Kapitel 1

Einleitung

In vielen Bereichen unseres täglichen Lebens werden wir mit Rechnern konfrontiert. Dabei werden Rechner nicht nur zur Unterhaltung oder Erstellung von Schriftsätzen eingesetzt, sondern finden auch Anwendung in sicherheitsrelevanten Bereichen. Zu diesen Bereichen zählt neben den Bankenanwendungen (z.B. Banktransaktionen bei elektronischen Überweisungen), der Automatisierungstechnik (z.B. Steuerung von Maschinen), der Bahntechnik (z.B. Signalsteuerung) und der Luft- und Raumfahrttechnik (z.B. Fly-By-Wire) seit ein paar Jahren vor allem auch die Automobiltechnik.

Gerade im Bereich der Automobiltechnik sind in den letzten Jahren immer neue sicherheitsrelevante Funktionen hinzugekommen, die durch Rechner gesteuert werden. Dazu zählen Airbagsysteme, das Antiblockiersystem, das Elektronische Stabilitätsprogramm und zukünftig X-by-Wire Anwendungen oder Verbundsysteme zur Fahrdynamikregelung und Fahrzeugführung. Diese Systeme haben gemeinsam, dass sie bei Auftreten von Fehlern unter Umständen ihre Funktion nicht mehr erfüllen oder fehlerhafte Aktionen durchführen können, die sogar Leben gefährden. Im Gegensatz zu anderen sicherheitsrelevanten Anwendungen ist der Automobilmarkt ein sehr hochvolumiger und preissensitiver Markt. Dies hat zur Folge, dass die gängigen Lösungen zur Fehlerentdeckung in einem Rechnersystem, zum Beispiel eine vollständige Duplizierung der verwendeten Prozessoren, vom Kunden aus Kostengründen nicht akzeptiert werden.

Einerseits steigt durch das vermehrte Aufkommen neuer sicherheitsrelevanter Anwendungen deren Berücksichtigung in den gesetzlichen Anforderungen, die in Normen, z.B. in der europäischen Norm EN61508, festgelegt sind. Andererseits nimmt auch die Anzahl transienter Fehler zu. Diese Zunahme ist eine Folge der Fortschritte in der Halbleitertechnologie. Durch den Einsatz neuer Halbleitertechnologien nehmen die Strukturbreiten der Halbleiterchips immer mehr ab [1], wodurch wiederum die Versorgungsspannungen der Prozessoren sinken. Eine Folge davon ist, dass der Signal-Rausch-Abstand der Logikschaltung abnimmt. Die Schaltungen werden dadurch anfälliger gegen transiente Fehler, was einen Anstieg dieser Fehler zur Folge hat [2, 3].

Um diese zwei gegensätzlichen Probleme lösen zu können, müssen die im Automobilbereich eingesetzten Prozessoren mit Fehlerentdeckungsmechanismen ausgestattet werden. Diese müssen auf der einen Seite eine hohe Fehlerentdeckungsrate bezüglich permanenter und transienter Fehler aufweisen und auf der anderen Seite sollen die Fehlerentdeckungsmechanismen möglichst kostengünstig implementierbar sein.

Ziel dieser Arbeit ist es, die Möglichkeiten der Fehlerentdeckung in einem Prozessor aufzu-

zeigen und speziell für die Anwendung im Automobilbereich neue Lösungskonzepte zu finden. Die besonderen Anforderungen und Umgebungsbedingen im Automobil müssen dabei berücksichtigt werden. Dabei ist zu beachten, dass die zukünftig verwendeten Prozessorarchitekturen eine Harvard-Architektur sein werden. Der Instruktionsspeicher wird ein Flash-Speicher sein und der Datenspeicher ein SRAM-Speicher. Zudem muss das System in einem weiten Temperaturbereich von -20 bis +80° Celsius betrieben werden können.

Bewertet werden die einzelnen Lösungsvarianten bezüglich ihrer Fehlerentdeckungsrate, ihrer Anfälligkeit gegen Common Cause Fehler, ihrem Flächenmehraufwand und ihrem Entwicklungsaufwand. Wenn möglich, sollen die Probleme und Lösungsmöglichkeiten der Lösungsvarianten zur Wiederherstellung eines sicheren Systemzustandes nach einem Fehlerfall beachtet werden [4, 5].

Im folgenden wird in dieser Arbeit zunächst im Kapitel 2 ein Überblick über den für diese Arbeit relevanten Stand der Technik gegeben; anschließend wird die Sicherheitsanalyse vorgestellt, die als Grundlage für die Analyse der einzelnen Prozessorarchitekturen dient. Kapitel 4 behandelt die neu entwickelten und untersuchten Prozessorarchitekturen. Es erfolgt jedoch zuerst die Vorstellung eines sicheren Speichers, der für die einzelnen Prozessorarchitekturen benötigt wird. In Kapitel 5 werden die Vor- und Nachteile der einzelnen Prozessorarchitekturen verglichen und schließlich endet die Arbeit mit einer Zusammenfassung und einem Ausblick.

Kapitel 2

Stand der Technik

Dieses Kapitel soll einen Einblick geben, wie heutige Prozessoren gegen Fehler abgesichert werden. Um die verschiedenen Maßnahmen miteinander vergleichen zu können, müssen die möglichen Fehlermodelle bekannt sein. Daher erfolgt zuerst eine Einführung in die Fehlermodelle und anschließend werden die Möglichkeiten zur Fehlerentdeckung bei Prozessoren betrachtet.

2.1 Fehlermodell

Es gibt verschiedene Eigenschaften von Fehlern, die bei der Auslegung der Fehlerentdeckungsmechanismen beachtet werden müssen. Die wichtigsten Punkte sind, ob es sich um Einzel- oder Mehrfachfehler, schlafende Fehler oder Common Cause Fehler handelt. Des weiteren wird auch noch nach Design- und Produktionsfehler unterschieden. Diese werden jedoch nicht betrachtet, da der Schwerpunkt dieser Arbeit auf der Entdeckung von Fehlern liegt, die im Betrieb auftreten können.

- **Einfach- oder Mehrfachfehler:** Es ist wichtig zu wissen, ob der Fehlerentdeckungsmechanismus in der Lage sein muss einen oder mehrere Fehler gleichzeitig zu entdecken. Ebenso ist wichtig zu wissen, ob sich die Fehlerursache in der Verfälschung eines oder mehrerer Bits auswirkt. Tritt ein Einfachfehler an einem Fehlerort auf, an dem die Auswirkung durch den verwendeten Fehlererkennungsmechanismus nicht erkannt werden kann, so wird dieser als **Single Point of Failure** bezeichnet. Dies tritt unter anderem auf, wenn ein fehlerhafter Gatterausgang 2 Bits in einem durch ein Parity abgesichertes Datenwort verfälschen kann. Ebenso stellen meist alle nicht redundant ausgeführten Komponenten einen Single Point of Failure dar.

- **Schlafende Fehler:** Schlafende Fehler sind Fehler die physikalisch vorhanden sind, aber sich noch nicht ausgewirkt haben [6]: Dies ist z.B. der Fall, wenn der betroffene Schaltungsteil über einen längeren Zeitraum nicht benutzt wird. Eine Gefahr bei schlafenden Fehlern ist, dass sie sich akkumulieren können. Wenn ein Fehlererkennungsmechanismus zur Entdeckung von maximal einem Fehler ausgelegt ist, aber sich mehrere Fehler über einen längeren Zeitraum akkumulieren, so wird er dadurch wirkungslos. Schlafende Fehler können auch während einer längeren Periode der Nichtbenutzung auftreten. Um eine Fehleranhäufung schlafender Fehler zu vermeiden, ist ein Test beim Systemstart sinnvoll.

- **Common Cause Fehler:** Common Cause Fehler sind Fehler, die eine Ursache haben und dabei mehrere redundante Komponenten gleichartig stören [7]. Diese Fehler können nicht durch einen Mehrheitsentscheid erkannt werden [4, 8, 9]. Als Fehlerursache kommen vor allem Störungen in der Spannungsversorgung, elektromagnetische Störungen (EMV) und Umwelteinflüsse in Frage.

Des weiteren werden Fehler auch nach ihrer zeitlichen Wirkdauer klassifiziert:

- **Permanente Fehler:** Permanente Fehler sind dauerhafte Fehler. Ursache ist meist ein physikalischer Defekt, der durch Alterung oder Umgebungseinflüsse auftreten kann. Permanente Fehler können durch einen Test oder durch räumliche Redundanz erkannt werden, nicht jedoch durch zeitliche Redundanz, das heißt zweimaliges Berechnen eines Ergebnisses. Bei einer Absicherung mit zeitlicher Redundanz würde sich der Fehler zu beiden Zeitpunkten gleichartig auswirken.

- **Transiente Fehler:** Transiente Fehler sind Fehler, die zufällig auftreten und wieder selbstständig verschwinden können. Sie haben im Gegensatz zu permanenten Fehlern nur eine kurze Wirkdauer. Ursachen sind meist externe Umgebungseinflüsse wie EMV, kosmische Strahlung oder Störungen in der Spannungsversorgung. Transiente Fehler können nicht durch einen Test erkannt werden, da sie bei der Testausführung überschrieben werden. Diese Fehler können nur durch Redundanz erkannt werden.

- **intermittierende Fehler:** Diese Fehler weisen ein ähnliches Verhalten wie transiente Fehler auf. Sie treten jedoch nicht nur einmal sondern repetitiv auf.

2.2 Möglichkeiten zur Fehlerentdeckung

Nach DIN V VDE 0801 gibt es zwei Maßnahmen zur Erkennung von Fehlern und somit zwei grundsätzliche Möglichkeiten, einen fehlertoleranten Mikrocontroller zu implementieren: zyklische Tests oder Redundanz. Redundanz gibt es als zeitliche, räumliche oder funktionelle Redundanz [10]. Zeitliche Redundanz ist das mehrmalige nacheinanderfolgende Berechnen eines Ergebnisses auf der gleichen Einheit. Räumliche Redundanz ist die gleichzeitige Berechnung eines Ergebnisses auf mehreren Einheiten und funktionelle Redundanz ist die Berechnung des Ergebnisses auf unterschiedliche Weise. Oftmals wird auch noch der Begriff der hybriden Redundanz verwendet. Hierzu zählt z.B. eine ALU mit Parity Prediction. Dabei wird das Parity sowohl räumlich getrennt vom eigentlichen Ergebnis berechnet, als auch mit einem anderen Algorithmus als das Ergebnis.

Aufgrund der unterschiedlichen Wirkungsmechanismen der einzelnen Fehlererkennungsarten sind sie nicht für alle Fehlerarten gleich wirksam:

- **Testen:** Durch Testen können nur Fehler erkannt werden, die während der Ausführung des Tests aktiv sind. Transiente Fehler, die während der Laufzeit der Applikation auftreten, können somit nicht erkannt werden.

- **zeitliche Redundanz:** Mit zeitlicher Redundanz lassen sich transiente Fehler entdecken, bei denen die Wirkdauer der Störung kleiner als der Zeitabstand zwischen dem mehrmaligen Berechnen ist. Dadurch kann sich die Störung nicht bei beiden Berechnungen gleichartig auswirken. Permanente Fehler lassen sich durch die alleinige Verwendung

einer zeitlichen Redundanz jedoch nicht erkennen, da sich dieser Fehler bei beiden Berechnungsdurchläufen gleichartig auswirkt und somit nicht durch einen Vergleich der beiden Ergebnisse erkannt werden kann.

- **räumliche Redundanz:** Ausgehend davon, dass der Wirkbereich einer Störung kleiner ist als der räumliche Abstand der redundanten Komponenten, lassen sich mit dieser Methode sowohl transiente als auch permanente Fehler erkennen. Es kann jedoch nicht unterschieden werden, ob ein transienter oder permanenter Fehler aufgetreten ist.

- **funktionelle Redundanz:** Mit einer funktionellen Redundanz lassen sich ebenso wie bei der räumlichen Redundanz permanente und transiente Fehler erkennen. Eine Unterscheidung zwischen den beiden Fehlerarten ist ebenfalls nicht möglich. Funktionelle Redundanz ist auch wirksam bei der Entdeckung von Designfehlern. Diese Fehlerart wird hier jedoch nicht näher betrachtet.

2.2.1 Test

Bei einem Test wird der Rechner durch Testmuster in einen definierten Systemzustand gebracht. Dabei wird die Systemantwort auf die Testmuster gemessen und gegen eine zuvor aufgenommene Referenz verglichen. Die Systemantwort muss nicht nur die Ausgangssignale umfassen sondern kann z.B. ebenso der Stromverbrauch sein. Es gibt drei verschiedene Testarten:

- physikalische Tests
- funktionelle Tests
- strukturelle Tests

Tests können zudem sowohl in Hardware als auch in Software ausgeführt werden.

Physikalische Tests: Physikalische Tests zielen auf das Erkennen von physikalischen Defekten und nicht auf das Erkennen von funktionellen Fehlern ab. Es wird entweder die an das Testobjekt angelegte Spannung oder ein vom Testobjekt aufgenommener Strom (IDDQ-Test) gemessen. Diese Tests können sowohl im Ruhezustand, als auch bei einer Aktivität des Testobjektes durchgeführt werden. Der I_{DDQ}-Test wird zum Beispiel in [11, 12, 13, 14] vorgestellt. I_{DDQ}-Tests wurden zwar als On-Line Tests untersucht [15, 16], sind aber im Automobilbereich aus verschiedenen Gründen nicht einsetzbar: Durch das Fortschreiten der Halbleitertechnologie kann die geforderte Messgenauigkeit wegen den geringer werdenden Versorgungsspannungen und den größer werdenden Leckströme aufgrund der höheren Integrationsdichte On-Line nur sehr schwer gewährleistet werden. Ein weiterer Einfluss auf das Messergebnis des I_{DDQ}-Test resultiert aus den schwankenden Umgebungseinflüssen, wie z.B. Temperatur und Luftfeuchtigkeit. Des weiteren muss der I_{DDQ}-Test, um ihn in einer vertretbaren Zeitdauer ausführen zu können, in kleine schnelle Tests partitioniert werden. Das bedeutet, dass Teile der Schaltungen einzeln und parallel testbar sein müssen, was wiederum einen erhöhten Flächenbedarf zur Implementierung der Testschaltung zur Folge hat. Ein weiterer Nachteil der Messung der Leckströme ist, dass die Schaltung durch die analoge Messtechnik eine sogenannte Mixed-IC Schaltung wird und dadurch der Herstellungsprozess der Schaltung aufwendiger wird.

Funktionelle Tests: Beim funktionellen Testen wird die Schaltung speziell auf ihre Funktion getestet. Dies ist beispielsweise bei Speichern sehr effektiv, da es sich hierbei um Komponenten handelt, die nur eine geringe Funktionalität aufweisen. Die Vor- und Nachteile dieser

Methode für Speicher wird explizit noch einmal im Kapitel 4.1 erläutert. Für komplexe Logik ist funktionelles Testen zu aufwendig, da die Testdauer exponentiell mit der Komplexität der Funktionalität ansteigt [17, 18]. Das Aufwendige beim funktionellem Testen ist, dass jeder mögliche Zustandswechsel betrachtet werden muss. Dies ist bei heutigen Prozessoren mit ihrer hohen Komplexität in Kombination mit internen Speicherelementen praktisch nicht durchführbar.

Strukturelle Tests: Bei Prozessoren werden meist strukturelle Tests eingesetzt. Dabei wird nicht versucht, den kompletten Funktionsumfang des Prozessors zu testen, sondern es wird versucht, jeden Gattereingang durch ein Testmuster zu aktivieren. Hierzu können die Testsignale entweder von außen angelegt werden [19], oder die Testmuster werden in der Schaltung intern durch Testmustergeneratoren erzeugt. Aufgrund der quadratisch steigenden Dauer für den Test bei größer werdenden Schaltungen werden heutzutage meist interne Testmustergeneratoren eingesetzt und die Schaltung hierzu in kleine Schaltungsteile partitioniert, den sogenannten „Scan-Chains“ [20, 21, 22, 23].

Da mit Tests keine transienten Fehler erkannt werden können, wird dieses Verfahren zur Fehlerentdeckung hier nicht weiter untersucht. Würden die Schaltungen durch technische Maßnahmen resistent gegen transiente Fehler gemacht, wie z.B. durch eine Reduzierung der α-Partikel im Material [24, 25], durch eine Erhöhung der Knotenkapazität [26] oder einer Verminderung der Ladungssammlung [24, 27], könnte dies in Kombination mit einem Test ausreichend sein um die Sicherheitsanforderung zu erfüllen. Die Resistenz gegen transiente Fehler ist jedoch vom Halbleiterprozess abhängig und muss bei einem Technologie- oder Zulieferwechsel erneut betrachtet werden.

Aufgrund der langen Lebenszeit bei Prozessoren in der Automobiltechnik von teilweise mehr als 10 Jahren – während der auch längere Stillstandsperioden auftreten können, besteht die Gefahr einer Akkumulation von schlafenden Fehlern. Da es sich hierbei um permanente Fehler handelt, kann ein Test beim Systemstart notwendig sein, um diese angehäuften Fehler entdecken zu können, wie es in [6] aufgezeigt wird.

2.2.2 Redundanz auf Prozessorebene

Ein Prozessor, bei dem an der internen Struktur keine Änderungen vorgenommen werden dürfen, kann gegen Fehler entweder durch Testen oder durch Redundanz abgesichert werden. Da beim Testen, wie bereits erwähnt, keine transienten Fehler erkannt werden können, werden in dieser Arbeit die Möglichkeiten der Redundanz auf Prozessorebene betrachtet. Das Ziel ist das Erkennen von Einfachfehlern, wozu eine einfache Redundanz ausreichend ist. Es gibt grundsätzlich folgende Möglichkeiten:

- Duplizierung des kompletten Prozessorsystems, wobei dies erreicht werden kann durch eine
  - zweikanalige Struktur mit diversitärer Hard- und Software
  - zweikanalige Struktur mit diversitärer Hardware und homogener Software
  - zweikanalige Struktur mit homogener Hardware und diversitärer Software
  - zweikanalige Struktur mit homogener Hard- und Software
- Duplizierung des Prozessorkerns und Absicherung der Speicher durch Codierung
- Überwachung des Prozessors durch einen Watchdog

Zweikanalige Struktur: Bei einer zweikanaligen Struktur wird der komplette Prozessor mit seiner Peripherie wie Instruktions- und Datenspeicher dupliziert. Die beiden Prozessoren benutzen keine gemeinsamen Ressourcen; die Ausgangsdaten der beiden Rechner werden miteinander verglichen. Durch den Vergleich dieser Daten können Fehler in einem der beiden räumlich redundanten Prozessoren erkannt werden.

Wenn für diese Implementierung zwei unterschiedliche Rechner verwendet werden und auf diesen unterschiedliche Software mit unterschiedlichen Algorithmen jedoch mit der gleichen Funbktionalität abgearbeitet werden, wird von einer ***zweikanaligen Struktur mit diversitärer Hard- und Software*** gesprochen. Solch eine Implementierung wird meist in sehr sicherheitsrelevanten Anwendungen eingesetzt, bei denen im Fehlerfall ein hoher Sach- oder Personenschaden auftreten kann. Dazu zählen unter anderem die Luft- und Raumfahrttechnik [28]. Um eine strikte Diversität der Implementierung zu erreichen, müssen die Prozessoren einen unterschiedlichen Befehlssatz aufweisen, in unterschiedlichen Fabriken gefertigt werden und von unterschiedlichen Herstellern sein. Die Software muß von unterschiedlichen Entwicklergruppen erstellt worden sein und ebenso dürfen keine identischen Bibliotheken, Compiler oder Entwicklungstools verwendet werden. Durch diese strikte Diversität wird eine sehr hohe Robustheit gegen Common Cause Fehler erreicht [29], da davon ausgegangen werden kann, dass ein Fehler, der beide Prozessoren gleichzeitig betrifft, sich in beiden Prozessoren unterschiedlich auswirkt und somit durch einen Vergleich der Ausgangsdaten sicher erkannt werden kann. Ein weiterer Vorteil ist das zusätzliche Erkennen von Design-Fehlern, da beide Prozessoren und die Software auf diversitärer Weise entwickelt wurden. Nachteilig an dieser Struktur ist, dass oft kein bitgenauer Vergleich der Ergebnisse erfolgen kann und somit ein Vergleich nur bei Zwischenergebnissen sinnvoll durchzuführen ist. Daher ist es schwierig, das Vergleichsfenster der beiden Ausgangsdaten eng festzulegen. Ein Fehler, der sich am Anfang nur sehr schwach auswirkt kann dadurch meist erst spät erkannt werden [8]. Ein weiterer Nachteil sind die hohen Entwicklungs- und Logistikkosten, die durch die Verwendung und Bereithaltung von zwei unterschiedlichen Prozessoren über den Produktlebenszyklus unverhältnismäßig hoch ausfallen.

Um die Entwicklungskosten zu senken, werden oftmals auch ***zweikanalige Strukturen mit diversitärer Hardware und homogener Software*** eingesetzt. Hierbei wird die Software plattformunabhängig erstellt und anschließend für die beiden unterschiedlichen Prozessoren kompiliert. Eingesetzt wird sie ebenfalls in sicherheitsrelevanten Anwendungen, wie zum Beispiel in der Luft- und Raumfahrttechnik [30]. Diese Implementierungen sind ebenfalls sehr resistent gegen Common Cause Fehler, bieten jedoch keinen Schutz gegen Softwaredesignfehler. Die sonstigen Eigenschaften entsprechen denen der vorigen Struktur.

Eine weitere Ausführungsart besteht aus einer ***zweikanaligen Struktur mit homogener Hardware und diversitärer Software***, wie sie auch in der Raumfahrttechnik eingesetzt [31] wird. Durch den Einsatz von zwei gleichen Rechnern sind die Logistikkosten niedriger als bei den bisherigen Konzepten. Durch die Verwendung von diversitärer Software werden jedoch teilweise auch Designfehler in der Hardware entdeckt. Bei Verwendung von Standardhardware, die z.B. auch in Massenprodukten, wie im Handy oder PC eingesetzt wird, kann davon ausgegangen werden, dass die schwerwiegendsten Hardwaredesignfehler bekannt sind. Der Schutz gegen Common Cause Fehler ist ebenfalls sehr gut.

Ein Nachteil bei den bisherigen Implementierungen ist, dass der Vergleich der Ausgangsdaten nicht bitgenau vorgenommen werden kann. Ebenso arbeiten die beiden Rechner bei den bisherigen Implementierungen nicht taktsynchron. Ein Vergleich der Ausgangsdaten kann erst dann erfolgen, wenn beide Rechner das Ergebnis bereitgestellt haben. Da die Strukturen der

Rechner oder der Software unterschiedlich sind, muß meist ein Rechner auf den zweiten warten, was eine Performanzeinbuße zur Folge hat. Ein taktgenauer Vergleich der Ausgangsdaten ist somit nicht möglich. Bei einer ***zweikanaligen Struktur mit homogener Hard- und Software*** ist dies jedoch möglich. Hier wird dieselbe Software auf zwei identischen Rechnern ausgeführt. Da die beiden Prozessoren synchron ihre Daten abarbeiten, kann ein Vergleich der Daten sowohl am Ende eines Tasks, als auch bei den einzelnen Zwischenschritten der Berechnung erfolgen. Durch den taktgenauen Vergleich kann der Fehlerort und Fehlerzeitpunkt sehr genau bestimmt werden. Ein Vergleich kann z.B. bei den Daten des internen Prozessorbusses stattfinden. Die Busdaten sind im fehlerfreien Fall identisch, weshalb ein bitgenauer Vergleich erfolgen kann. Da die Komparatoren kein Vergleichsfenster auswerten müssen, können sie sehr einfach aufgebaut werden. Komparatoren sind meist nur einfach vorhandene Komponenten und stellen somit einen Single Point of Failure dar, sind aber durch den einfachen Aufbau leicht gegen Fehler absicherbar. Durch die Verwendung gleicher Hardware und die enge Kopplung der beiden Mikrocontroller durch die Taktsynchronität nimmt die Anfälligkeit gegen Common Cause Fehler und Designfehler sowohl in der Hardware als auch in der Software zu. Eine Untersuchung dieser Common Cause Fehler bei Duplex-Systemen wurde in [32] vorgenommen. Empfehlenswert bei dieser Implementierung ist zur Absicherung gegen Common Cause Fehler die Verwendung eines Watchdog.

Einkanalige Struktur mit doppelt ausgelegter CPU: Bei einer Flächenbetrachtung eines Prozessorsystems fällt auf, dass der Instruktions- und Datenspeicher die meiste Chipfläche benötigt. Der Prozessor selbst ist dagegen ziemlich klein. Da Speicher sehr effektiv mit Fehlerentdeckungs- oder Fehlerkorrekturcodes abgesichert werden können, bietet es sich an, nur den Prozessor zu duplizieren. Um die Flächenersparnis durch einen gemeinsamen Speicher effektiv nutzbar zu machen, müssen beide Prozessoren identisch sein und auch dieselbe Software abarbeiten.

Diese einkanalige Struktur mit duplizierter CPU wird auch „Dual Core“ genannt. Die Ausgangssignale der beiden CPUs werden durch einen Komparator bei jedem Zugriff auf den Speicher oder auf die Peripherie verglichen [33]. Diese Struktur ist sehr anfällig gegen Common Cause Fehler, da die beiden CPUs taktsynchron arbeiten müssen, um einen bitgenauen Vergleich vornehmen zu können. Möglichkeiten diese Common Cause Fehler einzuschränken, sind das taktversetzte Abarbeiten der Tasks auf beiden CPUs oder auch die CPUs um 90° verdreht zueinander auf dem Chip zu platzieren. Dadurch wirken sich Störungen, die durch EMV verursacht werden, unterschiedlich auf beide CPUs aus, und eine Fehlererkennung durch die dadurch unterschiedlich verfälschten Ausgangsbits ist somit gewährleistet. Ein Beispiel, bei dem diese Architektur angewendet wird, ist der Infineon Safety-TriCore.

Einkanalige Struktur mit Watchdog: Watchdog-Prozessoren sind Schaltungen die das Verhalten eines Prozessors überwachen und dabei um einiges weniger komplex als die zu überwachende Schaltung selbst sind. Sie sind geeignet, um die Glaubwürdigkeit der Ergebnisse zu überprüfen, um Fehler in dem Speicherzugriffsverhalten, im Kontrollfluss oder in den Kontrollsignalen zu entdecken. Die hierzu notwendigen Kontrollinstanzen sind zum Beispiel ein in der Funktionalität reduzierter Checker-Prozessor, eine Programmablaufüberwachungseinheit oder eine Speicherzugriffskontrolleinheit [34, 35, 36, 37, 38]. Da diese Einheiten räumlich getrennt sind, können mit ihnen sowohl transiente als auch permanente Fehler entdeckt werden und weil sie funktionell diversitär sind, bieten sie einen Schutz gegen Common Cause Fehler. Zur Ver-