartige und spezialisierte Rechenkerne
  - heterogene Multicore-Mikrocontroller

Software-Migration von Singlecore nach Multicore
Im ersten Schritt sieht die Aufgabe der Software-Migration von Singlecore nach Multicore ganz einfach aus: Lassen wir einfach die vorhandene Software auf mehreren Cores parallel abarbeiten. Ein Core entspricht der Rechenleistung x, dann haben wir bei einem Dual-Core also zweimal x. Effektiv ergibt sich aber nur eine Leistungssteigerung von ca. 50 %, oder in einigen Fällen sogar noch weniger. Woran liegt das?

Die Architektur von Singlecore-Software ist in der Regel nicht für die erfolgreiche Ausführung auf einem Multicore-System geeignet. Folgende Herausforderungen lassen sich hier ausmachen:

- Singlecore-Software wird sequenziell ausgeführt; Multicore-Software benötigt für das gleiche Resultat Synchronisationspunkte.
- Ein Core greift immer nur auf eine Ressource zu (z.B. einen Datenspeicher, ein Peripheriemodul, etc.); mehrere Cores können nur mit zusätzlicher Koordination gleichzeitig auf dieselbe Ressource zugreifen, damit inkonsistente Datenzustände vermieden werden.
- Ein Rechenkern benutzt ein Bussystem, mehrere parallel arbeitende Rechenkerne können sich bei Buszugriffen in die Quere kommen und damit hohe Zugriffsverzögerungen auf Ressourcen hervorrufen.

Die größte Herausforderung liegt darin, die Verwendung gemeinsamer Ressourcen in der Software zu koordinieren und zu überwachen. Eine Lösung hierfür ist das Ressourcen-Management für die Entwicklung von Multicore-Softwarekomponenten. Ein erfolgreicher Umstieg von Singlecore- auf Multicore-Systeme basiert auf der richtigen Auswahl und Anwendung des einzusetzenden Multicore-Mikrocontrollers.

Schritte zur erfolgreichen Multicore-Mikrocontroller-Auswahl:

- Festlegung der Methodik/ Vorgehensweise
- Anforderungen zur Ressourcen-Identifikation
- Safety- und Security-Aspekte
- Art des Cores, Rechenleistung und Anzahl der Cores
- Datenspeicher/ Programmspeicher
- Interrupt- und DMA-Controller
- Peripherien und Ports
- Bussystem(e)

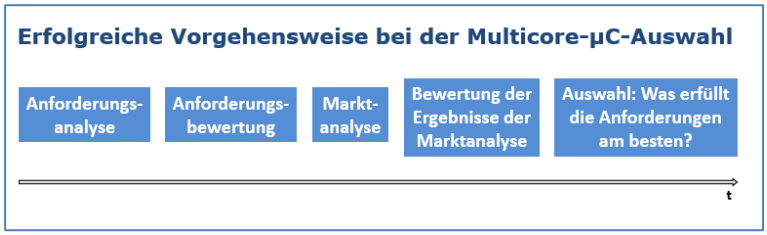

Bild 1: Erfolgreiche Vorgehensweise bei der Multicore-Mikrocontroller-Auswahl

Zunächst gilt es, eine **Anforderungsanalyse** für die **Software** und **Hardware** durchzuführen und die Ergebnisse zu dokumentieren. Im Anschluss sollte eine **Bewertung** der **aufgenommenen Anforderungen** nach der Wichtigkeit für das Projekt erfolgen:

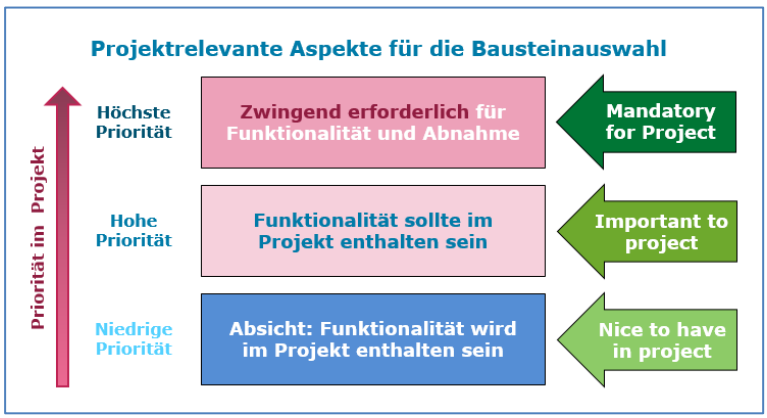

Bild 2: Zuweisung der Anforderungen zu Prioritätsklassen für das Projekt

Die **Prioritätsklassen** bestimmen, ob die Anforderung im Projekt enthalten

- **sein muss** (verpflichtend, höchste Priorität),
- **sein sollte** (wichtig für das Projekt, hohe Priorität) oder
- **sein wird** (wäre gut, wenn es enthalten wäre, niedrige Priorität).

Bevor nun eine **Marktanalyse** erfolgen kann, gilt es, z.B. für die Komponente eines Mikrocontrollers in der Systemsteuerung die genaueren Aspekte, die an den Steuerungsrechner gestellt werden, zu definieren.

Diese Angaben sind dann die Basis für die Mikrocontroller-Auswahl:

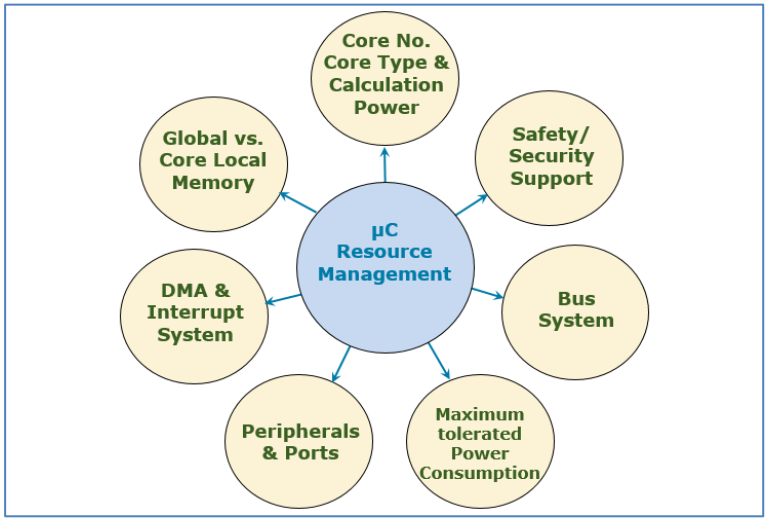

Bild 2: Benötigte Ressourcen im Projekt bestimmen die Mikrocontroller-Auswahl

So ergibt sich beispielsweise für eine **geforderte** bzw. **Rechenkapazität** (in MIPS) und der maximal tolerierbaren Stromaufnahme die Forderung nach einer Multicore-Architektur.

Für verschiedene Applikationen werden Arithmetik-Unterstützung wie Floating-Point oder DSP-Support gefordert. Beispiele hierfür sind komplexe Motorsteuerungen oder Bild- und Ton-Verarbeitung. In den meisten Fällen können die Forderungen nur durch spezifische Core-Architekturen erfüllt werden. Als Mikrocontroller kommen hier fast ausschließlich heterogen Rechenkerne zum Einsatz.

Kommen Anforderungen zu **Safety** und **Security** ins Spiel, ist eine umfassende Anforderungsanalyse für diese Aspekte sehr wichtig. So wird z.B. zum Erreichen des **ASIL-Levels C** gefordert, dass die Ergebnisse des Main-Core-Programms mit den Ergebnissen eines im Checker-Core zeitversetzt bearbeiteten Programms verglichen werden. Dies ist notwendig, damit eine eventuell während der Programmverarbeitung im Main-Core auftretende Störung erkannt werden kann. Als Antwort auf erkannte Fehler muss das System imstande sein, geeignete Fehlerantworten zu generieren. So muss in speziellen Fällen eine Fehlerantwort ohne Beteiligung von Software das System in einen sicheren Zustand versetzen können. Hier wird eine Safety-Hardware (Safety Management Unit) benötigt, die für jeden erkannten Fehler automatisch eine vom Anwender voreingestellte Fehlerantwort generiert (z.B. Exception/ Trap Routine, Reset, CPU Idle State oder ein externes Fehlersignal).

Bild 3: Arbeitsschritte in der Projektentwicklung für Safety-relevante Systeme

Für die Ermittlung der Anforderungen, die bei **Safety-relevanten Applikationen** berücksichtigt werden müssen, ist zunächst eine Analyse der möglichen Ursachen und Risiken durchzuführen.

Der geforderte Safety-Level bestimmt dann z.B., welche Safety-Hardware in einem Mikrocontroller enthalten sein muss, damit das System den zu erreichenden Grad der Sicherheit sicherstellt.

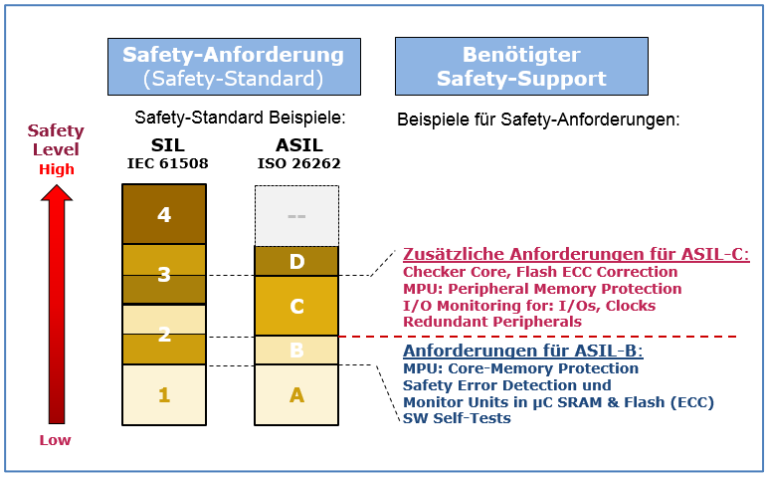

Bild 4: Beispiel für Safety-Anforderungen für ASIL-B und ASIL-C

Eine ähnliche Vorgehensweise gilt für die Security-Thematik:

Bild 5: Security – Sicherheitssysteme für Steuerungen

Die **Security-Ziele** (**Security Goals**) und **Angriffspotentiale**, die ein Angreifer auf das System hat, sind zu bestimmen und zu untersuchen. Ferner ist zu bewerten, ob diese möglichen Angriffe Auswirkungen auf die Systemsicherheit haben oder ob die Privatsphäre des Anwenders geschützt werden muss, damit sie nicht verletzt werden kann.

Beispiele hierfür sind die folgenden Aspekte:

- Muss das System Schutzmechanismen für die Software der Applikation gegen Manipulation enthalten?
- Gibt es die Möglichkeit, im Software-Bootprozess nicht-autorisierte Zugriffe über externe Interfaces zu erkennen und zu verhindern?
- Gibt es einen Hardware-Support für passwortgeschützte Kommunikation und die Möglichkeit, Viren in der Kommunikation zu erkennen und unschädlich zu machen?

Das Ergebnis dieser Analyse bestimmt, welcher Security-Support im System benötigt wird. Für die Auswahl des Mikrocontrollers, der für die Erfüllung der Security-Ziele verantwortlich ist, gilt es zu untersuchen, ob die dazu notwendigen Hardware-Voraussetzungen erfüllt sind:

- Secure Software Boot und Crypto Bootloader für sicheren Software-Start und Flash-Updates
- Flash-Protection-Mechanismen
- Zugriffsgeschützter Safety Controller (von der Applikation getrennte (private) Flash- und SRAM-Bereiche)
- Support passwortgeschützter Kommunikation
- etc.

Welche **Art von Cores** wird benötigt, und welche **Rechenleistung** sollen diese bei einer **festgelegten maximalen Stromaufnahme** liefern können? Gibt es den benötigten Baustein innerhalb einer Bausteinfamilie mit unterschiedlichen Core-Implementierungen und verschiedenen Gehäusen (Port-Pin-Anzahl)? Diese entscheidenden Aspekte sind wichtig für künftig zu realisierende Projekte.

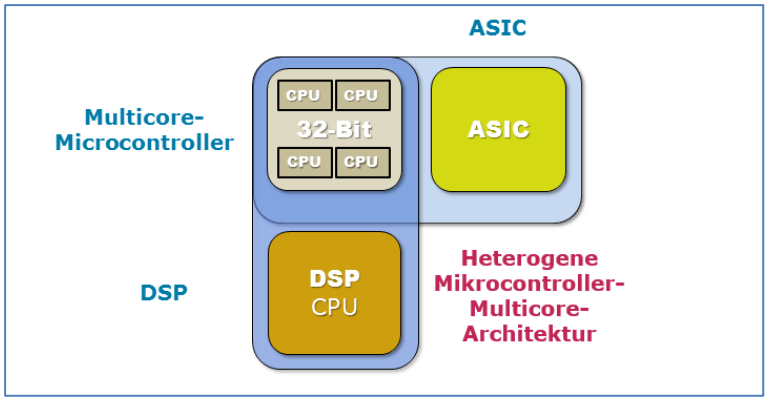

Bild 6: Heterogene Baustein-Architekturen

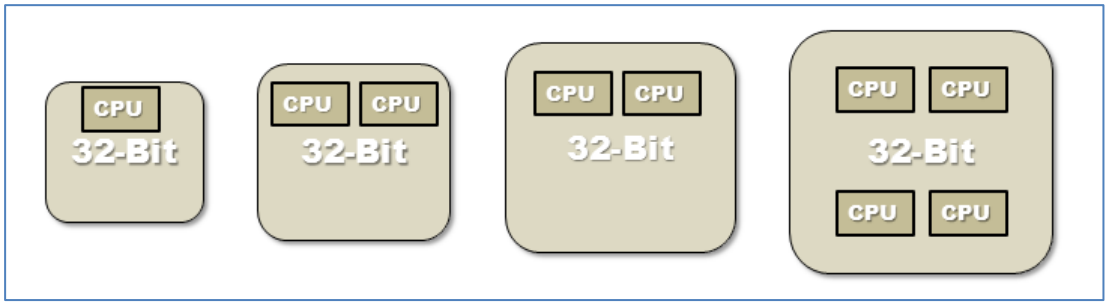

Bild 7: Singlecore-/Multicore-Bausteinfamilienkonzept

Ein Pin – mehrere Peripherie-Funktionen
Die ausreichende Pin-Anzahl ist neben der zur Verfügung stehenden Rechenkapazität ein entscheidender Projekt-Erfolgsfaktor. Alle Mikrocontroller haben gemeinsam eine Einschränkung: Es gibt immer mehr Peripheriefunktionen als verfügbare Pins. Also muss untersucht werden, ob die zwingend benötigten Peripherie-Module alle auch über die erforderlichen Ports-Pins verfügen:

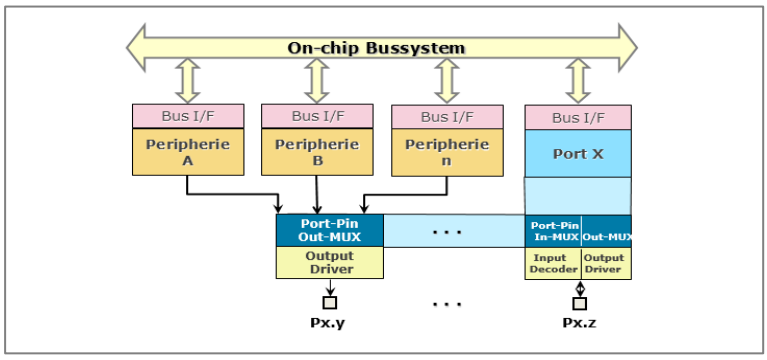

Bild 8: Mehrere Peripherie-Funktionen – nur ein verfügbarer Pin

Performance-Steigerung durch Core-private Speicher

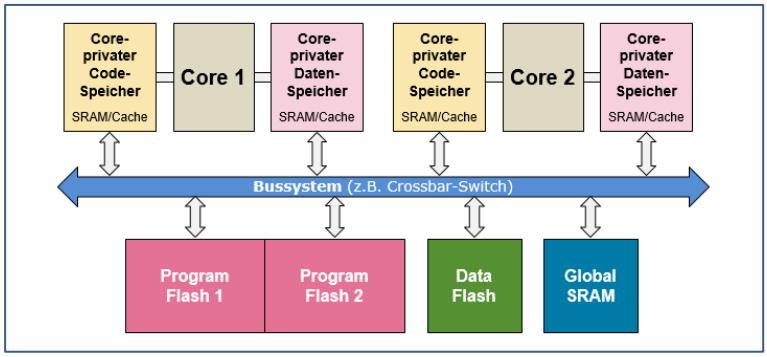

Bild 9: Core-private und System-globale Speicher

Die Multicore-Mikrocontroller-Architekturen unterscheiden sich maßgeblich bei der Speicher-Implementierung:

- viel globaler Speicher und wenig Core-lokaler Speicher
- wenig globaler Speicher und viel Core-lokaler Speicher

Wird sehr hohe Rechenperformance benötigt, kann dies durch viele Core-lokale Speicher und eine hohe Core-Taktung erreicht werden. Ferner können die Programme in lokalen Speichern besser vor unberechtigtem Zugriff anderer Cores gesichert werden.

Das On-chip-Bussystem - eine Performance-Bremse?
Bussysteme, die nur serielle Kommunikation erlauben, können sich schnell zu einem „Bottleneck“ in der Applikation entwickeln, wenn viele Daten kommuniziert werden müssen. So kann es z.B. bei der Anwendung von Ethernet-Modulen vorkommen, dass diese Einheiten in der Regel nicht über genug privaten Speicher verfügen. Besser sind die Implementierungen von Bus-Matrix-Systemen, wie z.B. einem Crossbar-Switch. Diese Implementierung wird heute als Bus-Interface für die Verbindung von Cores zu den globalen Ressourcen eingesetzt. Meist sind alle Peripherie-Module über ein gemeinsames oder zwei serielle Bussysteme angeschlossen. Hier lohnt sich eine Performance-Analyse, damit es im realen System nicht zu unerwarteten Kommunikationsverzögerungen kommt.

Fazit:
Für einen erfolgreichen Umstieg einer Hard-Realtime-Applikation von Singlecore nach Multicore ist ein Ressourcen-Management bei der Mikrocontroller-Auswahl sehr ratsam.

Weiterführende Informationen - Technisches Training:
1. Anforderungsanalyse
2. Software-Architektur
3. Multicore-Mikrocontroller-Grundlagen
4. Safety - Funktionale Sicherheit
5. Betriebssystem-Grundlagen
6. Embedded C und Embedded C++

Autor:

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

Kontakt: +49 (0)89 450617-71
info@microconsult.de
www.microconsult.de