© Infineon Technologies

Automatisiertes Fahren mit beherrschbarer Systemkomplexität

Automatisiertes Fahren wird den Straßenverkehr bequemer und sicherer machen. Hemmnisse, die gegen eine schnelle Einführung sprechen, sind weniger Verfügbarkeit und Kosten von Sensoren, Aktuatoren und schnellen Rechnern, sondern eher die Entwicklung von nachweislich verlässlicher Systemarchitektur und Software.

Komplexität lässt sich beherrschen durch Modularisierung mit klaren Schnittstellen, durch Wiederverwendung und durch evolutionäre Weiterentwicklung. Die letzten beiden Punkte erfordern für das System und die Software eine stabile und breit einsetzbare Hardware-Plattform, wie sie eine gut abgestimmte Mikrocontroller-Familie wie AURIX bereitstellt.

Firmeninterne Richtlinien und Industriestandards wie ISO 26262 sind Hilfsmittel, um ein funktional sicheres System zu entwerfen. Sie entlassen den Entwicklungsingenieur jedoch nicht aus seiner Verantwortung. Für hochkomplexe Systeme ist zum Beispiel das V-Modell zu simpel, da es Neuentwick-

lungen favorisiert und dabei kühn annimmt, dass auf jeder Abstraktionsebene Richtigkeit und Vollständigkeit erreicht werden können. Andererseits muss auch die Wiederverwendung bewährter Komponenten rigoros hinterfragt werden. Der 1996 gescheiterte Jungfernflug der Rakete „Ariane 5" ist hier ein warnendes Beispiel. Dieser Beitrag gibt einen Überblick über die Grundarchitekturen der verschiedenen Steuerungen für ein automatisiertes Fahrzeug und die dabei geltenden Randbedingungen.

Von Fail-Safe zu Fail-Operational

Fail-Safe bedeutet, dass ein System erkennen kann, wenn es nicht mehr funktioniert, und dann in einen passiven sicheren Zustand geht. Dieses Verhalten ist akzeptabel für die Lenkkraftunterstützung eines Fahrers, aber nicht für eine automatische Lenkung. Eine solche Lenkung muss als Fail-Operational ausgelegt sein Die Fehlererkennung für Fail-Safe und Fail-Operational ist eine sehr große Herausforderung. Folgendes ist derzeit Stand der Technik auf der Hardware-Ebene: Speicher und Busse haben Error Correction Codes (ECC), die Multi-Bit-Fehler erkennen, und Prozessorkerne besitzen redundante Rechenwerke (Lockstep). Daneben unterstützt die Hardware die Erkennung von Fehlern in der Software, z. B. mit adressbasierten Speicherschutzvorrichtungen in gemeinsam genutzten Speichern und Peripherie-Modulen und auch direkt in den Prozessorkernen (MPU). Es gibt noch viele andere Maßnahmen, die Fehler erkennen und teilweise sogar korrigieren können. Letzteres ist lokal für Bitfehler in Speichern möglich (ECC), bei anderen Alarmen in den Hardware- und Software-Überwachungseinheiten wird die Fehlerreaktion fast immer ein Reset und Neustart sein. Der Grund für den Neustart: Kennt man die genaue systematische Fehlerursache, beseitigt man diese in der Entwicklungsphase, anstatt komplexe und damit auch wieder anfällige Fehlerbehandlungsstrategien zu entwickeln.

Der Fehlerbehandlungscode ist häufig der am schlechtesten getestete Code, da er oft schwer zu stimulieren ist und zu beliebigen Zeitpunkten aktiviert werden kann. Wenn das Fehlersymptom nicht eindeutig ist, könnte er auch in Situationen aktiviert werden, für die er gar nicht vorgesehen ist, und damit wie ein falsches Medikament unkontrollierbaren Schaden verursachen.

Bei traditionellen Mikrocontrollern mit Embedded-Flash liegt die Totzeit durch einen Reset im Bereich von bis zu wenigen 10 Millisekunden – und ist damit akzeptabel für langsame physikalische Systeme wie ein Fahrzeug. Dieses gut verstandene und kostenoptimierte Grundelement eines Fail-Safe-Steuerungskanals von den Sensoren bis zu den Aktoren lässt sich für ein Fail-Operational-System wiederverwenden (Bild 1).

Im einfachsten Fall wird der Steuerungskanal komplett verdoppelt, mit eigenen Sensoren, Aktuatoren, deren Verkabelung und insbesondere auch einem völlig unabhängigem Stromversorgungs-, Clock- und Resetsystem. Eine gewisse Asymmetrie der Kanäle aus Kostengründen ist möglich, da nicht alle Funktionen sicherheitsrelevant sind, und diese deshalb ohne Redundanz gerechnet werden können.

Ein Lenkungssystem hat eine klare und gut überprüfbare Aufgabe. Es muss einen vorgegebenen Lenkwinkel durch die Ansteuerung der Motoren umsetzen. Das erfolgt durch überprüfbare kausale Abläufe, was eine einfache Fehlererkennung erlaubt. Die beim automatisierten Fahren benötigte Steuerung, die aus Kamera- und Radardaten den Lenkwinkel berechnet, hat hier eine weit komplexere Aufgabe.

Einfache und kostenoptimierte Fail-Operational-Systeme

Für solche komplexen Systeme kann eine hierarchische Architektur mit einer Fail-Operational ausgelegten Überprüfungs- und Rückfallebene eingesetzt werden. Einen ähnlichen Ansatz hat auch die Evolution mit Großhirn, Kleinhirn, zentralem und vegetativem Nervensystem entwickelt. Die meisten Menschen können zum Beispiel nicht durch bewusstes Anhalten des Atems ohnmächtig werden. Und selbst wenn, dann setzt mit der Ohnmacht die Atmung wieder ein.

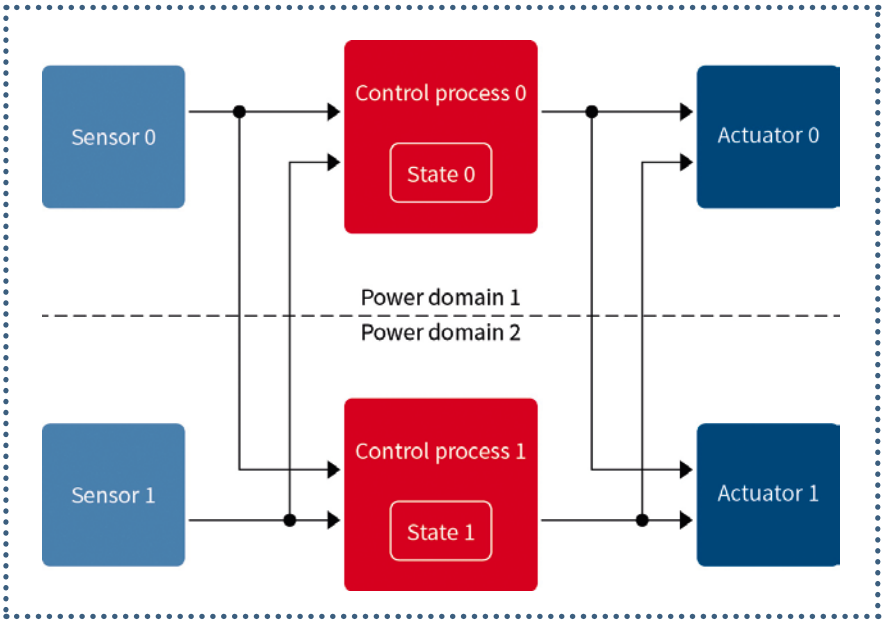

Bild 1: Fail-Operational-Grundarchitektur mit zwei unabhängigen Fail-Silent-Kanälen. Dadurch wird kein Entscheider (Voter) benötigt, der bei Ausfall zum Ausfall des Gesamtsystems führt. (© Infineon Technologies)

Die Grundidee ist, dass die Rückfallebene ein vereinfachtes und konservatives Zustandsmodell und Steuerungsregeln hat, mit dem die Entscheidung des Hauptrechners überprüft werden kann. Falls diese nicht plausibel erscheint (z. B. Hindernisse im Fahrkanal beim Überholvorgang), übernimmt die Rückfallebene mit einer konservativen »

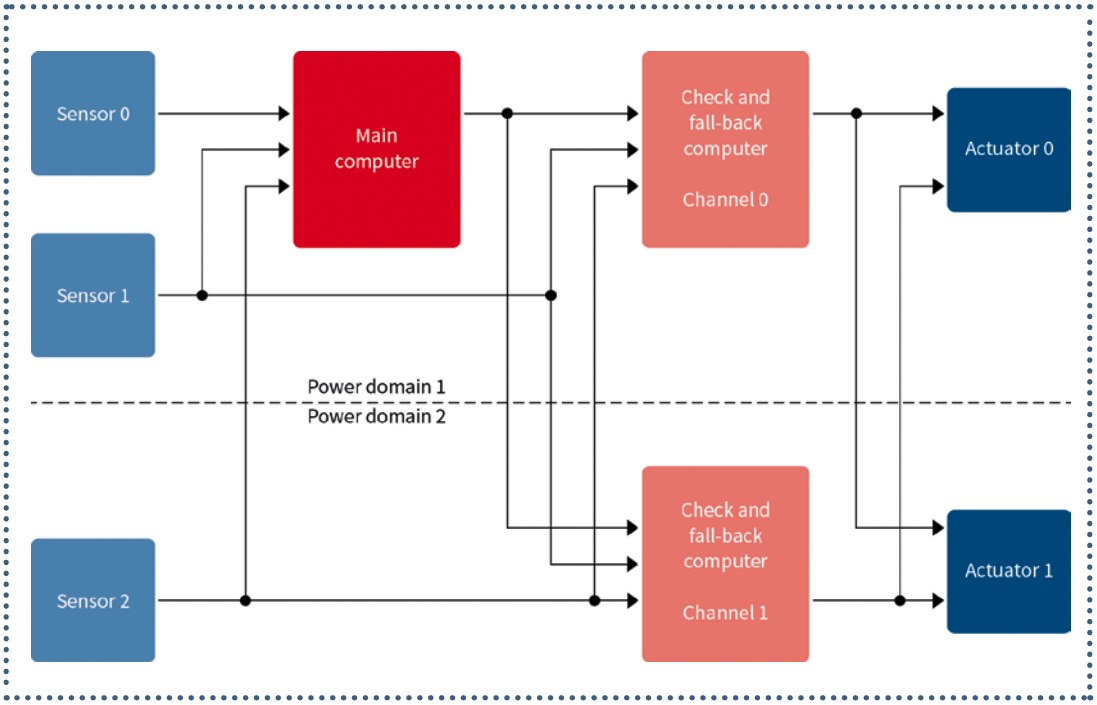

Bild 2: Hierarchische Steuerung mit Fail-Operational Überprüfungs- und Rückfallebene, die Ausfallsicherheit mit reduzierten Kosten und Komplexität verbindet. (© Infineon Technologies)

Fahrstrategie (z. B. wir bleiben hinter dem Lkw). Die Rückfallebene kann dann den Hauptrechner wieder neu starten, und wenn dieser plausible Ergebnisse liefert, übernimmt er wieder die Steuerung (Bild 2).

Dieser Ansatz hat mehrere Vorteile: Zum einen muss nur die Rückfallebene den höchsten Automotive-Sicherheitsanforderungen (ASIL-D) genügen und fail-operational sein. Das spart nicht nur Kosten und Verlustleistung, sondern es ist auch für leistungsfähige, PC-artige Prozessorarchitekturen nur schwer möglich, ASIL-D zu unterstützen. Die Software des Hauptrechners ist auch so komplex, dass sie in der Praxis kaum vollständig nach ASIL-D entwickelt werden kann.

Safety, Security, Software und sonstige Schmerzen

Nach Einschätzung der meisten Experten für funktionale Sicherheit werden die Risiken von Hardware-Fehlern (z. B. Soft-Errors durch radioaktive Strahlung) durch konservatives Addieren der Fehlerwahrscheinlichkeiten systematisch überschätzt. Gleichzeitig werden die Risiken der Software unterschätzt, da es hier keine belastbaren Fehlermodelle gibt. Das Problem wird noch größer, wenn die Software nicht auf einer Hardware läuft, die verlässlich Hardware-Fehler erkennt, und wenn Datensicherheit (Security) und harte Echtzeitanforderungen hinzukommen.

Prozessorkern für harte Echtzeit (z. B. TriCore)	Applikationsprozessorkern (z. B. ARM A-Series, Intel)
Optimiert für harte Echtzeit: kleine Chipfläche und Verlustleistung	Optimiert für Rechenleistung: lange Pipeline, spekulative Ausführung
Optimiert für Safety	Kein Lockstep ▪ kein ASIL-D
Embedded-Flash ▪ sehr schnelle Bootzeit	Speicherhierarchie mit externen DRAM ▪ sehr großer Speicher ▪ Effizienz durch Cache-Hierarchie
MPU ▪ Safety und harte Echtzeit	MMU ▪ Unterstützung von Linux, etc.
Volle Trace-Unterstützung	Begrenzter Trace-Support, oft auch kein Trace

Tabelle 1: Vergleich von Prozessorkernen für harte Echtzeit mit Applikationsprozessorkernen. (© Infineon Technologies)

Alles, was die Software-Komplexität und damit die Fehleranfälligkeit erhöht, sollte vermieden werden. Die Datensicherheit (Security) muss z. B. sehr stark an den Außengrenzen des Systems sein (Gateway, Software-Updates). Innerhalb des Systems muss, basierend auf realistischen Angriffsszenarien, ein vernünftiger Kompromiss zwischen Security, Safety und Komplexität gefunden werden.

Ein großes Thema der letzten Jahre war die Abbildung von harter Echtzeit-Software auf Multicore-Mikrocontroller. Der schmerzhafteste Schritt – und naturgemäß meistens der erste – war die Verteilung einer existierenden Single-Core-Software auf einen Multicore-Prozessor. Hier gibt es aber eine deutliche Entspannung, da die Anwender Multicore-Erfahrung und -Methodik gesammelt haben. Neue Software wird dann nach diesen Richtlinien entwickelt und diese Entwicklung wird durch entsprechende Tools, z. B. zur Multicore-Timing-Analyse, immer besser unterstützt.

Das Ende von Moore's Law

In den letzten 50 Jahren war das Grundgesetz der Halbleiterindustrie, dass in der nächsten Prozesstechnologie alles deutlich besser wird: Rechenleistung,

Stromverbrauch und Kosten. Das ist unterhalb von 28-nm-Strukturbreite nicht mehr der Fall. Die Integrationsdichte steigt zwar weiter an, aber die Kosten pro Transistor sinken nicht mehr entsprechend. Auch verschlechtern sich die Analogeigenschaften. Und es gibt keine 5-Volt-IO mehr und kein Non-Volatile-Memory, das geringe Kosten mit Automotive-Tauglichkeit verbindet.

Generell steigen mit immer feineren Strukturbreiten auch die Kosten pro Chipdesign exponenziell an und lassen sich nur bei entsprechenden Stückzahlen rechtfertigen. Da die Anzahl der jährlich produzierten Autos nur moderat wächst, lassen sich diese Stückzahlen nur erreichen, wenn der gleiche Chip in mehreren Anwendungen eingesetzt werden kann, z. B. in der Motorsteuerung und der Bremse. Das führt dazu, dass solche Chips signifikante Flächenanteile für Peripheriemodule haben, die nur in bestimmten Anwendungen benötigt werden.

Andererseits sind solche Mikrocontroller für Toolhersteller kommerziell attraktiv; mit entsprechend positiver Auswirkung auf die Qualität der Tools, die von vielen verschiedenen Kunden für verschiedene Designs eingesetzt werden.

Auswirkungen auf den Mikrocontroller

Ein klassischer Mikrocontroller hat Embedded-Flash und kann mit sehr wenig externer Zusatzbeschaltung betrieben werden. Auch unter Automotive-Bedingungen (Temperaturbereich, tolerante Stromversorgung, höchste Ausfallsicherheit über die gesamte Lebenszeit) gibt es inzwischen sehr leistungsfähige Multicore-Mikrocontroller mit Taktfrequenz von bis zu 300 MHz, wie beispielsweise die AURIX-Familie.

Bei noch höheren Anforderungen bei Rechenleistung und insbesondere Speicher (10 bis 100 MByte) können dann „PC-artige“ Architekturen mit Applikationsprozessoren eingesetzt werden. Tabelle 1 vergleicht beide Typen von Prozessorkernen. Beide Typen von Prozessorkernen haben ihre Vor- und Nachteile, die oft durch die Gesamt-Chip-Architektur weiter verstärkt werden. Mit einer MMU (Memory Management Unit), einer Multicore-Cache-Hierarchie mit Datenkohärenz und externem DRAM lässt sich keine harte Echtzeit garantieren. Hat man allerdings eine verlässliche Rückfallebene, dann ist diese Eigenschaft kein Ausschlusskriterium mehr, um einen solchen Chip in einer harten Echtzeitumgebung einzusetzen.

Fazit

Die Komplexität auf allen Ebenen ist der begrenzende Faktor für die schnelle Einführung von automatisiertem Fahren. In hochkomplexen Systemen lässt sich die notwendige funktionale Sicherheit und Zuverlässigkeit nur durch den Einsatz von bewährten und verstandenen Komponenten und Architekturen erreichen.

Immer leistungsfähigere Multicore-Mikrocontroller bleiben das Rückgrat für Fail-Operational-Systeme – wegen ihrer Zuverlässigkeit, ihrer Unterstützung für höchste Safety-Anforderungen und ihren Echtzeiteigenschaften, inklusive schnellem Reset.

Für die komplexesten Steuerungsfunktionen zeichnet sich als bester Kompromiss zwischen Leistung, Kosten und Komplexität eine hierarchische Systemarchitektur ab. Die besteht aus einem hochleistungsfähigen Hauptrechner und einer als fail-operational ausgelegten Entscheidungs- und Rückfallebene, die im Fehlerfall die Steuerung übernehmen kann, bis der Hauptrechner wieder funktionsfähig oder ein sicherer Zustand erreicht ist. ■ (oe)

» **www.infineon.com**

Dr. Albrecht Mayer ist Systemarchitekt für die nächste Generation von AURIX-Multicore-Mikrocontrollern im Unternehmensbereich Automotive bei Infineon Technologies.