Lehrstuhl für Realzeit-Computersysteme

Ein Konzept zum Anschluß von Peripherie an ein fehlertolerantes Rechnersystem

Eric Dönges

Vollständiger Abdruck der von der Fakultät für Elektrotechnik und Informationstechnik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften (Dr.-Ing.)

genehmigten Dissertation.

Vorsitzender:	Univ.-Prof. Dr.-Ing., Dr.-Ing. h.c. D. Schröder
Prüfer der Dissertation:	1. Univ.-Prof. Dr.-Ing. G. Färber
	2. Univ.-Prof. Dr.-Ing., Dr.-Ing. habil. F. Schneider, i.R.

Die Dissertation wurde am 20.5.2003 bei der Technischen Universität München eingereicht und durch die Fakultät für Elektrotechnik und Informationstechnik
am 28.10.2003 angenommen.

Danksagung

Diese Dissertation entstand als Ergebnis meiner Tätigkeit als wissenschaftlicher Mitarbeiter am Lehrstuhl für Realzeit-Computersysteme der Technischen Universität München. Teile der Arbeit wurden von der *Deutschen Forschungsgemeinschaft* (DFG) als Teil des Schwerpunktprogramms „Enwurf und Entwurfsmethodik eingebetteter Systeme“ unter dem Förderkennzeichen Fa 109/13-1 („Architecture for Dependable Embedded Systems“) gefördert.

Mein besonderer Dank geht an Prof. Färber, der diese Arbeit ermöglicht hat. Ebenso danke ich Prof. Schneider für seine Bereitschaft, als Gutachter zu wirken.

Weitherhin danke ich allen Mitarbeitern des Lehrstuhls für die gute Zusammenarbeit und das kollegiale Arbeitsklima. Besonderer Dank gebührt dem Kollegen Herrn Thomas Herbig, ohne den diese Arbeit nicht zustande gekommen wäre.

Last but not least danke ich meiner Familie, die mich stets unterstützt und angespornt hat („wann bist Du denn endlich mit Deiner Dissertation fertig ?“)

München, im Mai 2003

Inhaltsverzeichnis

Verzeichnis der verwendeten Symbole

ADES	architecture for dependable embedded systems
ASIC	application specific integrated circuit
CAN	control area network
COTS	commercial off the shelf
CRC	cyclic redundancy check
E/A	Ein/Ausgabe
FTU	fault tolerant unit
IFG	inter frame gap
IFS	interface file system
MBaud	Megabaud, entspricht einer millionen Bits pro Sekunde
MIPS	million instructions per second
RODL	round descriptor list
SOS	slightly off specification
SRU	smallest replaceable unit
TTP	time triggered protocol
TTSB	time triggered sensor bus

Zusammenfassung

ADES („Architecture for Dependable Embedded Systems“) ist eine Architekturbeschreibung, nach der sichere, bzw. sichere und hochverfügbare Steuerungssysteme entwickelt werden können. Diese Arbeit beschreibt, wie Peripheriegeräte (Sensoren und Aktoren) an ein solches System angebunden werden können. Es werden zwei verschiedene, auf der Nutzung von Feldbussen basierende Lösungsansätze vorgestellt und miteinander verglichen.

Der erste Lösungsansatz erreicht die erforderliche Sicherheit und Verfügbarkeit allein durch strukturelle Maßnahmen (Hardwareredundanz); die Peripherieanbindung kann nicht getrennt vom Gesamtsystem betrachtet werden. Der zweite Lösungsansatz basiert auf fail-silent Verhalten der Feldbusse, das durch zusätzliche Fehlererkennungsmaßnahmen und spezielle Bushardware erreicht wird. Die Peripherieanbindung kann dann unabhängig vom Gesamtsystem betrachtet und entwickelt werden. Der zweite Ansatz benötigt spezielle fehlersichere Bushardware, bietet dafür aber eine höhere Sicherheit und Verfügbarkeit als ein vergleichbares System, das nach dem ersten Lösungsansatz realisiert wird. Die Machbarkeit des zweiten Lösungsansatzes wurde anhand einer einfachen Beispielanwendung demonstriert, die am Ende dieser Arbeit beschrieben wird.

1 Einleitung

Immer mehr Steuerungs- und Regelungsaufgaben werden mit Hilfe von Computersystemen automatisiert. In vielen Fällen kann dabei ein Ausfall des Computersystems hohe Kosten verursachen oder sogar Mensch und Umwelt gefährden. Normale Computersysteme, wie sie z. B. im Bürobereich üblich sind, können für diese Aufgaben nicht verwendet werden, da sie die erhöhten Anforderungen an Sicherheit und Verfügbarkeit nicht erfüllen können. Daher wurden im Laufe der Jahre für die verschiedensten Anwendungsgebiete eine Vielzahl unterschiedlicher Systeme vorgeschlagen und teilweise auch in der Tat realisiert und eingesetzt. Es handelt sich dabei meist um hochspezialisierte Hard- und Software, die auf genau eine Anwendung maßgeschneidert ist; entsprechend langwierig und teuer ist die Entwicklung dieser Systeme. Die lange Entwicklungszeit führt dazu, daß die Systeme im Vergleich zu „normalen“ Systemen meist technisch bereits überholt sind, wenn sie zum ersten Mal eingesetzt werden.

Für die Klasse der zukünftigen „X-by-Wire“ Systeme (Break-by-Wire, Steer-by-Wire, etc.) aus dem Automobilsektor sind die langen Entwicklungszeiten und die hohen Entwicklungskosten kein so großes Hindernis, da hier – bedingt durch die zu erwartenden hohen Stückzahlen – die Entwicklungskosten gegenüber den Produktionskosten und anderen Faktoren wie Platz- und Energiebedarf kaum ins Gewicht fallen. Hier ist eine genau auf die Anwendung abgestimmte Spezialentwicklung praktisch zwingend erforderlich ([21]), da nur so der dominierende Hardwareaufwand minimiert werden kann.

Bei den meisten anderen Regelungs- und Steuerungsaufgaben der Automatisierungstechnik ist die Situation jedoch genau umgekehrt – die Stückzahlen liegen meist in einem Bereich, wo die Entwicklungskosten maßgeblich über die Gesamtkosten entscheiden. Der Entwickler muß daher versuchen, sein System soweit wie nur irgend möglich aus Standardbaugruppen („commercial off the shelf“, COTS) aufzubauen, um von bereits geleisteter Entwicklungsarbeit profitieren zu können. Ein weiterer Vorteil der COTS-Komponenten ergibt sich durch ihren breiten Einsatz in vielen verschiedenen Anwendungsgebieten, so daß eventuell vorhandene Schwächen und Methoden bekannt sind, mit denen diese Schwächen umgangen werden können. Die Komponenten können eine gewisse Betriebsbewährtheit für sich in Anspruch nehmen.

Die stetig steigende Komplexität der Anwendungen führt zu einer steigenden Komplexität der notwendigen Software, deren Entwicklung als Folge immer aufwendiger wird und länger dauert. Neben den Entwicklungskosten steigt damit auch die Wahrscheinlichkeit, fehlerhafte Software auszuliefern. Daher muß die Softwareentwicklung so weit wie möglich vereinfacht werden. Ideal wäre eine Plattform, die Sicherheit und Verfügbarkeit als Systemeigenschaften bereitstellt, die unabhängig von der konkreten Anwendung entwickelt und zertifiziert werden kann. Diese Arbeit müßte nur einmal gemacht werden und könnte dann in einer Reihe von Anwendungen wiederverwertet werden. Da sich der Anwendungsentwickler völlig auf seine anwendungs-

spezifische Funktionalität beschränken könnte, müßte er auch kein Experte für Sicherheitstechnik sein. Das von der Deutschen Forschungsgemeinschaft unter dem Kennzeichen Fa 109/13-1 geförderte Forschungsprogramm „Architecture for Dependable Embedded Systems“ (ADES) versucht genau hier anzusetzen.

ADES ist dabei kein fertiges System, sondern vielmehr eine Architekturbeschreibung, nach der sichere bzw. sichere und zusätzlich hochverfügbare Systeme aufgebaut werden können. Die Architektur ist für Anwendungen mit kleinen bis mittleren Stückzahlen gedacht, bei denen die Entwicklungskosten einen entscheidenden Einfluß auf die Gesamtkosten haben. Im Vordergrund steht nicht eine möglichst optimale Ausnutzung vorhandener Hardwareresourcen, sondern eine möglichst einfache und schnelle (Software-)Entwicklung.

ADES verwendet ein zentralistisches Systemmodell, in dem alle Steuerungs- und Regelungsaufgaben von einem zentralen Rechnerkern bearbeitet werden. Der Rechnerkern besteht – je nachdem, ob ein sicheres oder ein sicheres und verfügbares System benötigt wird – aus zwei oder drei normalen Computern, die über ein eigenes Kommunikationssystem miteinander gekoppelt sind. Auf allen Rechnern des Kerns läuft funktional identische Anwendungssoftware. Zur Synchronisation der Rechner untereinander werden alle Ergebnisse und interne Zustände über das Kommunikationssystem unter den Rechnern ausgetauscht und votiert. Der Anwendungsentwickler muß dabei lediglich angeben, wann und welche Daten synchronisiert werden sollen; die dafür notwendige Funktionalität wird vom ADES Laufzeitsystem zur Verfügung gestellt. Das Laufzeitsystem setzt auf ein normales, unmodifiziertes Realzeitbetriebssystem auf; alle für die Redundanzverwaltung notwendige Funktionalität ist im Laufzeitsystem gekapselt. Der Anwendungsentwickler wird damit von einer der aufwendigsten und schwierigsten Aufgaben bei der Entwicklung redundanter Systeme entlastet und kann sich auf die Entwicklung seiner Anwendung konzentrieren.

Neben dem Rechnerkern benötigt ein nach ADES entwickeltes System Peripherie (Sensorik und Aktorik), um mit der Umwelt interagieren zu können. Diese muß geeignet an den Rechnerkern gekoppelt werden. Im Kontext von ADES bedeutet „geeignet“, daß die Interaktion mit der Umwelt denselben Anforderungen an Sicherheit und Verfügbarkeit genügen muß wie der Rechnerkern selber. Diese Arbeit beschreibt ein mögliches Konzept zur Anbindung von Peripherie an einen ADES Rechnerkern unter Verwendung der Feldbustechnik. Andere Anbindungsarten sind denkbar, werden hier aber nicht weiter behandelt.

Die Peripherieanbindung ist sehr anwendungsspezifisch und kann daher nicht allgemeingültig detailliert betrachtet werden. Allerdings kann man zwei grundsätzliche Philosophien unterscheiden. Bei der ersten Philosophie wird die Sicherheit und Verfügbarkeit der Peripherie ausschließlich auf der Ebene des Gesamtsystems betrachtet. Die einzelnen Komponenten der Peripherieanbindung müssen daher keinen besonderen Anforderungen an Sicherheit und Verfügbarkeit genügen. Die zweite Philosophie setzt tiefer in der Systemhierarchie an. Das Gesamtsystem wird nun aus Subsystemen zusammengesetzt, die für sich betrachtet ebenfalls sicher bzw. sicher und verfügbar sind. Die einzelnen Komponenten einer solchen Peripherieanbindung fallen allerdings deutlich komplexer aus, da sie nun ebenfalls erhöhte Anforderungen an Sicherheit und Verfügbarkeit erfüllen müssen. Dafür wird die Komplexität des Gesamtsystems reduziert.

Im Rahmen dieser Arbeit werden beide Ansätze vorgestellt und miteinander verglichen. Die

Arbeit ist dabei wie folgt gegliedert. In Kapitel 2 werden für das Verständnis notwendige grundlegende Begriffe und Techniken erläutert. Es folgt in Kapitel 3 ein Überblick über den aktuellen Stand der Technik, der am Beispiel einiger realer Systeme erläutert wird. Das Grundkonzept von ADES und der ADES Rechnerkern werden am Anfang von Kapitel 4 kurz vorgestellt; der Rest des Kapitels beschreibt das Peripheriekonzept im Detail. Dabei werden zwei mögliche Strukturen des Peripherieanschlusses vorgestellt, die in den Kapiteln 5 und 6 eingehend miteinander verglichen werden.

Am Lehrstuhl für Realzeit-Computersysteme der Technischen Universität München wurde zur Untersuchung der prinzipiellen Machbarkeit des ADES Architekturkonzeptes ein einfaches Anwendungsbeispiel (eine in einem Magnetfeld schwebende Kugel) prototypisch realisiert. Die konkrete Implementierung des Peripherieanschlusses nach dem hier vorgestellten Konzept wird in Kapitel 7 beschrieben. Kapitel 8 schließt die Arbeit mit praktischen Erfahrungen und Ergebnissen ab, die am Prototypen gewonnen werden konnten.

2 Grundlagen und Begriffsklärung

Dieses Kapitel gibt einen kurzen Überblick über die Begriffe und Verfahren der Sicherheitstechnik für Automatisierungsanlagen.

2.1 Problemstellung

Ein nach dem ADES-Architekturkonzept erstelltes System besteht aus dem physikalischen Prozeß, der gesteuert bzw. geregelt[1)] werden soll, dem ADES Rechnerkern und der Peripherie (Sensorik und Aktorik) samt Anbindung an den ADES Rechnerkern. Sowohl die Peripherie als auch die Verbindung zwischen Peripherie und Rechnerkern müssen denselben Anforderungen an Sicherheit und Verfügbarkeit genügen wie der Rechnerkern, um ein sicheres bzw. ein sicheres und verfügbares Gesamtsystem realisieren zu können.

ADES beschreibt dabei nur den Rechnerkern und die Verbindung zu den Peripherieeinheiten; der Aufbau der Peripherieeinheiten selber ist weder Teil von ADES noch dieser Arbeit.

2.2 Begriffsdefinition

Vor einer weiteren Diskussion der Problemstellung sollten zuerst einige Begriffe eingeführt und erklärt werden. Die im folgenden verwendeten Definitionen orientieren sich dabei an [36].

Ein System ist nach DIN 66201 eine Anordnung aufeinander einwirkender Gebilde, die von der Umgebung abgegrenzt angesehen werden kann. Ein Prozeß ist (ebenfalls nach DIN 66201) die Summe aller physikalischen, chemischen oder biologischen Vorgänge innerhalb eines Systems, durch die Energie, Materie oder Information transportiert, gespeichert oder umgeformt wird. Im Rahmen dieser Arbeit werden nur Prozesse betrachtet, die mit technischen Mitteln erfaßt und beeinflußt werden können.

Technische Systeme, wie sie in dieser Arbeit betrachtet werden, besitzen eine Reihe von Meßpunkten, an denen Prozeßgrößen gemessen werden können, sowie Stellpunkte, an denen Prozeßgrößen mit Hilfe von Aktoren manipuliert werden können. Sensoren und Aktoren werden unter dem Oberbegriff Peripherieeinheit zusammengefaßt, während für Meß- und Stellpunkte der Oberbegriff E/A-Punkte verwendet wird.

[1)] Im folgenden wird immer von geregelten Prozessen ausgegangen, da sich für die Anbindung an den technischen Prozeß keine Unterschiede zwischen einer Steuerung und einer Regelung ergeben.

Ein Fehler ist jede Abweichung vom erwünschten Verhalten. Man unterscheidet zwischen aktiven Fehlern, bei denen das System eine unerwünschte Funktion ausführt, und passiven Fehlern, bei denen das System eine erwünschte Funktion nicht ausführt. Die englische Fachliteratur unterscheidet weiter zwischen der Ursache eines fehlerhaften Zustandes (engl. *fault*), dem fehlerhaften Zustand selber (engl. *error*), und seiner möglichen Auswirkung, dem Ausfall bzw. Versagen (engl. *failure*). Ein fehlerhafter Zustand muß nicht zwangsläufig zu einem Ausfall führen, wenn die betroffene Funktionalität vom System nicht in Anspruch genommen wird.

Es gibt zwei Möglichkeiten, Systemausfälle zu verhindern. Zum einen kann man durch entsprechenden Entwurf und Konstruktion verhindern, daß überhaupt ein Fehler auftritt. Bei mechanischen Systemen ist dies meist – zumindest bis zu einem gewissen Grad – möglich, indem man die einzelnen Komponenten für eine Belastung auslegt, die deutlich über der im normalen Einsatz erwarteten Belastung liegt.

Bei elektronischen und insbesondere digitalen Systemen ist ein solcher Ansatz nicht möglich, da das Ausfallverhalten völlig verschieden ist. Beim Entwurf muß daher generell mit dem Auftreten von Fehlern gerechnet und festgelegt werden, wie das System darauf reagieren soll.

Im Rahmen dieser Arbeit gilt ein System als normal, wenn keine besonderen Vorkehrungen für den Fall eines Fehler vorgesehen sind. Es gilt als sicher, wenn es nach dem Auftreten eines beliebigen Fehlers automatisch eine sichere Endlage annimmt (dies ist in der Regel der energielose Zustand), in der weder Mensch noch Maschine zu Schaden kommen können. Dieses Verhalten wird auch fail-safe Verhalten genannt. Ein System gilt als hochverfügbar, wenn es auch nach einem Fehler seine Funktion mit hoher Wahrscheinlichkeit weiter erfüllen kann. Ein solches System ist nicht zwangsläufig sicher, d. h. ein Fehler kann auch zu unkontrolliertem Verhalten führen.

Ist ein System sowohl sicher, als auch hochverfügbar, spricht man von fail-operational Verhalten. Wegen des sehr hohen Aufwandes werden fail-operational Systeme meist nur in Anwendungen eingesetzt, bei denen entweder keine sichere Endlage existiert, oder in denen die sichere Endlage nur durch aktive Maßnahmen und nicht durch einfaches Abschalten eingenommen werden kann. Bekannte Beispiele hierfür sind Fly-By-Wire Systeme in Flugzeugen.

2.3 Fehlerklassifikation und Ausfallmodelle

Für die Entwicklung eines sicheren Systems ist es unerläßlich, die erwarteten Fehler zu analysieren. In Tabelle 2.1 sind verschiedene Kriterien angegeben, nach denen Fehler klassifiziert werden können.

- Aktivität
  Führt ein Fehler zu einer sofort erkennbaren Fehlerausprägung, spricht man von einem aktiven Fehler. Kann eine Fehlerausprägung nicht sofort erkannt werden, spricht man dagegen von einem latenten Fehler. Latente Fehler sind schwierig zu lokalisieren, da ihre Ursache und Wirkung zeitlich weit auseinander liegen können.

Kriterium	**Fehler**
Aktivität	latent, aktiv
Dauer	transient, permanent
Wahrnehmung	symmetrisch, asymmetrisch
Ursache	zufällig, systematisch, gezielt
Auswirkung	gutartig, bösartig
Anzahl	einfach, mehrfach
Zeit (Mehrfachfehler)	gleichzeitig, einzeln
Ursache(Mehrfachfehler)	unabhängig, abhängig

Tabelle 2.1: Fehlerklassifikation nach [5]

- Dauer
  Ein transienter Fehler ist nur von kurzer Dauer (z. B. durch elektromagnetische Einstreuung verursachte Störungen auf Übertragungsleitungen); seine Ursache verschwindet wieder von alleine. Ein permanenter Fehler dagegen kann nur durch Austausch oder Reparatur der fehlerhaften Komponente entfernt werden. Die Unterscheidung zwischen transienten und permanenten Fehlern ist wichtig, da transiente Fehler wesentlich häufiger vorkommen als permanente Fehler, eine Erholung aber prinzipiell ohne externe Intervention vom System selber aus möglich ist.

- Wahrnehmung
  Alle funktionsfähigen Komponenten eines Systems beurteilen einen symmetrischen Fehler gleich, während ein asymmetrischer Fehler unterschiedlich bzw. von manchen funktionsfähigen Komponenten auch gar nicht wahrgenommen wird. Asymmetrische Fehler sind in redundanten Systemen besonders kritisch, da sie zu Inkonsistenzen zwischen den internen Zuständen an sich fehlerfreier Komponenten führen können. Zum Erkennen und Beheben solcher Fehler sind besondere Verfahren erforderlich (die klassische Lösung ist in [24] beschrieben). Ist ein System gegen asymmetrische Fehler tolerant, spricht man auch von „interaktiver Konsistenz“.

- Ursache
  Zufällige Fehler werden zufällig durch die Umwelt des Systems verursacht; ihr genaues Eintreten kann nicht vorhergesagt werden. Ein systematischer Fehler dagegen hat seine Ursache in einem Entwurfs- oder Produktionsfehler. Systematische Fehler sind insofern problematisch, weil sie alle gleichartigen Komponenten gleichermaßen betreffen und übliche Redundanzstrategien unwirksam machen können (siehe auch Abschnitt 2.4). Ein gezielter Fehler schließlich wird absichtlich von außen in das System eingebracht (entweder als Sabotage oder um die Sicherheit des Systems zu verifizieren). Im Rahmen dieser Arbeit werden gezielte Fehler nicht weiter berücksichtigt.

- Auswirkung
  Ein gutartiger Fehler kann von allen funktionsfähigen Komponenten erkannt werden. Ein bösartiger Fehler ist dagegen eventuell nicht direkt erkennbar und kann bei verschiedenen Komponenten zu unterschiedlichen Symptomen führen. Kann über den Fehler überhaupt

keine a priori Annahme gemacht werden, spricht man von „byzantinischen“ Fehlern (in Anlehnung an „The Byzantine Generals Problem“ [24], in dem diese Klasse von Fehlern zuerst beschrieben und eine Strategie zur Beherrschung solcher Fehler vorgeschlagen wurde.

- Mehrfachfehler
  Ein wichtiges Kriterium bei der Betrachtung möglicher Systemfehler ist die Frage, ob während des Betrachtungszeitraumes nur ein einziger Fehler auftreten kann oder ob Mehrfachfehler möglich sind. Mehrfachfehler können voneinander abhängig oder unabhängig sein. Ein abhängiger Mehrfachfehler wäre z. B. der Ausfall einer einzelnen Stromversorgung, an der mehrere andere Komponenten angeschlossen sind, die dann ebenfalls ausfallen. Schließlich wird noch unterschieden, ob die Fehler einzeln nacheinander oder gleichzeitig auftreten.

Neben der Betrachtung der möglichen Fehlerarten ist es erforderlich, für ein System ein passendes Ausfallmodell aufzustellen. Das Ausfallmodell legt fest, welche Art von Ausfällen vom System berücksichtigt werden können. Je einfacher dieses Modell ist, desto einfacher kann das System entworfen werden. Wird jedoch ein zu einfaches Modell angenommen, das die Wirklichkeit nicht ausreichend berücksichtigt, kann das System die notwendigen Sicherheitskriterien nicht erfüllen. Folgende Ausfallmodelle sind gebräuchlich:

- Fail-Silent Ausfälle
  Bei einem fail-silent Ausfall fällt eine Komponente vollständig aus und reagiert nicht mehr auf ihre Umgebung. Um k fail-silent Ausfälle tolerieren zu können, werden mindestens $k+1$ redundante Komponenten benötigt. Die Entscheidung, welche der redundanten Ergebnisse richtig sind, ist denkbar einfach: jedes Ergebnis ist korrekt, da eine ausgefallene Komponente gar kein Ergebnis liefert.

- Fail-Consistent Ausfälle
  Ein fail-consistent Ausfall wird von allen nicht ausgefallenen Komponenten auf die gleiche Art beurteilt; es kann also nicht vorkommen, daß eine funktionsfähige Komponente den Ausfall erkennt und eine andere nicht. Um k fail-consistent Ausfälle tolerieren zu können, benötigt man mindestens $2k + 1$ redundante Komponenten, wobei nach einem m von n Auswahlverfahren mit $n = 2k + 1$ und $m = k + 1$ (d. h. mindestens m von n Teilnehmern müssen dasselbe Ergebnis haben) entschieden wird, welches Ergebnis richtig ist.

- Fail-Uncontrolled Ausfälle
  Ein fail-uncontrolled Ausfall ist beliebig – es können keine Einschränkungen hinsichtlich der möglichen Auswirkungen des Ausfalls gemacht werden. Dies schließt insbesondere den Fall eines asymmetrischen Fehlers mit ein. In [24] wird gezeigt, daß mindestens $3k+1$ redundante Komponenten benötigt werden, um k fail-uncontrolled Ausfälle tolerieren zu können.

Eine wichtige Frage ist die nach der Abdeckungswahrscheinlichkeit (engl. *coverage*) des Ausfallmodells, d. h. wie gut das gewählte Modell die tatsächlich auftretenden Ausfälle beschreibt. In [28] werden die Auswirkungen der Abdeckungswahrscheinlichkeit des verwendeten Aus-

fallmodells auf die tatsächliche Ausfallwahrscheinlichkeit untersucht.

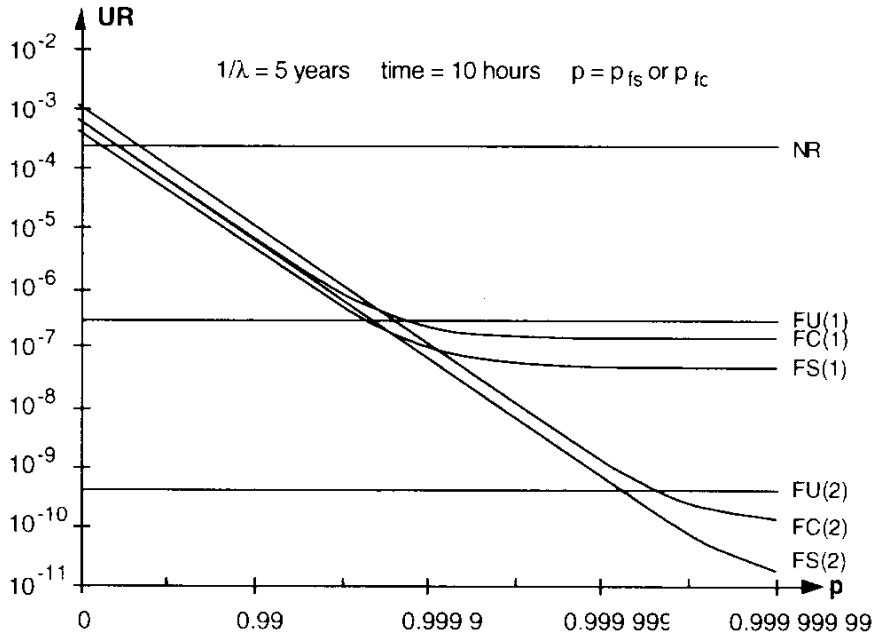

Bild 2.1: Abhängigkeit der tatsächlichen Ausfallwahrscheinlichkeit UR von der Abdeckungswahrscheinlichkeit p des Ausfallmodells (aus [28])

Die Resultate sind in Bild 2.1 dargestellt. UR ist dabei die Wahrscheinlichkeit für einen Systemausfall während einer Betrachtungszeit von 10 Stunden in Abhängigkeit von der Wahrscheinlichkeit p, daß die mit gegebener Wahrscheinlichkeit ($\lambda^{-1} = 5$ Jahre) auftretenden Komponentenausfälle tatsächlich dem gewählten Ausfallmodell (fail-consistent oder fail-silent) entsprechen. $FU(x)$, $FC(x)$ und $FS(x)$ bezeichnen dabei jeweils ein System, das x fail-uncontrolled (FU), fail-consistent (FC) oder fail-silent (FS) Ausfälle tolerieren kann, während NR ein nicht redundantes System darstellt. Deutlich zu erkennen ist, daß Systeme, die nach einem fail-silent bzw. fail-consistent Ausfallmodell entwickelt wurden, ab einer bestimmten Abdeckungswahrscheinlichkeit ihres angenommenen Fehlermodells sicherer sind als Systeme, die auch fail-uncontrolled Ausfälle tolerieren können. Dies liegt daran, daß zur Beherrschung der eingeschränkten Ausfallmodelle weniger aufwendige Maßnahmen notwendig sind, die selber wiederum weniger fehleranfällig sind. Bei Systemen, die jeweils einen Komponentenausfall tolerieren können, ist die Grenze bei einer Abdeckungswahrscheinlichkeit von ca. $0,9999$ erreicht. Obwohl eine solche Abdeckungswahrscheinlichkeit für ein fail-silent Modell eher fragwürdig ist (99,99% aller Fehler müßten vom System selber erkannt werden), ist sie für ein fail-consistent Modell durchaus denkbar.

2.4 Redundanz

In der Praxis wird fail-safe bzw. fail-operational Verhalten meist durch eine oder mehrere Arten von Redundanz erreicht. Redundanz bedeutet, daß mehr technische Mittel vorhanden sind, als

für die vorgesehene Funktion unbedingt notwendig sind. Redundanz ist dabei nicht auf physikalische Komponenten beschränkt. So ist es z. B. möglich, einen Algorithmus mehrfach hintereinander (zeitredundant) zu durchlaufen und die Ergebnisse miteinander zu vergleichen. Auf diese Art können vorübergehende Störungen in der Bearbeitungseinheit erkannt werden, ohne daß redundante Bearbeitungseinheiten benötigt werden. Bei der Speicherung oder Übertragung von Daten verwendet man Informationsredundanz, um die Daten gegen Verfälschungen abzusichern. Dabei werden den eigentlichen Nutzdaten zusätzliche Informationen zugefügt, anhand derer eine Verfälschung erkannt und – je nach verwendetem Verfahren – auch behoben werden kann. Eine weitere Form der Redundanz ist die analytische Redundanz, bei der zusätzliches Wissen über die Wechselwirkungen innerhalb des betrachteten Prozesses genutzt wird, um Fehler erkennen zu können (analytische Redundanz alleine kann keine Fehler beheben, sondern muß mit einer anderen Form von Redundanz kombiniert werden). Bild 2.2 zeigt zwei mögliche Blockstrukturen einer Anwendung analytischer Redundanz. Das Problem bei analytischer Redundanz liegt darin, die hierzu benötigten Prozessmodelle mit ausreichender Genauigkeit zu bestimmen. Dies ist meist nur mit erheblichem Aufwand möglich.

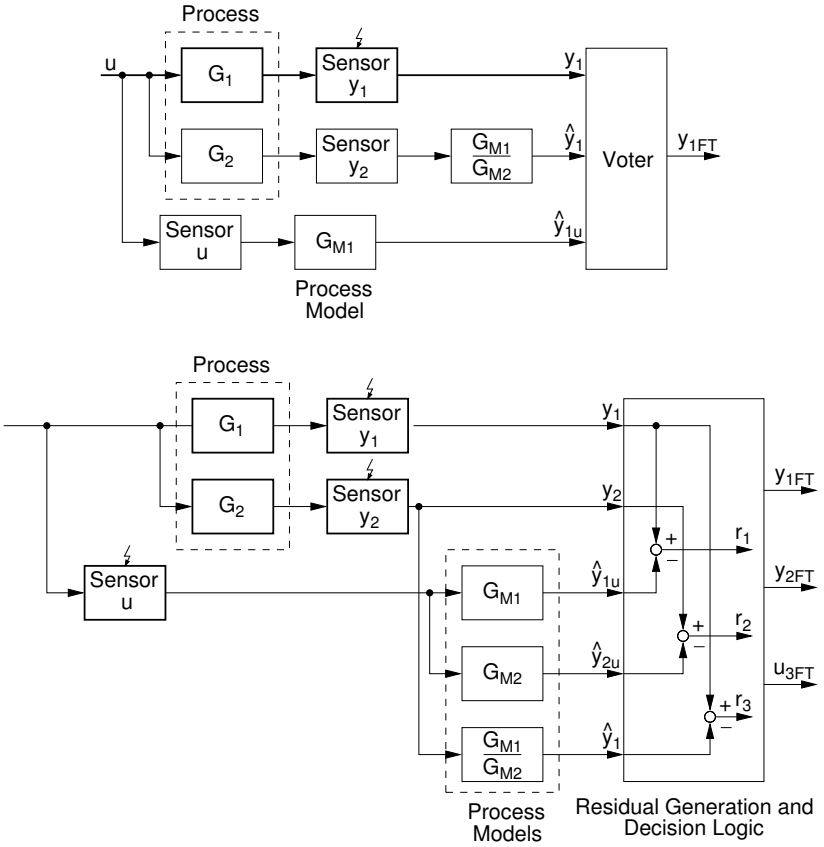

Bild 2.2: Zwei Beispiele für mögliche Realisierungen analytischer Redundanz (nach [21])

Redundanzmaßnahmen lassen sich ganz grob in die zwei große Klassen statische und dynamische Redundanz einteilen. Bild 2.3 zeigt verschiedene Strukturen statischer und dynamischer Redundanz.

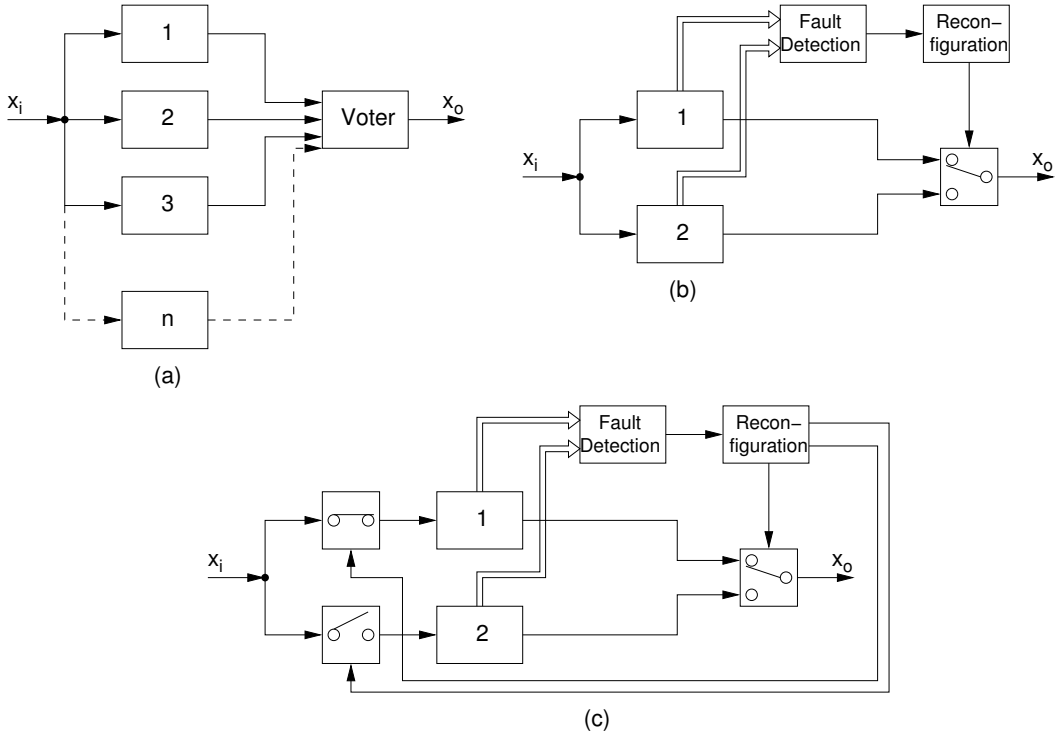

Bild 2.3: Verschiedene Klassen redundanter Strukturen: (a) statische Redundanz mit Mehrheitsentscheider, (b) dynamische Redundanz mit hot-standby, (c) dynamische Redundanz mit cold standby (nach [21]).

Bei der statischen Redundanz sind alle redundanten Komponenten gleichzeitig parallel in Betrieb. Ein sog. Voter muß die Ausgaben aller Komponenten miteinander vergleichen und die richtigen Ausgaben auswählen („votieren“). Fehlerhafte Komponenten werden entweder abgeschaltet oder ihre Ausgaben werden ausgeblendet („maskiert“). Eine der am häufigsten verwendeten Formen statischer Redundanz sind die sog. m-aus-n Systeme (mit $m \leq n$ und $n \geq 2$). Bei ihnen müssen von insgesamt n funktional identischen Komponenten mindestens m Komponenten funktionsfähig sein, damit das Gesamtsystem funktionsfähig ist. Der häufige Sonderfall $m = n$, wobei dann meist $n = 2$ gilt, wird zur Fehlererkennung verwendet. Ein anderer häufiger Spezialfall ist $m = 2$ und $n = 3$, der auch als „tripple modular redundancy“ (TMR) bekannt ist. Diese Systemstruktur kann einen fail-consistent Ausfall tolerieren.

Bei der dynamischen (engl. *standby*) Redundanz werden die redundanten Komponenten erst nach einem Ausfall der primären Komponente hinzugeschaltet. Hier wird ein Umschalter benötigt, der Fehler in der aktiven Komponente erkennen und auf eine Ersatzkomponente umschalten kann. Man unterscheidet zwischen zwei Varianten der dynamischen Redundanz – bei der heißen Reserve (engl. *hot standby*) sind die redundanten Komponenten parallel zur eigentlichen Funktionseinheit in Betrieb, ihre Ergebnisse werden aber nicht verwendet. Ein Umschalten kann daher ohne Zeitverlust erfolgen. Bei der kalten Reserve (engl. *cold standby*) dagegen sind die redundanten Komponenten nicht sofort betriebsbereit. Das Umschalten auf die kalte Reserve kann unter Umständen vergleichsweise viel Zeit benötigen. Daher wird diese Art der Redundanz in der Automatisierungstechnik eigentlich nicht verwendet. Eine Ausnahme bilden Anwendungen

in der Raumfahrt, wo Reservekomponenten teilweise abgeschaltet werden, um Strom zu sparen oder eine Abnutzung der Reservekomponenten zu vermeiden.

Die Verwendung identischer redundanter Komponenten ist nur gegen stochastisch voneinander unabhängige Fehler wirksam. Bei Anwendungen mit extremen Anforderungen an Sicherheit oder Verfügbarkeit ist dies nicht ausreichend; hier müssen auch Entwurfs- und Produktionsfehler der einzelnen Komponenten beherrschbar sein. Daher verwendet man in solchen Fällen diversitäre Komponenten, die zwar die gleiche Funktion erfüllen, aber von unterschiedlichen Entwicklungsgruppen entwickelt wurden und eventuell auch ein ganz anderes Funktionsprinzip verwenden. Diversität ist mit einem sehr hohen Entwicklungsaufwand verbunden und wird daher nach Möglichkeit vermieden (eine Entwicklung muß nicht nur einmal, sondern mehrfach durchgeführt werden – und zwar völlig eigenständig, da sonst das Ziel diversitärer Entwicklung nicht erreicht werden kann). Ein weiteres Problem liegt darin begründet, daß viele Entwurfsfehler bereits in der Spezifikationsphase gemacht werden, und daher die Spezifikation ebenfalls diversitär erstellt werden müßte; dies würde den notwendigen Aufwand aber nochmals erheblich erhöhen, bzw. ist eventuell gar nicht konsequent durchführbar, da die nach den diversitären Spezifikationen entwickelten Subsysteme reibungslos zusammenarbeiten können müssen.

Aus Kostengründen kann es unerwünscht sein, funktional identische Komponenten zu verwenden; statt dessen kann man in ihrer Funktionalität stark eingeschränkte Komponenten einsetzen, die nur zur Fehlererkennung dienen oder einen Notbetrieb ermöglichen. Ein bekanntes Beispiel hierfür sind die in ECC-Halbleiterspeichern verwendeten zusätzlichen Prüfbits, mit deren Hilfe Bitfehler erkannt und korrigiert werden können.

2.5 Redundante Sensorik und Aktorik

Bei der Verwendung redundanter Sensorik und Aktorik ergeben sich einige Schwierigkeiten. Grundsätzlich können redundante Peripherieeinheiten nicht am exakt gleichen Ort installiert werden. Ebensowenig sind baugleiche Peripherieeinheiten völlig identisch. Es ist daher immer mit einem Abweichen der redundanten Einheiten untereinander sowohl im Zeit-, wie auch im Wertebereich zu rechnen. Diese Abweichungen können von kurzer Dauer (z. B. bedingt durch den unterschiedlichen Einbauort), oder permanent sein (z. B. bedingt durch Fertigungstoleranzen der einzelnen Peripherieeinheiten). Es muß daher ein geeigneter Votierungsalgorithmus gefunden werden, mit dem bestimmt werden kann, welche Werte korrekt und welche fehlerhaft sind. Einen sehr guten Überblick über mögliche Votierungsalgorithmen wird in [25] gegeben.

Bei redundanter Aktorik ergibt sich im Vergleich zur Sensorik das zusätzliche Problem, daß sich Aktorik i. allg. nur mit hohem Aufwand redundant verwenden läßt. Aktoren, die Masse- oder Energieströme an- oder abschalten, können relativ leicht zu redundanten Anordnungen zusammengefaßt werden (siehe Bild 2.4); allerdings ergeben sich hier bereits erste Probleme, wenn die Durchflußmenge geregelt und nicht nur an- oder abgeschaltet werden soll. In diesem Fall muß die Durchflußmenge von jedem Zweig der Anordnung gemessen und die Ansteuerung der einzelnen Aktoren koordiniert werden; man hat es dann nicht mehr mit einzelnen redundanten Aktoren, sondern mit einem komplexen Subsystem zu tun.

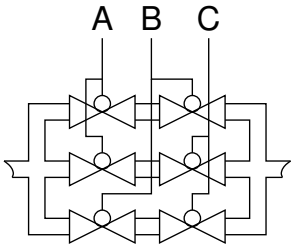

Bild 2.4: Ventilharfe, nach [18]

Bei redundanten Antrieben ergeben sich dagegen ernorme Schwierigkeiten, da die Antriebe alle letztlich auf dieselbe Mechanik einwirken müssen. Im Fehlerfall müssen die funktionsfähigen Antriebe noch in der Lage sein, die Funktion auch gegen den aktiven Widerstand eines ausgefallenen oder falsch angesteuerten Antriebes zu erfüllen.

Eine wesentliche Vereinfachung läßt sich erreichen, wenn die Aktoren fail-silent Verhalten besitzen, da es dann keine aktive Störung durch einen ausgefallenen Aktor geben kann. Bei Sensoren ist fail-silent Verhalten ebenfalls erwünscht, weil dann der Redundanzgrad vermindert werden kann (siehe Abschnitt 2.3).

2.6 Konventionelle Sicherheitstechnik

In der konventionellen Sicherheitstechnik werden die sicherheitsrelevanten Sensoren (z. B. Notaustaster, Lichtgitter, Endlagenschalter, etc.) und Aktoren (meist Leistungstrennschütze, mit denen die Antriebe von ihrer Energieversorgung getrennt werden können) direkt mit der Sicherheitsschaltung oder -steuerung verbunden, die sich normalerweise im Schaltschrank befindet. Da jedes Signal eine eigene Leitung verwendet, erreichen auch kleine Anlagen eine beträchtliche Anzahl von zu verlegenden Leitungen; man spricht daher auch von Parallelverdrahtung. Um den Kabelaufwand zu reduzieren, werden in ihrer Funktion zusammenhängende Gruppen von Sensoren (z. B. alle Notaus-Taster) zusammengefaßt und an einer Leitung in Serie geschaltet. Die Leistungstrennschütze befinden sich meist im Schaltschrank und schalten die Stromversorgung für viele Aktoren gleichzeitig. Es ist daher nicht möglich, die genaue Ursache einer Sicherheitsreaktion zu bestimmen, oder nur einzelne Teile einer Anlage gezielt abzuschalten.

Die Sicherheitsfunktionalität selber wird meist mit einfachen elektromechanischen, pneumatischen oder hydraulischen Schaltungen realisiert – man spricht daher auch von „kontaktbehafteter“ Technik – die von der eigentlichen Steuerungsfunktionalität völlig getrennt sind. Die Sicherheitsfunktionalität ist daher auf einfache logische Verknüpfungen beschränkt. Es existieren zwar inzwischen auch Sicherheitsschaltkombinationen, die typische elektromechanische Sicherheitsschaltungen mit Hilfe von Mikroelektronik nachbilden; sie bilden aber nur die Funktion der elektromechanischen Schaltungen nach und bieten keine erweiterte Funktionalität. In ihrer Funktion sind diese Schaltungen „Zustimmschaltungen“, d. h. dem Einleiten einer potentiell gefährlichen Aktion seitens der eigentlichen Steuerung muß von außen zugestimmt werden,

bevor sie ausgeführt werden kann.

In Bild 2.5 ist als Beispiel die Überwachung einer Schutztür mit Zuhaltung in kontaktbehafteter Technik dargestellt. Die Stromversorgung der für die zu überwachende Gefahr verantwortlichen Aktoren (z. B. ein Spindelantrieb) wird über den mit „Freigabe“ bezeichneten Pfad geführt; ist die Schutztür nicht geschlossen, sind die Aktoren energielos geschaltet. Die verwendeten Steuerschütze (K1, K2, K3 und K4) sind alle zwangsgeführt, d. h. es werden entweder alle Kontakte oder gar keine geöffnet bzw. geschlossen. Die Schalter S1 und S2 sind zwangsöffnend, d. h. ihre Kontakte werden bei Betätigung garantiert geöffnet. Diese Eigenschaften reduzieren die möglichen Ausfallmöglichkeiten und vereinfachen damit den Schaltungsentwurf wesentlich; sie werden durch entsprechende (mechanische) Auslegung der Komponenten erreicht.

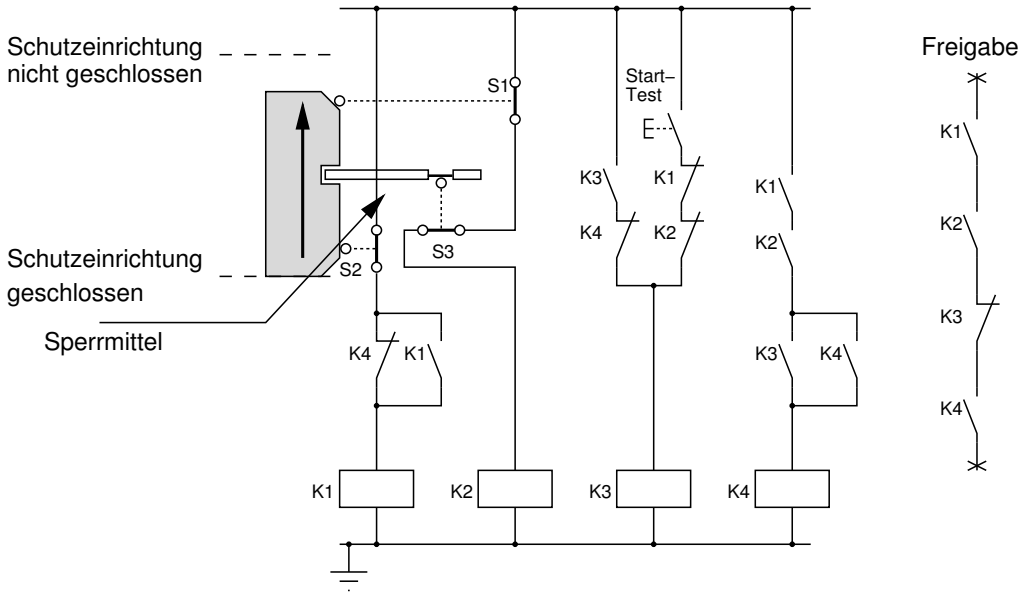

Bild 2.5: Stellungsüberwachung beweglicher Schutzeinrichtungen mit Zuhaltung und Anlauftestung nach EN 954 Kategorie 4, nach [22]

Vorteilhaft an dieser Technik ist, daß die Schaltungen relativ einfach sind und daher ihre korrekte Funktion leicht nachzuweisen ist. Da grundsätzlich Leistungen geschaltet werden, sind sie auch recht unempfindlich gegenüber elektromagnetischen Störungen. Diesen Vorteilen stehen jedoch eine Reihe von gravierenden Nachteilen gegenüber. Die strenge Trennung zwischen der Sicherheitsfunktionalität und der Steuerungsfunktionalität schränkt die Flexibilität der Steuerung stark ein. So ist z. B. ein bahntreues Abbremsen der Vorschubantriebe nur möglich, wenn die Sicherheitsfunktion direkt in die Achssteuerung mit integriert wird. Das bahntreue Abbremsen ist bei modernen, hochdynamischen Werkzeugmaschinen notwendig, da ein einfaches (nicht bahntreues) Anhalten der Vorschubantriebe im Falle eines Notaus sowohl das Werkzeug, als auch das Werkstück beschädigen kann. Eine weitere Einschränkung kontaktbehafteter Technik ergibt sich durch die mit ca. 20-50ms recht lange Abfallzeit von Schützen, wodurch die erreichbare Reaktionsgeschwindigkeit stark eingeschränkt wird.

Daher gibt es verstärkt auch sichere programmierbare Steuerungen, die neben den Sicherheitsfunktionen auch die normale Steuerung der Anlage übernehmen können. Die notwendige siche-

re Aktorik und Sensorik ist jedoch meist noch konventionell parallelverdrahtet mit den entsprechenden E/A-Baugruppen der Steuerung verbunden.

2.7 Feldbustechnik

Wenn mehrere Geräte zum Datenaustausch gemeinsam eine oder mehrere zusammengehörende digitale Datenleitungen nutzen, spricht man in der Technik von einem Bus. Ein Feldbus ist ein Bus, der für die Kommunikation zwischen einem Steuerungsrechner und „im Feld“ (also außerhalb der Leitwarte bzw. des Schaltschrankes) installierten Sensoren und Aktoren verwendet wird. Signale (Messwerte und Stellwerte) werden digital kodiert, in ein vom System abhängiges Nachrichtenformat umgewandelt und dann über den Bus gesendet. Alle bekannten Feldbusse arbeiten bitseriell, d. h. eine Nachricht wird als Strom einzelner Bits in Serie übertragen. Auf diese Weise können sehr viele verschiedene Signale gemeinsam über eine einzige Datenleitung übertragen werden.

Insbesondere in Anlagen mit großer physikalischer Ausdehnung läßt sich auf diese Art der Verdrahtungaufwand deutlich reduzieren; daher wurden Feldbusse auch zuerst in der chemischen Industrie verwendet, wo zwischen der Leitwarte und den Feldgeräten oft sehr große Entfernungen liegen. Der größte Vorteil der Bustechnik liegt aber darin, daß neben den eigentlichen Nutzdaten ohne erheblichen Mehraufwand zusätzliche Daten zur Diagnose oder Parametrierung von komplexeren Geräten übertragen werden können und somit Fernwartung ermöglicht wird.

Ein Mittelding zwischen der Parallelverdrahtung und der Feldbustechnik stellt das in der Prozesstechnik verwendete HART-Protokoll („Highway Addressable Remote Transducer“) dar. Bei dieser Technik werden digitale Daten dem in der Prozesstechnik üblichen analogem $4 \ldots 20$mA Normsignal überlagert. Auf diese Weise kann die zusätzliche Funktionalität eines Feldbusses über bereits bestehende konventionelle Verkabelung genutzt werden. Die Datenübertragung mit HART ist allerdings recht langsam, und man benötigt für jedes Gerät eine eigene Leitung, so daß HART eher als eine Überganglösung bis zum Umstieg auf ein „richtiges“ Feldbussystem zu sehen ist.

Die Vorteile der Feldbustechnik haben dazu geführt, daß sie inzwischen in praktisch allen Bereichen der Automatisierungstechnik für die Anbindung nicht sicherheitskritischer Aktoren und Sensoren eingesetzt wird. Im folgenden wird untersucht, unter welchen Bedingungen die Feldbustechnik auch für die Anbindung sicherheitsrelevanter Sensorik und Aktorik verwendet werden kann.

2.7.1 Typische Systemstruktur

Ein Feldbussystem besteht normalerweise aus einem zentralen Steuerungssystem und vielen dezentralen Peripheriegeräten, die über ein gemeinsames Medium miteinander gekoppelt sind. Man spricht dann von dezentraler Peripherie. Die meisten Feldbusse erlauben zwar zusätzlich auch die Kommunikation zwischen verschiedenen Steuerungssystemen; hierfür werden aber

verstärkt Kommunikationssysteme aus der Bürotechnik für die speziellen Bedürfnisse der Automatisierungstechnik angepaßt und verwendet (z. B. Industrial Ethernet). Diese bieten gegenüber Feldbussen wesentlich höhere Datenraten und die Möglichkeit, die Steuerungen an das Rechnernetz des Unternehmens anzubinden.

Einfache Sensoren oder Aktoren wie z. B. induktive Näherungssensoren oder Magnetventile werden aus Kostengründen üblicherweise nicht direkt über eine eigene Busanschaltung an den Bus gekoppelt, sondern über dezentrale E/A-Baugruppen, die einen Busanschluß für mehrere konventionelle Ein- und Ausgänge enthalten.

Der Begriff „Feldbus“ ist etwas irreführend, da die physikalische Struktur eines Feldbussystems nicht zwingend als Bus ausgeführt sein muß. In der Praxis werden die drei in Bild 2.6 dargestellten Topologien verwendet.

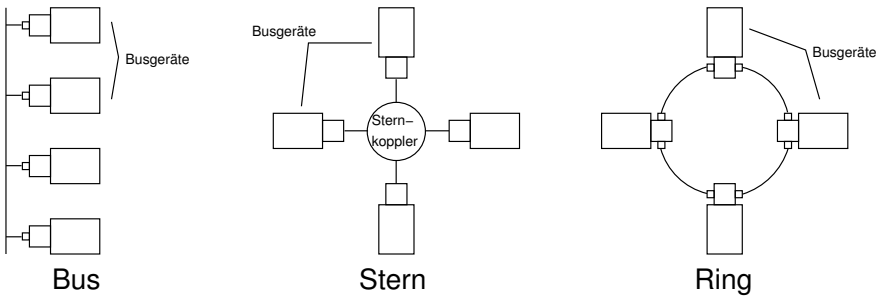

Bild 2.6: Gebräuchliche Topologien für Feldbussystem

Bei der Bustopologie sind alle Teilnehmer über eine Busleitung miteinander verbunden (dies entspricht der klassischen Definition eines „Busses“). Bei niedrigen Übertragungsraten können die einzelnen Teilnehmer über Stichleitungen an den gemeinsamen Bus angeschlossen werden; bei höheren Übertragungsraten werden spezielle Stecker mit durchgeführten Kontakten benutzt. Dies ist bei Verwendung von Kupferkabeln die am häufigsten eingesetzte Topologie.

Bei der Sterntopologie sind alle Teilnehmer mit einer zentralen Koppeleinrichtung über Punkt-zu-Punkt Verbindungen vernetzt. Jede Kommunikation zwischen den einzelnen Teilnehmern muß über den Sternkoppler erfolgen. Da ein Ausfall des zentralen Sternkopplers zum Ausfall des gesamten Systems führt, muß dieser besonders zuverlässig ausgelegt werden. Beim Einsatz von Lichtwellenleitern ist dies die üblicherweise verwendete Topologie. Im Vergleich zur Bustopologie muß mehr Kabel verlegt werden; dafür ergibt sich jedoch der Vorteil, daß neue Teilnehmer leicht in das System integriert werden können, ohne vorhandene Verkabelung ändern zu müssen. Dies ist ein großer Vorteil bei modular aufgebauten Anlagen mit kleinen physikalischen Abmessungen.

Bei der Ringtopologie sind alle Teilnehmer in einem Ring angeordnet. Direkte Verbindungen bestehen nur zwischen den am Ring benachbarten Teilnehmern; Nachrichten an weiter entfernte Teilnehmer müssen von den unmittelbaren Nachbarn weitergereicht werden. Dieses Verfahren hat den Nachteil, daß der Ausfall eines Teilnehmers oder einer Verbindung zwischen zwei Teilnehmern zum Ausfall des gesamten Rings führt. Diese Topologie wird daher nur von wenigen Feldbussystemen verwendet. Der bekannteste Vertreter der Ringstruktur ist Interbus-S, bei dem

das Medium allerdings nicht ringförmig verlegt wird, sondern die Ringform durch eine geeignete Verschaltung der Datenleiter erreicht wird (siehe [6]).

Aus Sicht der Sicherheitstechnik hat keine der hier vorgestellten Strukturen einen eindeutigen Vorteil gegenüber den anderen. In jedem Fall kann ein einziger Ausfall zum Zusammenbruch der Kommunikation führen. Bei der Busstruktur ist das Busmedium selber die kritische Komponente; bei der Sternstruktur ist es der Sternkoppler, und bei der Ringstruktur jedes Busgerät oder das Verbindungsstück zwischen zwei benachbarten Busgeräten. Für die Ringstruktur ist diese Aussage nicht ganz richtig, da Algorithmen existieren, mit denen ein „Bruch“des Rings toleriert werden kann; dies erfordert allerdings bidirektionale Verbindungen zwischen allen Teilnehmern, und ändert unter Umständen das zeitliche Verhalten der Anordnung massiv. Es muß also in jedem Fall Redundanz vorgesehen werden, wenn die Kommunikationsfähigkeit auch nach einem Ausfall gewährleistet sein muß.

Im folgenden Text ist mit dem Begriff „Bus“ immer ein System zur Kommunikation mehrerer Teilnehmer gemeint und nicht eine Anordnung in Bustopologie.

2.7.2 Zeitliches Verhalten

Ein wesentlicher Unterschied zwischen der konventionellen und der busbasierten Technik liegt in einem sehr differenziertem Zeitverhalten. In der konventionellen Technik wird die Reaktionszeit des Systems im wesentlichen durch die Reaktionszeiten der Sensoren und Aktoren, sowie die Verarbeitungszeiten in der Sicherheitsschaltung oder der sicheren Steuerung bestimmt. In der busbasierten Technik kommt zusätzlich noch die Übertragungszeit über das Medium und die Wartezeit hinzu, bis ein Busteilnehmer senden darf – da sich mehrere Teilnehmer ein Medium teilen, darf nicht jeder Teilnehmer sofort senden, wenn er gültige Daten besitzt, sondern muß warten, bis ihm das Medium zugeteilt wird. Die gebräuchlichen Zuteilungsverfahren sind:

- TDMA – Time Division Multiple Access
  Bei TDMA wird die Kommunikation über den Bus in Zeitscheiben aufgeteilt. Die Zeitscheiben werden vor dem Start des Systems statisch bestimmten Teilnehmern zugeteilt; ein Teilnehmer darf nur innerhalb seiner Zeitscheiben über den Bus senden. Die maximale Wartezeit, bis ein bestimmter Teilnehmer auf den Bus zugreifen darf, ist auf diese Art leicht zu bestimmen. TDMA Protokolle sind bei zyklischem Nachrichtenaustausch die effizientesten, da sie keine Adreßinformationen versenden müssen. Der Sender einer Nachricht kann anhand der verwendeten Zeitscheibe eindeutig identifiziert werden. Voraussetzung hierfür ist jedoch eine genaue Uhrensynchronisation der einzelnen Teilnehmer. Ein bekannter Bus mit TDMA-Zugriff ist der InterBus-S; hier erfolgt die „Uhrensynchronisation“ durch die Rückgewinnung des Taktsignals aus dem empfangenen Datenstrom. Ein weiteres Bussystem mit TDMA-Buszugriffsteuerung ist TTP/C ([8]), das in Zukunft wachsende Bedeutung im Flugzeug- und Automobilbau erhalten wird.

- Master/Slave
  Beim Master/Slave-Verfahren darf ein Slave nur dann senden, wenn er vom Master dazu aufgefordert wird. Die maximale Wartezeit bis zum Senden hängt davon ab, wann der

Master einem Teilnehmer die Sendeerlaubnis erteilt, und läßt sich ebenfalls einfach ermitteln. Master/Slave ist flexibler als TDMA, da der Master sein Zugriffsschema jederzeit verändern kann. Es ist aber auch weniger effizient, da der Master eine explizite Sendeerlaubnis versenden muß. Das Master/Slave-Verfahren wird wegen seiner Einfachheit von vielen Bussen verwendet, z. B. von Profibus (zwischen Master- und Slave-Geräten) und dem „Factory Instrumentation Protocol“ (WorldFIP).

- CSMA/CD – Collision Sense Multiple Access / Collision Detection
  Bei CSMA/CD wartet ein sendewilliger Teilnehmer, bis die Busleitung frei ist und fängt dann an zu senden. Kommt es zu einer Datenkollision, weil zwei Teilnehmer gleichzeitig eine freie Leitung erkannt haben, brechen beide Sender ab und warten eine zufällig bestimmte Zeit, bis sie es erneut probieren. Bei hoher Buslast (ab ca. 70%) behindern sich die einzelnen Teilnehmer gegenseitig so stark, daß der erreichbare Datendurchsatz einbricht. Gleichzeitig kann die maximale Wartezeit bis zum Senden einer Nachricht nicht bestimmt werden. Für sicherheitskritische Anwendungen ist dieses Verfahren daher ungeeignet. Das bekannteste Bussystem mit CSMA/CD ist IEEE 802.3 („Ethernet“), das zur Vernetzung von Bürocomputern verwendet wird.

- CSMA/CA – Collision Sense Multiple Access / Collision Avoidance
  CSMA/CA Funktioniert wie CSMA/CD, nur werden Kollisionen durch eine Arbitrierungsphase vermieden. Dazu gibt es zwei gebräuchliche Methoden. Bei der ersten bekommt jede Nachricht eine Priorität zugewiesen. Während der Arbitrierungsphase kann der Sender einer Nachricht mit höherer Priorität den Sender einer Nachricht mit niedrigerer Priorität überstimmen; dieser bricht seinen Sendeversuch dann ab. Nach der Arbitrierungsphase kann eine Nachricht aber nicht mehr unterbrochen werden. Der bekannteste Vertreter dieser Methode ist das sog. „control area network“ (CAN). Die Priorität ist hier implizit in der Nachrichtenkennung enthalten; die Arbitrierungsphase findet während des Sendens dieser Kennung statt. Da die Arbitrierung bitweise geschieht, muß die Signallaufzeit zwischen zwei Teilnehmern deutlich niedriger sein als die Zeit zum Senden eines Bits. Die physikalische Ausdehnung eines CAN-Netzwerkes ist daher gegenüber anderen Bussystemen stark beschränkt (auf ca. 25-40m bei der maximal möglichen Übertragungsrate von 1MBaud).

  Bei der zweiten Methode zur Vermeidung von Kollisionen muß jeder Busteilnehmer eine festgelegte Zeit warten, nachdem er ein freies Busmedium erkannt hat, bevor er selber senden darf. Die Wartezeit muß für jeden Busteilnehmer unterschiedlich lang sein. Dadurch wird die Anzahl der sinnvoll an einem Bus betreibarer Geräte eingeschränkt. Dieses Verfahren, auch Minislotting genannt, wird z. B. von ARINC 629 (einem Bus für Avionikanwendungen) verwendet.

- Token-Passing
  Beim Token-Passing darf derjenige Teilnehmer senden, der den sog. Token besitzt. Ist er mit der Übertragung fertig, gibt er den Token an den nächsten Teilnehmer weiter. Meist wird eine maximale Zeit vereinbart, in der ein Teilnehmer den Token besitzen darf; in einem solchen Fall ist die maximale Zeit bekannt, die ein Teilnehmer auf eine Sendeerlaubnis warten muß. Obwohl Token-Passing konzeptionell ähnlich einfach ist wie Master/Slave, werden in der Praxis komplizierte Algorithmen benötigt, um bei Störungen den Verlust oder die Verdoppelung des Tokens zu vermeiden. Die bekanntesten Bussysteme mit Token-Passing sind der Token-Ring aus der Bürokommunikation und Profibus aus der Automatisierungstechnik. Profibus verwendet Token-Passing aber nur zur Kommunikation mehrerer Master untereinander; ansonsten verwendet Profibus ein Master/Slave-Buszugriffsverfahren.

Die Übertragungszeit ist abhängig von der Gesamtnachrichtenlänge, d. h. der Anzahl der zu übertragenden Nutzdaten zuzüglich der vom verwendeten Kommunikationsprotokoll benötigten Zusatzinformationen wie Adreßinformationen, Prüfsummen, etc., sowie der physikalischen Übertragungsrate des verwendeten Busses. Die Effizienz eines Protokolls ergibt sich aus dem Verhältnis von Brutto- zu Nettodatenraten, d. h. wie viele Nachrichtenbits gesendet werden müssen, um eine bestimmte Menge an Nutzdaten übertragen zu können. Typische Übertragungsraten liegen zwischen ca. 38.4KBit/s für eigensichere, d. h. in explosionsgefährdeten Bereichen einsetzbare Feldbusse wie z. B. Profibus-PA, und bis zu 12MBit/s bei Profibus-DP. Die Effizienz ist abhängig vom verwendeten Protokoll und der Nutzdatenlänge pro versendeter Nachricht; die Spanne bei der für dezentrale Peripheriesysteme üblichen geringen Anzahl von Nutzdaten pro Gerät reicht von ca. 50% bei TDMA-basierten Systemen wie Interbus-S bis zu unter 10% bei ineffizienten Protokollen wie Profibus (in allen Ausführungen). Für die Sicherheitstechnik sind Bussysteme mit einer hohen Effizienz wünschenswert, da eine hohe Effizienz eine niedrigere und damit weniger störanfällige Übertragungsrate ermöglicht.

2.7.3 Übertragungsfehler

Der für die Sicherheitstechnik größte Unterschied zwischen konventioneller und busbasierter Anschlußtechnik besteht in den bei der Bustechnik zusätzlich möglichen Fehlern.

In vielen Anwendungen der Automatisierungstechnik gilt die Annahme, daß der sichere Zustand – meist die Abschaltung der Aktoren – durch autonome Einzelmaßnahmen in den Peripheriegeräten erreicht werden kann, d. h. es ist kein Zusammenwirken zwischen verschiedenen Anlagenteilen notwendig. Dann ist es für die Sicherheit der Anlage nicht notwendig, die Übertragung einer Nachricht über das Medium zu garantieren; es müssen lediglich Übertragungsfehler sicher erkannt werden.

Ein Busteilnehmer besteht immer aus einer Verarbeitungseinheit, die für die eigentliche Funktion verantwortlich ist, einem Buscontroller und einem Transceiver, der die physikalische Anbindung an das Medium darstellt. Der Buscontroller kann auch mit der Verarbeitungseinheit physikalisch auf dem selben Chip integriert sein; bei einfachen (oder langsamen) Bussen kann die Funktion des Buscontrollers von der Verarbeitungseinheit mit übernommen werden. Bei

sehr einfachen Busteilnehmern kann umgekehrt die eigentliche Funktion in den Buscontroller integriert werden. Auf die weitere Betrachtung der möglichen Fehlerquellen hat dies aber keine entscheidende Bedeutung. Prinzipiell können Fehler in jeder dieser Komponenten auftreten; hier sollen jedoch Fehler in der Verarbeitungseinheit nicht weiter betrachtet werden, da sie nicht Teil der Busübertragung sind.

In Tabelle 2.2 sind die bei der Übertragung von Nachrichten über den Bus zu erwartenden Fehler zusammen mit möglichen Erkennungsmaßnahmen aufgeführt. Diese werden im folgenden näher erläutert.

	Maßnahmen						
Fehler	Nachrichten–zähler	Zeit–stempel	Timeout	Echo	Sender–kennung	Prüf–summe	Redundanz mit Kreuz–vergleich
Verfälschung				X		X	X
Verlust	X		X	X			X
Verzögerung		X	X				
Wiederholung	X	X					X
Einfügung	X			X	X		X
Vertauschung	X	X					X

Tabelle 2.2: Fehlermatrix nach [33]

Eine Verfälschung ist die Änderung eines oder mehrerer Bits einer Nachricht. Dies ist der am häufigsten zu erwartende Fehler; er wird meist durch elektromagnetische Störungen auf dem verwendeten Medium verursacht. In Tabelle 2.3 sind typische Wahrscheinlichkeiten für Einzelbitfehler verschiedener Übertragungsmedien (einschließlich Transceiver) zusammengefaßt.

Ein Verlust ist der Totalverlust einer Nachricht, wobei der vorgesehene Empfänger nicht weiß, daß er eine Nachricht hätte empfangen sollen. Dieser Fehler kann z. B. durch den Ausfall des Senders verursacht werden.

Bei einer Verzögerung trifft eine Nachricht zu spät beim Empfänger ein, um die gewünschte Reaktion noch rechtzeitig auslösen zu können. Dies kann z. B. geschehen, wenn das Medium längere Zeit von anderen Teilnehmern belegt ist.

Bitfehler–wahrscheinlichkeit	**Medium**
10^{-3}	Funkstrecken
10^{-4}	ungeschirmte Telefonleitung
10^{-5}	geschirmte, verdrillte Zweidrahtleitung
$10^{-6} - 10^{-7}$	digitale Telefonleitungen der Deutschen Telekom
10^{-9}	Koaxialkabel in lokalen Anwendungen
10^{-12}	Lichtwellenleiter

Tabelle 2.3: Anzunehmende Bitfehlerwahrscheinlichkeiten nach [33]

Bei einer Wiederholung wird eine früher bereits gesendete, gültige Nachricht zu einem späteren Zeitpunkt wiederholt, zu dem sie dann nicht mehr gültig ist.

Bei einer Einfügung wird eine ungültige Nachricht gesendet. Dies schließt den Fall ein, daß ein Sender eine Nachricht sendet, die vom Empfänger für die Nachricht eines anderen Senders gehalten wird. Letzterer Fall kann insbesondere bei TDMA-Systemen auftreten, wenn ein Teilnehmer während des Zeitschlitzes eines anderen Teilnehmers sendet.

Bei einer Vertauschung empfängt ein Teilnehmer zwei oder mehr aufeinanderfolgende Nachrichten in der falschen Reihenfolge.

Zur Beherrschung dieser Fehler stehen eine Reihe verschiedener Maßnahmen zur Verfügung, die im folgenden kurz vorgestellt werden.

Der Nachrichtenzähler ist ein Zähler, der alle Nachrichten fortlaufend durchnummeriert. Der aktuelle Zählerstand wird mit jeder Nachricht gesendet; der Empfänger kann so prüfen, ob er alle Nachrichten in der richtigen Reihenfolge erhalten hat. Bis auf Verfälschung und Verzögerung lassen sich so bereits alle angesprochenen Fehler erkennen. Allerdings kann ein Nachrichtenverlust erst erkannt werden, wenn eine nachfolgende Nachricht beim Empfänger vorliegt. Werden keine weiteren Nachrichten empfangen, kann der Verlust nicht erkannt werden.

Der Zeitstempel ist die momentane Systemzeit, zu der die Nachricht generiert wurde. Er wird an die Nachricht angehängt und mit verschickt. Wiederholungen und Vertauschungen lassen sich wie bei einem Nachrichtenzähler erkennen, während Verzögerungen erst erkannt werden können, wenn die verzögerte Nachricht beim Empfänger angekommen ist. Ein Nachrichtenverlust kann jedoch nicht erkannt werden.

Ein Timeout ist eine zwischen Sender und Empfänger vereinbarte Zeit, nach der spätestens eine gültige Nachricht empfangen worden sein muß. Sind keine Daten verfügbar, wird eine Leernachricht als „Lebenszeichen“ gesendet (wird im engl. oft als „heartbeat“ bezeichnet). Dies ist das einzige Verfahren, mit dem Nachrichtenverluste und Verzögerungen rechtzeitig erkannt werden können.

Beim Echo sendet der Empfänger die Nachricht an den Sender zurück, der damit prüfen kann, ob der Empfänger die Nachricht richtig empfangen hat. Der Empfänger hat jedoch keine Möglichkeit, selber einen Fehler zu erkennen. Auf den ersten Blick erscheint dieses Verfahren eng mit dem in der Nachrichtentechnik sehr häufig verwendeten Quittieren von Nachrichten verwandt zu sein. Ein Empfänger teilt dabei dem Sender über eine Quittierungsnachricht (engl. *acknowledge*) mit, ob die Nachricht korrekt empfangen wurde. Bei einem erkannten Fehler kann der Sender die Nachricht wiederholen. Dies setzt aber voraus, daß der Empfänger die Korrektheit der empfangenen Nachricht bestimmen kann; dies ist beim Echo jedoch nicht der Fall.

Die Senderkennung ist eine spezielle Kennung, anhand derer der Empfänger den Sender der Nachricht eindeutig identifizieren und so Einfügungen eindeutig erkennen kann.

Eine Prüfsumme wird vom Sender nach einer beliebigen mathematischen Vorschrift über den Nachrichteninhalt berechnet und den eigentlichen Nutzdaten angehängt. Der Empfänger berechnet eine eigene Prüfsumme nach derselben Vorschrift und vergleicht diese mit der empfangenen Prüfsumme. Zur Berechnung der Prüfsummen sollte natürlich ein zur Fehlererkennung

möglichst gut geeignetes Verfahren gewählt werden. In der (drahtgebundenen) Übertragungstechnik wird häufig der sog. „cyclic redundancy check“ (CRC) eingesetzt. Eine gute Übersicht verschiedener Prüfsummenverfahren findet sich in [40]. Aus der Nachrichtentechnik sind auch Prüfsummenverfahren bekannt, mit denen Übertragungsfehler vom Empfänger korrigiert werden können. Im Vergleich zu reinen Fehlererkennungsverfahren mit gleichwertiger Fehlererkennung ist ihr Aufwand grundsätzlich höher, so daß sie nur eingesetzt werden, wenn das Medium sehr störanfällig ist (wie z. B. Satelitenfunkübertragungsstrecken) und ohne Fehlerkorrektur sonst sehr häufig Nachrichten wiederholt werden müßten. Bei leitungsgebundener Übertragungstechnik mit relativ störsicheren Übertragungsmedien (im Idealfall Lichtwellenleiter) erscheint ihr Einsatz verzichtbar.

Bei der Redundanz mit Kreuzvergleich wird jede Nachricht doppelt versendet. Der Empfänger prüft dann beide empfangenen Nachrichten auf Gleichheit. Es sind eine Reihe von Varianten zur Implementierung dieses Verfahrens denkbar; so können z. B. redundante Buscontroller verwendet werden, die sich ein Busmedium teilen. Wird nur ein Buscontroller verwendet, muß die zweite Nachricht leicht abgewandelt werden (z. B. jedes Bit invertiert werden), um den Fall zu vermeiden, daß ein Defekt im Buscontroller beide Nachrichten gleichermaßen verfälscht.

Es ist weder sinnvoll noch notwendig, alle diese Maßnahmen gleichzeitig in einem System zu verwenden. Es ist ausreichend, Maßnahmen miteinander zu kombinieren, die alle Fehlermöglichkeiten abdecken. Ein Timeout wird aber immer benötigt, da nur auf diese Art Nachrichtenverluste und -verzögerungen rechtzeitig erkannt werden können.

Zwei weitere mögliche Fehler sind in obiger Tabelle nicht enthalten. Bei einem sog. „babbling idiot“-Ausfall eines Busteilnehmers blockiert ein Teilnehmer den Bus durch das ständige Senden von Nachrichten, so daß andere Teilnehmer nicht mehr senden können und die Kommunikation zum Erliegen kommt. Bei einem „slightly off specification“ (SOS) Fehler liegt ein Sender etwas außerhalb der Spezifikation (entweder der physikalischen Spezifikation für die Datenübertragung oder der zeitlichen Spezifikation für den Protokollablauf). Da die Empfänger gewisse Toleranzen haben müssen, kann es vorkommen, daß einige Empfänger eine Nachricht empfangen, andere aber nicht. Dies ist nur in Systemen ein Problem, bei denen eine Nachricht von mehr als einem Empfänger empfangen werden soll. In diesem Fall sind spezielle Protokolle notwendig, die sicherstellen, daß entweder keiner der Empfänger oder alle Empfänger eine Nachricht empfangen (siehe z. B. [32]); ansonsten liegt ein asymmetrischer Fehler vor.

2.7.4 End-To-End Prüfsummen

In der Praxis stellt sich die Frage, an welcher Stelle die im vorherigen Abschnitt vorgestellten Maßnahmen zur Fehlererkennung konkret umgesetzt werden sollen. Handelsübliche Buscontroller berechnen ein Prüfsumme, um Nachrichtenverfälschungen zu erkennen. Diese Prüfsumme kann aber i. allg. keine Fehler im Buscontroller selber oder bei der Kommunikation zwischen Buscontroller und Verarbeitungseinheit erkennen. Es bleiben letztlich drei Möglichkeiten, sichere Übertragung über einen Feldbus zu realisieren.

Die erste Möglichkeit besteht darin, den Bus gar nicht als sicheres Kommunikationsmedium zu

verwenden, sondern die Sicherheit der Anlage durch strukturelle Maßnahmen auf Systemebene zu gewährleisten. In Kapitel 4.4.4 wird diese Möglichkeit eingehender beschrieben.

Die zweite Möglichkeit ist die erwähnte Redundanz mit Kreuzvergleich, wobei der Vergleich der Nachrichten in einer der Buscontroller übergeordneten Instanz durchgeführt wird. Der gesamte Datenpfad durch die unterhalb dieser Instanz liegenden Komponenten kann so geprüft werden. Die Fehlererkennung ist unabhängig von der verwendeten Busanschaltungshardware, so daß keine Änderungen am Protokoll oder den verfügbaren Bausteinen vorhandener Busse notwendig sind. Nachteilig an dieser Lösung ist, das jede Nachricht doppelt versendet werden muß; bei der Verwendung redundanter Buscontroller entsteht auch noch ein zusätzlicher Hardwareaufwand.

Als letzte Möglichkeit bleibt die Verwendung einer sog. End-To-End Prüfsumme. Bild 2.7 stellt die prinzipielle Funktion einer End-To-End Prüfsumme dar. Sie wird von einer dem Buscontroller übergeordneten Instanz berechnet und den eigentlichen Nutzdaten zugefügt. Wie bei der Redundanz mit Kreuzvergleich kann auf diese Weise der gesamte Datenpfad zwischen der Erzeugung der Prüfsumme im Sender und der Prüfung im Empfänger überwacht werden. Wie dort müssen die verwendeten Bussysteme nicht modifiziert werden. Im Gegensatz zur Redundanz mit Kreuzvergleich ist der zusätzliche Resourcenaufwand jedoch geringer, da Nachrichten nur etwas verlängert werden, und nicht doppelt gesendet werden müssen.

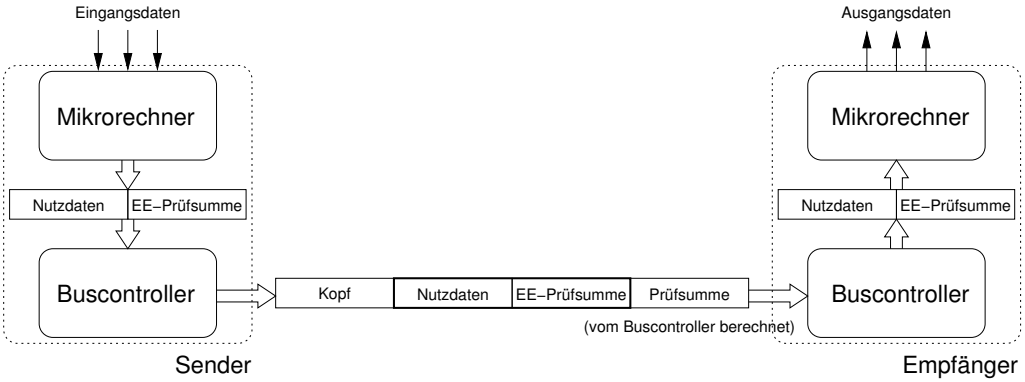

Bild 2.7: Prinzip einer „End-To-End“ Prüfsumme

3 Stand der Technik

3.1 Beispiele für fehlertolerante Systeme

Die große Vielfalt von möglichen Anwendungen spiegelt sich in realen Systemen auch in der Art des Peripherieanschlusses wieder; hier gibt es kein eindeutig bevorzugtes Lösungskonzept. Auch für ähnliche Anwendungsgebiete ergeben sich teilweise erhebliche Unterschiede bei den gewählten Lösungen.

Dieses Kapitel ist in zwei große Teile untergliedert. Im ersten Teil werden verschiedene Peripherieanschlußkonzepte vollständiger Systeme aus unterschiedlichen Anwendungsgebieten vorgestellt. Im zweiten Teil werden dann Sicherheitsbussysteme vorgestellt, die speziell für Anwendungen im Werkzeugmaschinen- und Anlagenbau sowie der chemischen Industrie entwickelt worden sind.

3.1.1 Das fehlertolerante Mehrrechnersystem FUTURE

Ein früher Ansatz für ein universelles, für Automatisierungsaufgaben geeignetes fehlertolerantes Rechnersystem ist das am ehemaligen Lehrstuhl für Prozeßrechner (heute Lehrstuhl für Realzeit-Computersysteme) der Technischen Universität München Anfang der 80er Jahre entstandene FUTURE. Obwohl FUTURE außerhalb der Forschung keine Bedeutung erlangt hat, ist dieses System insofern von Interesse, weil es als Grundlage für nachfolgende Systeme gedient hat (z. B. die im folgenden Abschnitt vorgestellte „Rechnergesteuerte Bahnübergangstechnik“, die erfolgreich eingesetzt wird).

Die prinzipielle Struktur eines FUTURE-Systems ist in Bild 3.1 dargestellt. Es besteht aus 4 bis 10 einzelnen Rechnern. Jede Aufgabe des Systems wird, je nach Anforderungen an die Sicherheit oder Verfügbarkeit, mit der sie ausgeführt werden soll, von mehreren Rechnern parallel redundant bearbeitet; diese bilden eine sog. „Taskbearbeitungsgruppe“. Die Rechner einer solchen Taskbearbeitungsgruppe tauschen die Ergebnisse von Berechnungen über das Kommunikationssystem miteinander aus und votieren sie. Aufgaben ohne besondere Anforderungen können auch nur von einem Rechner ausgeführt werden; eine Votierung entfällt dann.

Für das FUTURE-System sind eine Reihe von verschiedenen Peripherieanschlußkonzepten vorgeschlagen worden ([14]), von denen die Variante „Modulbus“ prototypisch realisiert wurde (siehe z. B. [9] und [12]). Bei diesem Konzept wird die Peripherie über intelligente E/A-Module angeschlossen, die jeweils über eine eigene serielle Busleitung an die Rechner angeschlossen

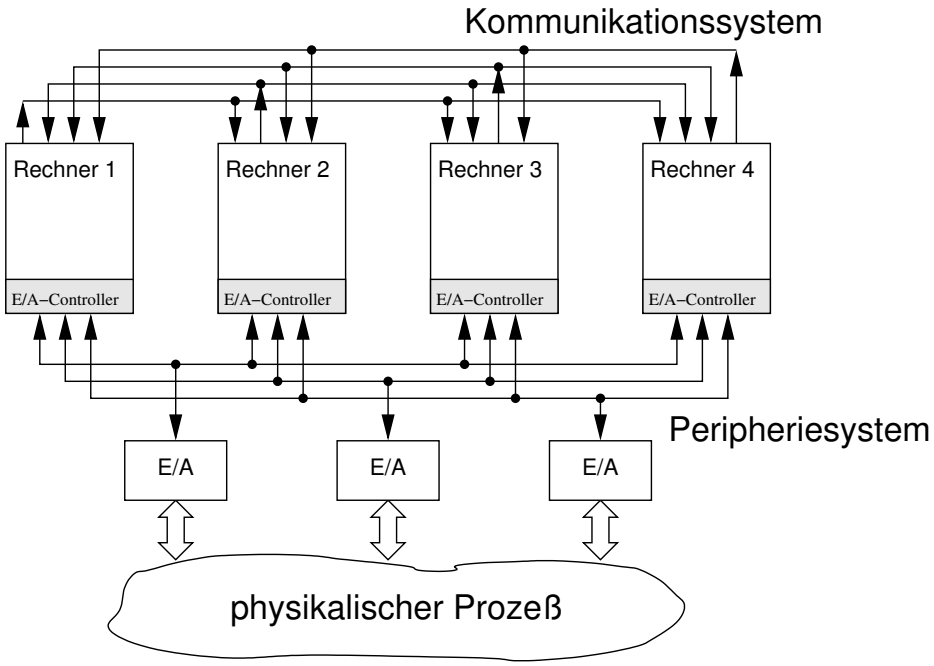

Bild 3.1: Aufbau eines FUTURE-Systems (nach [12])

sind. Jeder Rechner besitzt dazu einen E/A-Controller mit acht asynchronen seriellen Schnittstellen. Ein E/A-Modul wird an alle Rechner angeschlossen, die zur Erfüllung ihrer zugewiesenen Aufgaben Daten mit diesem Modul austauschen müssen. Jeweils ein Rechner übernimmt eine Masterfunktion und sendet Befehle an das Modul, während die übrigen Rechner nur passiv die vom Modul gesendeten Antworten empfangen. Die Masterfunktion ist nicht statisch festgelegt, sondern kann dynamisch im Betrieb gewechselt werden.

Um zu verhindern, daß ein ausgefallener Rechner die Kommunikation der anderen Rechner mit den E/A-Modulen stört, sieht das Konzept vor, daß jeder Rechner eine Abschaltleitung besitzt, mit der die Spannungsversorgung seines Buskopplers abgeschaltet werden kann, wenn mindestens zwei andere Rechner dies fordern. Ebenso ist vorgesehen, ein spezielles Protokoll zur Erkennung von Übertragungsfehlern zu verwenden; die genauen Details wurden aber nicht erarbeitet. Die existierende Implementierung sichert lediglich das erste Byte eines Befehls mit einem (8,4)-Hamming Code ab, der Doppelfehler erkennen und Einfachfehler beheben kann. Die übrigen Bytes werden lediglich durch ein Paritätsbit gesichert.

3.1.2 Rechnergesteuerte Bahnübergangstechnik

„Rechnergesteuerte Bahnübergangstechnik“ (kurz RBÜT, siehe [1]) ist eine Entwicklung der Firma Pintsch Bamag zur Steuerung von Bahnübergängen. Ein RBÜT-System besteht aus drei redundanten Steuerungsrechnern, einem zentralen Abschalter, sowie einer je nach Bahnübergang konfigurierbarer Anzahl von Ein-/Ausgabemodulen. Die Rechner arbeiten im 2-von-3 Betrieb identische Software ab. Der Abschalter kann im Fehlerfall einzelne Baugruppen oder die

gesamte Anlage abschalten. Die Module werden über einen dreifach redundanten Peripheriebus angeschlossen, der ein proprietäres TDMA-basiertes Protokoll verwendet. Neben der Ansteuerung der E/A-Module verwenden die Rechner die Peripheriebusse auch zur Synchronisation untereinander.

In Bild 3.2 ist die prinzipielle Struktur des Peripherieanschlusses dargestellt. Ein Rechner sendet jeweils nur auf einem der angeschlossenen Busse; von den anderen Bussen wird nur empfangen. Es gibt drei Arten von E/A-Modulen – eine zur Auswertung der Zugerkennungssensorik, eine zur Ansteuerung der Motoren für die Schranken, und eine für die Lichtsteuerung. Ihr interner Aufbau ist aber bis auf die Leistungselektronik weitgehend identisch. Ein anwendungsspezifischer integrierter Schaltkreis (ASIC) ist für die Ankoppelung an die Peripheriebusse und die Votierung der empfangenen Nachrichten verantwortlich. Ein E/A-Modul enthält jeweils zwei solcher ASICs, die sich gegenseitig überwachen. Bei einer Abweichung wird das gesamte Modul abgeschaltet.

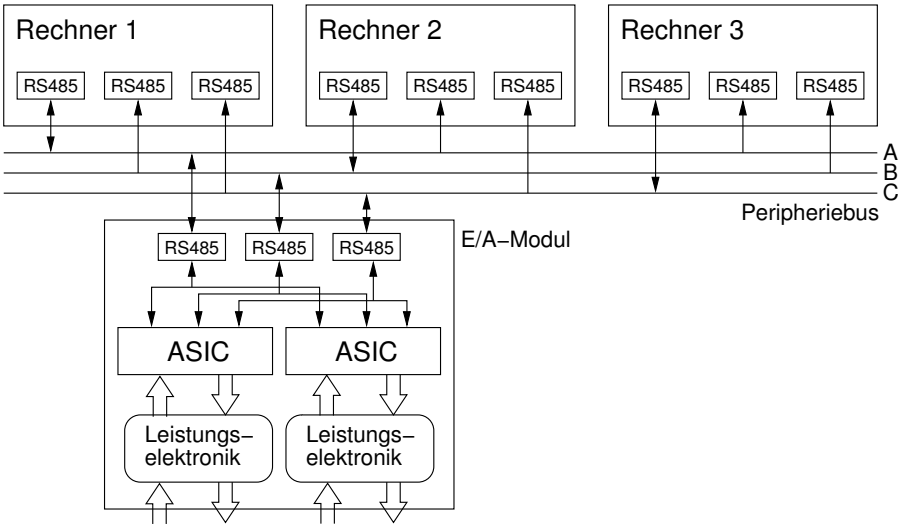

Bild 3.2: Peripherieanschluß bei RBÜT, nach [1]

Rechner und E/A-Module befinden sich alle im selben Gehäuse. Die eigentliche Sensorik und Aktorik sind über Parallelverdrahtung an die Leistungselektronik der E/A-Module angeschlossen; es handelt sich also nicht um ein (Feld)Bussystem im Sinne dieser Arbeit.

3.1.3 Airbus A320 Flugsteuerung

Der Airbus A320 ist das erste zivile Flugzeug, das vollständig über ein digitales Fly-by-Wire System gesteuert wird ([10]). Die Steuerkommandos des Piloten bzw. des Autopiloten werden elektrisch an die Flugsteuerungsrechner übermittelt, die daraus die entsprechenden Anstellwinkel für die Steuerflächen des Leitwerks berechnen und einstellen. Die Rechner verhindern dabei gefährliche Flugzustände wie z. B. Strömungsabriß durch zu niedrige Geschwindigkeit oder eine Überschreitung der maximal zulässigen mechanischen Belastung durch zu hohe Sinkgeschwindigkeit.

In Bild 3.3 ist das Leitwerk der A320 mit den beteiligten Rechnern schematisch dargestellt. Sämtliche Steuerflächen werden von hydraulischen Aktoren eingestellt. Das Flugzeug besitzt drei unabhängige Hydraulikkreise, von denen einer ausreicht, um das Flugzeug fliegen zu können. Nicht dargestellt ist die notwendige Sensorik – dies sind zum einen Rückmeldungen der Aktoren, und zum anderen Sensoren zur Bestimmung der Fluglage und der Befehle des Piloten (Kreiselkompaß, Steuerknüppel, etc.). Jeder Sensor ist mindestens doppelt redundant vorhanden; jede vom System verwendete Information wird von mindestens zwei verschiedenen Quellen ermittelt.

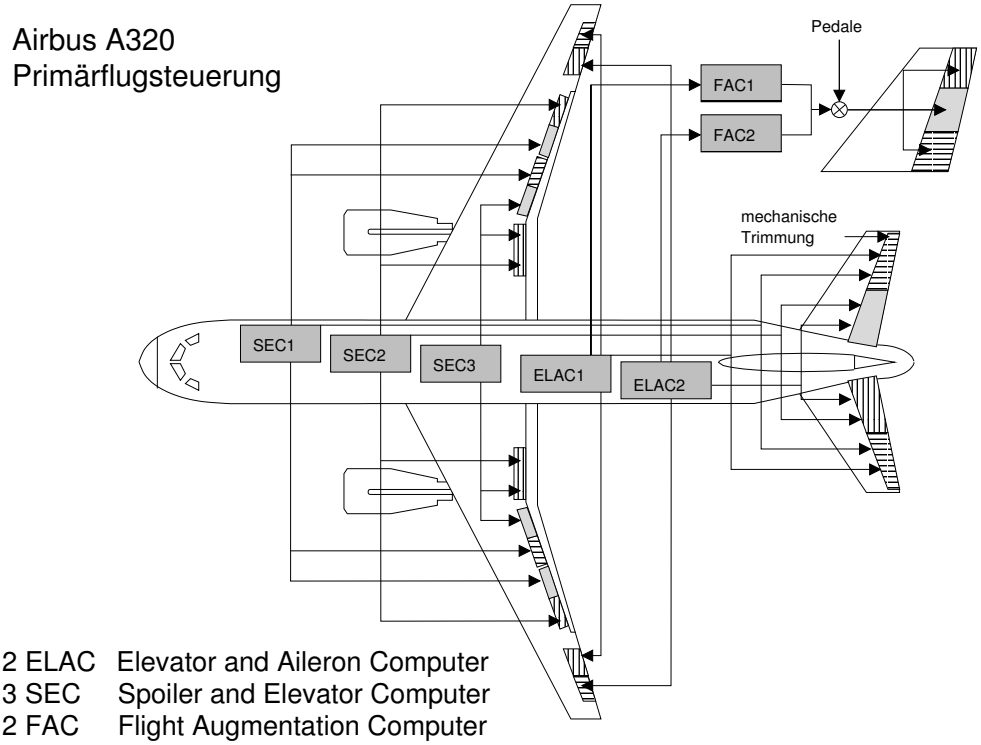

Bild 3.3: Airbus A320 Fly-by-Wire Flugsteuerung ([17])

Das System verwendet drei verschiedene Rechnertypen – den „Elevator and Aileron Computer“ (ELAC), „Spoiler and Elevator Computer“ (SEC), und den „Flight Augmentation Computer“ (FAC). Diese Rechner besitzen alle dieselbe Grundstruktur, die in Bild 3.4 dargestellt ist. Ein Rechner besteht aus einem Steuerungskanal, der die Aktorik ansteuert, und einem Überwachungskanal, der die Funktion des Steuerungskanals überprüft. Tritt eine Abweichung zwischen beiden Kanälen auf, wird der komplette Rechner über Relais von den Aktoren getrennt. Um auch systematische Fehler zu berücksichtigen, wurden die verschiedenen Rechnertypen von unterschiedlichen Firmen entwickelt. Sie verwenden unterschiedliche Mikroprozessoren (die ELACs verwenden Motorola 68010, während die SECs Intel 80186 Prozessoren verwenden). Ebenso wurde die Software vollständig diversitär entwickelt – die Software für Steuerungskanal und Überwachungskanal von ELAC und SEC wurden jeweils von unterschiedlichen Entwicklergruppen in unterschiedlichen Programmiersprachen erstellt.

Jeder Aktor wird nur von genau einem Rechner angesteuert. Die Aktoren sind so auf die ein-

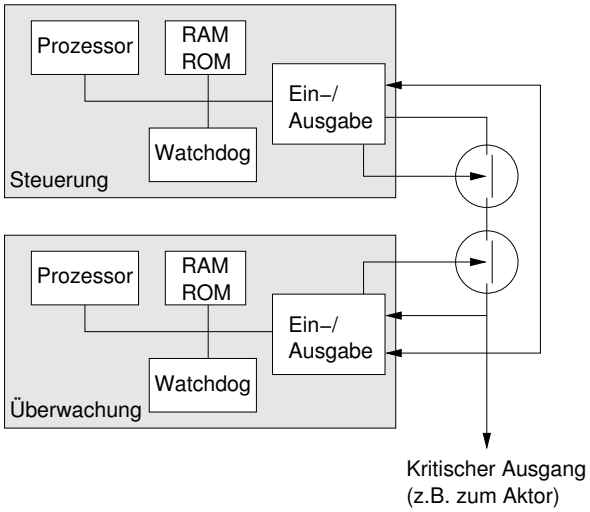

Bild 3.4: Flugsteuerungscomputer des Airbus A320 Fly-by-Wire Systems (nach [10])

zelnen Rechner aufgeteilt, daß auch nach dem Ausfall mehrerer Rechner das Flugzeug noch sicher geflogen werden kann. So wird z. B. im normalen Betrieb der Steigwinkel über das linke und rechte Höhenruder und die Höhenflossentrimmung eingestellt, die von ELAC2 angesteuert werden. Fällt ELAC2 oder einer seiner Aktoren aus, übernimmt ELAC1 seine Funktion; ebenso kann ELAC1 von SEC2 und SEC2 von SEC1 ersetzt werden. Die geforderte Sicherheit könnte auch mit nur drei Rechnern erreicht werden; das Flugzeug darf daher auch mit einem bereits beim Start ausgefallenen Rechner starten.

Die Entscheidung der Entwickler, die Aktorik direkt an die Steuerungsrechner anzuschließen, hat ihren Preis – die verwendeten Rechner sind hochspezialisiert und können nicht für andere Anwendungen verwendet werden, nicht einmal für andere Flugsteuerungssysteme. Die Airbusmodelle A330 und A340 haben mehr und größere Steuerflächen und verwenden daher andere Steuerrechner.

3.1.4 Fehlertolerantes Rechnersystem für Raumfahrtanwendungen

Ein völlig anderes Peripherieanschlußkonzept wird von einem fehlertoleranten Computer (FTC) der (ehemaligen) Daimler-Benz Aerospace, heute European Aerospace and Defence Systems (EADS) verfolgt ([39]). Dieses System ist für verschiedene Avionikanwendungen in der Raumfahrt gedacht. Bild 3.5 zeigt einen Entwurf für die Antriebssteuerung des Automated Transfer Vehicle (ATV) der ESA. Das ATV ist als Transporterfahrzeug für die Versorgung der Internationalen Raumstation ISS gedacht.

Der FTC besteht aus maximal vier sog. „Spuren“ (engl. *lanes*). Eine Spur ist in drei Ebenen aufgeteilt – je eine für das Avionik-Interface (engl. *avionics interface layer*, AVI), die Fehlertoleranzmechanismen (engl. *fault management layer*, FML) und die eigentliche Anwendung

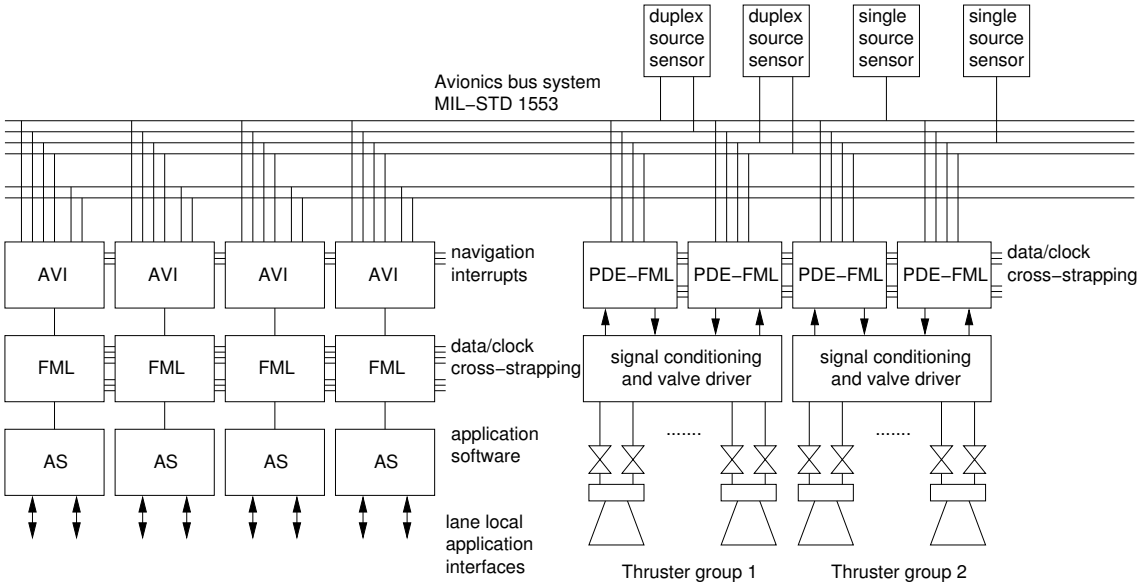

Bild 3.5: Fehlertolerante Architektur für Raumfahrtanwendungen (nach [39])

(engl. *application layer*, AL[1)]). Jede dieser Ebenen kann in einem eigenen Rechner implementiert sein. Peripherie kann entweder lokal, oder über das Avionikbussystem an die AL angeschlossen werden. Das Avionikbussystem besteht aus sechs MIL-STD 1553A Bussen, von denen vier vom FTC als Master gesteuert werden (je einer dieser Busse wird von jeder Spur gesteuert). Auf den übrigen zwei Bussen tritt der FTC als Slave auf; auf diese Weise kann der FTC mit anderen Steuerungssystem im Raumfahrzeug kommunizieren.

Die einzelnen Spuren sind über die FML lose miteinander gekoppelt und werden zehn mal pro Sekunde miteinander synchronisiert. Die Spuren kommunizieren über ein eigenes Punkt-zu-Punkt Kommunikationssystem miteinander (in Bild 3.5 mit „data cross-strapping„ bezeichnet). Als Zeitgeber steht in jeder Spur eine eigene, fehlertolerante Uhr zur Verfügung. Die Uhren der Spuren sind ebenfalls miteinander verbunden („clock cross-strapping“).

Die Antriebssteuerung hat die Aufgabe, die Sauerstoff- und Treibstoffzufuhr der einzelnen Antriebsdüsen zu steuern. Die Steuerung soll auch nach zwei beliebigen (fail-uncontrolled) Ausfällen noch funktionsfähig sein. Die Struktur der einzelnen Komponenten des Antriebs sind in Bild 3.6 dargestellt.

Die Antriebsdüsen sind in zwei Gruppen aufgeteilt. Die Gruppierung ist dabei so gewählt, daß das Raumfahrzeug auch mit nur einer Gruppe von Antriebsdüsen manövrierfähig ist, sofern eine einzelne Antriebsdüse fail-silent Verhalten aufweist. Die Treibstoff- und Sauerstoffzufuhr werden über drei hierarchisch angeordnete Ebenen von Ventilen geregelt. Fällt eine Antriebsdüse aus, werden die entsprechende Ventile der untersten Ebene („level 1“ im Bild) geschlossen. Kann ein Ventil nicht geschlossen werden, werden die Ventile einer Untergruppe von drei Antriebsdüsen auf der mittleren Ebene geschlossen („level 2“im Bild; die Untergruppen sind nicht dargestellt). Kann auch eines dieser Ventile nicht geschlossen werden, werden die Ventile ei-

[1)] Im Bild ist die Anwendungsebene mit „AS“ gekennzeichnet.

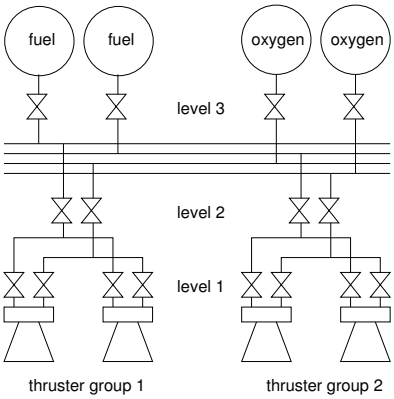

Bild 3.6: Schematische Darstellung der Aktorik der Antriebssteuerung für das ATV ([39])

ner vollständigen Gruppe von Antriebsdüsen auf der obersten Ebene („level 3“) geschlossen. Auf diese Weise kann das fail-silent Verhalten einer Antriebsdüse auch nach zwei beliebigen Ausfällen garantiert werden.

Die Ventile einer Antriebsgruppe werden von je zwei Antriebssteuereinheiten (engl. *propulsion drive electronic*, PDE) gesteuert, die über das Avionikbussystem mit dem FTC verbunden sind. Jede PDE-Einheit entspricht dabei einer FML-Baugruppe des FTC, bei dem die Avionikanbindung und die Anwendungsebene zusätzlich mit in die Baugruppe integriert sind. Zusammen bilden die vier PDE-Einheiten einen eigenen vierspurigen fehlertoleranten Rechner nach dem Muster des FTC.

Die für die Antriebssteurung notwendige Sensorik wird ebenfalls über MIL-STD 1553A Busterminals angeschlossen. Es gibt Sensoren mit einem oder mit zwei Busanschlüssen.

3.1.5 Generic Upgradeable Architecture for Real-Time Dependable Systems

Die bisher beschriebenen Systeme sind für ein relativ kleines, klar umrissenes Anwendungsgebiet gedacht. Im Gegensatz dazu ist die „Generic Upgradable Architecture for Real-Time Dependable Systems“ (GUARDS, siehe [27]) für ein breites Anwendungsgebiet gedacht. Die Architektur wurde von einem Konsortium aus akademischen Forschungseinrichtungen und Industriepartnern entwickelt. Die anvisierten Anwendungsgebiete reichen dabei von Steuerungssystemen für die Eisenbahn, über Steuerungssysteme für nukleare Unterseeboote bis hin zu Raumfahrtanwendungen mit extremen Anforderungen an die Sicherheit und Verfügbarkeit.

Grundidee von GUARDS ist es, eine generische Architektur zu definieren, aus der dann anwendungsspezifische Instanzen abgeleitet werden können. Da die Architektur auf möglichst viele Anwendungsgebiete anwendbar sein soll, bietet GUARDS dem Entwickler viele Freiheiten beim Systementwurf. Bild 3.7 zeigt eine Übersicht über den Entwurfsraum.

Das Grundelement von Guards ist der Kanal. Die Fehlertoleranz des Gesamtsystems basiert

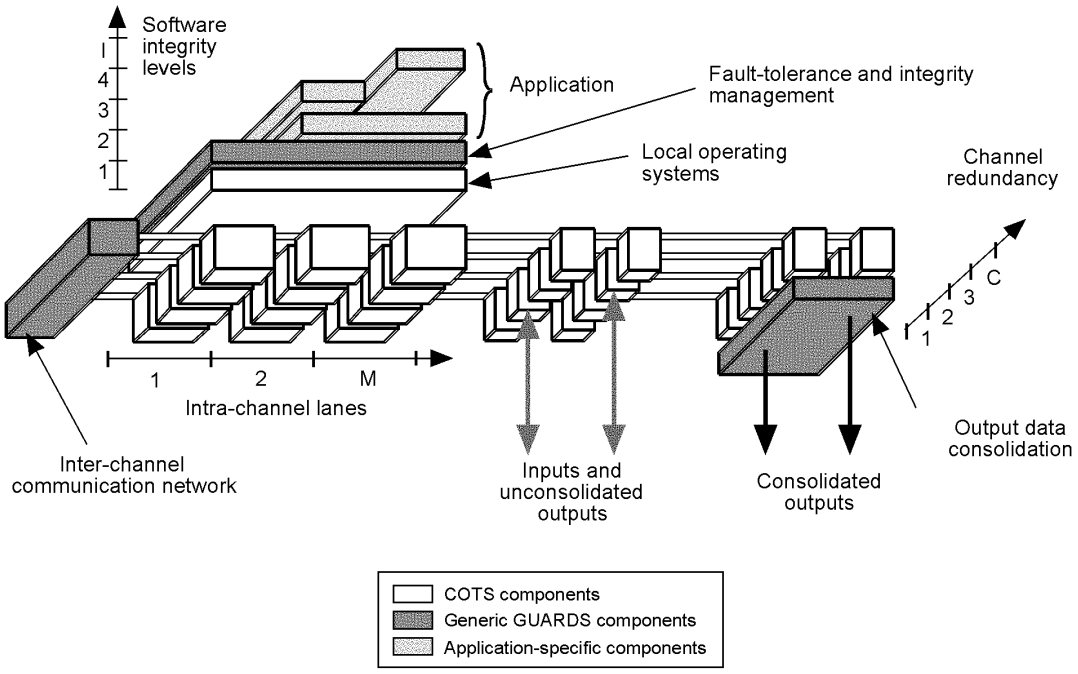

Bild 3.7: Mögliche Struktur eines GUARDS-Systems ([27])

auf dem parallelen Betrieb von $1..n$ Kanälen, die jeweils dieselben Eingangsdaten erhalten, und im fehlerfreien Betrieb dieselben Ausgangsdaten berechnen. Jeder Kanal besitzt einen eigenen Kommunikationsprozessor, der über Punkt-zu-Punkt Verbindungen mit allen anderen Kanälen verbunden ist (im Bild sind alle Kommunikationsprozessoren zum sog. *inter-channel communication network* zusammengefaßt). Der Kommunikationsprozessor votiert alle Nachrichten selbstständig und synchronisiert auch die Uhren der Kanäle untereinander.

Ein einzelner Kanal besteht aus einem oder mehreren sog. Spuren (engl. *lanes*). Jede Spur ist eine eigenständige Verarbeitungseinheit. Die auf einem Kanal ablaufende Verarbeitung kann auf mehreren Spuren redundant ausgeführt werden, um mittels Vergleich zwischen den Ergebnissen der Spuren die Fehlererkennung innerhalb eines Kanals zu verbessern. Alternativ kann die Verfügbarkeit eines Kanals erhöht werden, wenn ausgefallene Spuren abgeschaltet werden können. Es ist aber auch möglich, die zusätzlichen Spuren zur Steigerung der Rechenleistung des Kanals zu verwenden.

Ferner besitzt jeder Kanal eigene Sensorik und Aktorik, die über COTS E/A-Baugruppen an den Kanal angeschlossen wird. GUARDS spezifiziert nicht, wie diese Anschaltung konkret aussieht, d. h. es sind sowohl parallel verdrahtete, als auch busbasierte Systeme möglich (in letzterem Fall hätte jeder Kanal seinen eigenen Feldbus). Die redundanten Ausgänge aller Kanäle werden über eine zusätzliche Votierungsschicht, die sog. *output consolidation layer*, auf die physikalischen Ausgänge des Systems abgebildet. Um einen Engpass bei Sicherheit oder Verfügbarkeit zu vermeiden, sollte diese Schicht so weit wie nur möglich in den physikalischen Prozeß hineinreichen. Bei einem Flugsteuerungssystem könnten z. B. mehrere redundante Aktoren auf eine Steuerfläche einwirken. Die Steuerfläche fungiert als physikalischer Voter, indem die von den

Aktoren aufgebrachten Kräfte summiert werden und somit eine einfache Mehrheitsentscheidung getroffen wird. Alternativ könnte die Steuerfläche von einem ausgewählten Kanal von nur einem Aktor gesteuert werden. Fällt der Kanal oder der Aktor aus, wird auf einen Reserveaktor umgeschaltet, der an einem anderen Kanal angeschlossen ist (auf diese Weise arbeitet auch die Flugsteuerung des Airbus A320, siehe auch Abschnitt 3.1.3). Dies funktioniert aber nur dann, wenn der Aktor fail-silent ist. Ebenso sind je nach Anwendungsgebiet auch andere Lösung möglich. Für GUARDS ist die Votierungsschicht daher eine anwendungsspezifische Komponente, die nicht näher spezifiziert wird.

Lediglich für diskrete digitale und analoge Ausgänge existiert ein generisches Anschlußkonzept. Die redundanten Ausgänge aller Kanäle werden elektrisch voneinander isoliert und über eine sog. „wired-or“ Schaltung miteinander überlagert. Mittels zusätzlicher Sensorik überwachen alle Kanäle den Zustand der eigenen Ausgänge und den der anderen Kanäle; eine einfache Mehrheitsentscheidung bestimmt, welcher Kanal die Aktorik tatsächlich ansprechen darf. Die Ausgänge eines fehlerhaften Kanals können von den anderen Kanälen über eine zusätzliche Auswahlschaltung vom Rest des Systems getrennt werden. Da beim Umschalten zwischen den Kanälen kurzzeitig ungültige Ausgangspegel anliegen können, muß das Gesamtsystem entsprechend entworfen werden.

3.1.6 Time Triggered Architecture

Die von der Arbeitsgruppe um Professor Kopetz an der Technischen Universität Wien entwickelte „Time Triggered Architecture“ (TTA, siehe [8]) ist eine Architektur für verteilte Realzeitsysteme in Anwendungen mit hohen Anforderungen an Sicherheit und Verfügbarkeit. Eine Reihe von Firmen evaluieren derzeit diese Architektur für Anwendungen wie z. B. elektronisch gesteuertes Bremsen im Auto („break by wire“).

Der zentrale Gedanke der TTA ist die strikte Zeitsteuerung aller Systemaktivitäten, d. h. jede Aktivität wird zu einer beim Entwurf des Systems festgelegten Zeit ausgeführt (vgl. TDMA-Buszugriff, Kapitel 2.7.2). Dies betrifft nicht nur die Kommunikation der verteilten Knoten untereinander, sondern auch die Verarbeitung auf den Knoten selber.

Ein TTA-System besteht aus mehreren fail-silent Rechnerknoten, den sog. „smallest replaceable units“ (SRU). Die SRUs werden über das Bussystem TTP/C miteinander gekoppelt, das eine fehlertolerante Kommunikation zwischen den SRUs ermöglicht. Mehrere SRUs können (logisch) zu einer sog. „fault tolerant unit“ (FTU) zusammengefaßt werden. Die SRUs einer FTU synchronisieren sich über den TTP/C Bus; eine eigene Fehlertoleranzschicht innerhalb der SRUs sorgt dafür, daß dies für die auf den SRUs laufende Anwendungssoftware transparent geschieht.

Innerhalb der TTA wird Peripherie über den TTP/A Feldbus angeschlossen. TTP/A ist wie TTP/C ein zeitgesteuerter Bus; es handelt sich aber um ein wesentlich einfacheres Bussystem ohne die Fehlertoleranzmechanismen von TTP/C. Eine mögliche Konfiguration ist in Bild 3.8 dargestellt. Drei SRUs sind zu einer FTU zusammengefaßt. Die Peripherie wird über drei TTP/A Busse angeschlossen; jede SRU ist mit jedem Bus verbunden. Jeweils eine SRU der FTU

ist der Master für einen solchen Bus; fällt sie aus, kann eine andere SRU ihre Rolle übernehmen. Sensorbusanschaltungen werden nur an jeweils einen Bus angeschlossen. Eine SRU empfängt Sensordaten über alle angeschlossenen TTP/A Busse und votiert die empfangenen Daten. Eine Aktorbusanschaltung wird dagegen an alle Busse angeschlossen und empfängt Befehle von allen SRUs. Die empfangenen Befehle werden in der Busanschaltung votiert und die Aktorik entsprechend angesteuert.

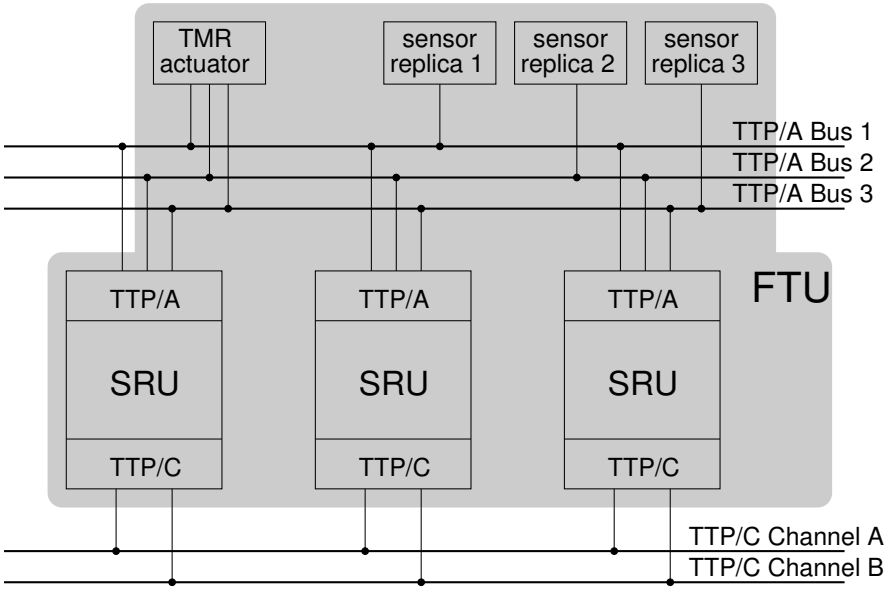

Bild 3.8: Peripherieanschluß in einem TTA-System, nach [8]

3.2 Sicherheitsbussysteme

Bei den bisher vorgestellten Systemen werden die für den Peripherieanschluß verwendeten Bussysteme nicht als sicheres Kommunikationsmedium eingesetzt, d. h. mögliche Fehler bei der Datenübertragung werden durch strukturelle Maßnahmen auf höherer Ebene (z. B. mehrere parallel betriebene Busse) erkannt. Dies ist für viele Anwendungen im (Werkzeug)Maschinenbau aus Kostengründen nicht akzeptabel. Nachdem mit der Neufassung der Europäischen Norm EN60204 im Jahre 1998 der Einsatz von Bussystemen zur Übertragung von sicherheitsrelevanten Signalen nicht mehr ausdrücklich verboten ist, sind daher für diesen Bereich verschiedene busbasierte Lösungen entwickelt worden.

Die ersten verfügbaren Sicherheitsbussysteme wie z. B. SafetyBus p oder „Elan Safety Local Area Network“(ESALAN) sind speziell für die Sicherheitstechnik entwickelt worden und erlauben keinen gemischten Betrieb mit normalen, nicht sicheren Buskomponenten. Da der Anschluß nicht sicherheitsrelevanter Ein- und Ausgänge an eine sichere Busanschaltung unwirtschaftlich ist, wird der Sicherheitsbus dann parallel zu einem normalen Feldbus verwendet. Dies ist in größeren Anlagen (z. B. bei Pressenstraßen) vom Aufwand her vertretbar; bei kleineren Ma-

schinen und Anlagen soll aber möglichst nur ein Bus verwendet werden. Daher wurden bzw. werden die meisten Standardfeldbusse für den Einsatz in der Sicherheitstechnik erweitert.

Im folgenden werden fünf verschiedene Lösungsansätze vorgestellt. In allen Fällen wird die sichere Übertragung von Daten durch eine zusätzliche Protokollschicht erreicht, die dem normalen Busprotokoll aufgesetzt wird; dies gilt auch für den reinen Sicherheitsbus SafetyBus p, der direkt auf dem Control Area Network (CAN) aufsetzt. Alle vorgestellten Konzepte sind nur für einen fail-safe Betrieb ausgelegt. Die Lösungen unterscheiden sich dennoch erheblich voneinander; zum einen in der Art der verwendeten Fehlererkennungmaßnahmen, zum anderen in dem Sicherheitskonzept für das Gesamtsystem.

3.2.1 SafetyBus p

SafetyBus p der Firma Pilz ([16]) war eines der ersten verfügbaren sicheren Bussysteme für die Automatisierungstechnik. Dieser Bus verwendet CAN (Control Area Network) als Grundlage für die Datenübertragung, dem ein eigenes Protokoll aufgesetzt wird. Im Gegensatz zu den meisten anderen Feldbussystemen, bei denen die Kommunikation mit der Peripherie zyklisch abläuft, bleibt SafetyBus p dem ereignisorientierten Ansatz von CAN treu, d. h. ein Teilnehmer sendet nur dann eine Nachricht, wenn ein Ereignis (z. B. die Änderung eines Sensorwertes) aufgetreten ist. Um dennoch Teilnehmerausfälle erkennen zu können, sendet das sog. „Management Device“ periodisch eine Verbindungskontrollnachricht an alle Busgeräte, die von den Empfängern beantwortet werden muß.

Eine SafetyBus p Nachricht enthält einen Nachrichtenzähler, eine explizite Sender- und Empfängeradresse, sowie eine CRC-Prüfsumme, die als End-To-End Prüfsumme genutzt wird (siehe Kapitel 2.7.3 und 2.7.4). Diese Informationen werden zusammen mit den Nutzdaten in einer normalen CAN Nachricht „verpackt“ und gesendet. Der Empfänger muß den Empfang einer Nachricht mit einer Quittierungsnachricht an den Sender bestätigen.

SafetyBus p behält die aus der konventionellen Sicherheitstechnik bekannte strikte Trennung zwischen sicherheitsrelevanter und normaler Funktionalität bei. Der Anschluß normaler Komponenten an den Bus ist nicht vorgesehen; SafetyBus p wird daher zusätzlich zu einem normalen Feldbus verwendet.

3.2.2 ProfiSafe

ProfiSafe ist die sichere Erweiterung von Profibus DP und Profibus PA. Zentraler Bestandteil der Erweiterung ist eine zusätzliche End-To-End Prüfsumme (wahlweise CRC-16 oder CRC-32, je nach Länge der zu sicherenden Nachricht), die normalen Profibus-Nachrichten zugefügt wird. Diese Prüfsumme wird nicht nur über die eigentlichen Nutzdaten, sondern auch über die sog. F-Parameter des Senders berechnet. Die F-Parameter enthalten alle für die sichere Kommunikation benötigten Parameter; dies sind z. B. eine eindeutige Kennung für die Sender-/Empfängerkombination einer Nachricht (damit kann der Sender eindeutig identifiziert werden),

eine Überwachungszeit, innerhalb derer eine Kommunikation mit dem Gerät stattgefunden haben muß, die Länge der End-To-End Prüfsumme (2 oder 4 Bytes), usw.. Zusätzlich können auch noch gerätespezifische Parameter wie die Positionsdaten von Schutzfeldern eines Laserscanners in die Prüfsumme mit einbezogen werden. Auf diese Weise können nicht nur Fehler bei der Kommunikation, sondern auch Veränderungen an der Konfiguration eines Gerätes erkannt werden. Weiterhin werden alle Nachrichten mit einem Nachrichtenzähler nummeriert, um Nachrichtenwiederholungen erkennen zu können (siehe Kapitel 2.7.3).

Die F-Parameter werden während der Projektierungsphase festgelegt und ändern sich im Betrieb nicht. Daher kann ein Teil der End-To-End Prüfsumme bereits während der Projektierung im voraus berechnet und als Startwert für die für jede Nachricht zu berechnende Prüfsumme verwendet werden; auch große Parametermengen lassen sich daher ohne negative Auswirkungen auf das Laufzeitverhalten absichern.

Das Gesamtkonzept sieht vor, daß ein ProfiSafe-System mindestens eine sichere Steuerung als Master enthält. Nur ein sicherer Master kann mit sicheren Peripheriegeräten kommunizieren. Eine sichere Kommunikation zwischen sicheren Mastern ist in der derzeitigen Spezifikation nicht vorgesehen ([26]), ist aber für zukünftige Erweiterungen geplant.

3.2.3 INTERBUS Safety

INTERBUS Safety ist die sichere Erweiterung des INTERBUS ([20]). Wie bei ProfiSafe wird vom Sender eine zusätzliche Prüfsumme in den normalen Datenstrom eingefügt und in den Empfängern geprüft.

Das Gesamtkonzept unterscheidet sich jedoch stark von ProfiSafe, da die Steuerung der Anlage weiterhin von einer normalen, nicht sicheren Steuerung durchgeführt wird. Die für die Kommunikation mit den sicheren Peripherieanschaltungen notwendige Prüfsumme wird dabei nicht von der Steuerung, sondern von einer zusätzlichen Überwachungseinheit („SafeControl“) berechnet und in den Datenstrom eingefügt. Dies ist möglich, da INTERBUS-Systeme grundsätzlich eine Ringtopologie verwenden, und Nachrichten daher von einem Teilnehmer zum nächsten „weitergeschoben“ werden. Die Überwachungseinheit erhält dieselben Eingangsdaten wie die Steuerung über den Bus und kann anhand der von der Steuerung gesendeten Ausgangsdaten überwachen, ob die Steuerung korrekt funktioniert. Dabei kann das Überwachungsprogramm meist sehr viel einfacher ausfallen als das eigentliche Steuerungsprogramm, da lediglich eine Gefährdung der Sicherheit erkannt und nicht die eigentliche Funktion erfüllt werden muß. Konzeptionell entspricht dies der bei der kontaktbehafteten Sicherheitstechnik üblichen „Zustimmschaltung“(siehe Kapitel 2.6).

Vorteilhaft an diesem Konzept ist, das keine sichere Steuerung benötigt wird. Eine separate Überwachungseinheit ist allerdings nur solange wirtschaftlich, wie der Aufwand für eine normale Steuerung zuzüglich Überwachungseinheit kleiner ist als für eine sichere Steuerung mit der gleichen Funktionalität – die Funktionalität des Überwachungsprogramms ist daher eingeschränkt. Ein weiterer Nachteil ergibt sich aus der Notwendigkeit, ein gesondertes Überwachungsprogramm entwickeln und pflegen zu müssen.

3.2.4 CANopen-Safety

CANopen-Safety ist die sichere Erweiterung von CANopen, einem auf CAN basierendem Feldbussystem ([35]). Im Gegensatz zu den anderen vorgestellten Lösungsansätzen wird bei CANopen-Safety keine zusätzliche Prüfsumme in den Datenstrom eingefügt, sondern alle sicherheitsrelevanten Nachrichten werden doppelt gesendet. Der Empfänger prüft, ob beide Kopien der Nachricht identisch sind. Die Nachrichtenkennung der redundanten Nachricht muß sich dabei in mindestens zwei Bitpositionen unterscheiden. Zusätzlich wird das Datenfeld der zweiten Nachricht bitweise invertiert, um Fehler innerhalb der beteiligten CAN-Controller entdecken zu können.

Wie bei den anderen Bussystemen auch gibt es eine definierte Zeitspanne, innerhalb derer ein sicheres Gerät eine gültige Nachricht gesendet haben muß, um nicht als ausgefallen zu gelten. Zur globalen Sicherschaltung einer Anlage gibt es das sog. „Global Failsafe Command“(GFC), das als hochpriore CAN Nachricht versendet wird. Im Gegensatz zu den bisher vorgestellten Ansätzen ist in CANopen-Safety eine direkte Kommunikation zwischen zwei sicheren Peripherieanschaltungen (z. B. einem Notaus-Taster und einer Motorsteuerung) möglich; eine sichere Steuerung ist also nicht unbedingt erforderlich.

3.2.5 AS-Interface

„AS-Interface Safety at Work“ist die sichere Variante von AS-Interface ([4], [3], [19]). An dieses Bussystem sind nur sichere Eingangsmodule anschließbar, wobei ein solches Modul lediglich die (binäre) Information „frei“ oder „nicht frei“ übertragen kann. Die prinzipielle Struktur eines sicheren AS-Interface Systems ist in Bild 3.9 dargestellt.

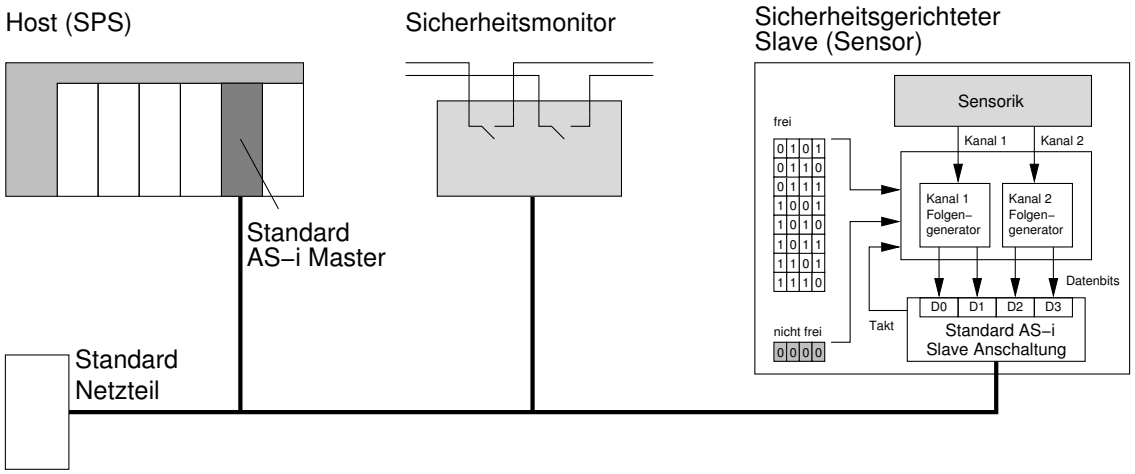

Bild 3.9: Prinzipielle Struktur einer „AS-Interface Safety at Work“Systems (nach [19])

Da eine normale AS-Interface Nachricht mit nur vier Nutzdatenbits zu kurz ist, um eine sinnvolle Prüfsumme enthalten zu können, wird zur Datensicherung eine sog. dynamische Codetabelle

verwendet. Ein sicherer Slave muß in aufeinanderfolgenden Nachrichten die in dieser Tabelle abgelegten Codewörter senden, wenn er im Zustand „frei“ ist. Jeder sichere Slave hat eine eigene Tabelle; dies soll verhindern, daß sich ein sicherer Slave für einen anderen ausgeben kann.

Die Überwachung der Funktionsfähigkeit der sicheren Slaves geschieht nicht im AS-Interface Busmaster, sondern in einem sog. Sicherheitsmonitor (vgl. mit der Überwachungseinheit von INTERBUS Safety). Ein Sicherheitsmonitor verfügt über einen oder zwei Abschaltkreise, die in Abhängigkeit des Zustandes der sicheren Eingänge geschaltet werden können. Konzeptionell handelt es sich auch hier um eine reine Zustimmschaltung. „AS-Interface Safety at Work“ erlaubt gegenüber der konventionellen Sicherheitstechnik keine neue Funktionalität, sondern nur eine vereinfachte Verdrahtung der dort typischerweise genutzten Sensorik (Not-Aus-Taster, Näherungssensoren, Lichtschranken und Türschutzkontakte).

4 ADES

4.1 Das ADES Architekturkonzept

ADES (Architecture for Dependable Embedded Systems) ist ein Architekturkonzept für den Aufbau eines fail-safe bzw. eines fail-operational Rechnersystems. Ein nach dem ADES Architekturkonzept erstelltes reales System besteht aus dem physikalischen Prozeß, der geregelt werden soll, einem Rechnerkern, auf dem die Regelalgorithmen ablaufen, sowie Sensorik und Aktorik, mit denen der Rechnerkern auf den Prozeß einwirken kann. Das ADES Architekturkonzept beschreibt den Aufbau und die Anbindung des Rechnerkerns an die Peripherie; der Aufbau der Peripherieeinheiten selber ist nicht Bestandteil des Konzepts.

ADES ist für Systeme mit kleinen bis mittleren Stückzahlen gedacht, bei denen die Entwicklungskosten einen Großteil der Gesamtkosten ausmachen. Das Ziel ist daher eine möglichst einfache und schnelle Entwicklung, nicht jedoch die möglichst effiziente Nutzung von Hardwareresourcen. Dieses Ziel wird einerseits durch den Einsatz von standard Hard- und Softwarekomponenten erreicht, den sog. „commercial off the shelf“ (COTS) Komponenten. Neben einem erheblichen Preisvorteil im Vergleich zu Sonderentwicklungen haben etwas länger am Markt verfügbare COTS-Komponenten eine gewisse Betriebsbewährung, da eventuelle Designfehler mit hoher Wahrscheinlichkeit bereits in anderen Anwendungen entdeckt wurden. Diese Fehler sind dann inzwischen entweder behoben worden, oder es wurden Möglichkeiten entdeckt, sie zu umgehen. Darüber hinaus stellt ADES für den fail-safe oder fail-operational Betrieb notwendige Fehlertoleranz als inhärente Systemeigenschaft zur Verfügung, so daß sich der Entwickler auf die eigentliche Funkionalität seiner Anwendung konzentrieren kann.

Bei Anwendungen mit sehr hohen Stückzahlen (z. B. im Automobilbau) können durch genau auf die Anwendung maßgeschneiderte Sonderentwicklungen gegenüber einem auf COTS-Komponenten basierendem System große Einsparungen bei den Hardwarekosten erreicht werden; gleichzeitig kann der zusätzliche Entwicklungsaufwand auf die hohe Stückzahl umgelegt werden. ADES ist daher für solche Anwendungen nicht wirtschaftlich.

Das in seinen Zielen sehr ähnliche Architekturkonzept GUARDS ([27]) erlaubt dem Entwickler wesentlich mehr Freiheiten in der Struktur seines Systems (siehe Kapitel 3.1.5). Es ist daher auch auf eine sehr viel breitere Klasse von Anwendungen anwendbar. Allerdings wird diese Flexibilität mit einer erhöhten Komplexität erkauft, die in ADES vermieden werden soll. Daher beschränkt sich ADES auf nur zwei Varianten, wie der Rechnerkern realisiert werden kann.

4.2 Fehlermodell

Hinsichtlich der zu erwartenden Fehler geht ADES von folgenden Annahmen aus:

1. Das zu steuernde bzw. zu regelnde System hat einen sicheren Endzustand, den beizubehalten keine Energiezufuhr benötigt. Dafür sind unter Umständen zusätzliche Maßnahmen wie z. B. mechanische Haltebremsen notwendig; diese werden im Rahmen von ADES nicht behandelt und daher im folgenden nicht weiter betrachtet.
2. Mögliche Fehlerursachen sind ausschließlich zufällige Komponentenausfälle und durch elektromagnetische Einstrahlung verursachte Störungen. Diese Fehler können permanenter oder transienter Natur sein. Mehrfachfehler mit gemeinsamer Ursache (sog. common-mode Fehler) werden durch den Systementwurf verhindert. Entwurfsfehler und gezielte Fehlereinstreuung von außen (wie z. B. Sabotage) werden nicht berücksichtigt.
3. Innerhalb der Fehlererkennungs- und Behandlungszeit des Systems tritt maximal ein Fehler auf.

Treffen diese Annahmen nicht zu, kann ADES nicht wie hier beschrieben verwendet werden, sondern muß um die entsprechende Funktionaltität erweitert werden.

4.3 Der Rechnerkern

Der ADES Rechnerkern ist – mit Ausnahme einer möglichen Vorverarbeitung von Sensordaten in den Peripheriegeräten – für die gesamte Datenverarbeitung innerhalb des Systems verantwortlich. Er besteht aus mehreren, nicht taktsynchron arbeitenden Einzelrechnern, deren Programmabläufe und Ergebnisse untereinander synchronisiert werden. Dazu besitzt der Rechnerkern ein eigenes Kommikationssystem, über das die einzelnen Rechner Daten austauschen, die dann in jedem Rechner votiert werden (es gibt also keinen zentralen Voter). Für ein fail-safe System werden zwei Rechner im zwei-von-zwei, für ein fail-operational System drei Rechner im zwei-von-drei Betrieb verwendet. Fällt ein Rechner aus, muß sich das fail-safe System abschalten; in diesem Fall kann das fail-operational System als fail-safe System im zwei-von-zwei Betrieb weiterarbeiten, bis ein weiterer Rechner ausfällt.

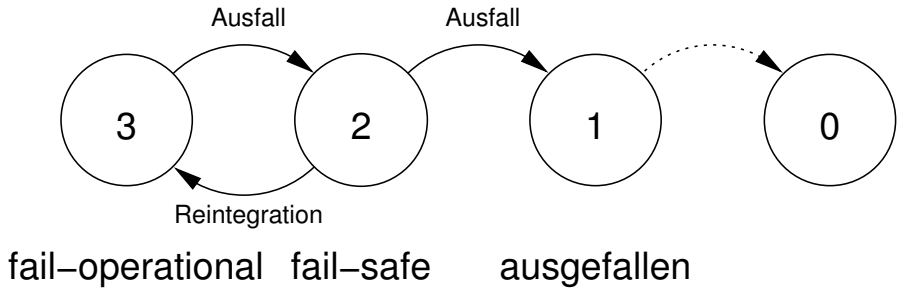

Bild 4.1: Mögliche Zustände des Rechnerkerns. Die Zahl innerhalb der Kreise gibt die Anzahl der funktionsfähigen Rechner an.

Da die Rechner nicht taktsynchron arbeiten, muß keine identische Rechnerhardware verwendet werden – der Kern kann auch diversitär aufgebaut werden. Der Aufwand für eine diversitäre Entwicklung ist allerdings sehr hoch und daher nur für wenige Anwendungen zu rechtfertigen.

Auf allen Rechnern des Kerns läuft funktional identische Software, d. h. aus gleichen Eingaben müssen auf allen Rechnern gleiche Ausgaben berechnet werden. Eine Verteilung von Funktionen findet nicht statt; es handelt sich nicht um ein sog. verteiltes System, auch wenn eine räumliche Trennung der Rechner durchaus möglich ist, um das System gegen physikalische Schäden wie z. B. Überflutung oder Feuer zu schützen.

Wie bei der Hardware könnte auch die Software diversitär implementiert werden. Aber ebenso wie dort spricht der sehr hohe Entwicklungsaufwand nur in Ausnahmefällen für eine solche Vorgehensweise.

Die auf dem Rechnerkern ablaufende Software ist in mehrere Schichten aufgeteilt, deren generelle Struktur in Bild 4.2 dargestellt ist. Die oberste Schicht bildet das Anwendungssystem, das für die eigentliche Funktionalität des Systems verantwortlich ist. Das Anwendungssystem wird durch die Fehlertoleranzschicht weitgehend von den für die Fehlertoleranz notwendigen Mechanismen isoliert. Die Anwendung kann daher unabhängig davon entwickelt werden, ob sie später auf einem fail-safe oder einem fail-operational System ablaufen soll; es ist ebenso möglich, die Entwicklung der eigentlichen Anwendung auf einem normalen Rechnersystem durchzuführen, das zwar dieselben Schnittstellen, aber keine Sicherheitsfunktionalität aufweist. Durch die strikte Trennung der Schichten wird die Anwendung unabhängig von der Art des Peripherieanschlusses.

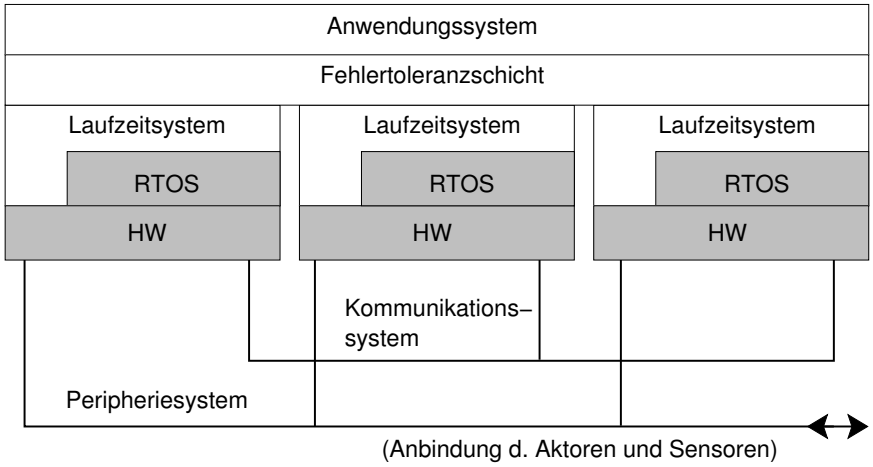

Bild 4.2: ADES Schema

Die Fehlertoleranzschicht setzt auf das ADES Laufzeitsystem auf, das die für die Kommunikation und Synchronisation der Rechner notwendige Funktionalität bereitstellt und für die Ansteuerung der Peripherie des jeweiligen Rechners zuständig ist.

Das Laufzeitsystem wiederum setzt auf einem Realzeitbetriebssystem auf (im Bild als „RTOS“ bezeichnet), das die notwendige Grundfunktionalität (Speicher- und CPU-Verwaltung, Datei-

zugriff, etc..) bereitstellt. Das Betriebssystem des jeweiligen Rechners muß die redundante Struktur des Rechnerkerns nicht berücksichtigen, so daß ein beliebiges Realzeitbetriebsystem verwendet werden kann (der in Kapitel 7 beschriebene Prototyp verwendet z. B. das Betriebssystem QNX).

Auf unterster Ebene ist schließlich die Hardware, die vom Betriebssystem gesteuert wird. Die für die Ansteuerung des Peripheriesystems und des zum Rechnerkern gehörenden Kommunikationssystems notwendige Hardware muß dabei i. allg. von ADES-spezifischer Treibersoftware angesteuert werden; diese gehört konzeptionell zum ADES Laufzeitsystem, nicht zum Betriebssystem, da sie auf die besonderen Bedürfnisse von ADES zugeschnitten ist.

Die Synchronisation der Rechner erfolgt nur an den sog. Synchronisationspunkten. Ein Synchronisationspunkt ist ein bestimmter Punkt im Ablauf der Software, an dem der zu diesem Punkt gehörende Datensatz mit den anderen Rechnern des Rechnerkerns über das Kommunikationssystem ausgetauscht und anschließend votiert wird. Erst danach darf die Anwendungssoftware weiter abgearbeitet werden. Eine Zeitüberwachung verhindert, daß ein ausgefallener Rechner den übrigen Rechnerkern blockieren kann.

Der Entwickler der Anwendungssoftware muß sich nicht explizit um die Synchronisation kümmern; er legt lediglich den Synchronisationspunkt und den zu synchronisierenden Datensatz fest. Die Kommunikation, Votierung und eventuelle Fehlerreaktion übernimmt die Fehlertoleranzschicht. In Bild 4.3 sind die Synchronisationspunkte aus Sicht der Anwendungssoftware dargestellt. Synchronisiert werden müssen alle Ein- und Ausgabedaten, sowie alle internen Zwischenergebnisse, bei denen ein fehlerhafte Veränderung längere Zeit unbemerkt bleiben kann. Zusätzlich dazu müssen periodisch alle internen Zustände ausgetauscht werden, damit ein ausgefallener Rechner wieder in das System reintegriert werden kann. Bei der Wahl der Synchronisationspunkte muß aber auch berücksichtigt werden, daß die Synchronisation – je nach Implementierung des Kommunikationssystems – eine sehr zeitaufwendige Operation sein kann. Daher sollte versucht werden, mit möglichst wenigen Synchronisationspunkten auszukommen, bzw. möglichst viele zusammenhängende Daten auf einmal zu synchronisieren.

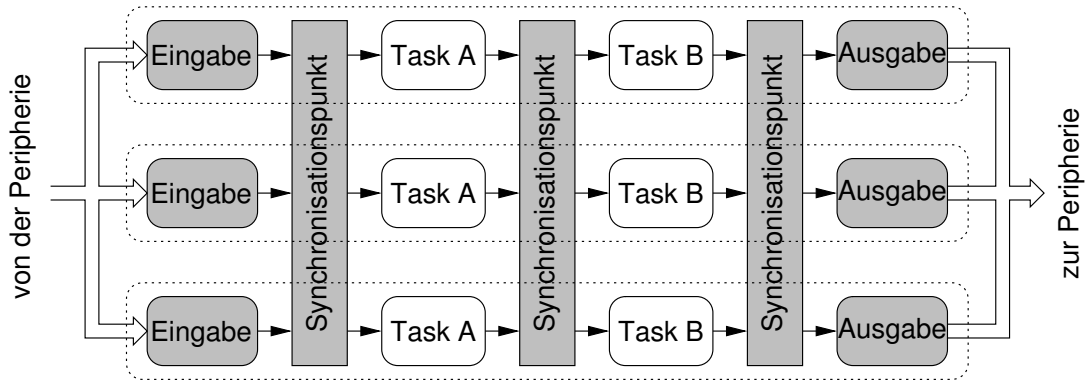

Bild 4.3: Synchronisationspunkte im Datenfluß

Die Votierung ist fast im gesamten System eine einfache Mehrheitsentscheidung auf Gleichheit, da alle funktionsfähigen Rechner immer identische Datensätze besitzen sollten. Lediglich die

Eingangsdaten vom Peripheriesystem müssen eventuell mit einem speziell auf die Art der Daten angepaßten Algorithmus votiert werden (siehe Kapitel 2.5). Kann keine eindeutige Mehrheit gefunden werden, muß das System sicher geschaltet werden.

4.4 Das Peripheriesystem

Dieser Abschnitt beschreibt das grundlegende Konzept für den Peripherieanschluß an ADES, sowie mehrere mögliche Realisierungsvarianten. In den Kapiteln 5 und 6 werden diese miteinander verglichen und die Leistungsfähigkeit und Zuverlässigkeit des gesamten Konzeptes abgeschätzt.

Das Peripheriesystem ist zuständig für die Verbindung zwischen dem Rechnerkern und den Sensoren und Aktoren (der Peripherie), über die der Rechnerkern mit dem physikalischen System verbunden sind. Im Rahmen dieser Arbeit wird nur ein busbasiertes Peripheriesystem betrachtet, d. h. die Peripherieeinheiten werden nicht direkt an den Rechnerkern angeschlossen (siehe auch Kapitel 2.7). Die Peripherie selber wird nur insoweit diskutiert, wie dies für die Verbindung zum Rechnerkern notwendig ist.

Da das ADES Peripheriesystem sowohl verschiedene Anforderungen erfüllen kann, als auch mehrere Implementierungsmöglichkeiten offen läßt, ist für das Verständnis der folgenden Abschnitte eine Nomenklatur notwendig, anhand derer eine Systemvariante eindeutig identifiziert werden kann. Daher wird eine Systemvariante mit einer Folge aus Großbuchstaben bezeichnet, die jeden Freiheitsgrad festlegen. Zuerst wird zwischen fail-safe und fail-operational Systemen unterschieden. Erstere werden mit einem **S**, letztere mit einem **O** gekennzeichnet. Der nächste Buchstabe beschreibt die Art der Nutzung der Peripheriekanäle. ADES erlaubt hier zwei verschiedene Varianten – die exklusive Nutzung, bezeichnet durch den Buchstaben **E**, und die gemeinsame Nutzung, bezeichnet durch den Buchstaben **G**. Beide Varianten werden in den Abschnitten 4.4.4 und 4.4.5 ausführlich beschrieben. Bei einem System mit exklusiver Kanalnutzung ergeben sich zwei Untervarianten – es können normale Busanschaltungen, bezeichnet mit dem Buchstaben **N**, oder fail-safe bzw. fail-operational Busanschaltungen, bezeichnet mit dem Buchstaben **S**, verwendet werden. Systeme mit gemeinsamer Kanalnutzung können nur mit fail-safe Busanschaltungen betrieben werden, daher entfällt hier eine weitere Kennzeichnung. Eine genaue Beschreibung normaler, fail-safe und fail-operational Busanschaltungen folgt in Abschnitt 4.4.2.

Tabelle 4.1 zeigt alle möglichen Kombinationen. Ist eine Variationsmöglichkeit für die weitere Diskussion unerheblich, wird dies durch ein **X** als Platzhalter gekennzeichnet, wenn weitere Buchstaben folgen; ansonsten wird der Buchstabe weggelassen. Die Menge aller fail-safe Systeme heißt nach diesem Schema „S-Systeme“, während die Menge aller Systeme mit exklusiver Kanalnutzung „**XE**-Systeme“ genannt werden.

	fail-safe		fail-operational	
	exklusive Kanalnutzung	gemeinsame Kanalnutzung	exklusive Kanalnutzung	gemeinsame Kanalnutzung
normale Busanschaltung	**SEN**	—	**OEN**	—
sichere Busanschaltung	**SES**	**SG**	**OES**	**OG**

Tabelle 4.1: Nomenklatur verschiedener Realisierungsvarianten des Peripheriesystems

4.4.1 Das Peripheriemodell — physikalische, reale und virtuelle E/A-Punkte

In Bild 4.4 ist das ADES Peripheriesystem schematisch dargestellt. Auf der untersten Ebene befindet sich die Peripherie, mit der die physikalischen Größen des Prozesses beeinflußt bzw. gemessen werden können. Die einzelnen Meß- und Stellpunkte werden unter der Bezeichnung physikalische E/A-Punkte zusammengefaßt; sie stellen die eigentliche, physikalische Schnittstelle zwischen dem Prozeß und dem Peripheriesystem dar. Jeder physikalische E/A-Punkt kann eindeutig genau einer Peripherieeinheit zugeordnet werden.

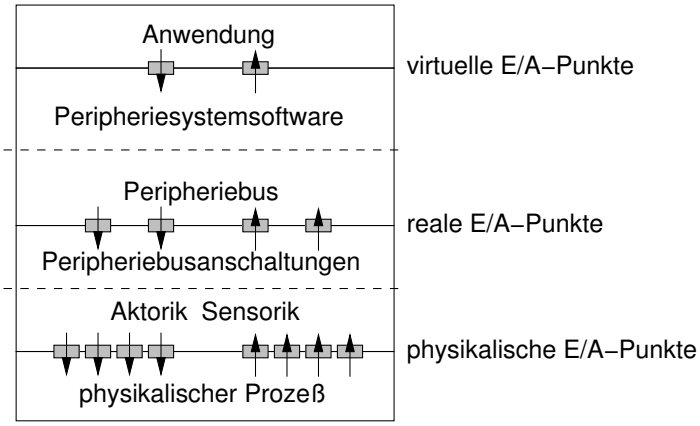

Bild 4.4: Peripheriemodell

Eine Busanschaltung bildet die an sie angeschlossenen physikalischen E/A-Punkte auf sog. reale E/A-Punkte ab. Diese Punkte werden „real“ genannt, da sie in ihrer Gesamtheit die für den Rechnerkern wahrnehmbare „Realität“ darstellen. Neben dem Meß- bzw. Stellwert besitzt jeder reale E/A-Punkt auch eine Diagnoseinformation, die eine Auskunft über den momentanen Status („ausgefallen“, „nicht ausgefallen“) des Punktes gibt. Im einfachsten Fall ist ein realer E/A-Punkt die (digitale) Abbildung der physikalischen Größe des korrespondierenden physikalischen E/A-Punktes; dann ist die Diagnoseinformation nicht gesichert und muß durch übergeordnete Maßnahmen ergänzt werden. Die für eine Busanschaltung benötigte Eigenintelligenz kann aber auch für eine Vorverarbeitung zur Datenreduktion oder Fehlererkennung (siehe auch den folgenden Abschnitt 4.4.2) genutzt werden. In diesem Fall wird ein realer E/A-Punkt

auf mehrere physikalische E/A-Punkte abgebildet (ein physikalischer E/A-Punkt wird dagegen niemals auf mehrere reale E/A-Punkte abgebildet). Ein realer E/A-Punkt ist immer genau einer Busanschaltung eindeutig zugeordnet.

Der Rechnerkern tauscht die zu den realen E/A-Punkten gehörenden Daten (Wert und Status) über das Bussystem mit den Busanschaltungen aus und bildet die realen E/A-Punkte auf die sog. virtuellen E/A-Punkte ab. Diese Punkte heißen „virtuell“, weil sie die Abstraktion des physikalischen Systems auf die für die Anwendung wesentlichen Größen darstellen und keine direkte Entsprechung in realen oder physikalischen E/A-Punkten besitzen müssen. Sie bilden den Mechanismus, mit dem die Komplexität des Peripheriesystems vor der Anwendungssoftware verborgen wird, insbesondere hinsichtlich der Verwaltung redundanter Sensoren und Aktoren.

Ein kleines Beispiel, dargestellt in Bild 4.5, soll das Modell verdeutlichen. Gegeben sei eine (recht einfache) Brennersteuerung, bei der die Temperatur der Brennkammer geregelt werden soll. Die Kammer besitze zur Brennstoffzuführung eine Brennstoffpumpe, ein selbstschließendes Brennstoffventil und eine elektrische Zündvorrichtung. Die Pumpe sei so entworfen, daß ein beliebiger Fehler innerhalb der Pumpe nicht zum unerwünschten Anlaufen der Pumpe führen kann. Die Messung der Brennkammertemperatur erfolgt mittels zweier redundanter Temperaturfühler. Die Funktionsfähigkeit der Brennstoffzufuhr soll über eine einfache modellbasierte Diagnose („bei Brennstoffzufuhr steigt die Temperatur“) ohne zusätzliche Diagnose erfolgen. Für die gesamte Anwendung werde ein Einfehlermodell angenommen (für eine reale Brennersteuerung wäre eine solche Annahme nicht zulässig; hier geht es aber nur um ein Beispiel zur Verdeutlichung des Peripheriemodells).

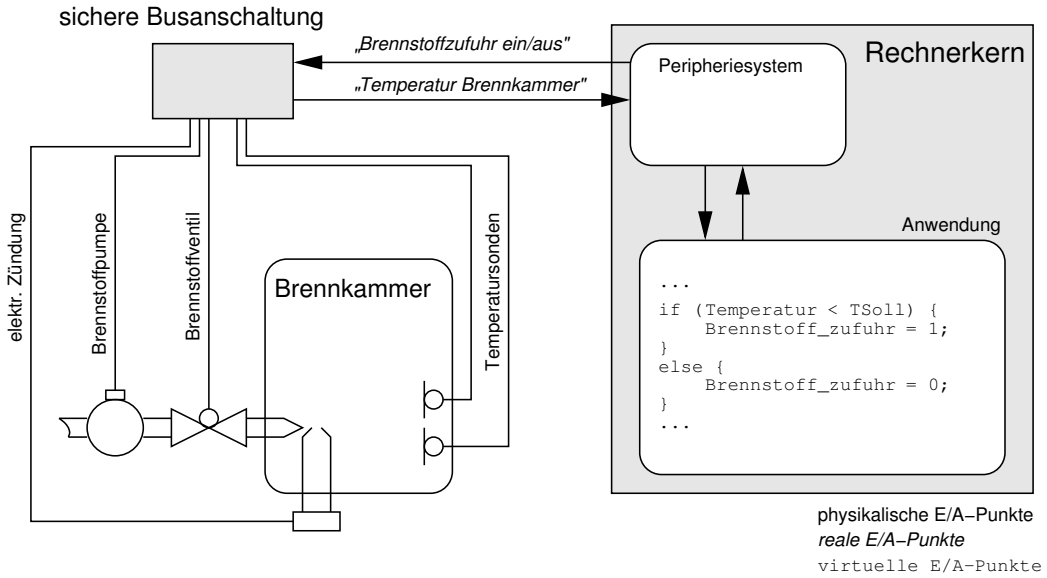

Bild 4.5: Beispiel: Brennkammersteuerung

Das System besitzt insgesamt fünf physikalische E/A-Punkte, wovon drei Aktoren und zwei Sensoren sind. Diese werden über eine einzige fail-safe Busanschaltung an das Peripheriebussystem angekoppelt. Die Busanschaltung sei so konstruiert, daß bei Auftreten eines beliebigen Fehlers in der Busanschaltung die Brennstoffpumpe sicher abgeschaltet und das Ventil zur

Brennstoffzuführung sicher geschlossen wird. Die Meßwerte beider Temperaturfühler werden von der Busanschaltung miteinander verglichen und der Mittelwert über den Bus übertragen; liegt die Abweichung der beiden Meßwerte zueinander außerhalb der vorgegebenen Toleranzen, wird ein Sensorfehler gemeldet. Die Steuerung von Brennstoffpumpe, -ventil und Zündmechanismus ist ebenfalls in die Busanschaltung integriert; der Rechnerkern sendet lediglich den gewünschten Zustand „Brennstoffzufuhr ein“ bzw. „Brennstoffzufuhr aus“. Das System besitzt damit zwei reale E/A-Punkte, die Brennkammertemperatur und die Brennstoffzufuhr. Die Diagnoseinformation des E/A-Punktes „Temperatur“ ist dabei vollständig, d. h. es kann ohne weitere Maßnahmen eine Aussage über die Funktionsfähigkeit der Temperaturmessung gemacht werden, während die Diagnoseinformation für den E/A-Punkt „Brennstoffzufuhr“ nur bis in die Busanschaltung hinein gültig ist (ein Ausfall der Brennstoffpumpe kann z. B. an dieser Stelle nicht erkannt werden).

Die beiden realen E/A-Punkte werden eins zu eins auf zwei virtuelle E/A-Punkte abgebildet. Die Anwendung muß dabei anhand des hinterlegten Prozeßmodells die Funktionsfähigkeit der Brennstoffzufuhr überwachen, da die hier verwendete Fehlererkennungsstrategie bereits zu anwendungsspezifisch ist, als das sie sinnvoll in die Peripheriesystemsoftware integrierbar wäre.

Soll das System später durch Erhöhung des Redundanzgrades der Peripherie für den fail-operational Betrieb erweitert werden, muß die Art und Anzahl der virtuellen E/A-Punkte und damit die Anwendungssoftware nicht geändert werden; lediglich die Abbildung der (nun zahlreicheren) realen E/A-Punkte auf die virtuellen E/A-Punkte muß angepaßt werden.

Wenn im folgenden von „normalen“, „fail-safe“ und „fail-operational“ E/A-Punkten die Rede ist, sind immer die virtuellen E/A-Punkte des Systems gemeint, klassifiziert nach den Anforderungen an Sicherheit und Verfügbarkeit, bzw. die Menge aller zur Erfüllung dieser Anforderungen notwendigen physikalischen E/A-Punkte. Ein „fail-safe“ E/A-Punkt wird dabei in der Regel aus zwei redundanten, physikalischen E/A-Punkten bestehen.

4.4.2 Normale, fail-safe und fail-operational Busanschaltungen

Das busbasierte ADES Peripheriesystem erlaubt die Verwendung einer Mischung von normalen, fail-safe oder fail-operational Busanschaltungen, um physikalische E/A-Punkte an den Rechnerkern anzubinden.

Eine normale Busanschaltung verhält sich im Fehlerfall undefiniert, d. h. sie kann auf beliebige Art und Weise ausfallen. Diese Art von Busanschaltungen ist für alle gängigen Bussysteme in großer Vielfalt für fast jede Art von E/A-Punkt erhältlich und kann ohne Änderungen verwendet werden. Alle für einen fail-safe bzw. fail-operational Betrieb notwendigen Zusatzfunktionen (Votierung, Test, Fehlerabschaltung) müssen dann durch eine Kombination von zusätzlicher Software im Rechnerkern, redundanter Peripherieeinheiten und Busanschaltungen, sowie zusätzlicher Votierungs- und Testhardware zwischen Busanschaltungen und Aktorik realisiert werden (siehe Bild 4.6). Diese muß insbesondere in der Lage sein, einen Ausfall des Rechnerkerns zu erkennen und die Aktorik entsprechend sicher zu schalten. Es ist dabei unbedingt zu beachten, daß die für redundante Peripherie verwendeten redundanten Busanschal-

tungen niemals an den selben physikalischen Bus angeschlossen werden dürfen, da sonst eine ausgefallene Busanschaltung ihr redundantes Gegenstück imitieren und korrekte Funktion vorgaukeln könnte. Für die Verwendung normaler Busteilnehmer spricht, das keine besondere Hardware benötigt wird, sondern die für alle gängigen Bussysteme in großer Vielfalt vorhandenen Busanschaltungen verwendet werden können.

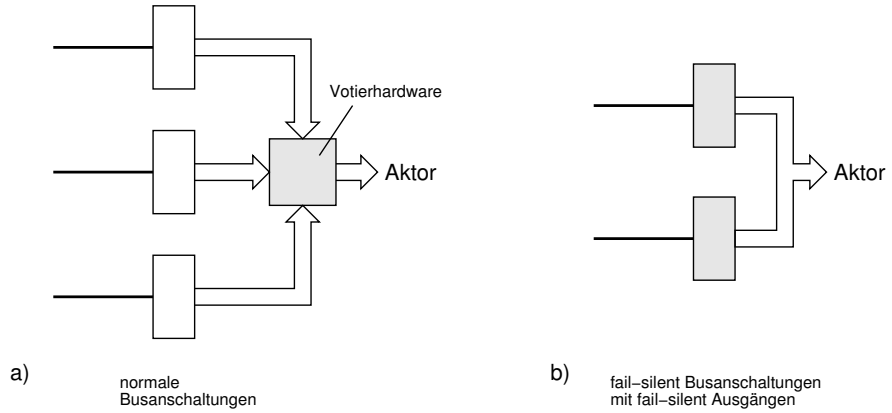

Bild 4.6: Fail-operational Anschluß eines Aktors mit normalen Busanschaltungen (a) und mit fail-safe Busanschaltungen, die über fail-silent Ausgänge verfügen (b)

Eine fail-safe Busanschaltung schaltet sich im Fehlerfall selbständig in einen sicheren Zustand. Dazu ist eine Mischung aus Fehlererkennung und interner Redundanz notwendig, die einen Mehraufwand gegenüber einer normalen Busanschaltung bedeutet. Dieser ist nur dann sinnvoll, wenn – wie im vorherigen Abschnitt bereits angedeutet – zusätzlich einfache Fehlererkennungs- und Fehlerbehebungsmaßnahmen in die Busanschaltung integriert werden. So können redundante Eingangsdaten bereits in der Busanschaltung votiert, und Aktoren über fehlersichere Ausgänge angesteuert werden. Aktoren, deren Funktion durch einfaches Rücklesen zusätzlicher Sensorik überwacht werden kann, d. h. keine komplizierte modellbasierte Diagnose benötigen (z. B. ein Ventil mit Positionsrückmeldung), können dann direkt durch die Busanschaltung überwacht werden. Ebenso können aktive Tests von Aktoren lokal durchgeführt und somit Rechnerkern und Peripheriesystem entlastet werden. Eine Reihe von Tests lassen sich nur lokal sinnvoll durchführen. So kann z. B. die Schaltfähigkeit eines elektrischen Ausganges ohne Störung der normalen Funktion getestet werden, in dem periodisch mit einer so hohen Frequenz umgeschaltet wird, daß die nachgeschaltete Aktorik auf Grund ihrer Trägheit den Schaltvorgang nicht wahrnimmt. Solch kurze Schaltzeiten im Bereich von einer Millisekunde und kürzer sind über die heute üblichen Bussysteme nicht bzw. nur mit sehr hohem Aufwand realisierbar.

Fail-safe Busanschaltungen bieten neben besseren Testmöglichkeiten entscheidende Vorteile bei der Entwicklung des Gesamtsystems. Eine fail-safe Busanschaltung ist ein eigenständiges, sicheres System mit klaren Schnittstellen und kann daher unabhängig vom Gesamtsystem entwickelt und in anderen Systemen weiterverwendet werden. Bedingt durch den begrenzten Funktionsumfang einer einzelnen Busanschaltung im Vergleich zum Gesamtsystem wird dadurch die (Software-)Entwicklung stark vereinfacht — ein entscheidender Vorteil bei der Entwicklung sicherheitskritischer Software.

Im Vergleich zu einer Lösung mit redundanten normalen Busanschaltungen wird eine fail-safe Busanschaltung in der Regel kompakter ausfallen, da bei dieser nicht alle Komponenten redundant vorhanden sein müssen, während sonst zwei vollständige normale Busanschaltungen erforderlich sind. Beim Anschluß von Aktoren benötigt eine Lösung mit normalen Busanschaltungen zusätzlich noch eine externe Votierungshardware, die zwischen den Aktor und die Busanschaltungen plaziert wird. Der Gesamtaufwand kann dann bei geringerer Funktionalität insgesamt höher ausfallen.

Für die fail-operational Busanschaltung gilt im Prinzip das gleiche wie für die fail-safe Busanschaltung; allerdings müssen die Funktionen nicht nur sicher, sondern auch noch hochverfügbar durchgeführt werden, was wiederum den notwendigen Aufwand gegenüber einer fail-safe Busanschaltung bedeutend erhöht.

Um den Entwicklungsaufwand möglichst gering zu halten, kann es in vielen Fällen sinnvoller sein, statt einer fail-operational Busanschaltung zwei redundante, fail-safe Busanschaltungen, wie in Bild 4.6 rechts gezeigt, parallel zu verwenden. Diese Busanschaltungen können auch in einem fail-safe System verwendet werden. Da sie parallel und nicht in Serie betrieben werden, ergeben sich allerdings zwei zusätzliche Anforderungen an die fail-safe Busanschaltung:

1. Zwei fail-safe Busanschaltungen können einen Aktor nur dann gemeinsam ansteuern, wenn sie zusätzlich fail-silent Verhalten an ihren Ausgängen aufweisen; eine ausgefallene Busanschaltung darf die Ansteuerung des Aktors durch die andere Busanschaltung nicht behindern.
2. Die Busanschaltung muß zwei Fehler tolerieren können, ohne in einen unsicheren Zustand zu geraten.

Der Mehraufwand gegenüber einer „normalen“ fail-safe Busanschaltung hält sich in Grenzen und liegt unter dem für eine fail-operational Busanschaltung, die nach einem Fehler ihre Funktion noch erfüllen können muß.

4.4.3 Das Peripheriebussystem allgemein

Das Peripheriebussystem besteht, je nach Anforderungen an Sicherheit, Verfügbarkeit und gewählter Busstruktur, aus ein bis drei Peripheriebuskanälen. Ein Peripheriebuskanal ist eine logische Einheit und kann durchaus aus mehreren physikalischen Bussen bestehen (z. B. um Beschränkungen bei der maximalen Teilnehmerzahl zu umgehen); sowohl horizontal (mehrere Busse werden parallel betrieben), als auch vertikal (an einen Bus werden über sog. Gatewayknoten untergeordnete Busse betrieben). Dabei kann es sich auch um völlig verschiedene Bustypen handeln. An dem eigentlichen Konzept ändert das nichts. Um das Prinzip des Peripheriebussystems besser veranschaulichen zu können, wird daher im folgenden davon ausgegangen, daß jeder Peripheriekanal aus genau einem physikalischen Bus besteht.

Ein ADES Peripheriebus besitzt einen Busmaster (dies ist immer einer der Rechner des Rechnerkerns) und einen oder mehrere Slaves. Die Kommunikation zwischen Master und Slaves ist grundsätzlich in sog. Buszyklen aufgeteilt, die vom Master initiiert werden. Ein Buszyklus ist

wiederum in einzelne Zeitscheiben aufgeteilt, die einzelnen Teilnehmern fest zugeteilt werden. Ein Teilnehmer darf nur innerhalb seiner Zeitscheiben senden. Damit Nachrichtenverluste erkannt werden können, muß ein Teilnehmer alle seine Zeitscheiben während eines Buszyklus zum Senden von Nachrichten nutzen. Hat ein Teilnehmer keine neuen Daten, sendet er eine leere Nachricht. Die Zuteilung der Zeitscheiben erfolgt off-line beim Systementwurf; dadurch kann garantiert werden, daß es im Betrieb nicht zu einer Überlastung des Bussystems kommen kann.

Das von ADES verwendete Buszuteilungsverfahren ist an sich ein normales TDMA-Zuteilungsverfahren, wie es in Kapitel 2.7.2 beschrieben wurde. Soll dieses Konzept mit Hilfe eines Bussystems realisiert werden, das eine andere Buszuteilungsstrategie verwendet, muß TDMA auf das andere Verfahren abgebildet werden. Der Master teilt in diesem Fall dem Slave den Anfang seiner Zeitscheibe durch eine explizite Masteranfrage mit; der Slave antwortet nur auf diesen Anstoß vom Master (dies entspricht dem in Kapitel 2.7.2 beschriebenen Master/Slave Zuteilungsverfahren). Für den zeitlichen Ablauf ist daher alleine der Master verantwortlich; die Slaves müssen lediglich innerhalb einer bestimmten Zeit antworten. Bei Bussystemen wie z. B. Profibus-DP, die von Haus aus eine Master/Slave Zuteilungsstrategie verwenden, wird das gewünschte Verhalten bereits vom Buscontroller des Slaves implementiert; bei anderen Bussystemen wie z. B. CAN, bei denen alle Teilnehmer (aus Sicht des Busses) gleichberechtigt sind und es daher keinen Busmaster gibt, kann dieses Verhalten in der Firmware der Slaves mit geringem Aufwand nachgebildet werden. Im Vergleich zu einem Bussystem mit „richtigem“ TDMA ist die Effizienz aber in jedem Fall geringer.

Generell wird die Wahrscheinlichkeit für einen transienten Fehler bei der Übertragung von Nachrichten deutlich größer sein als die Wahrscheinlichkeiten für alle anderen Fehlerarten (siehe Kapitel 2.7). Im Interesse einer möglichst hohen Verfügbarkeit muß das System daher so ausgelegt sein, daß eine bestimmte Zahl von Übertragungsfehlern tolerierbar sind. Für das ADES Peripheriesystem ist es jedoch weder sinnvoll, Fehlerkorrekturverfahren zu verwenden, da dann die Fehlererkennung und damit die Sicherheit reduziert würden, noch fehlerhaft übertragene Nachrichten zu wiederholen, da dafür feste Sendeslots eingeplant werden müßten und das System im Normalbetrieb die verfügbare Bandbreite nur extrem schlecht ausnutzen könnte. Statt dessen werden fehlerhaft übertragene Nachrichten verworfen und der letzte empfangene Wert verwendet. Für jeden realen E/A-Punkt wird festgelegt, wie viele Buszyklen lang nacheinander bzw. insgesamt in einem bestimmten Zeitintervall keine gültigen Daten empfangen worden sein dürfen, ohne daß eine Sicherheitsreaktion eingeleitet werden muß.

Über den Peripheriebus ausgetauschte Daten haben somit eine Zustandssemantik, d. h. sie spiegeln den momentan bekannten Zustand wieder. Ein Ereignis kann daher nur über die Änderung eines Zustandes übertragen werden. Ändert sich der Zustand bereits wieder, bevor eine gültige Zustandsnachricht empfangen wurde, verpaßt der Empfänger das Ereignis. Um auch Ereignisse sicher erkennen zu können, muß ein eigenes (anwendungsspezifisches) Protokoll in die normale ADES-Kommunikation integriert werden.

4.4.4 Exklusive Kanalnutzung

Für den Anschluß des Rechnerkerns an das Peripheriebussystem gibt es zwei sinnvolle Möglichkeiten, die exklusive und die gemeinsame Kanalnutzung. Bei der exklusiven Kanalnutzung besitzt jeder Kernrechner einen eigenen Peripheriebuskanal, auf den kein anderer Kernrechner Zugriff hat. Für ein fail-safe System werden daher analog zum Rechnerkern zwei, für ein fail-operational System drei Kanäle benötigt. Jeder Peripheriekanal ist eine direkte Erweiterung seines angeschlossenen Kernrechners in die Peripherie hinein. Eine sichere Datenübertragung über nur einen Kanal ist nicht möglich, da ein einzelner Kernrechner nicht sicher ist.

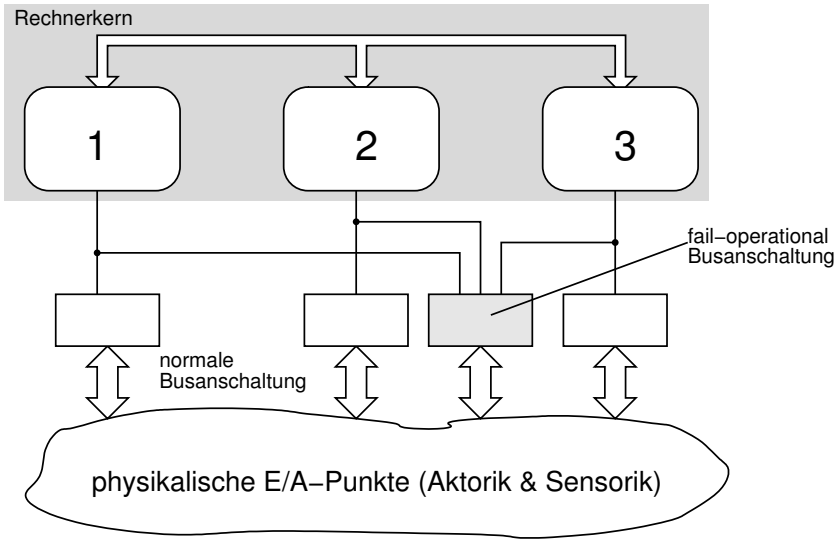

Bild 4.7: Exklusive Kanalnutzung

Sicherheitsunkritische E/A-Punkte können mit normalen, nicht sicheren Busanschaltungen an einen beliebigen Peripheriekanal angeschlossen werden (asymmetrischer Anschluß). Eine Fehlerrückwirkung von einer Busanschaltung auf den Kanal (z. B. ein „babbling idiot“-Ausfall, siehe auch Kapitel 2.7.3) betrifft nur diesen einen Kanal und kann sich nicht auf die anderen Kanäle auswirken.

E/A-Punkte, die fail-safe oder fail-operational angesteuert werden müssen, können wie im vorherigen Abschnitt beschrieben mit normalen, fail-safe oder fail-operational Busanschaltungen angeschlossen werden. Eine fail-operational Busanschaltung muß an alle drei in einem fail-operational System vorhandenen Peripheriekanäle angeschlossen werden, während für eine fail-safe Busanschaltung ein (asymmetrischer) Anschluß an zwei Kanäle ausreicht. Dabei muß aber unbedingt sichergestellt werden, daß ein Fehler in der Busanschaltung keinerlei Rückwirkungen auf einen Peripheriekanal haben kann, da sonst eine Busanschaltung zwei Kanäle blockieren und damit die Verfügbarkeit des Systems in Frage stellen könnte. Soll das System ausschließlich sicher sein, kann auf diese Forderung verzichtet werden, sofern gewährleistet ist, daß die Fehlerrückwirkung erkannt werden kann. Weiterhin ist zu beachten, daß im Gegensatz zur noch zu besprechenden gemeinsamen Kanalnutzung ein fail-operational E/A-Punkt nicht durch die Verwendung zweier asymmetrisch angeschlossener fail-safe Busanschaltungen gebildet werden

kann. Diese müßten, wie in Bild 4.8 dargestellt, einen gemeinsamen Peripheriekanal verwenden. Würden auf dem gemeinsamen Kanal ungültige Daten gesendet, müßten beide Busanschaltungen in den sicheren Zustand wechseln, da keine der beiden fail-safe Busanschaltungen feststellen kann, welche der beiden von ihr empfangenen Nachrichten fehlerhaft ist. Dann aber wäre die Funktion der entsprechenden E/A-Punkte nicht mehr gewährleistet.

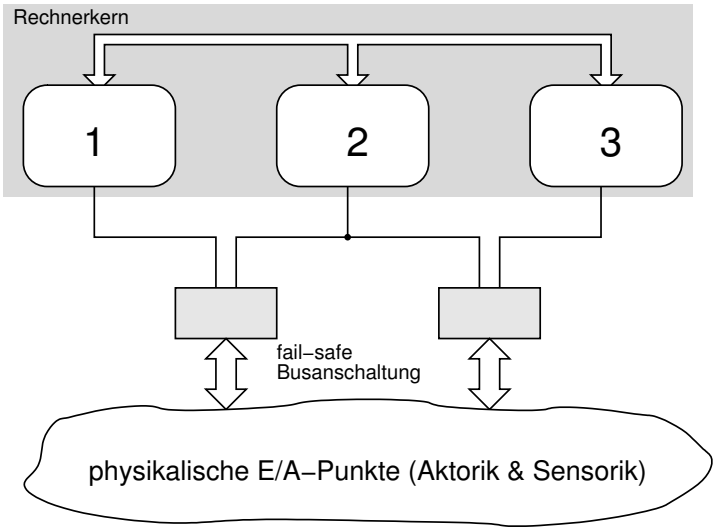

Bild 4.8: Asymmetrischer Busanschluß zweier fail-safe Busanschaltungen

Die exklusive Kanalnutzung kann erweitert werden, so daß jeder Rechner die auf den Peripheriekanälen der anderen Rechner gesendeten Nachrichten mit empfangen kann. Eine mögliche Fehlerrückwirkung eines ausgefallenen Rechners auf die Peripheriekanäle der anderen Rechner muß durch entsprechende Schaltungsmaßnahmen verhindert werden; dies ist bei einer reinen Empfangsmöglichkeit aber nicht weiter schwierig. Mit Hilfe dieser zusätzlichen Empfangsmöglichkeit kann eine fehlerhafte Ansteuerung der Peripherie durch einen Rechner direkt erkannt und lokalisiert werden, ohne indirekt über die Auswirkungen in der Peripherie Rückschlüsse ziehen zu müssen. Auf einen Austausch der empfangene Daten über das Rechner/Rechner Kommunikationssystem kann aber nicht verzichtet werden, da ein SOS-Fehler (siehe Kapitel 2.7.3) dazu führen könnte, daß eine Nachricht nicht von allen Rechnern empfangen werden kann. Es gibt also keine grundsätzliche Änderung gegenüber einer vollständig exklusiven Kanalnutzung.

4.4.5 Gemeinsame Kanalnutzung

Die zweite Möglichkeit für den Anschluß des Rechnerkerns an die Peripherie ist die gemeinsame Kanalnutzung, wie in Bild 4.9 dargestellt.

Bei der gemeinsamen Kanalnutzung kann jeder Kernrechner auf jeden Peripheriekanal zugreifen. Jeweils ein Rechner übernimmt die Rolle des Masters und steuert das Busgeschehen (siehe Abschnitt 4.4.3), während die anderen Rechner Beobachter sind und nur am Bus „mithören“.

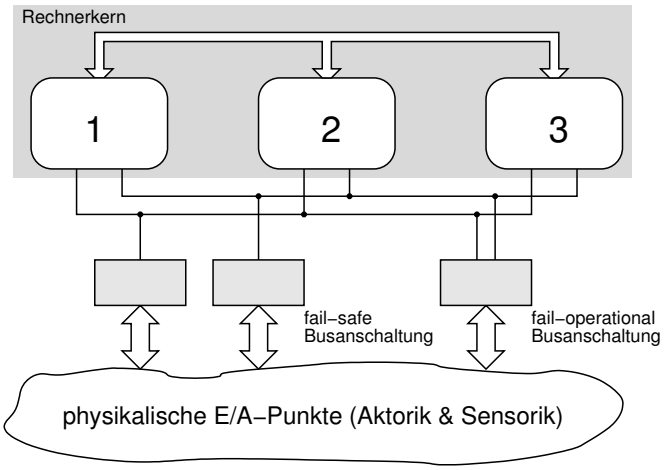

Bild 4.9: Gemeinsame Kanalnutzung

Ein Zugriffsverfahren muß entscheiden, welcher Rechner jeweils der aktuelle Master ist. Ein sinnvolles Verfahren ist die zyklische Zuordnung der Masterrolle an alle aktiven Kernrechner in einer vorgegeben Reihenfolge, wie in Bild 4.10 dargestellt. Damit wird vermieden, daß sich in einem Rechner Fehler anhäufen können, die nur beim Senden von Nachrichten erkannt werden können. Die Masterrolle sollte immer für alle Kanäle gleichzeitig vergeben werden, da der Master die Buszyklen auf den parallelen Kanälen dann gleichzeitig beginnen kann („gleichzeitig“ im Vergleich zur möglichen zeitlichen Synchronisation zweier Kernrechner). Da die Bestimmung des Busmasters sehr kritisch für die Funktion des Gesamtsystems ist, muß diese Entscheidung ebenso wie die zu sendenden Ausgangsdaten votiert werden.

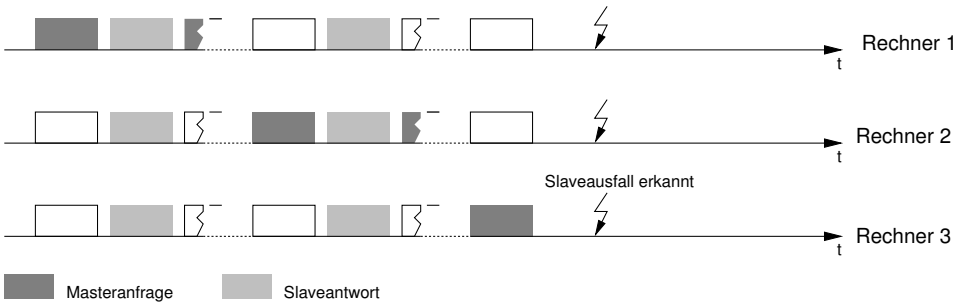

Bild 4.10: Buszugriffsverfahren bei der gemeinsamen Kanalnutzung. Die Masterrolle wird zyklisch zwischen den einzelnen Rechnern durchgetauscht.

Ein entscheidendes Merkmal der gemeinsamen Kanalnutzung ist, daß für die am Bus angeschlossenen Peripheriegeräte nicht erkennbar ist, welcher Kernrechner gerade den Busmaster stellt – für sie existiert der Rechnerkern als ein einziger monolithischer Rechner. Dies hat negative Auswirkungen bei Verwendung von Standardbussystemen wie Profibus-DP oder Interbus-S, bei denen die erhältlichen Masterbaugruppen keinen Master/Beobachterbetrieb erlauben. Profibus-DP erlaubt zwar ausdrücklich den Betrieb mehrerer Master an einem Bus; ein Beobachterbetrieb ist allerdings nicht vorgesehen. Sollen diese Busse dennoch eingesetzt werden,

müssen daher modifizierte Masterbaugruppen verwendet werden, die um die notwendige Funktionalität erweitert worden sind.

Im Vergleich zur exklusiven Kanalnutzung macht die gemeinsame Kanalnutzung nur dann Sinn, wenn durch seinen Einsatz Resourcen eingespart werden können. Dies ist möglich, wenn ein einzelner Peripheriebuskanal fail-silent Verhalten aufweist, d. h. eine Nachricht entweder korrekt oder gar nicht empfangen wird. Ein fail-silent System benötigt dann nur einen, ein fail-operational zwei Peripheriebuskanäle (anstatt zwei bzw. drei Kanälen bei der gemeinsamen Kanalnutzung).

Fail-silent Verhalten kann erreicht werden, wenn der Empfänger die in Kapitel 2.7.3 beschriebenen Fehler erkennen kann und fehlerhafte Nachrichten verwirft. Daher werden alle Nachrichten zusätzlich mit einer End-To-End Prüfsumme abgesichert (siehe Kapitel 2.7.4). Diese Prüfsumme umfaßt auch einen Zykluszähler und eine eindeutige Identifikation des Senders; eine genaue Beschreibung folgt in Kapitel 7.3.3.

Allerdings wird die End-To-End Prüfsumme in den für sich betrachtet nicht sicheren Kernrechnern berechnet und geprüft. Wenn ein Fehler genau zwischen der Votierung der zu sendenden Daten und der Berechnung der Prüfsumme auftritt, kann es daher passieren, daß der ausgefallene Rechner eine syntaktisch korrekte, aber semantisch falsche Nachricht sendet, d. h. eine Nachricht mit ungültigen Daten, die aber eine gültige End-To-End Prüfsumme besitzt. Eine Peripheriebusanschaltung kann diesen Fehler nicht erkennen. Daher müssen die berechneten End-To-End Prüfsummen vor dem Senden genau wie die eigentlichen Ausgangsdaten über die Rechner/Rechnerkommunikation ausgetauscht und votiert werden. Da dies gleichzeitig mit der Bestimmung des aktiven Busmasters geschehen kann, muß dafür kein zusätzlicher Synchronisationspunkt eingeführt werden.

Dies alleine genügt jedoch nicht, da ein Rechner prinzipiell beliebig ausfallen und daher nicht ausgeschlossen werden kann, daß trotz aller Votierungsmaßnahmen eine für die Peripheriebusanschaltungen nicht als solche zu erkennende ungültige Nachricht gesendet wird. Ebenso kann der (deutlich wahrscheinlichere) Fall auftreten, daß ein ausgefallener Rechner Nachrichten sendet, die zwar erkennbar falsch sind, mit diesen Nachrichten aber alle angeschlossenen Peripheriekanäle blockiert und damit das Gesamtsystem ausfallen läßt (ein „babbling idiot“-Ausfall). Es wird daher ein Mechanismus benötigt, mit dem ein fehlerhafter Rechner zuverlässig vom Peripheriebussystem getrennt werden kann, ein sog. Busguardian.

Der Busguardian kann entweder zentral oder verteilt sein, d. h. es gibt entweder einen Busguardian pro Peripheriekanal, an den alle Kernrechner angeschlossen werden, oder jeder Kernrechner besitzt seinen eigenen Busguardian (siehe Bild 4.11). Bei der zentralen Anordnung führt der Ausfall eines Busguardians zum Ausfall des gesamten zugehörigen Peripheriekanals, während bei der verteilten Lösung der angeschlossene Kernrechner ausfällt. Bei der verteilten Lösung muß der Busguardian fail-silent sein, da sonst beide Peripheriekanäle durch einen Ausfall blockiert werden könnten.

Das Abtrennen eines Kernrechners vom Peripheriebus muß von den übrigen Rechnern des Kerns bestimmt werden, d. h. der jeweilige Busguardian muß eine Kommunikationsmöglichkeit mit den übrigen Kernrechnern besitzen. Dies könnte ein dediziertes Kommunikationssystem mit

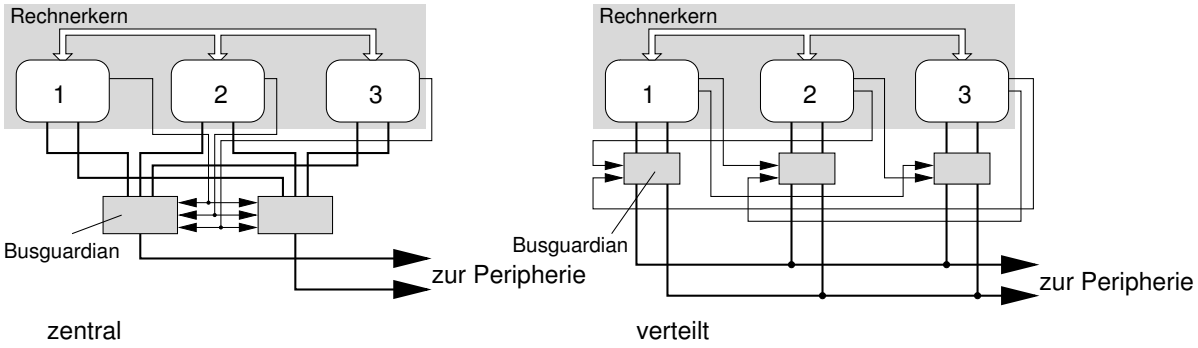

Bild 4.11: Busguardian für ein System mit gemeinsamer Kanalnutzung

eigenem Medium sein; es ist aber auch denkbar, über das normale Rechner/Rechner Kommunikationssystem Nachrichten zu verschicken, die dann an den Busguardian weitergeleitet werden. Diese Nachrichten müßten dann speziell signiert werden, so daß der Busguardian den Absender der Nachricht eindeutig identifizieren kann.

Für die Steuerung des Abschaltvorgangs sind zwei verschiedene Strategien denkbar. In beiden Fällen benötigt der momentane Busmaster eine „Sendeerlaubnis“ von mindestens einem Beobachterrechner (logisch gesehen gibt sich der Busmaster selber eine Sendeerlaubnis, so daß hier ein normaler Mehrheitsentscheid vorliegt, wie er bei m-von-n Systemen üblich ist).

Bei der ersten Strategie bekommt ein Rechner diese Erlaubnis a priori erteilt, bis ein Fehler in diesem Rechner erkannt wird. Im ungünstigsten Fall kann ein ausgefallener Rechner dann eine fehlerhafte Nachricht senden, die von den Peripherieanschaltungen nicht als fehlerhaft erkannt werden kann; ein Aktor darf daher eine potentiell gefährliche Aktion (z. B. das Zünden eines Airbags) nur dann ausführen, wenn der Befehl für diese Aktion mindestens zweimal hintereinander korrekt empfangen wurde. Diese Strategie hat den Vorteil, daß der dafür benötigte Busguardian einen Rechner einfach vollständig vom Bus trennen kann. Dies ist wesentlich einfacher zu realisieren als ein selektives Unterbinden nur des Sendens über den Bus. Soll ein ausgefallener Rechner wieder in das System integriert werden, muß er die Peripheriedaten über die Rechner/Rechnerkommunikation erhalten. Dies stellt zwar keine zusätzliche Belastung für den Rechnerkern dar, da die Eingangsdaten sowieso votiert werden müssen, aber die Funktionsfähigkeit der Busanschaltung kann während der Reintegrationsphase nicht überprüft werden.

Bei der zweiten Abschaltstrategie wird eine Sendeerlaubnis erst direkt nach der Votierung der End-To-End Prüfsumme erteilt und sofort nach dem Ende des aktuellen Peripheriezyklus wieder entzogen. Da ein Rechner im Normalzustand Nachrichten empfangen können muß, aber nicht in der Lage sein darf, Nachrichten zu senden, muß der Busguardian in diesem Fall die Möglichkeit haben, selektiv nur das Senden zu verhindern. Dies ist wesentlich aufwendiger als eine komplette Trennung, wenn der Busguardian keinen direkten Zugriff auf die Steuersignale des Bustransceivers eines Kernrechners hat. Bei Standardhardware ist dieser Zugriff i. allg. nicht gegeben (bei speziell entwickelter Hardware ist dies dagegen kein Problem, siehe z. B. den fail-silent Peripheriebusteilnehmer für den ADES-Prototypen in Kapitel 7.3.5).

Da die End-To-End Prüfsumme im Sender und Empfänger außerhalb des Buscontrollers berechnet bzw. geprüft wird, ist die Absicherung des Nachrichtenverkehrs für den verwendeten Feldbus völlig transparent, so daß keine Änderungen an den Standard-Buscontrollern notwendig sind. Normale Busanschaltungen können (wie im Falle der exklusiven Kanalnutzung) für sicherheitsunkritische E/A-Punkte gemeinsam mit fail-safe Busanschaltungen am selben Kanal betrieben werden, da eine normale Busanschaltung nicht in der Lage ist, eine gültige End-To-End Prüfsumme zu generieren. Im ungünstigsten Fall kann eine normale Busanschaltung einen Peripheriekanal blockieren; alle fail-safe Busgeräte können diesen Fehler aber erkennen und entsprechend reagieren.

Fail-safe E/A-Punkte dagegen müssen über fail-safe Busanschaltungen angeschlossen werden. Ein Anschluß über redundante normale Busanschaltungen ist nicht möglich, da redundante normale Busanschaltungen nicht an denselben physikalischen Bus angeschlossen werden dürfen (siehe Kapitel 4.4.2).

Fail-operational E/A-Punkte können entweder über fail-operational Busanschaltungen (die an beide Kanäle angeschlossen werden müssen) oder über jeweils zwei fail-safe Busanschaltungen angeschlossen werden. Je nach anzusteuernder Aktorik ist unter Umständen analog zur exklusiven Kanalnutzung eine zusätzliche, externe Hardware notwendig, die den Zugriff zweier fail-safe Busanschaltungen auf die eigentliche Aktorik steuert.

Ein besonderes Problem bei der gemeinsamen Kanalnutzung stellt ein Ausfall des gerade aktiven Masters dar. Es wäre vom Konzept her durchaus möglich, einen solchen Ausfall während des Zyklus per Timeout zu erkennen und die Masterrolle noch während des laufenden Zyklus einem anderen Rechner zuzuweisen; dies ist aber je nach konkreter Realisierung des Peripheriesystems nur mit hohem Aufwand möglich. Daher erlaubt das ADES-Peripheriekonzept in einem solchen Fall den gesamten Buszyklus als ausgefallen zu betrachten.

4.4.6 Vergleich beider Zugriffsverfahren

Die Wahl zwischen beiden Zugriffsverfahren läßt sich nicht allgemein entscheiden. Vielmehr hängt sie von einer Reihe von anwendungsspezifischen Faktoren ab.

Auf den ersten Blick erscheint die exklusive Kanalnutzung kostengünstiger, da handelsübliche Standardfeldbuskomponenten verwendet werden können und keine Busguardians benötigt werden. Sofern die verwendeten Komponenten das in Abschnitt 4.4.3 beschriebene Zugriffsverfahren unterstützen, können sie ohne Veränderungen genutzt werden. Dies ist prinzipiell bei fast allen heute verwendeten Feldbussen der Fall, bzw. erfordert nur geringe Softwareänderungen. Ein weiterer Vorteil der exklusiven Kanalnutzung liegt darin, daß beim Ausfall eines Rechners die Ansteuerung der Peripherie nicht betroffen ist, während bei der gemeinsamen Kanalnutzung der Ausfall des aktuellen Busmasters zum Ausfall eines kompletten Peripheriezyklus führen kann.

Allerdings benötigt man für die Ansteuerung der Aktorik zusätzliche fail-safe bzw. fail-operational Hardwarevotierer; je nach Anwendungsfall könnten diese mit deutlich weniger Aufwand in eine fail-safe bzw. fail-operational Busanschaltung integriert werden. Darüber hinaus können

fail-safe und fail-operational Busanschaltungen zusätzliche Funktionalität bei Test und Diagnose bieten (siehe Abschnitt 4.4.2) und damit Peripheriebus und Rechnerkern entlasten. In Kapitel 6 wird gezeigt, daß ein System mit fail-safe Busanschaltungen daher i. allg. auch sicherer ist als eins mit redundant verwendeten normalen Busanschaltungen. Zwar können fail-safe und fail-operational Busanschaltungen auch zusammen mit der exklusiven Kanalnutzung verwendet werden; allerdings ist dann keine Skalierbarkeit gegeben, d. h. für sichere und fail-operational Systeme werden unterschiedliche Busanschaltungen benötigt.

Für eine Reihe von Anwendungen ist neben den Kosten auch der Platzbedarf ein wichtiger Faktor. Ein System mit exklusiver Kanalnutzung wird generell mehr Platz benötigen als ein vergleichbares System mit gemeinsamer Kanalnutzung, da ein zusätzliches Buskabel benötigt wird, und zwei normale Busanschaltungen (jeweils mit eigenem Steckverbinder, Stromversorgung und Gehäuse) mehr Platz als eine äquivalente fail-safe Busanschaltung benötigen. Wird die exklusive Kanalnutzung mit fail-safe oder fail-operational Busanschaltungen verwendet, ist immer noch mit einem leicht höheren Platzbedarf zu rechnen, da die Busanschaltungen zusätzliche Steckverbindungen für den Busanschluß benötigen – industrietaugliche Steckverbinder haben einen gegenüber der Busanschaltung nicht zu vernachlässigenden Platzbedarf.

Neben den eher praktischen Aspekten „Kosten“und „Platzbedarf“ besteht auch ein ganz wesentlicher konzeptioneller Unterschied zwischen beiden Verfahren: bei der exklusiven Kanalnutzung ist ein einzelner Kanal eine Erweiterung des Kernrechners in die Peripherie hinein und kann nicht getrennt betrachtet werden. Bei der gemeinsamen Kanalnutzung ist das Peripheriesystem dagegen ein eigenständiges Subsystem. Rechnerkern, Peripheriebussystem und die Busanschaltungen können daher weitgehend unabhängig voneinander betrachtet und entwickelt werden. Obwohl die einzelnen Komponenten komplizierter sind als ihre Gegenstücke bei exklusiver Kanalnutzung, ist das Gesamtsystem durch die bessere Modularisierung insgesamt einfacher zu entwickeln, zu testen und auch zu erweitern.

Insgesamt erscheint es daher sinnvoll, bei relativ einfachen Anwendungen, bei denen keine besonderen Anforderungen an den Platzbedarf oder die nachträgliche Erweiterbarkeit bestehen, ein System mit exklusiver Kanalnutzung und normalen Standardbusanschaltungen zu verwenden – insbesondere für Anwendungen, die lediglich fail-safe sein sollen. Ansonsten sprechen die in den vorherigen Abschnitten aufgeführten Vorteile für den Einsatz der gemeinsamen Kanalnutzung.

4.5 Die Peripheriesystemsoftware

Die Peripheriesystemsoftware im Rechnerkern bildet die Schnittstelle zwischen Anwendung und Peripheriesystem. Sie arbeitet wie das Peripheriebussystem ebenfalls zyklisch. Ein Peripheriezyklus besteht aus der Ermittlung der Sendedaten aus den virtuellen Ausgangspunkten, dem eigentlichen Buszyklus und der Ermittlung der virtuellen Eingangspunkte aus den empfangenen Daten.

Prinzipiell wäre es vorteilhaft, Eingangsdaten direkt nach ihrem Empfang zu verarbeiten und die

berechneten Ausgangsdaten noch im selben Buszyklus an die Aktoren zu senden. Dies würde jedoch eine deutlich größere Anzahl von Synchronisationspunkten verlangen, da nicht mehr alle Eingangs- und alle Ausgangsdaten auf einmal votiert werden könnten. Wie in Abschnitt 4.3 beschrieben, sollte die Anzahl der Synchronisationspunkte jedoch möglichst gering gehalten werden. Daher ist ein Buszyklus aus Sicht des Rechnerkerns atomar, d. h. die Daten für die realen Ausgangspunkte müssen vor Beginn des Zyklus vollständig vorliegen und Daten der realen Eingangspunkte können erst nach dem Ende des Zyklus abgerufen werden. Dies entspricht der Praxis heutiger Automatisierungssysteme, insbesondere speicherprogrammierbarer Steuerungen, bei denen ein sog. „Prozeßabbild“ von der Sensorik ermittelt und in der Steuerung verarbeitet wird; danach werden die Ergebnisse auf einmal an die Aktorik ausgegeben.

Als erster Schritt innerhalb eines Peripheriezyklus wird das momentane Abbild der virtuellen Ausgangspunkte den entsprechenden realen Ausgangspunkten zugeordnet. Diese Daten sind durch das ADES Laufzeitsystem bereits während der Übergabe an das Peripheriesystem votiert worden. In einem XE-System ist die Vorverarbeitung damit im Prinzip abgeschlossen und der Buszyklus kann begonnen werden. In einem XG-System dagegen müssen zuerst noch die End-To-End Prüfsummen der zu sendenden Nachrichten bestimmt und votiert, sowie der aktuelle Busmaster bestimmt werden, bevor der Busmaster senden darf. Die Beobachter prüfen dabei, ob der Busmaster auch tatsächlich die vorher vereinbarten Daten sendet.

Nach Abschluß des Buszyklus überprüft jeder Rechner, ob es während des Zyklus zu Übertragungsfehlern oder Nachrichtenverlusten gekommen ist, und setzt die Statusinformation der realen E/A-Punkte entsprechend. Dann werden Wert und Status der virtuellen Eingangspunkte aus den zugeordneten realen Eingangspunkten ermittelt. Die Anwendung kann nun die neuen Daten abholen und weiter verarbeiten. Um zu verhindern, daß die einzelnen Kernrechner im Falle eines asymmetrischen Fehlers unterschiedliche Eingangsdaten verarbeiten, müssen diese ebenfalls votiert werden. Die Notwendigkeit zu votieren ist unabhängig von der verwendeten Architektur. Auch bei gemeinsamer Kanalnutzung kann es durch SOS-Fehler (siehe Kapitel 2.7.3) zu asymmetrischen Fehlern kommen. Die Votierung selber wird vom ADES Laufzeitsystem während der Datenübergabe an die Anwendung durchgeführt.

Neben dem Datenaustausch mit der Anwendung überwacht die Peripheriesystemsoftware fortlaufend den Zustand des Peripheriebussystems und meldet entdeckte Fehler dem ADES Laufzeitsystem, damit dieses entsprechende Maßnahmen einleiten kann.

Durch den atomaren Buszyklus ergibt sich ein Konflikt, der in Bild 4.12 dargestellt ist. Um eine möglichst niedrige Totzeit zu erreichen, sollte die Bearbeitung der empfangenen Eingangsdaten in der Anwendung abgeschlossen sein, bevor der nächste Peripheriezyklus beginnt, damit die berechneten Ausgabewerte im nächsten Buszyklus gesendet werden können. Für eine möglichst optimale Auslastung des Peripheriebusses sollte aber ein neuer Peripheriezyklus möglichst direkt auf das Ende des vorherigen Buszyklus folgen und sowohl die Ermittlung der virtuellen Eingangspunkte als auch die Berechnungen innerhalb der Anwendung überlappend zum Busgeschehen ablaufen; die berechneten Ausgaben können dann aber erst im übernächsten Buszyklus gesendet werden.

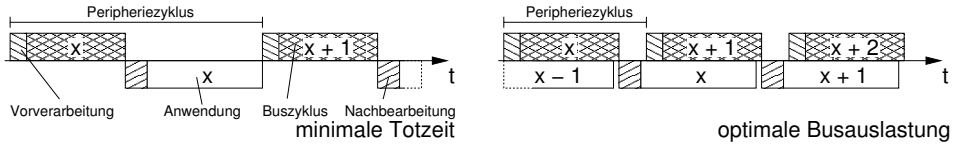

Bild 4.12: Konflikt zwischen minimaler Totzeit und optimaler Busauslastung

5 Abschätzung der erreichbaren Reaktionszeiten

5.1 Einleitende Überlegungen

In diesem Kapitel werden die Faktoren betrachtet, die konzeptbedingt Auswirkungen auf die erreichbaren Reaktionszeiten des ADES Peripheriesystems haben. Dazu wird von dem Beispiel „Brennersteuerung“ aus Kapitel 4.4.1 ausgegangen. Die Temperatur innerhalb der Brennkammer soll mit Hilfe einer einfachen Zweipunktregelung geregelt werden. Bei diesem Beispiel gibt es zwei relevante Reaktionszeiten. Dies ist zum einen die Zeit zwischen dem Unterschreiten einer unteren Temperaturschwelle bis zur Zündung des Brenners und zum anderen die Zeit zwischen dem Überschreiten einer oberen Temperaturschwelle bis zum Abschalten des Brenners.

Es werden nur Faktoren berücksichtigt, auf die das Peripheriesystem direkt Einfluß hat, d. h. eine Temperaturschwelle gilt erst dann als über- oder unterschritten, wenn entsprechende Meßwerte an einer Temperatursonde verfügbar sind, und die gewünschte Reaktion gilt als eingeleitet, wenn entsprechende Befehle an die Aktorik übergeben worden sind. Da es keinen Unterschied zwischen den Reaktionen eines Fail-operational und eines fail-safe Systems gibt, wird nur letzteres behandelt. Demgegenüber gibt es sehr wohl einen Unterschied zwischen Systemen mit exklusiver oder gemeinsamer Kanalnutzung (im folgenden mit SE bzw. SG abgekürzt; siehe Kapitel 4.4). Für das SG-System wird ein einfacher Busguardian verwendet, der einen Rechner nur vollständig vom Peripheriebus trennen kann.

Die notwendige Peripherie für diese Aufgabe wird wie folgt angenommen:

- Sensorik
  Die Sensorik besteht aus zwei Temperatursonden, die im SE-System an zwei normale, und im SG-System an eine fail-safe Busanschaltung angeschlossen sind. Die Sonden werden nur am Anfang eines Peripheriezyklus abgetastet und das Ergebnis in zwei Bytes kodiert noch im selben Zyklus an den Rechnerkern gesendet.

- Aktorik
  Sowohl die Brennstoffpumpe, als auch die Zündvorrichtung sind an eine fail-safe Busanschaltung angeschlossen, die vom Rechnerkern die Befehle „Brennstoffzufuhr ein“ und „Brennstoffzufuhr aus“ erwartet und ausführen kann. Die Busanschaltung meldet in jedem Zyklus ihren aktuellen Status an den Rechnerkern. Der Befehl und die Statusrückmeldung werden beide mit jeweils einem Byte kodiert. Bei einem System mit

gemeinsamer Kanalnutzung muß ein „Brennstoffzufuhr ein“-Befehl in zwei aufeinanderfolgenden Peripheriezyklen empfangen werden, um gültig zu sein. Kann kein gültiger Befehl empfangen werden, wird der Brenner abgeschaltet.

In der Praxis wird ein solch einfaches System kaum vorkommen, bzw. nicht über einen Bus betrieben werden. Daher wird angenommen, daß die Temperaturregelung innerhalb der Brennkammer nur ein Teil eines größeren Systems ist. Um eine realistischere Busauslastung zu erreichen, enthalte das System daher 36 weitere normale Busanschaltungen im Falle der exklusiven Kanalnutzung bzw. 18 fail-safe Busanschaltungen im Falle der gemeinsamen Kanalnutzung. An jeweils die Hälfte dieser Teilnehmer sei Sensorik, an die andere Hälfte Aktorik angeschlossen, wobei die Anzahl der zu transportierenden Nutzdaten denen der für die Temperaturregelung benötigten Busanschaltungen entspreche. Die genaue Funktion der übrigen Geräte ist für die weitere Betrachtung unwichtig.

5.2 Abschätzung der Dauer der Busübertragung

Im Gegensatz zu parallel verdrahteter Peripherie entstehen durch die Datenübertragung über einen Feldbus nicht zu vernachlässigende Datenübertragungszeiten. Sie hängen ab von der Länge der eigentlichen Nutzdaten, dem durch das verwendete Busprotokoll verursachten zusätzlichen Kommunikationsoverhead und der Übertragungsrate des verwendeten Busses.

Für den ADES-Prototypen wurde als grundlegendes Bussystem TTP/A verwendet. Eine detaillierte Begründung für diese Entscheidung wird in Kapitel 7.3.1 dargestellt. Da TTP/A ein TDMA-Buszugriffsverfahren verwendet und keine besonderen Kodierungen nutzt, ist die Berechnung der Übertragungszeit trivial möglich; gleichzeitig ist TTP/A z. Zt. noch nicht besonders weit verbreitet. Daher soll in diesem Abschnitt die zu erwartende Übertragungsbandbreite anhand eines anderen Bussystems, des Control Area Networks (CAN), abgeschätzt werden.

CAN ist kein Feldbus im eigentlichen Sinne, da es nur die ersten beiden Schichten des OSI-Referenzmodelles für Kommunikationssysteme spezifiziert (physikalisches Medium und Nachrichtentransport über dieses Medium), auf die höhere Protokolle aufgesetzt werden können. CAN verwendet zur Datenübertragung vier verschiedene Arten von Nachrichten. Der erste Nachrichtentyp ist eine normale Datennachricht. Sie enthält eine eindeutige Kennung, eine Anzahl von Steuerbits, 0 bis 8 Bytes Nutzdaten und eine 15 Bit CRC-Prüfsumme. Die Nachricht wird nicht an einen bestimmten Empfänger adressiert, sondern an alle anderen Teilnehmer gesendet; diese können anhand der Kennung feststellen, ob sie die Daten auswerten sollen. Die Nachricht enthält ebenfalls keinen Absender – bei Bedarf muß eine höhere Protokollschicht für eine eindeutige Zuordnung zwischen Kennung und Sender sorgen. Der zweite Nachrichtentyp ist eine Datenanforderungsnachricht, mit der ein Sender die zu einer bestimmten Kennung gehörenden Daten anfordert; der angesprochene Teilnehmer antwortet auf diese Nachricht mit einer normalen Datennachricht. Eine Datenanforderungsnachricht ist genauso aufgebaut wie die zugehörige Datennachricht, enthält aber kein Nutzdatenfeld. Die beiden anderen Nachrichtentypen, die Fehler- und die Überlastnachricht, bestehen aus jeweils 6 gleichen Bits und einer nachfolgenden Pause von 8 Bitübertragungszeiten. Eine Fehlernachricht wird gesendet, wenn

ein Teilnehmer eine fehlerhafte Nachricht empfangen hat; der Sender der fehlerhaften Nachricht versucht daraufhin, seine Nachricht erneut zu senden. Eine genauere Beschreibung von CAN findet sich in [2] und [15].

Um sicherzustellen, daß für die Synchronisierung der einzelnen Teilnehmer genügend Flanken im Bitstrom vorhanden sind, verwendet CAN das sog. Bitstuffing, d. h. nach jeweils fünf Bits mit der gleichen Polarität wird ein Bit mit umgekehrter Polarität eingefügt (bei Überlast- und Fehlernachrichten wird diese Regel bewußt verletzt). Der Empfänger erkennt diese Einfügung und entfernt das entsprechende Bit wieder aus dem Bitstrom. Nimmt man für den Datenbereich den ungünstigsten Fall an (alle Bits haben die gleiche Polarität), berechnet sich die Zeit für die Übertragung einer CAN Nachricht nach [38] damit zu:

$$t_{\mathrm{N}} = \left(\left\lfloor \frac{34 + 8d}{5} \right\rfloor + 47 + 8d \right) * t_{\mathrm{Bit}}{}^{1)} \tag{5.1}$$

t_{Bit} Bitzeit (abhängig von der Datenrate des Bussystems).

d Die Länge der Nutzdaten in Bytes

In einem CAN-System sind alle Teilnehmer gleichberechtigt; es gibt keine zentrale Buszuteilung. Statt dessen wird das in Kapitel 2.7.2 beschriebene CSMA/CA-Verfahren verwendet. Das für ADES vorgesehene zyklische Peripheriebusprotokoll (siehe Kapitel 4.4.3) wird daher auf CAN aufgesetzt. In einem SG-System muß zusätzlich zu den eigentlichen Nutzdaten auch noch die End-To-End Prüfsumme mit übertragen werden. Bei den sehr kurzen Nachrichtenlängen reicht dabei eine 16 Bit Prüfsumme aus. Ausgehend von Formel 5.1 kann nun die Anzahl der pro Nachricht tatsächlich zu übertragenden Bits bestimmt werden. Es werden dabei insgesamt fünf verschiedene Arten von Nachrichten benötigt:

1. Ankündigung eines neuen Peripheriezyklus
   Mit dieser Nachricht teilt der Rechnerkern allen anderen Teilnehmern mit, daß ein neuer Peripheriezyklus beginnt. Die Sensorik wird damit gleichzeitig dazu aufgefordert, ihre Meßwerte zu bestimmen und zum Abruf durch den Rechnerkern bereitzuhalten. Die Nachricht enthält keine Nutzdaten und ist daher 53 Bits lang.

2. Datenanforderung vom Rechnerkern an die Sensorik
   Mit dieser Nachricht fordert der Rechnerkern die Sensorik auf, den aktuellen Meßwert zurückzugeben. Dieser Nachrichtentyp enthält ebenfalls keine Nutzdaten und ist somit ebenfalls 53 Bits lang.

3. Datenantwort von einer Sensorbusanschaltung an den Rechnerkern
   Diese Nachricht enthält den aktuellen Meßwert, ausgedrückt in zwei Bytes Nutzdaten. In einem SG-System kommen noch zwei Bytes für die Prüfsumme hinzu, so daß diese Nachricht in einem SE-System maximal 73 und in einem SG-System maximal 92 Bits lang ist (die genaue Länge ist wegen des verwendeten Bitstuffings datenabhängig).

4. Aktorbefehle vom Rechnerkern an eine Aktorbusanschaltung

[1)] Die Konstanten ergeben sich aus den Längen der übrigen Felder einer CAN-Nachricht.

Hier gelten die gleichen Überlegungen wie für eine Datenantwort einer Sensorbusanschaltung, nur das lediglich ein Byte Nutzdaten übertragen werden. Die maximale Länge ist in einem SE-System somit 63 und in einem SG-System 82 Bits.

5. Statusantwort einer Aktorbusanschaltung an den Rechnerkern
   Diese Nachricht entspricht in seiner Länge einem Aktorbefehl und ist damit ebenfalls maximal 63 bzw. 82 Bits lang.

Mit den anfangs gemachten Annahmen über die Zusammensetzung der Peripheriebusteilnehmer müssen in einem Peripheriezyklus neben einer Ankündigungsnachricht jeweils 10 Nachrichten von jedem Typ übertragen werden. Für die weitere Betrachtung sollen optimale Übertragungsbedingungen angenommen werden, d. h. die Nachrichten werden ohne Zwischenpausen direkt aufeinander abfolgend gesendet. Die reine Datenübertragung für einen Peripheriezyklus benötigt damit in einem SE-System mit der mit CAN maximal möglichen Datenrate von 1MBaud ca. 2.5ms, während in einem SG-System ca. 3.1ms benötigt werden.

Bei CAN muß ferner berücksichtigt werden, daß ein CAN-Buscontroller versucht, seine Nachricht mehrfach zu wiederholen, wenn einer der anderen Teilnehmer einen Übertragungsfehler entdeckt hat. In ADES ist dieses Verhalten zwar unerwünscht, da dies den Peripheriezyklus verlängert und damit das Zeitverhalten stört. Es läßt sich aber nicht vermeiden, da dies eine grundlegende Eigenschaft von CAN ist. Um zu verhindern, daß jede Übertragungsstörung den Peripheriezyklus durcheinanderbringt, muß daher zusätzliche Zeit für die mögliche Wiederholung mehrerer Nachrichten eingeplant werden.

Im folgenden wird kurz abgeschätzt, wieviele Wiederholungen sinnvoll einzuplanen sind. Es wird dabei eine Bitfehlerwahrscheinlichkeit von 10^{-5} (siehe Tabelle 2.3) und binomialverteilte Übertragungsfehler angenommen. Eine CAN-Nachricht ist maximal 130 Bits lang, so daß die Wahrscheinlichkeit p, daß eine Nachricht bei der Übertragung gestört wird, in etwa $1,30 * 10^{-3}$ beträgt. Die Wahrscheinlichkeit $P(n)$, daß während eines Peripheriezyklus mehr als n Nachrichten wiederholt werden müssen, kann mit der folgenden Formel bestimmt werden:

$$P(n) = 1 - \sum_{i=0}^{n} \left(\begin{array}{c} 41+n \\ i \end{array} \right) p^i (1-p)^{41+n-i} \tag{5.2}$$

Nimmt man z. B. $n = 6$ an, ergibt sich eine Wahrscheinlichkeit $P(6) \approx 3.77 \cdot 10^{-13}$. Bei dieser Wahrscheinlichkeit wäre erst nach über 200 Jahren Dauerbetrieb mit einer Störung durch die automatische Wiederholung im Fehlerfall zu rechnen. Die oben berechnete benötigte Zeit für die Datenübertragung muß dann für ein SE-System auf etwa 3ms, und für ein SG-System auf etwa 3.7ms verlängert werden (dieser Wert errechnet sich aus der benötigten Zeit, um die längste vorkommende Nachricht sechs mal zu wiederholen).

Daraus läßt sich noch nicht die tatsächlich erreichbare Peripheriezykluszeit bestimmen. Neben der reinen Übertragungszeit müssen noch Verarbeitungszeiten im Rechnerkern und den Busanschaltungen, sowie Synchronisationszeiten im Rechnerkern berücksichtigt werden. Diese lassen sich nicht allgemein abschätzen, sondern können immer nur für die konkrete Implementierung angegeben werden. Der im Rahmen dieser Arbeit entstandene ADES-Prototyp (siehe Kapitel 7)

benötigt ca. 4ms, wobei die reine Verarbeitungszeit vernachlässigt werden kann. Wie in Kapitel 8 ausführlicher dargestellt, ist diese Größenordnung auch für Systeme zu erwarten, die deutlich komplexer sind als der Demonstrator. Insgesamt erscheint damit ein Peripheriezyklus von ungefähr 8ms durchaus realistisch.

5.3 Bestimmung der Reaktionszeit

Bild 5.1 zeigt eine Reihe von Faktoren, die Einfluß auf die tatsächlich zu erreichende Reaktionszeit haben.

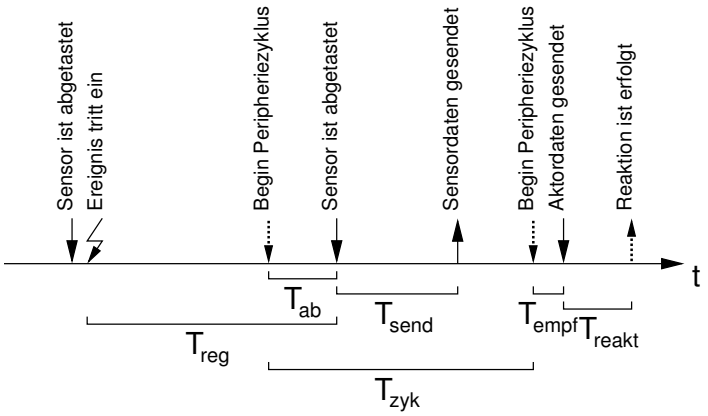

Bild 5.1: Bestimmung der Reaktionszeit

T_{zyk} ist die Dauer eines Peripheriezyklus, T_{reg} die Zeit zwischen dem Eintritt des Ereignisses (Über- oder Unterschreiten einer Temperaturschwelle) und der Abtastung des Sensors. T_{reg} ist maximal einen Peripheriezyklus lang, wenn das Ereignis direkt nach dem vorhergehenden Abtastzeitpunkt eintritt. Die übrigen Zeiten T_{ab} (Dauer der Abtastung vom Anfang des Peripheriezyklus), T_{empf} (Zeitdauer, bis Aktor Befehle nach Anfang des Peripheriezyklus erhalten hat) und T_{reakt} (Zeitdauer zwischen dem Empfang eines Befehls und seiner Ausführung im Aktor) sind durch die Implementierung der Busanschaltungen und die Aufteilung des Peripheriezyklus vorgegeben und konstant. Es ergibt sich eine maximale Reaktionszeit von $2T_{zyk} - T_{ab} + T_{empf} + T_{reakt}$. Im Interesse einer minimalen Reaktionszeit sollte innerhalb des Peripheriezyklus die Sensorik daher möglichst spät abgetastet und die Aktorik möglichst früh angesteuert werden. In einem System mit gemeinsamer Kanalnutzung dauert das Einschalten der Brennstoffzufuhr einen Peripheriezyklus länger, da zwei gültige Befehle in aufeinanderfolgenden Peripheriezyklen empfangen werden müssen. Das Abschalten dagegen erfolgt genauso schnell wie bei einem System mit exklusiver Kanalnutzung.

Wie bereits in Kapitel 2.5 beschrieben werden die beiden Temperatursonden von einander abweichende Werte aufweisen. Es wird daher ein Votierungsalgorithmus benötigt, mit dessen Hilfe der korrekte Wert bestimmt bzw. ein Ausfall erkannt werden kann. Bei diesem Beispiel ist es zwar unwahrscheinlich, daß die beiden Sensoren kurzzeitig außerhalb der erlaubten Maximalabweichung liegende Meßwerte melden werden; bei binären Sensoren ist dagegen durchaus mit

diesem Fall zu rechnen. Unabhängig davon, wo die Votierung durchgeführt wird (im Rechnerkern oder einer fail-safe Busanschaltung) ergibt sich daher im ungünstigsten Fall eine weitere Verzögerung. Ihre Dauer beträgt die maximal erlaubte Dauer der Abweichung (vorher kann kein Fehler erkannt werden), aufgerundet auf ein ganzzahliges Vielfaches eines Peripheriezyklus. Allerdings wird diese Verzögerung von der durch die Abtastung verursachten Verzögerung T_{reg} überlagert und hat daher keinen weiteren Einfluß auf die maximale Reaktionszeit, wenn ihre Dauer kürzer als T_{reg} ist.

Bei den obigen Überlegungen wurden Übertragungsfehler nicht berücksichtigt. In einem realen System ist aber auf jeden Fall mit transienten Störungen der Datenübertragung zu rechnen, die nicht zu einem sofortigen Abschalten des Systems führen sollten. An dieser Stelle muß unterschieden werden, welches Bussystem als Grundlage für das Peripheriesystem verwendet wird.

Bei einem CAN-System werden, wie oben beschrieben, fehlerhafte Pakete vom Sender wiederholt. Die in CAN eingebaute Fehlererkennung kann zwar nur Fehler erkennen, die bei der Datenübertragung zwischen zwei CAN-Controllerbausteinen auftreten; dies ist aber der am häufigsten zu erwartende Fehlerfall. Da CAN mit seiner 15 Bit CRC-Prüfsumme eine brauchbare Fehlererkennung bietet, ist davon auszugehen, daß nur mit Hilfe der End-To-End Prüfsumme erkennbare Fehler auf schwerwiegendere Störungen im System hinweisen. Es macht daher Sinn, bei einer fehlerhaften End-To-End Prüfsumme das System sofort abzuschalten und nicht erst abzuwarten, ob dies nur eine transiente Störung war, und sich ansonsten auf die automatische Nachrichtenwiederholung von CAN zu verlassen (siehe auch den vorhergehenden Abschnitt). Die Reaktionszeit beträgt dann zwei Zyklen für das SE-, und drei Zyklen für das SG-System (ein Aktorbefehl muß in zwei aufeinanderfolgenden Zyklen empfangen werden, um gültig zu sein); dies entspricht mit den in Abschnitt 5.2 getroffenen Überlegungen einer Reaktionszeit von 16 bzw. 24 Millisekunden.

Bei Bussystemen, die keine Wiederholung im Fehlerfall bieten, muß das System dagegen eine gewisse Zahl von fehlerhaften Nachrichten erlauben, ohne das abgeschaltet wird. Ein Sender wird erst dann als ausgefallen betrachtet, wenn er innerhalb eines bestimmten Zeitraumes nach dem Senden einer ungültigen Nachricht erneut eine ungültige Nachricht sendet. Sowohl bei der Bestimmung der Sensordaten als auch der Ansteuerung des Aktors muß daher im ungünstigsten Fall mit einem weiterer Peripheriezyklus Verzögerung gerechnet werden. Für das SE-System ergibt sich damit eine maximale Reaktionszeit von vier, für das SG-System von fünf Peripheriezyklen. Dies entspricht mit den in Abschnitt 5.2 getroffenen Überlegungen einer Reaktionszeit von 32 bzw. 40 Millisekunden.

Wird ein Busguardian verwendet, der selektiv nur das Senden unterbinden kann (siehe Kapitel 4.4.5), ist es nicht notwendig zu verlangen, daß ein Aktorbefehl nur dann gültig ist, wenn er in zwei aufeinanderfolgenden Peripheriezyklen empfangen wurde. Die Reaktionszeiten des SG-Systems lassen sich dann auf die des SE-Systems reduzieren. Tabelle 5.1 fasst die Reaktionszeiten für die verschiedenen Varianten zusammen.

Systemart	Wiederholung von Nachrichten im Fehlerfall	
	mit	ohne
SE	2	4
SG	3	5
SG (mit erweitertem Busguardian)	2	4

Tabelle 5.1: Maximale Reaktionszeiten in Buszyklen für verschiedene Varianten des Peripheriesystems

6 Qualitative Zuverlässigkeitsanalyse

In diesem Kapitel wird anhand eines sehr einfachen Beispiels die Verfügbarkeit und die Sicherheit eines OEN- mit dem eines OG-Systems verglichen. Als weiterer Vergleich werden beide Systeme einem normalen, nicht redundanten System gegenübergestellt.

6.1 Einführende Bemerkungen

Als Beispiel soll eine einfache Tankfüllstandsregelung dienen. Der Tank hat einen vom System nicht steuerbaren Zulauf; steigt der Flüssigkeitspegel im Tank über einen bestimmten Pegel, sollen die Ablaufventile geöffnet werden. Der sichere Zustand für dieses System sei, daß keine Flüssigkeit aus dem Tank auslaufen kann. In Bild 6.1 ist dieses Beispiel einmal als OEN- und einmal als OG-System dargestellt.

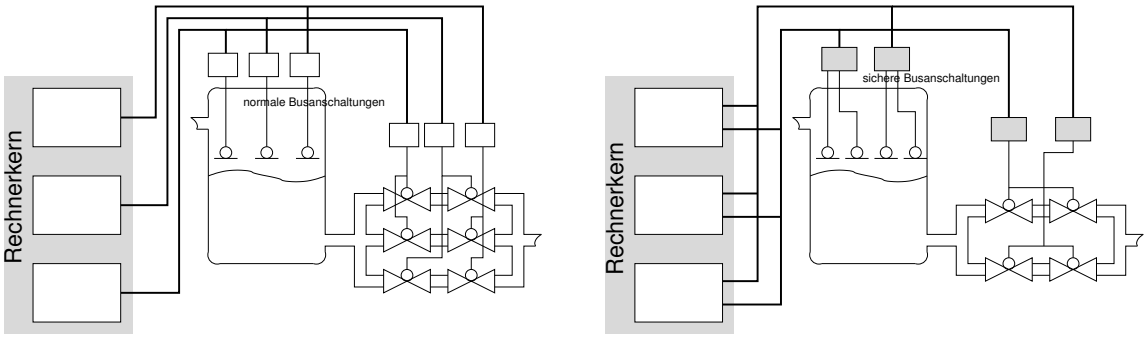

Bild 6.1: Tankfüllstandsregelung

Für diese Analyse werden die folgenden Einzelereignisse angenommen:

R Innerhalb des Beobachtungszeitraumes fällt ein beliebiger Kernrechner aus. Im Falle des OG-Systems wird davon ausgegangen, daß ein ausgefallener Rechner vom Bus getrennt wird und er die gemeinsamen Peripheriekanäle nicht blockieren kann.

R_k Ein Kernrechner fällt aus und schickt falsche Befehle an die Aktorik, ohne daß der Rechner von den beiden anderen Kernrechnern abgeschaltet werden kann.

M Ein beliebiges Busmedium fällt aus.

B, $\hat{B}$ Eine normale bzw. eine fail-safe Busanschaltung fällt auf beliebige Art aus. Es wird hier nicht zwischen Busanschaltungen für Sensoren und Aktoren unterschieden.

B_i, $\hat{B}_i$ Eine normale bzw. eine fail-safe Busanschaltung fällt aus und blockiert als „babbling idiot“ einen Peripheriekanal.

B_k Eine normale Aktorbusanschaltung öffnet fälschlicherweise ihre angeschlossenen Ventile.

$\hat{B}_k$ Eine fail-safe Aktorbusanschaltung fällt unsicher aus, d. h. entweder sie kann ihre Ventile nicht mehr schließen, oder sie öffnet sie zur falschen Zeit.

S Ein Sensor fällt aus. Es wird angenommen, daß sich der Füllstand häufig genug ändert, daß ein „stuck at“-Fehler eines Sensors nicht längere Zeit unbemerkt bleiben kann.

A Ein Ventil oder die Leistungselektronik der zugehörigen Busanschaltung ist ausgefallen. Die Leistungselektronik wird hier zum Aktor und nicht zur Busanschaltung gerechnet, da sie für jeden Aktor getrennt ausfallen kann.

A_k, $\hat{A}_k$ Ein Ventil öffnet sich selbständig durch einen Fehler im Ventil selber oder in der Leistungselektronik der zugehörigen normalen bzw. fail-safe Busanschaltung.

Andere mögliche Ausfallereignisse wie z. B. der Ausfall der Energieversorgung werden nicht berücksichtigt. In der Herleitung der Formeln für die Ausfallwahrscheinlichkeit des Gesamtsystems ist mit X nicht das entsprechende Ereignis selber, sondern die Wahrscheinlichkeit $P(X)$ für das Eintreten dieses Ereignisses gemeint. Dies macht die Formeln etwas lesbarer.

Die Lebensdauer einer Komponente kann in guter Näherung durch eine Exponentialverteilung mit der Wahrscheinlichkeitsdichtefunktion $f(t) = \lambda e^{-\lambda t}$ beschrieben werden. Damit ergibt sich die Wahrscheinlichkeit, daß eine Komponente i nach der Zeit T noch funktionsfähig ist, zu $F_i(T) = \int_0^T f(t)dt = 1-e^{-\lambda_i T}$. Geht man von einem festen Beobachtungszeitraum aus, können Komponentenausfälle vereinfachend als binomialverteilte Ereignisse mit der Wahrscheinlichkeit $P_i = 1 - F_i(T)$ angenommen werden. Für den Vergleich seien die Ausfallwahrscheinlichkeiten aller betrachteten Komponenten konstant, voneinander stochastisch unabhängig und $\leq 10^{-3}$. Eine Reparatur findet während des gesamten Beobachtungszeitraumes nicht statt. Transiente Fehler werden nicht gesondert betrachtet. Für die Betrachtung der Sicherheit wird im folgenden noch die zusätzliche Annahme getroffen, daß sich das System gleich häufig in den beiden Zuständen „Ventile offen“ und „Ventile geschlossen“ befindet, d. h. die Wahrscheinlichkeit in einem der beiden Systemzustände zu sein beträgt jeweils 0.5.

6.2 Unverfügbarkeit des OEN-Systems

Die Unverfügbarkeit des Systems ergibt sich direkt aus der Wahrscheinlichkeit für den Ausfall des Gesamtsystems. Um diese einfacher berechnen zu können, werden zuerst einige Systemkomponenten zusammengefaßt und die folgenden Hilfsereignisse eingeführt:

K Ein Peripheriekanal ist ausgefallen. Das ist der Fall, wenn entweder das Medium selber ausgefallen ist oder Sensor- oder Aktorbusanschaltung als „babbling idiot“ den Kanal

blockiert. Nach dem *Satz von Sylvester* ([7]) ergibt sich mit den in Abschnitt 6.1 eingeführten Größen folgende Wahrscheinlichkeit für K:

$$K = M + 2B_i - {B_i}^2 - 2B_iM + {B_i}^2M$$

S_e Eine Sensoreinheit ist ausgefallen, ohne dabei den gesamten Kanal zu blockieren. Die Sensoreinheit besteht aus der Busanschaltung und einem Sensor. Sie gilt als ausgefallen, wenn entweder die Busanschaltung oder der Sensor ausgefallen ist. Die Wahrscheinlichkeit S_e berechnet sich zu:

$$S_e = B + S - BS - (1 - S)B_i$$

A_e Eine Aktoreinheit ist ausgefallen, ohne dabei den gesamten Kanal zu blockieren. Die Aktoreinheit besteht aus einer Busanschaltung und beiden angeschlossenen Aktoren. Sie gilt als ausgefallen, wenn entweder die Busanschaltung oder einer der Aktoren ausgefallen ist. Diese Festlegung führt zu einem pessimistischen Ergebnis, da bei der gegeben Aktoranordnung Fälle möglich sind, bei denen ein Aktor jeder Busanschaltung ausgefallen ist (und damit nach dieser Festlegung jede Aktoreinheit), ohne zu einem Systemausfall zu führen. Die Berechnung wird aber stark vereinfacht, da die geometrische Anordnung der Aktoren nicht berücksichtigt werden muß. Die Wahrscheinlichkeit A_e berechnet sich zu:

$$A_e = B - (1 - A)^2 - B_i(1 - A)^2 - B(1 - (1 - A)^2) + 1$$

In Tabelle 6.1 ist aufgeführt, wieviele der betrachteten Ereignisse jeweils eintreten müssen, damit das Gesamtsystem ausfällt. Die Abhängigkeiten der Komponenten untereinander sind dabei in der Tabelle bereits berücksichtigt (wenn z. B. ein Kernrechner ausgefallen ist, sind alle an seinen Peripheriekanal angeschlossenen Peripherieeinheiten nicht mehr ansprechbar, egal ob sie selber ausgefallen sind oder nicht).

R	K	S_e	A_e
0/3	0/3	0..1/3	2..3/3
		2..3/3	—
	1/3	0/2	1..2/2
		1..2/2	—
	2..3/3	—	—
1/3	0/2	0/2	1..2/2
		1..2/2	—
	1..2/2	—	—
2..3/3	—	—	—

Tabelle 6.1: Ereigniskombinationen, die zum Ausfall des gesamten OEN-Systems führen. Dabei bedeutet $0..1/3$, daß 0 bis 1 von 3 insgesamt möglichen Ereignissen diesen Typs aufgetreten sind.

Die Berechnung der Ausfallwahrscheinlichkeit des Gesamtsystems kann nun durch die Aufsummierung der verschiedenen Fälle erfolgen. Es soll hier exemplarisch der erste (und zugleich aufwendigste) Fall der Tabelle betrachten werden:

$$P = (1-R)^3(1-K)^3[((1-S_e)^3 + \tbinom{3}{1}S_e(1-S_e)^2)(\tbinom{3}{2}A_e^2(1-A_e) + A_e^3)]$$

Es ist leicht zu erkennen, daß die Formel für die Gesamtausfallwahrscheinlichkeit recht groß und unhandlich wird. Bei der angenommenen Größenordnung der Einzelausfallwahrscheinlichkeiten ($\leq 10^{-3}$) werden Terme wie z. B. A_e^3 oder $S_eA_e^2$ keinen nennenswerten Beitrag zum Gesamtergebnis beisteuern. Daher liegt die Idee nahe, diese Terme einfach zu ignorieren. Daraus ergibt sich für obige Formel folgende Näherung:

$$P \approx 3(1-R)^3(1-K)^3(1-S_e)^3(1-A_e)A_e^2$$

Eine weitere Vereinfachung läßt sich erzielen, wenn man zusätzlich davon ausgeht, daß alle negierten Wahrscheinlichkeiten der Form $(1-R)$ den Wert eins besitzen. Der daraus resultierende Fehler ist bei den betrachteten Größenordnungen der Wahrscheinlichkeiten kleiner als 1% – d. h. der durch die Näherung verursachte Fehler wird damit i. allg. niedriger liegen als die Unsicherheit bei der Bestimmung der einzelnen Ausfallwahrscheinlichkeiten. Für das Gesamtsystem ergibt sich damit eine angenäherte Ausfallwahrscheinlichkeit von

$$P_{\text{exk}} \approx 3(R^2 + K^2 + S_e^2 + A_e^2 + 2RK + 2RS_e + 2RA_e + 2KS_e + 2KA_e) \tag{6.1}$$

Betrachtet man dieses Ergebnis etwas genauer, erkennt man, daß sich die angenäherte Formel für die Ausfallwahrscheinlichkeit aus der Summe der Wahrscheinlichkeiten aller primären Ausfallereignissen zusammensetzt. Ein primäres Ausfallereignis ist dabei die minimale Kombination aus Einzelereignissen, die zum Ausfall des Systems führen – z. B. der Ausfall von zwei Kernrechnern. Die angenäherte Formel kann daher auch direkt aus der Summe der primären Ausfallereignisse bestimmt werden, ohne zuerst die exakte Formel zu berechnen und diese dann mit den oben genannten Überlegungen zu vereinfachen.

Die vereinfachte Formel für die Wahrscheinlichkeit berücksichtigt nicht, daß die primären Ausfallereignisse sich teilweise überschneiden und daher einige Teilwahrscheinlichkeiten mehrfach gezählt werden. Die angenäherte Ausfallwahrscheinlichkeit bildet daher eine obere Schranke für die tatsächliche Ausfallwahrscheinlichkeit.

6.3 Unsicherheit des OEN-Systems

Das System „Tankfüllstandsregelung“ kann auf zwei Arten unsicher werden – zum einen, wenn die Ventile bereits geöffnet sind und nach einem Ausfall nicht mehr geschlossen werden können, und zum anderen, wenn die Ventile bereits geschlossen sind und fälschlicherweise geöffnet werden.

Da normale Busanschaltungen verwendet werden, führt fast jeder mögliche Systemausfall dazu, daß die Ventile nicht mehr geschlossen werden können. Die beiden einzigen Ausnahmen sind ein Ausfall der Sensorik, und ein vorzeitiges Schließen der Ventile durch einen aktiven Ausfall der Aktorik. Die Sensorik hat keinen Einfluß auf die Sicherheit des Systems, da angenommen wird, daß nur ein Sensor auf einmal ausfällt; ein Ausfall der gesamten Sensorik kann damit immer erkannt werden. Die Wahrscheinlichkeit für ein vorzeitiges Schließen der Ventile ist im Vergleich zur Gesamtausfallwahrscheinlichkeit gering (sofern $A_k \ll A$ oder $A_k \ll B$) und kann deshalb vernachlässigt werden. Daher kann die Wahrscheinlichkeit für den ersten Fall (Ventile sind geöffnet und können nicht mehr geschlossen werden) näherungsweise durch Gleichung 6.1 bestimmt werden, wenn dort die Wahrscheinlichkeit für einen Sensorausfall ignoriert wird.

Für den zweiten Fall, bei dem die geschlossenen Ventile zum falschen Zeitpunkt geöffnet werden, wird ein weiteres Hilfsereignis eingeführt:

$A_{e,k}$ Eine Aktoreinheit fällt aus, so daß sich die Ventile selbständig öffnen. Es gelten dieselben Randbedingungen und Annahmen wie für A_e, wobei aber B mit B_k und A mit A_k ersetzt werden müssen.

$$A_{e,k} = B_k - (1 - A_k)^2 - B_i(1 - A_k)^2 - B(1 - (1 - A_k)^2) + 1$$

Mit der in Abschnitt 6.2 eingeführten Näherungsmethode läßt sich die Wahrscheinlichkeit für diesen Fall nun relativ leicht zu $3(R_k^2 + A_{e,k}^2 + 2R_a A_{e,k})$ bestimmen. Insgesamt ergibt sich damit die Wahrscheinlichkeit für einen unsicheren Ausfall zu:

$$P_{exk,\textit{unsicher}} \approx \begin{array}{l} 1.5(R^2 + K^2 + A_e^2 + 2RK + 2RA_e + 2KA_e) + \\ 1.5(R_k^2 + A_{e,k}^2 + R_k A_{e,k}) \end{array} \quad (6.2)$$

6.4 Unverfügbarkeit des OG-Systems

Die Berechnung der Unverfügbarkeit des OG-Systems verläuft ähnlich der Berechnung für das OEN-System. Wie dort werden einige Hilfsereignisse eingeführt:

$\hat{K}$ Ein Peripheriekanal ist ausgefallen. Dieses Ereignis entspricht dem des OEN-Systems; es müssen aber die Einzelereignisse $\hat{B}$ und $\hat{B}_i$ an Stelle von B und B_i verwendet werden:

$$\hat{K} = M + 2\hat{B}_i - \hat{B}_i^2 - 2\hat{B}_i M + \hat{B}_i^2 M$$

$\hat{S}_e$ Eine Sensoreinheit ist ausgefallen, ohne dabei den gesamten Kanal zu blockieren. Die Sensoreinheit besteht aus der Busanschaltung und zwei Sensoren, wobei der Ausfall eines Sensors ausreicht, um die gesamte Sensoreinheit ausfallen zu lassen:

$$\hat{S}_e = \hat{B} - (1 - S)^2 - B(1 - (1 - S)^2) + 1$$

R	$\hat{K}$	$\hat{S}_e$	$\hat{A}_e$
0..1/3	0/2	0..1/2	2/2
		2/2	—
	1/2	0/1	1/1
		1/1	—
	2/2	—	—
2..3/3	—	—	—

Tabelle 6.2: Ereigniskombinationen, die zum Ausfall des gesamten Systems mit gemeinsamer Kanalnutzung führen. Dabei bedeutet $0..1/3$, daß 0 bis 1 von 3 insgesamt möglichen Ereignissen diesen Typs aufgetreten sind.

$\hat{A}_e$ Eine Aktoreinheit ist ausgefallen, ohne dabei den gesamten Kanal zu blockieren. Dieses Ereignis entspricht A_e des OEN-Systems; hier müssen ebenfalls $\hat{B}$ und $\hat{B}_i$ an Stelle von B und B_i verwendet werden:

$$\hat{A}_e = \hat{B} - (1-A)^2 - \hat{B}_i(1-A)^2 - \hat{B}(1-(1-A)^2) + 1$$

Analog zur Berechnung für die exklusive Kanalnutzung zeigt Tabelle 6.2 alle möglichen Kombinationen von Ereignissen, die zum Ausfall des Gesamtsystems führen. Die Gesamtausfallwahrscheinlichkeit kann dann mit Verwendung der in Abschnitt 6.2 beschriebenen Näherung bestimmt werden:

$$P_{\text{gem}} \approx 3R^2 + \hat{K}^2 + \hat{S}_e^2 + \hat{A}_e^2 + 2\hat{K}\hat{S}_e + 2\hat{K}\hat{A}_e \tag{6.3}$$

6.5 Unsicherheit des OG-Systems

Für die Berechnung der Unsicherheit muß gegenüber dem vorherigen Abschnitt ein weiteres Hilfsereignis eingeführt werden:

$\hat{A}_{e,k}$ Eine Aktoreinheit fällt unsicher aus. Dieses Ereignis tritt ein, wenn die Busanschaltung unsicher ausfällt oder beide Ventile sich entweder nicht mehr schließen lassen oder sich selbständig öffnen. Es wird angenommen, daß die Wahrscheinlichkeit, daß sich ein Ventil nicht mehr schließen läßt, in etwa der Gesamtausfallwahrscheinlichkeit A entspricht (also $A_k \ll A$). Daraus ergibt sich:

$$\hat{A}_{e,k} \approx \hat{B}_k + \hat{A}_k^2 - \hat{B}_k\hat{A}_k^2 + 0.5A^2 - 0.5\hat{B}_kA^2$$

Das Gesamtsystem fällt unsicher aus, wenn einer der Kernrechner (beliebig) ausgefallen ist und ein weiterer Rechner unsicher ausfällt (in diesem Fall kann der unsichere Rechner vom verbleibenden funktionsfähigen Rechner nicht abgeschaltet werden), oder wenn eine der failsafe Busanschaltungen unsicher ausfällt. Wie bei der Berechnung der Unsicherheit des OEN-Systems (siehe Abschnitt 6.3) muß der Ausfall beider Sensoreinheiten nicht berücksichtigt werden, da dieser Fall vom System immer erkannt wird. Die Wahrscheinlichkeit ergibt sich dann

angenähert zu:

$$P_{\text{gem,}\textit{unsicher}} \approx RR_k + 2\hat{A}_{e,k} \tag{6.4}$$

Aus der Formel für $\hat{A}_{e,k}$ ist ersichtlich, daß die Sicherheit des Gesamtsystems entscheidend von $\hat{B}_k$ beeinflußt wird, da diese Größe als einzige in der ersten Potenz vorkommt, ohne mit einer anderen Größe multipliziert zu werden.

6.6 Bewertung

Zur Bewertung der Formeln soll ein einfaches Zahlenbeispiel mit Vergleich eines nicht fehlertoleranten Systems verwendet werden. Die Ausfallwahrscheinlichkeit für ein solches System berechnet sich zu:

$$P_{\text{norm}} = 1 - (1-R)(1-M)(1-B_S)(1-S)(1-B_A)(1-A) \tag{6.5}$$

Der Ausfall des Systems ist immer kritisch, wenn das Ventil geöffnet ist. Die Wahrscheinlichkeit eines aktiven Ausfalls des Systems bei geschlossenem Ventil kann demgegenüber vernachlässigt werden, so daß gilt:

$$P_{\text{norm,}\textit{unsicher}} \approx 0.5 P_{\text{norm}} \tag{6.6}$$

Die Wahrscheinlichkeiten für die Ausfallereignisse werden nun willkürlich bezüglich R festgelegt, um den Verlauf der Wahrscheinlichkeiten für verschiedene Größenordnungen von R miteinander vergleichen zu können. Es gelten folgende Annahmen:

- Eine Busanschaltung ist ähnlich kompliziert wie ein Kernrechner und hat daher eine ähnliche Ausfallwahrscheinlichkeit: $B = R$
- Da eine fail-safe Busanschaltung komplizierter ist als eine normale, wird ihre Ausfallwahrscheinlichkeit insgesamt höher sein. Dafür ist die Wahrscheinlichkeit für einen kritischen Ausfall deutlich niedriger als bei einer normalen Busanschaltung: $\hat{B} = 1-(1-B)^2$, $\hat{B}_k = \hat{B}^2$
- Ein aktiver Ausfall von Kernrechner, Busanschaltung oder Ventil liegt deutlich unter der Wahrscheinlichkeit für einen beliebigen Ausfall: $R_k = 0.1R$, $B_k = 0.1B$, $A_k = 0.1A$
- Die Wahrscheinlichkeit für den Ausfall eines Busmediums und die Wahrscheinlichkeit für einen „babbling idiot“-Ausfall einer Busanschaltung sind im Vergleich zu den übrigen Wahrscheinlichkeiten niedrig: $M = B_i = \hat{B}_i = 0.01R$. Nicht berücksichtigt wird, daß ein „babbling idiot“-Ausfall einer fail-safe Busanschaltung durch konstruktive Maßnahmen niedriger sein wird als bei einer normalen Busanschaltung; da B_i klein gegenüber den anderen Wahrscheinlichkeiten ist, fällt dies nicht besonders ins Gewicht.

- Die Sensoren und Aktoren sind weniger komplex als ein Kernrechner und daher nicht so anfällig. Die Leistungselektronik eines fail-safe Busteilnehmers ist speziell zur Vermeidung unsicherer Ausfälle ausgelegt – nimmt man an, daß die Ausfallwahrscheinlichkeit eines Ventils von der Ausfallwahrscheinlichkeit seiner Leistungselektronik dominiert wird, dann gilt: $S = A = 0.1R$, $\hat{A}_k = A_k^2$.

In Bild 6.2 sind die Verläufe der Wahrscheinlichkeiten für beliebige und für unsichere Ausfälle der betrachteten Systemarchitekturen für verschiedene $P(R)$ in doppelt-logarithmischer Darstellung zu sehen. Da die Wahrscheinlichkeiten für die einzelnen Ereignisse willkürlich gewählt wurden, ist nur das Verhältnis der Ergebnisse und nicht ihr absoluter Wert von Relevanz.

Es ist deutlich zu sehen, daß beide fehlertolerante Systemarchitekturen deutlich zuverlässiger und sicherer sind als ein vergleichbares nicht fehlertolerantes System. Sowohl hinsichtlich der allgemeinen Ausfallwahrscheinlichkeit als auch der Wahrscheinlichkeit für einen unsicheren Ausfall schneidet das OG-System dabei besser ab als das OEN-System.

Für die Ausfallwahrscheinlichkeit läßt sich dieses Ergebnis dadurch erklären, daß durch die exklusive Kanalnutzung im OEN-System mehr Kombinationen von zwei ausgefallenen Komponenten (z. B. ein Kernrechner und eine nicht mit dem ausgefallenen Kernrechner verbundene Busanschaltung) zu einem Ausfall des gesamten Systems führen können. Dies hat einen höheren Einfluss als die durch die größere Komplexität bedingte höhere Ausfallwahrscheinlichkeit einer fail-safe gegenüber einer normalen Busanschaltung. Das OG-System hat bei der Sicherheit ebenfalls Vorteile gegenüber dem OEN-System, da zum Erreichen der sicheren Endlage keine Kommunikation über das Peripheriebussystem notwendig ist (die fail-safe Busanschaltungen können ja einen Kommunikationsausfall erkennen und dann selbständig abschalten), während ein möglicher Kommunikationsausfall einen großen Anteil an der Gesamtwahrscheinlichkeit für einen unsicheren Ausfall des OEN-Systems hat. Durch Verwendung eines OES- an Stelle eines OEN-Systems ließe sich dieses Problem vermeiden; allerdings gelten dann die in Abschnitt 4.4.4 beschriebenen Einschränkungen.

Im folgenden wird untersucht, wie empfindlich die einzelnen Architekturen auf die Änderung einzelner Ausfallwahrscheinlichkeiten reagieren. Es werden nur solche Größen betrachtet, die einen Einfluß auf die Sicherheit aller drei Architekturen haben, wobei die oben gemachten Annahmen zu den Abhängigkeiten der einzelnen Ausfallwahrscheinlichkeiten untereinander – mit Ausnahme der betrachteten Größe – weiterhin zu Grunde gelegt werden; zusätzlich wird die Ausfallwahrscheinlichkeit R fest auf den (willkürlich gewählten) Wert 10^{-4} gesetzt.

In Bild 6.3 wird die Ausfallwahrscheinlichkeit A für einen Aktor (einschließlich Leistungselektronik) unabhängig von R variiert. Der Kurvenverlauf für die Gesamtausfallwahrscheinlichkeit zeigt nichts Unerwartetes; er wird für $A \gg R$ von A dominiert, während A bei $A \ll R$ keinen spürbaren Einfluß mehr hat. Interessanter wird das Bild beim Kurvenverlauf für die Wahrscheinlichkeit eines unsicheren Ausfalls. Die „Sättigung“ der Kurve wird für das OG-System deutlich vor der des OEN-Systems erreicht; dies resultiert aus der Annahme, daß die Leistungselektronik einer fail-safe Busanschaltung deutlich sicherer sein wird als die einer vergleichbaren normalen Busanschaltung.

In Bild 6.4 wird anlog zu obiger Betrachtung die Auswirkung der Ausfallwahrscheinlichkeit ei-

ner Busanschaltung B untersucht. Da angenommen wird, daß eine fail-safe Busanschaltung eine etwas höhere Gesamtausfallwahrscheinlichkeit haben wird als eine vergleichbare normale Busanschaltung, verläuft die Kurve bei $B \gg R$ für das OG-System steiler als für das OEN-System; in beiden Fällen ist jedoch (wie erwartet) B für $B \ll R$ nicht mehr weiter relevant. Auch hier ist der Kurvenverlauf für die Wahrscheinlichkeit eines unsicheren Ausfalls deutlich interessanter. Aus dem Verlauf für das OG-System kann man die lineare Abhängigkeit der Sicherheit eines OG-Systems von der Sicherheit einer sicheren Busanschaltung ablesen (siehe auch Formel 6.4). Ebenfalls zu erkennen ist, daß eine Verringerung von B auf eine Größenordnung kleiner als R zu einer Verbesserung der Sicherheit des Gesamtsystems führt; eine weitere Verringerung von B hat aber keinen großen Einfluß mehr.

Die übrigen Größen eignen sich nicht für eine tiefergehende Betrachtung, da sie keinen Einfluß auf die Sicherheit aller drei betrachteter Systemstrukturen haben (z. B. hat die Kommunikationsfähigkeit auf die Sicherheit eines OG-Systems keinen Einfluß).

Eine weitere Verbesserung der Sicherheit beider betrachteten Systeme ließe sich in dem gewählten Beispiel durch Ventile erreichen, die sich selbstständig über eine Mechanik schließen, wenn sie nicht aktiv unter Energiezufuhr offengehalten werden.

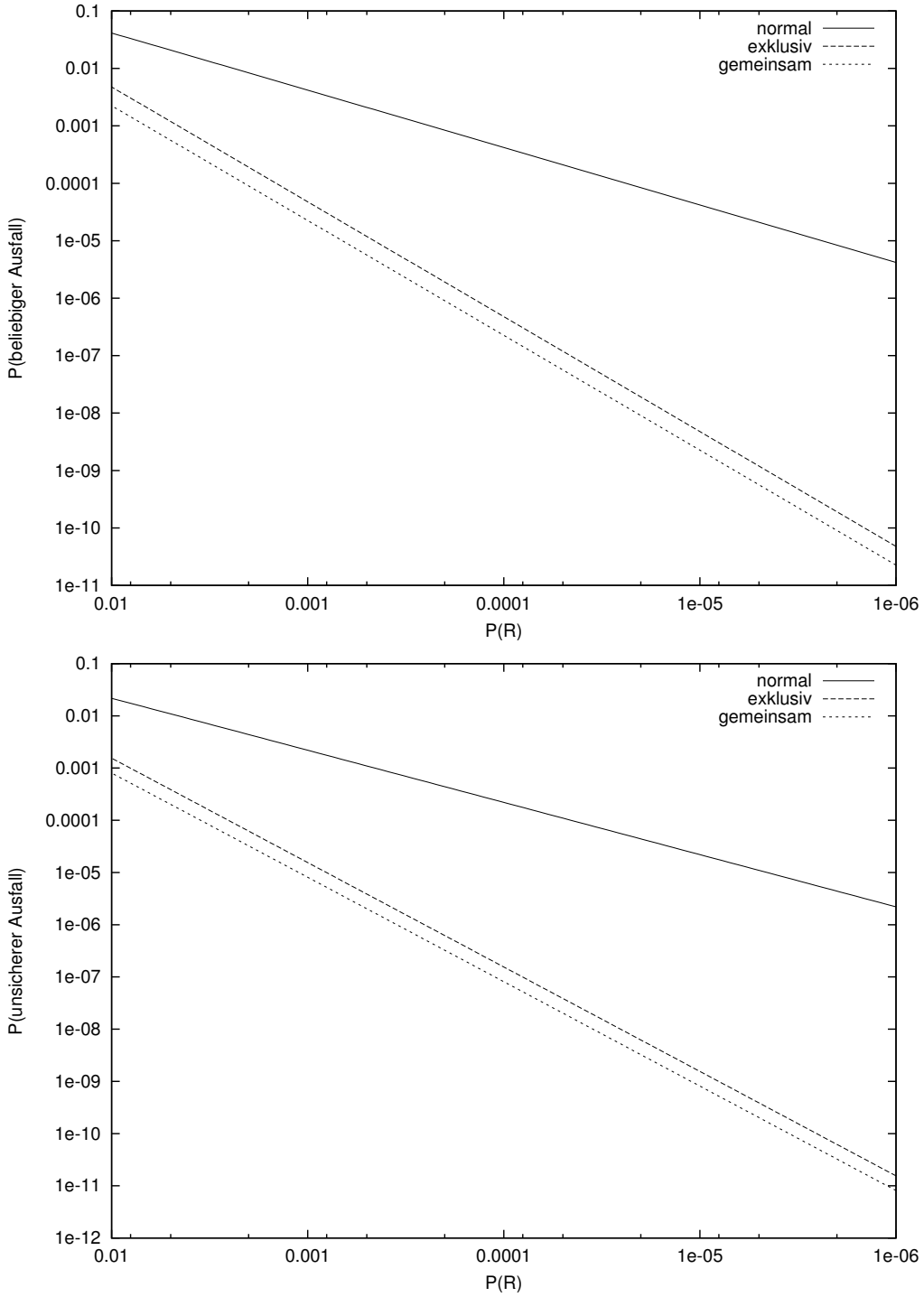

Bild 6.2: Wahrscheinlichkeiten für einen beliebigen Ausfall (oben) und einen sicherheitskritischen Ausfall (unten) für verschiedene Systemarchitekturen.

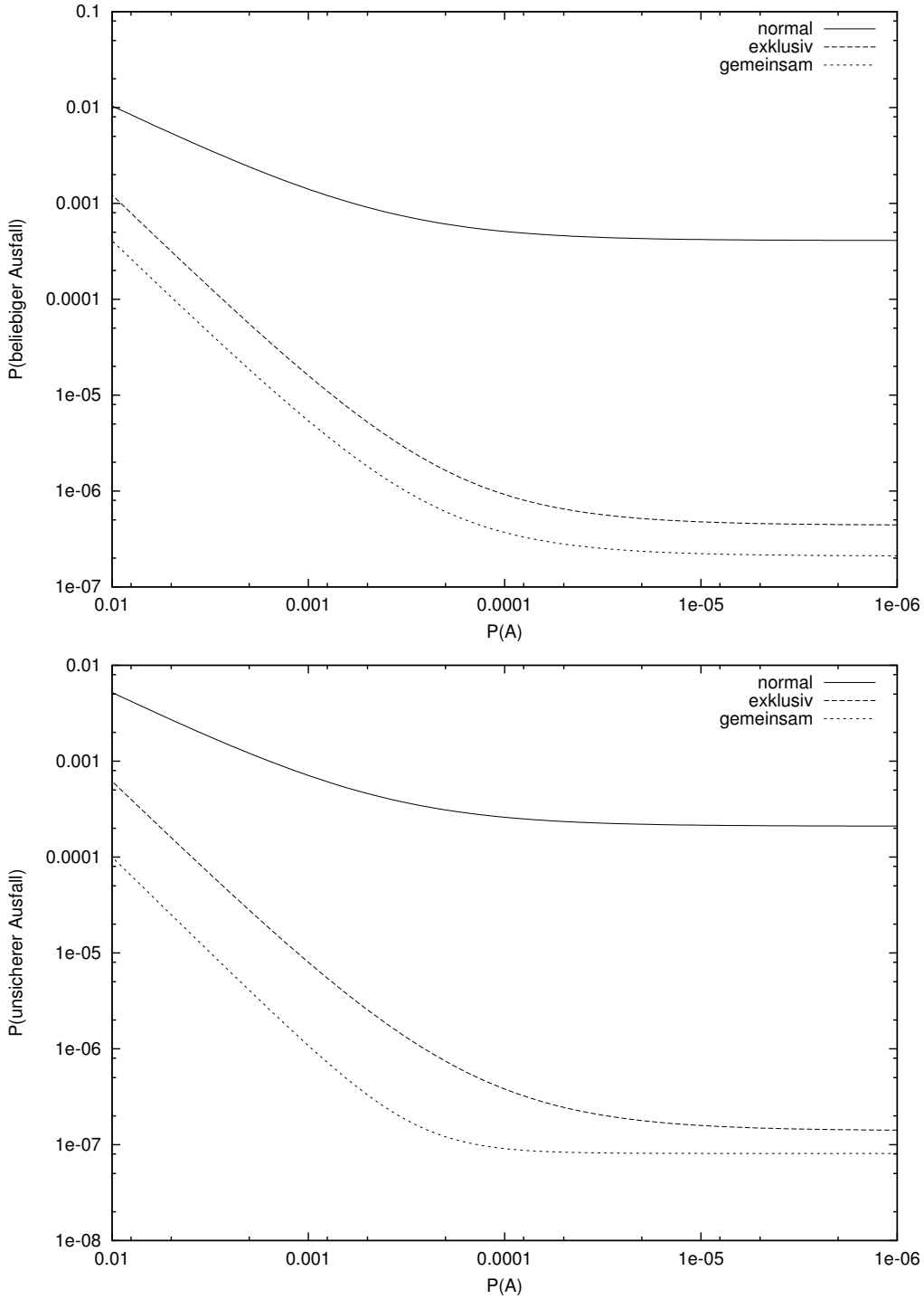

Bild 6.3: Einfluß der Ausfallwahrscheinlichkeit A eines Aktors (einschließlich Leistungselektronik) auf die Gesamtwahrscheinlichkeit für einen beliebigen Ausfall (oben) und einen sicherheitskritischen Ausfall (unten) bei fester Ausfallwahrscheinlichkeit $R = 10\text{e}^{-4}$.

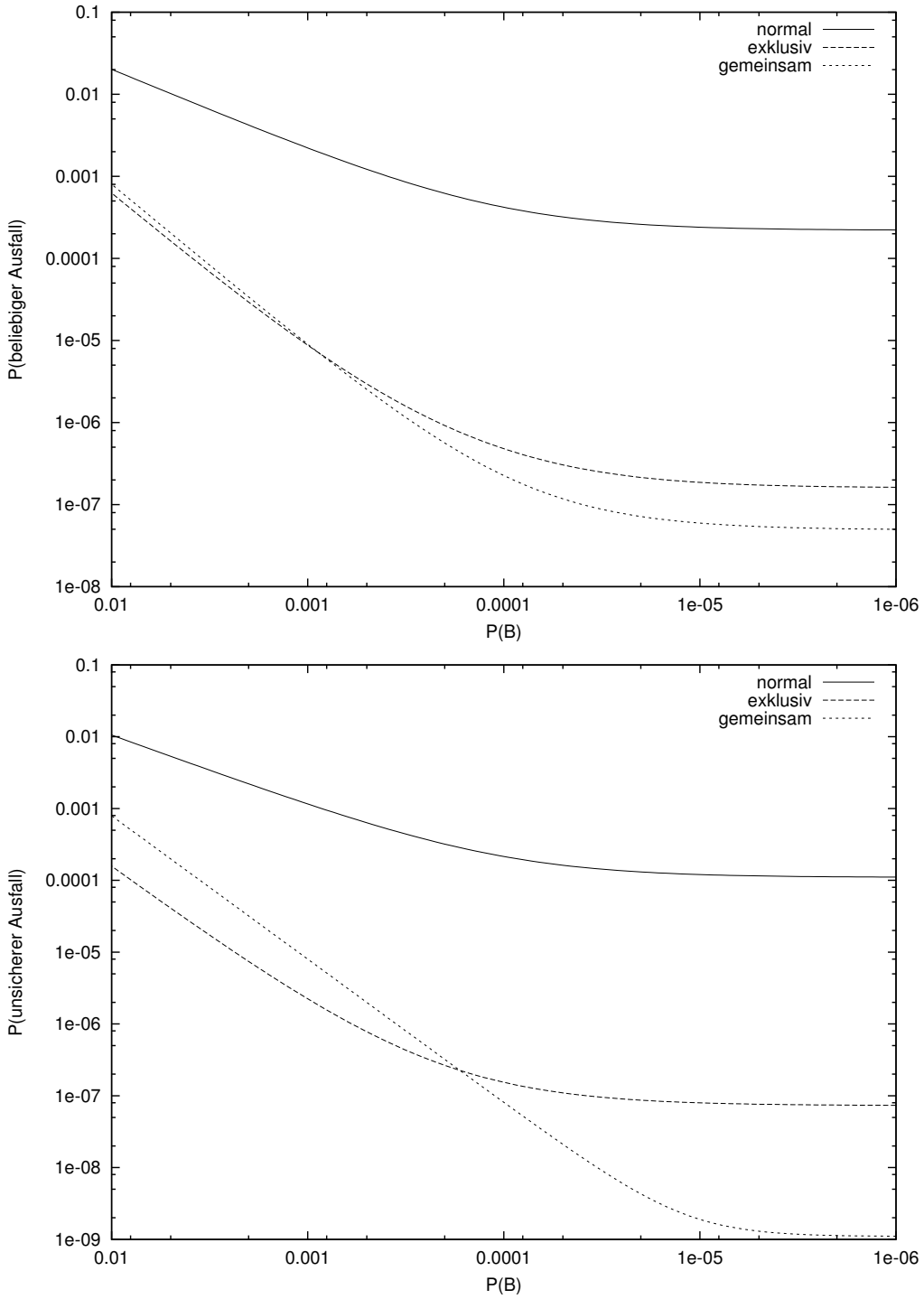

Bild 6.4: Einfluß der Ausfallwahrscheinlichkeit B einer Busanschaltung auf die Gesamtwahrscheinlichkeit für einen beliebigen Ausfall (oben) und einen sicherheitskritischen Ausfall (unten) bei fester Ausfallwahrscheinlichkeit $R = 10\text{e}^{-4}$.

7 Implementierung und Einsatzbeispiel

Dieses Kapitel beschreibt das ab 1997 am Lehrstuhl für Realzeit-Computersysteme prototypisch realisierte ADES-System, anhand dessen die prinzipielle Machbarkeit des Konzeptes demonstriert werden soll.

7.1 Die Beispielanwendung

Als Anwendungsbeispiel dient der in Bild 7.1 schematisch dargestellte Versuchsaufbau einer schwebenden Kugel. Ein metallischer Körper (keine richtige Kugel, sondern ein Zylinder mit Kugelkopf) wird mit Hilfe eines Magnetfeldes zum Schweben gebracht. Die aktuelle Position des Schwebekörpers wird mittels einer Lichtschranke gemessen – der auf eine Photozelle treffende Lichtstrahl wird dabei je nach Position des Körpers unterschiedlich stark verdeckt. Das Magnetfeld wird über eine Spule erzeugt, die von einem Stromverstärker gespeist wird.

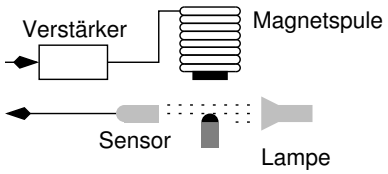

Bild 7.1: Prinzipieller Aufbau der schwebenden Kugel

Die Regelung der Lage des Körpers geschieht über einen digitalen PID-Regelalgorithmus, der auf dem ADES-Rechnersystem fail-operational mit einer Frequenz von 100Hz ausgeführt wird.

7.2 Der Rechnerkern

Der Rechnerkern des Prototypen besteht aus drei handelsüblichen Intel-kompatiblen Standard-PCs, die jeweils mit einem 300MHz AMD K6 Prozessor ausgestattet sind. Die Rechner sind untereinander über Ethernet Punkt-zu-Punkt Verbindungen voll vernetzt; dazu verwendet jeder Rechner zwei handelsübliche 100MBit/s Ethernetkarten im Vollduplex-Betrieb. Die Punkt-zu-Punkt Verbindungen vermeiden die sonst bei Ethernet typischen Paketkollisionen und ermög-

lichen somit ein deterministisches Zeitverhalten beim Datenaustausch. Eine weitere Ethernetkarte pro Rechner bindet den Rechnerkern an das (externe) Diagnose- und Entwicklungssystem an.

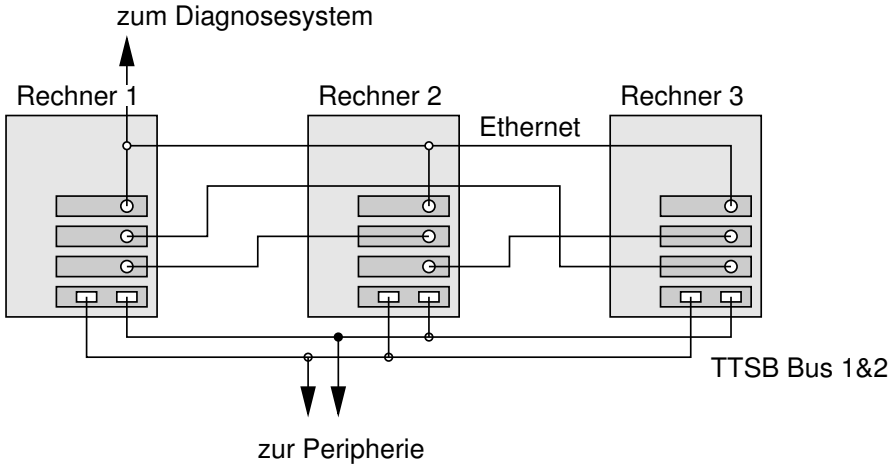

Bild 7.2: Rechnerkern des ADES Prototypen

Auf dem Rechnerkern läuft das Realzeitbetriebssystem QNX 4.25 unmodifiziert; die gesamte für den sicheren und fail-operational Betrieb notwendige Zusatzfunktionalität wird von der ADES Fehlertoleranzschicht zur Verfügung gestellt (siehe auch Kapitel 4.3). QNX ist ein sog. Mikrokernelsystem, d. h. die Funktionalität des Betriebssystemkerns (des sog. Kernels) ist gering; bei QNX beschränkt sie sich auf die Prozessorzuteilung, Speicherverwaltung und die Interprozeßkommunikation mittels Nachrichten. Die übrige Funktionalität (Dateisystem, Netzwerkkommunikation, etc.) wird von eigenständigen Treiberprozessen bereitgestellt, deren Dienste man über die Interprozeßkommunikation anfordern kann.

Diese Architektur paßt sehr gut zu den Anforderungen von ADES. Der Austausch von Daten zwischen einzelnen Prozessen über QNX-Nachrichten kann gleichzeitig als Synchronisationspunkt verwendet werden. Der Systemaufruf `Send` (sende Nachricht) wird dabei von der Fehlertoleranzschicht abgefangen und der Nachrichteninhalt mit den anderen Kernrechnern ausgetauscht und votiert. Erst die votierten Daten werden dann mit dem normalen QNX-Mechanismus an den Empfängerprozeß weitergegeben. Für die beteiligten Prozesse geschieht dies völlig transparent. QNX blockiert den sendenden Prozeß solange, bis der Empfänger mit dem Systemaufruf `Reply` auf die empfangene Nachricht geantwortet hat, wobei die Antwort zusätzlich Daten enthalten kann, die dann ebenfalls von der Fehlertoleranzschicht votiert werden müssen. Auf diese Weise kann gleich die Ausführungsreihenfolge der Prozesse auf den einzelnen Rechnern des Kerns ohne weitere Mechanismen synchronisiert werden. QNX kennt auch die Kommunikation über sog. Proxies – dies sind Nachrichten, die keinem Senderprozeß zugeordnet sind und daher auch nicht beantwortet werden können. Sie werden verwendet, um den Empfänger von bestimmten Ereignissen wie z. B. dem Auftreten eines Interrupts oder dem Ablaufen eines Timers zu benachrichtigen, nicht jedoch zur Interprozeßkommunikation. Die Fehlertoleranzschicht verwendet daher Proxies nicht als Synchronisationspunkte.

7.3 Das ADES Peripheriesystem

Das Peripheriesystem des Prototypen ist vollständig busbasiert. Auf Grund der Vorteile bei der (modularen) Entwicklung des Systems und der höheren zu erwartenden Sicherheit und Zuverlässigkeit wird ein System mit gemeinsamer Kanalnutzung verwendet (siehe Kapitel 4.4.5).

Das Peripheriesystem des Prototypen ist in Bild 7.3 dargestellt. Im Rest dieses Abschnittes werden die einzelnen Komponenten des Peripheriesystems kurz vorgestellt.

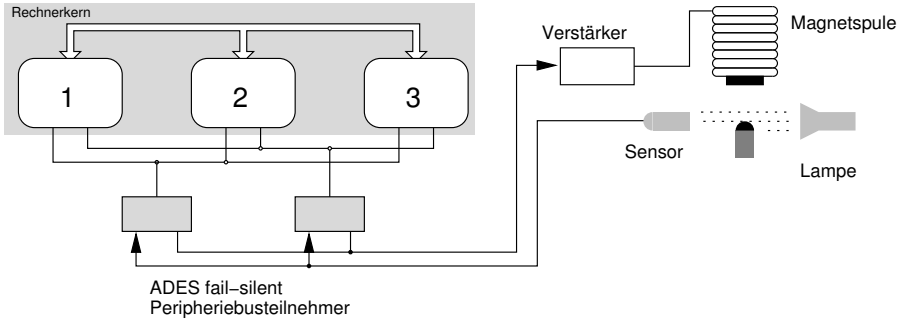

Bild 7.3: Peripherieanschluß des Prototypen

7.3.1 Auswahl eines geeigneten Feldbusses

In Kapitel 3 wurden eine Reihe verschiedener Feldbussysteme vorgestellt, die eine sichere Datenübertragung erlauben. Für keines dieser Bussysteme sind Masteranschaltungen verfügbar, die eine gemeinsame Kanalnutzung im Sinne von ADES erlauben. Prinzipiell wäre es möglich, spezielle Masteranschaltungen selber zu entwickeln, um wenigstens die verfügbaren fail-safe Busteilnehmer verwenden zu können. ProfiSafe und Interbus Safety scheiden hier für den Prototypen aus, da die verfügbaren Buscontrollerbausteine bereits einen Teil der höheren Protokollschichten implementieren und somit eine Anpassung an die besonderen Bedürfnisse von ADES erschweren. Es bleiben die CAN basierten Busse, für die sich mit relativ geringem Aufwand eigene Entwicklungen durchführen ließen. Anfang 2000 waren jedoch von keinem dieser Busse (SafetyBus-p, CANopen Safety) die Spezifikationen öffentlich zugänglich, so daß auch diese Busse nicht für den Prototypen verwendet werden konnten.

Statt nun einen eigenen, proprietären Bus auf Basis von CAN zu entwickeln, wurde beschlossen, eine Variante des TTSB-Busses zu verwenden. Der TTSB-Bus („Time Triggered Sensor Bus“) ist ursprünglich eine Entwicklung der Arbeitsgruppe um Professor Kopetz der Technischen Universität Wien. Er wurde im Rahmen eines von den Ländern Österreich, Baden-Würtemberg und Bayern geförderten Forschungsprojekts weiterentwickelt. Inzwischen ist TTSB unter dem Namen TTP/A bei der OMG („Object Management Group“) zur Standardisierung angenommen. TTSB verwendet ein sehr einfaches Protokoll und läßt sich daher mit wenig Aufwand implementieren. Zum Zeitpunkt der Arbeiten am Prototypen war die Spezifikation jedoch noch

nicht endgültig fertig gestellt; daher entspricht die Implementierung im ADES-Prototypen nur ansatzweise der aktuellen Spezifikation[1)].

Es soll hier nochmals betont werden, daß das ADES Peripheriekonzept an sich busneutral ist und auch mit anderen Bussen realisiert werden kann. Die Entscheidung zugunsten von TTSB für den Prototypen wurde nicht primär aus technischen, sondern aus praktischen Gründen gefällt, um möglichst schnell ein lauffähiges System präsentieren zu können.

7.3.2 Der TTSB Bus

Ein TTSB-System besteht aus einem Master und mindestens einem Slave, die in einer klassischen Bustopologie ein gemeinsames Medium benutzen. Die zu verwendende Busphysik ist dabei nicht festgelegt; es ist z. B. Signalisierung nach dem in der Automatisierungstechnik weit verbreitetem RS485 Standard oder dem von CAN verwendeten ISO 11898 Standard möglich. Für den Prototypen wurde eine Übertragung nach dem RS485 Standard gewählt, da die entsprechenden Treiberbausteine problemlos verfügbar und einfach anzuwenden sind.

Die prinzipielle Funktion der Datenübertragung bei TTSB ist in Bild 7.4 dargestellt. Die Kommunikation über den Bus ist in einzelne sog. TDMA Runden aufgeteilt, die vom Busmaster initiiert werden. Eine Runde wird in eine Anzahl von Zeitscheiben identischer Länge unterteilt, den sog. Slots. Ein Slot wiederum besteht aus einem UART Frame, gefolgt von einer kurzen Pause, dem sog. Inter Frame Gap (IFG), der jeweils zwei UART-Frames voneinander trennt. Die genaue Dauer des IFG beträgt je nach physikalischer Ausdehnung und verwendeter Baudrate 2 bis 5 Bitzeiten. Ein UART-Frame („universal asynchronous receiver/transmitter“) ist ein einfaches serielles Datenformat, bestehend aus einem Startbit, acht Datenbits, einem Paritätsbit und einem Stopbit, wie es bei asynchronen seriellen Schnittstellen üblich ist. Solche Schnittstellen finden sich in fast allen Mikrocontrollern oder lassen sich mit geringem Aufwand in Software nachbilden. Sie erlauben daher besonders kostengünstige Implementierungen.

TTSB unterscheidet zwischen Master/Slave-Runden, bei denen der Master einen spezifischen Slave adressiert (dies entspricht klassischem Master/Slave Betrieb), und den Multipartnerrunden, bei denen die einzelnen Slots über die sog. „round descriptor list“ (RODL) fest einzelnen Teilnehmern zugeordnet werden (dies entspricht klassischem TDMA Betrieb). Da keine expliziten Teilnehmeraddressen oder Nachrichtenlängen gesendet werden müssen (beides ist implizit in der RODL enthalten), erlauben Multipartnerrunden eine sehr effiziente Kommunikation, insbesondere dann, wenn die Nutzdatenmenge pro Busteilnehmer gering ist.

Um eine größere Flexibilität zu erreichen, erlaubt TTSB sieben verschiedene Multipartnerrunden mit jeweils eigener RODL. Der erste Slot einer Runde ist grundsätzlich für den sog. Fireworks- Frame reserviert; mit ihm signalisiert der Busmaster den genauen Anfangszeitpunkt und die Art einer Runde. Der Fireworks-Frame kann von normalen Datenframes am unterschiedlichen Paritätsbit unterschieden werden (Datenframes verwenden gerade Parität, Fireworks-Frames ungerade Parität). Reicht das Paritätsbit der einzelnen UART-Frames zur Er-

[1)] Implementiert wurde nach Spezifikation V1.13, aktuell ist Version 2.0 ([23])

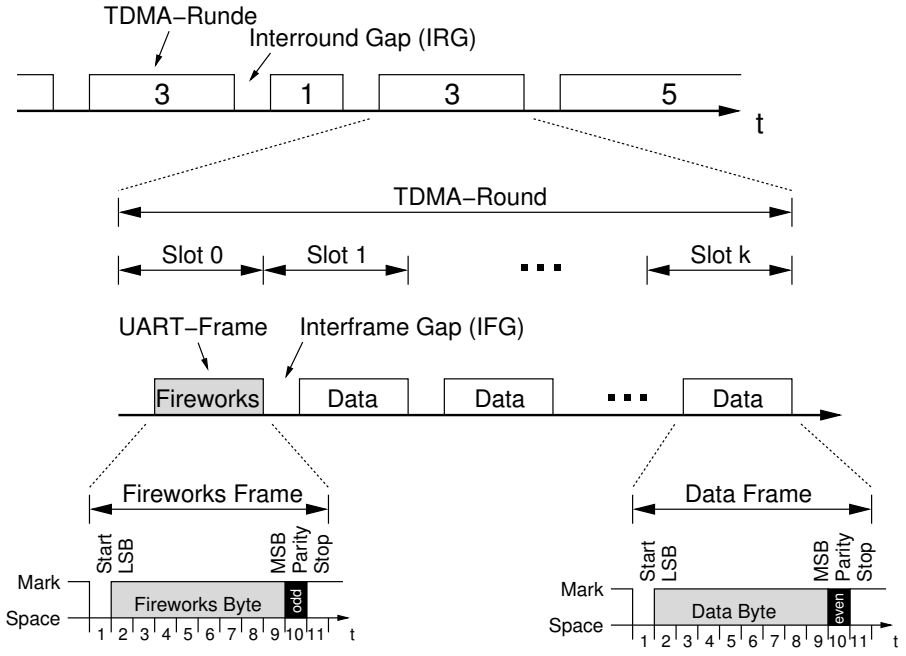

Bild 7.4: Funktionsprinzip von TTSB

kennung von Übertragungsfehlern nicht aus, kann für ausgewählte Slots mittels einer Exklusiv-Oder Verknüpfung eine zusätzliche Prüfsumme berechnet werden.

TTSB spezifiziert nicht nur die eigentliche Übertragung von Datenbytes, sondern definiert auch eine einheitliche Kodierung der zu übertragenden Informationen. Analogwerte werden z. B. nicht als Absolutwert übertragen, sondern als Bruchteil des Maximalwertes angegeben – vergleichbar mit der Darstellung von Meßwerten in der 4..20mA Übertragungstechnik, die in der Prozeßtechnik immer noch üblich ist. Mit 16 Bit Datenlänge können so Datenwerte in 0.025% Schritten (entspricht in etwa 12 Bit Genauigkeit) angegeben werden, einschließlich eines in vier Bit kodiertem Konfidenzmaß (damit kann der Sensor angegeben, für wie sicher er das Ergebnis hält) und der Möglichkeit, Fehlercodes mit im Datenwort (inband) zu transportieren.

Weiterhin definiert TTSB das sog. Interface File System (IFS). Der Zugriff auf das IFS erfolgt ausschließlich über Master/Slave Runden. Jedes Gerät kann bis zu 64 verschiedene „Dateien“ enthalten, die wiederum aus maximal 256 Datensätzen zu je vier Bytes bestehen. Datensätze können gelesen, geschrieben oder ausgeführt werden. Sie sind dazu gedacht, Informationen über das Gerät (z. B. Hersteller, Gerätetyp, Seriennummer) zu speichern, eine dynamische Konfiguration der RODL zu erlauben, oder den Zugriff auf Sonderfunktionalität zu ermöglichen. Konzeptionell ähnelt das IFS dem Objektverzeichnis von CANopen. Ein TTSB-Slave muß keine IFS Unterstützung bieten, so daß auch sehr einfach aufgebaute Slaves möglich sind.

7.3.3 Erweiterte Fehlererkennungsmaßnahmen

Die sichere Kommunikation über den Peripheriebus erfordert zusätzliche Maßnahmen zur Fehlererkennung, die TTSB nicht leisten kann. Von den in Kapitel 2.7.3 vorgestellten Verfahren werden daher die folgenden genutzt:

1. Jeder funktionsfähige Teilnehmer muß innerhalb einer bestimmten Zeit eine gültige Nachricht senden. Für den Prototypen bedeutet dies, daß ein Teilnehmer jeden für ihn reservierten Sendeslot auch nutzen muß. Kann ein Teilnehmer in zwei aufeinanderfolgenden Zyklen keine gültige Nachricht senden, gilt er als ausgefallen.

2. Jede Nachricht wird mit einer End-To-End Prüfsumme (siehe Kapitel 2.7.4) abgesichert. Dazu wird über den momentanen Wert des Zykluszählers, die Kennung des Peripheriebusteilnehmers, und die eigentlichen Nutzdaten eine 16 Bit CRC (cyclic redundancy check, siehe [40]) berechnet und über den Bus gesendet. Die Wahrscheinlichkeit, mit Hilfe einer CRC einen Übertragungsfehler erkennen zu können, wird neben der „Breite“ der CRC entscheidend von dem verwendeten Generatorpolynom beeinflußt – insbesondere dann, wenn der CRC-Code nicht in seiner „natürlichen“ Länge, sondern „verkürzt“ verwendet wird. Dies ist bei Datenübertragung fast immer der Fall, da die „natürliche“ Länge eines CRC-Codes mit der Breite r (die Breite entspricht der Anzahl der Prüfbits, die den eigentlichen Daten angehängt werden) 2^{r-1} Bits beträgt – dies ist für die meisten Anwendungen viel zu lang. Daher wird ein in [41] vorgestelltes Generatorpolynom ($x^{16} + x^{15} + x^{12} + x8 + x6 + x3 + x2 + 1$, „CRC-16Q“) verwendet, das bei sehr kurzen Nachrichtenlängen deutlich bessere Fehlererkennung ermöglicht als die bekannten, standardisierten Generatorpolynome wie z. B. CRC-CCITT oder CRC-16 mit derselben Breite.

3. Der Master verwaltet einen acht Bit Zykluszähler, der am Anfang jedes Zyklus mit gesendet wird. Der Zykluszähler wird bei der Berechnung der End-To-End Prüfsumme berücksichtigt; auf diese Art können die Empfänger erkennen, wenn ein Sender alte Nachrichten wiederholt. Der Zykluszähler dient auch dazu, den aktuellen Busmaster für einen Zyklus zu bestimmen. Dazu werden die funktionsfähigen Kernrechner von null beginnend durchnummeriert; der Zykluszähler modulo der Anzahl der funktionsfähigen Rechner ergibt dann die Nummer des aktuellen Busmasters.

4. Für jede Peripheriebusanschaltung gibt es eine eindeutige Kennung, die als Startwert für die Berechnung der End-To-End Prüfsumme jeder mit dem Master ausgetauschten Nachricht verwendet wird. Auf diese Weise kann der jeweilige Empfänger prüfen, ob die Nachricht tatsächlich vom erwarteten Sender stammt.

7.3.4 Die TTSB-Busmasterkarte

Der ADES-Rechnerkern wird über eine spezielle TTSB-Busmasterkarte an das Peripheriebussystem angeschlossen. Es handelt sich dabei um eine modifizierte ProfiBus-Masterkarte der Firma Softing. Die Karte besitzt zwei unabhängige Mikrocontrollerkanäle, die mit je einem

Infineon 80C165 Mikrocontroller bestückt sind. Ein dual-ported RAM ermöglicht die Kommunikation mit dem Hostrechner über den PCI Bus. Die ProfiBus-spezifischen ASICs wurden entfernt und statt dessen die auf jedem 80C165 vorhandene asynchrone serielle Schnittstelle mit den RS485 Treiberbausteinen der Karte verbunden. Die Firmware der Karte wurde vom Autor vollständig neu entwickelt, so daß jeweils ein 80C165 selbständig eine Untermenge des TTSB-Protokolls als Busmaster oder Beobachter ausführen kann.

Der Hostrechner schreibt dazu die zu sendenden Daten in das dual-ported RAM und teilt dem 80C165 mit, welche TDMA-Runde über den Bus abgewickelt werden soll, und ob Master- oder Beobachterbetrieb gewünscht ist. Sowohl im Master-, als auch im Beobachterbetrieb werden alle vom momentanen Busmaster tatsächlich gesendeten Daten empfangen und mit den im internen Speicher abgelegten Sendedaten verglichen. Am Ende des Zyklus kopiert die Karte die empfangenen Daten einschließlich eines Statuscodes für jedes Byte in das dual-ported RAM und signalisiert mit einer Interruptanforderung an den Hostrechner, daß die Runde abgeschlossen wurde. Die weitere Bearbeitung einschließlich der im vorherigen Abschnitt beschriebenen erweiterten Fehlererkennung findet im Hostrechner statt. Da für die sichere Datenübertragung über den Bus sowieso ein End-To-End Prüfsummenverfahren verwendet wird (siehe auch Kapitel 7.3.3), sind die von TTSB optional vorgesehenen Prüfsummen überflüssig und daher nicht implementiert.

Wie bereits erwähnt konnte das TTSB Protokoll nicht vollständig nach der Spezifikation implementiert werden, da die vorliegende, noch nicht endgültige Version der Spezifikation einige Fehler bzw. unlösbare Probleme enthielt, die im folgenden kurz erläutert werden.

Für den Demonstrationsaufbau wird eine wesentlich höhere Datenrate (100kBit/s) als die von der vorliegenden Spezifikation vorgesehenen 10kBit/s benötigt, was die Implementierung eines Software-UARTs unmöglich macht und eine Hardware-Lösung verlangt. Typische Hardware-UARTs, wie sie in einer Reihe von Mikrocontrollern zu finden sind (z. B. auch die verwendeten Controller vom Typ Infineon 80C165 und Motorola 68332), tasten den empfangenen Bitstrom mit 16-fachem Oversampling ab. Der für die Abtastung benötigte Takt wird aus dem Systemtakt generiert und für die Bitübertragung nochmals mittels eines Zählers durch 16 geteilt. Das Problem liegt in der Tatsache, daß der UART nur bei einem Überlauf des Zählers prüft, ob ein Byte gesendet werden soll. Der genaue Sendezeitpunkt kann daher mit diesen UARTs nur mit einer Abweichung von minimal einer Bitzeit festgelegt werden. Die vorliegende Spezifikation forderte jedoch, den Startzeitpunkt eines Slots auf eine halbe Bitzeit genau einzuhalten. Dies ist mit den UARTs der oben genannten Mikrocontroller technisch nicht möglich. Daher wurde für den ADES-Prototypen die Pause zwischen zwei Frames (der IFG) auf fünf Bitzeiten erhöht, um größere Abweichungen der Startzeitpunkte eines Slots zu ermöglichen (ein kürzerer IFG wäre technisch machbar; mit einem IFG von fünf Bitzeiten ist ein Slot aber genau 16 Bitzeiten lang. Das wiederum vereinfacht die interne Berechnung von Sendezeitpunkten).

Ein weiteres Problem ergibt sich durch die Bitrate bei Master/Slave-Runden. Die Spezifikation sieht keine Übertragungspause nach einer Datenanforderung des Masters vor, d. h. der Slave muß sofort im nächsten Slot antworten. Der Slave kann aber die Bearbeitung der Masteranfrage erst beginnen, wenn er alle Daten empfangen hat – ihm bleibt also letztlich nur eine einzige Bitzeit zur Verarbeitung, in der neben der Masteranfrage auch das normale Protokoll verarbeitet

werden muß. Dies entspricht bei einer Datenrate von 100kBit/s nur $10\ \mu$s – zu wenig, um mit einem kostengünstigen Mikrocontroller eine höhere Funktion ausführen zu können. Der Prototyp unterstützt daher Master/Slave-Runden nicht. Ohne Master/Slave-Runden kann das IFS nicht genutzt werden und wurde daher ebenfalls vollständig weggelassen.

Die oben genannten Probleme wurden wohl auch von den Entwicklern von TTSB erkannt. In einer späteren Version der Spezifikation ([23]) wurde der IFG daher auf 2 bis 5 Bitzeiten (abhängig von der gewünschten Übertragungsrate, die jetzt bis zu 1MBaud betragen darf, und der maximalen physikalischen Ausdehnung) verlängert. Die Master/Slave-Runde wurde in zwei Runden aufgeteilt. In der ersten Runde stellt der Master eine Anfrage an den Slave, dann folgt eine normale Multipartnerrunde, und in der nächsten Runde antwortet der angesprochene Slave. Diese überarbeitete Spezifikation war jedoch zum Implementierungszeitpunkt des Prototypen noch nicht verfügbar und konnte daher nicht berücksichtigt werden.

7.3.5 Der fail-silent Peripheriebusteilnehmer

Der Demonstrationsaufbau „schwebende Kugel“ besitzt zwei physikalische E/A-Punkte – die Position der schwebenden Kugel, ausgedrückt als analoges Spannungssignal von $\pm 5V$, und der gewünschte Stromfluß durch die Spule, ebenfalls durch ein Spannungssignal von $\pm 5V$ dargestellt. Um den notwendigen Hardwareaufwand dabei möglichst niedrig zu halten, wurde ein kombinierter fail-silent Peripheriebusteilnehmer für beide Signale entwickelt. Zwei dieser Geräte werden parallel geschaltet, um fail-operational Betrieb bis in die Busanschaltungen zu ermöglichen. Da Sensor und Aktor nur einfach vorhanden sind, ist ein fail-operational Betrieb des Gesamtsystems nicht möglich.

In Bild 7.5 ist der prinzipielle Aufbau des realisierten fehlersicheren Peripheriebusteilnehmers dargestellt. Er besteht aus zwei getrennten Verarbeitungskanälen, dem Masterkanal und dem Beobachterkanal. Beide Kanäle können über eine dedizierte Kommunikationsschnittstelle Daten austauschen und miteinander vergleichen, um Fehler in einem der Kanäle erkennen zu können. Jeder Kanal hat seine eigenen Meßeingänge; nur der Masterkanal verfügt über Ausgangstreiber. Die Ausgänge werden zur Überwachung wieder auf die Eingänge zurückgeführt. Im Fehlerfall können sie über redundante Abschalter von der Außenwelt getrennt werden. Der Bustransceiver ist ebenfalls nur einfach vorhanden. Während beide Kanäle Daten vom Bus empfangen können, kann nur der Masterkanal Daten senden. Im Fehlerfall kann der Beobachterkanal den Transceiver sperren und somit verhindern, daß der Masterkanal falsche Daten an den Rechnerkern sendet.

Für den Demonstrationsaufbau besitzt der Peripheriebusteilnehmer für die Eingangselektronik einen achtkanaligen (per Multiplex-Verfahren) 12 Bit A/D Wandler und für die Ausgangsschaltung einen einkanaligen 12 Bit D/A Wandler. Der D/A-Ausgang kann über zwei in Serie geschaltete Relais vollständig von der Außenwelt abgetrennt werden. Das Ausgangssignal wird zur Überwachung zum A/D-Wandler zurückgeführt. Sowohl der Master- als auch der Beobachterkanal bestehen aus einem Motorola 68332 Mikrocontroller mit eigenem Speicher und Taktversorgung; es handelt sich (wie auch beim ADES Rechnerkern) um ein lose gekoppeltes System. Die Prozessoren werden mit jeweils 16MHz getaktet; bei dieser Geschwindigkeit

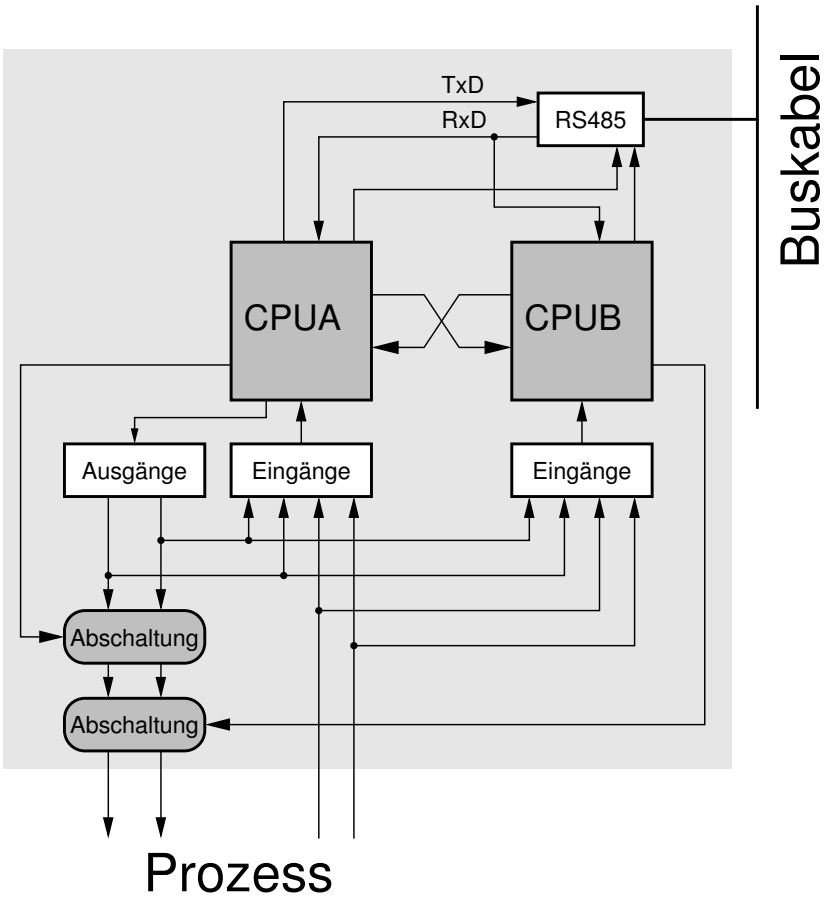

Bild 7.5: Internes Schema des sicheren Peripheriebusteilnehmers

leistet ein 68332 ca. 2-3 MIPS (Millionen Instruktionen pro Sekunde). Als Kommunikationsschnittstelle zwischen beiden Kanälen wird die mit 4 MBaud betriebene SPI-Schnittstelle („Serial Peripheral Interface“, eine synchrone serielle Schnittstelle) des Motorola Bausteins benutzt.

Die auf dem Teilnehmer implementierte Software ist ebenso wie die Hardware funktional sehr einfach gehalten. Nach dem Einschalten werden alle internen Datenstrukturen initialisiert und der D/A-Wandler kalibriert. Dazu gibt der Masterkanal alle möglichen Werte über den D/A-Wandler aus. Beide Kanäle messen die tatsächlich anliegenden Pegel und speichern die Werte in einer Tabelle; im weiteren Betrieb wird diese Tabelle dazu benutzt, den digitalen Wert zu bestimmen, der an den D/A-Wandler übergeben werden muß, um einen bestimmten Spannungspegel einzustellen. Während der Kalibrierung wird davon ausgegangen, daß der D/A-Wandler fehlerfrei arbeitet. Der Ausgang des Teilnehmers bleibt während des gesamten Vorgangs über die Abschaltrelais von der Außenwelt getrennt. Nach der Kalibrierung wartet der Teilnehmer auf den Empfang der ersten gültigen TDMA-Runde, um den aktuellen Wert des Zykluszählers vom Master zu erhalten. Danach wechselt das Gerät in den normalen Betrieb; erst jetzt wird der Ausgang freigegeben.

Im normalen Betrieb wartet das Gerät auf den Empfang einer gültigen TDMA-Runde über den Bus. Wird innerhalb einer bestimmten Zeit keine gültige Runde empfangen, wird ein interner Fehlerzähler erhöht und der Wartevorgang erneut gestartet. Erreicht der Fehlerzähler einen vorbestimmten Wert, schaltet sich das Gerät selbständig ab. Nach dem Empfang einer gültigen Runde vergleichen Master- und Beobachterkanal die empfangenen Daten auf Übereinstimmung; auch hier wird ein Fehlerzähler erhöht, wenn keine Übereinstimmung zwischen beiden Kanälen besteht. Der Masterkanal gibt nun den gewünschten Stellwert über den D/A-Wandler aus. Nun messen beide Kanäle das Sensorsignal und den tatsächlich am D/A-Wandler ausgegebenen Spannungspegel, der mit dem gewünschten Wert verglichen wird. Wieder folgt ein Datenaustausch und Vergleich zwischen beiden Kanälen. Hat bis hierher alles gestimmt, gibt der Beobachterkanal den RS485-Transceiver frei, so daß der Masterkanal die ermittelten Meßwerte über den Bus an den Rechnerkern senden kann. Wichtig ist, daß die interne Verarbeitung im Teilnehmer durch den TTSB-Buszyklus gesteuert wird – ohne gültigen Zyklus findet keine Verarbeitung statt, insbesondere wird die A/D-Wandlung nicht angestoßen.

Der Peripheriebusteilnehmer hat zwei reale E/A-Punkte, den Meßwert und den gewünschten Stellwert. Beide werden als TTSB-Standardanalogwert in zwei Bytes dargestellt, wobei das von diesem Format angebotene Konfidenzmaß nicht verwendet wird. Da mit der für den Prototypen entwickelten Untermenge von TTSB keine dynamische Konfiguration möglich ist, wurde die Slotaufteilung statisch abgelegt. Der Peripheriebusteilnehmer erwartet direkt nach dem Fireworks-Frame und dem Zykluszähler den vom Rechnerkern gewünschten Stellwert. Danach folgt eine relativ lange Pause von 11 Slotzeiten, die zur Bestimmung des Eingangswertes und zum Abgleich des Master- und des Beobachterkanals notwendig sind. Diese erstaunlich lange Zeit (fast 2ms) wird durch drei Umstände bedingt:

1. Der zur Verstärkung des D/A-Wandlers verwendete Operationsverstärker LMC 7111 ist mit einer Anstiegsgeschwindigkeit von nur $0.015V/\ \mu$s (nach Datenblatt) relativ langsam. In der realen Schaltung benötigt er ca. $360\ \mu$s, um das Ausgangssignal von $-5V$ auf $+5V$ zu ändern. Während dieser Zeit kann der Teilnehmer das Ausgangssignal nicht überprüfen, da es noch nicht stabil anliegt. Durch geschickte Wahl der Reihenfolge, in der die A/D-Wandlerkanäle abgefragt werden, kann dieser Effekt zwar etwas gemildert, aber nicht völlig beseitigt werden. Hierbei wird zuerst der externe Eingang abgefragt, und erst danach die internen Signalrückführungen zur Überprüfung des D/A-Wandlersignals. Der LMC 7111 könnte zwar durch den pinkompatiblen, wesentlich schnelleren LM324 ersetzt werden; dieser hat aber einen kleineren Aussteuerungsbereich.

2. Sowohl der D/A-, als auch der A/D-Wandler werden über eine synchrone serielle Schnittstelle angeschlossen. Dazu kann die schnelle Hardware-SPI-Schnittstelle des 68332 nicht verwendet werden, da diese bereits für die Kommunikation zwischen den beiden Prozessoren benutzt wird. Statt dessen wird eine weitere SPI-Schnittstelle über die sog. „Time Processing Unit“ (TPU) des 68332 nachgebildet. Die TPU ist eine eigenständige, mikroprogrammierebare Einheit, die in verschiedenen Mikrocontrollern der Firma Motorola verwendet wird und einfache Ein-/Ausgabefunktionen wie Pulsbreitenmodulation, Frequenzmessung, etc. völlig autonom vom Hauptprozessor ausführen kann und diesen damit entlastet. Die Geschwindigkeit dieser SPI-Schnittstelle ist jedoch auf ca. 150kBit/s begrenzt. Es müssen insgesamt fünf Worte zu je 12 Bit übertragen werden; dies entspricht

einer Zeit von $400\ \mu$s. Der A/D-Wandler schließlich benötigt für die Digitalisierung eines Kanals ca. $64\ \mu$s; bei drei Kanälen ergibt sich daraus eine zusätzliche Wartezeit von $192\ \mu$s.

3. Die Synchronisation zwischen den beiden Kanälen des Teilnehmers ist vergleichsweise aufwendig (der dafür benötigte Code macht einen großen Teil der gesamten Firmware aus). Gleichzeitig ist der verwendete 68332 mit ca. 2-3 MIPS nicht besonders schnell. So hat sich nach eingehenden Zeitmessungen herausgestellt, daß der Anfang des ersten Slots nach dem Fireworks-Byte nicht korrekt eingehalten wird, weil die $50\ \mu$s Zeit nicht ausreicht, die das Gerät nach Empfang des Fireworks-Byte für die Abarbeitung des Protokolls zur Verfügung hat. Der Teilnehmer funktioniert aber trotzdem, weil er im ersten Slot nichts senden muß – hier wird vom Master der Zykluszähler gesendet – und er daher erst am Ende des ersten Slots aktiv werden muß. Die Startpunkte der übrigen Slots werden aber korrekt eingehalten.

Da im Prototypen keine weiteren Peripheriebusteilnehmer vorhanden sind, besteht eine TTSB-Runde daher aus insgesamt 21 Slots: ein Fireworks-Frame, ein Slot für den Zykluszähler, für die beiden realen E/A-Punkte jeweils zwei Slots für die Daten und zwei Slots für die End-To-End Prüfsumme, sowie 11 Slots Verarbeitungszeit. Bei der verwendeten Bitrate von 100kBit$/s$ ergibt sich somit eine Rundendauer von insgesamt 3.36ms.

7.4 Peripheriesystemsoftware

Die Peripheriesystemsoftware erlaubt der Anwendung einen einfachen Zugriff auf die Peripherie, ohne daß diese sich um die Details der Busansteuerung oder der Fehlererkennung kümmern muß. Da eine genauere Beschreibung der Aufgaben der Peripheriesystemsoftware bereits in Kapitel 4.5 erfolgt ist, soll hier nur auf die Besonderheiten bei der Implementierung für den Prototypen eingegangen werden.

Da die Peripheriesystemsoftware aus einem anwendungsneutralen und einem anwendungsspezifischen Teil besteht, liegt es nahe, einen objektorientierten Ansatz zu wählen. Der Prototyp implementiert das Peripheriesystem daher als „Baukasten“ aus C++-Klassen, die den anwendungsneutralen Teil kapseln. Die Klassenhierarchie ist in Bild 7.6 dargestellt. Das Peripheriesystem enthält genau eine Instanz der Klasse `PControl`, die den äußeren Rahmen für das zyklische Verhalten des Peripheriesystems bereitstellt. Jedes Peripheriesystem hat mindestens eine von der abstrakten Basisklasse `PCycle` abgeleitete Klasse (im Bild `TTSBCycle`), die alle anwendungsspezifischen Details eines Peripheriezyklus kapselt. `PDriver` ist eine abstrakte Basisklasse für Gerätetreiber; für jede unterstützte Busanschaltungshardware existiert eine entsprechende abgeleitete Treiberklasse (für den Prototypen exisitiert nur die Treiberklasse `ppciDriver` für die TTSB-Busmasterkarte).

Um die Peripheriesystemsoftware an die Anwendung anzupassen, muß der Anwendungsentwickler folgende Aufgaben erfüllen:

- Beschreibung der realen E/A-Punkte.

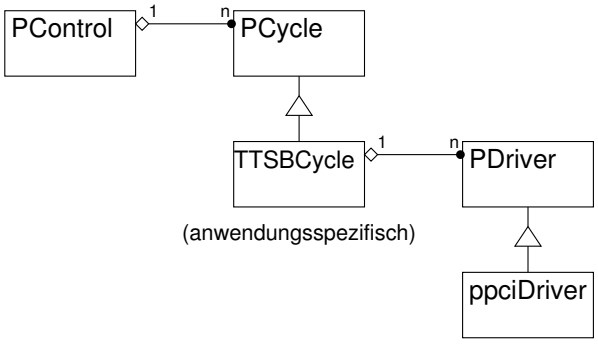

Bild 7.6: Klassenhierarchie des Peripheriesystems, am Beispiel der „Schwebenden Kugel“

- Festlegung der virtuellen E/A-Punkte und der Abbildungsvorschriften, mit denen reale und virtuelle E/A-Punkte aufeinander abgebildet werden können.
- Festlegung der Nachrichten, mit denen Peripheriesystemsoftware und Anwendungsprozesse miteinander kommunizieren.
- Festlegung des Zeitverhaltens, d. h. in welchen zeitlichen Abständen Peripheriezyklen durchgeführt werden sollen.

Die Anpassung des Peripheriesystems an die Anwendung muß momentan noch „von Hand“ geschehen, indem der Entwickler eine eigene Klasse von `PCycle` ableitet und dort den anwendungsspezifischen Code einträgt. Diese Arbeit könnte aber in Zukunft genauso von einem automatischen Werkzeug übernommen werden, dem der Anwender nur die entsprechenden Spezifikationen übergeben muß.

Die Kommunikation mit der Anwendung wird über normale ADES Nachrichten abgewickelt. Ein Anwendungsprozeß kann die Werte virtueller Ausgangspunkte für den nächsten Peripheriezyklus festlegen, bzw. die aktuellen Werte der virtuellen Eingangspunkte anfordern. Eine solche Anforderung ist blockierend, d. h. das Peripheriesystem antwortet erst, wenn ein neuer Zyklus durchgeführt worden ist und aktuelle Werte verfügbar sind. Auf diese Weise wird die Anwendung mit dem Peripheriezyklus synchronisiert. Der prinzipielle Ablauf der Kommunikation ist in Bild 7.7 anhand der Beispielanwendung dargestellt. Die Festlegung des aktuellen Spulenstromes und die Anforderung der aktuellen Position der Kugel sind in einer Nachricht zusammengefaßt, um die Anzahl der Synchronisationspunkte zu minimieren. In der Beispielanwendung ist dies problemlos möglich, da es nur einen Anwendungsprozeß gibt.

7.5 Die Anwendungssoftware

Die Anwendungssoftware ist für die Regelung der schwebenden Kugel zuständig. Sie erhält vom Peripheriesystem die aktuelle Position des Schwebekörpers und berechnet mit Hilfe eines einfachen PID-Regelalgorithmus den gewünschten Stromfluß durch die Magnetspule. Die

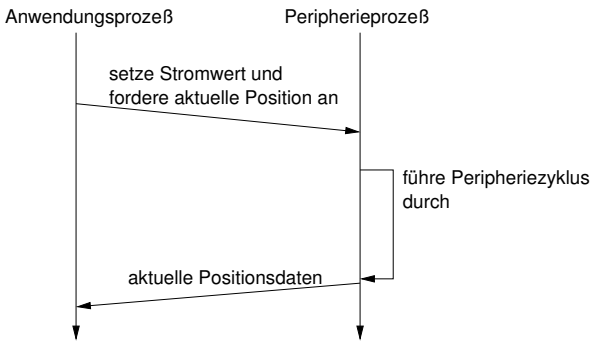

Bild 7.7: Kommunikation zwischen Anwendung und Peripheriesystem für die Beispielanwendung „Schwebende Kugel“

Beispielanwendung ist der bei weitem einfachste Teil des gesamten Systems, da hier die Besonderheiten eines redundant arbeitenden Systems überhaupt nicht berücksichtigt werden müssen.

7.6 Einschränkungen des Prototypen

Da der Prototyp nur zur prinzipiellen Erprobung des Konzepts gedacht ist und keine fertige Produktentwicklung ist, bestehen eine Reihe von Einschränkungen, die dem Einsatz in einer realen Anwendung entgegenstehen würden. Das Konzept selber ist davon nicht berührt, da eine Implementierung für die „reale Welt“ jederzeit möglich ist. Die Einschränkungen sollen aber trotzdem nicht unerwähnt bleiben.

1. Der vom Konzept geforderte Busguardian (siehe Kapitel 4.4.5) wurde nicht implementiert. Ein einfacher Busguardian, der einen Kernrechner im Falle eines Ausfalls komplett vom Bus abtrennt, wäre zwar relativ einfach zu realisieren, aber konzeptionell nur von geringem Interesse gewesen.

   Der wesentlich interessantere komplexe Busguardian, der nur das Senden eines Rechners selektiv verhindert, konnte nicht realisiert werden. Es existiert zwar eine Testschaltung, die extern an eine TTSB-Masterkarte angeschlossen wird und tatsächlich selektiv nur das Senden über das RS485-Medium verhindern kann. Die Realisierung eines vollwertigen Busguardians benötigt jedoch eine genaue zeitliche Synchronisation zwischen der Busmasterkarte und dem Busguardian. Da die externen Schnittstellen der verwendeten Rechner (Parallelport und RS-232 Schnittstellen) nicht schnell genug sind, müßte der Busguardian als zusätzlich zu entwickelnde Einsteckkarte realisiert werden.

2. Die Hardware der fail-safe Peripherieteilnehmer ist ebenfalls nur prototypisch realisiert und ist daher nicht vollständig ausgereift; insbesondere die Störsicherheit des Analogteils ist ungenügend.

3. Die Peripheriebusteilnehmer speichern den Fehlerzustand nicht, d. h. wenn eine Störung beseitigt ist, laufen sie ohne externe Intervention sofort wieder. Dies ist notwendig, um in

der Entwicklungs- und Testphase einen einigermaßen durchlaufenden Betrieb zu ermöglichen. In einem realen System dürfen einmal ausgefallene Komponenten sich aber auf keinen Fall selbständig in das System reintegrieren; ansonsten könnte ein Mehrfachfehler dazu führen, daß sich ein defektes Gerät irrtümlich reintegriert und die Funktion des Gesamtsystems beeinträchtigt.

4. Die entwickelte Software ist ein „proof of concept“, und nicht für den Einsatz in realen Systemen geeignet. Software für sicherheitskritische Systeme muß nach speziellen Regeln entwickelt, dokumentiert und getestet werden (siehe z. B. [30]). Der damit verbundene Aufwand ist enorm (erfahrungsgemäß dauert die Dokumentation und Verifikation sicherer Software deutlich länger als die eigentliche Entwicklung) und für einen nur zu Demonstrationszwecken gedachten Prototypen nicht zu rechtfertigen.

8 Ergebnisse und Ausblick

In diesem Kapitel werden die anhand des im vorherigen Kapitel beschriebenen Demonstrationsaufbaus gewonnen Erfahrungen und Ergebnisse vorgestellt und ein Ausblick auf mögliche weitere Arbeiten gegeben.

8.1 Bewertung des Konzepts

Um die Leistungsfähigkeit des Systems beurteilen zu können, wurden anhand der Beispielanwendung (siehe Kapitel 7.1) einige Laufzeiten ermittelt:

- Die eigentliche Anwendung, der PID-Regelalgorithmus, benötigt auf den verwendeten Rechnern etwa $8\ \mu$s Bearbeitungszeit.
- Die Vor- und Nachbearbeitung innerhalb der Peripheriesystemsoftware (Umwandlung von virtuellen in reale E/A-Punkte und umgekehrt, sowie die Ansteuerung der Bushardware) benötigt jeweils ca. $45 - 50\ \mu$s.
- Ein Buszyklus selber dauert 3.36ms; während dieser Zeit ruht der Rest des Systems.
- Der Prototyp erlaubt eine minimale Zykluszeit von 8ms. Die Synchronisation der Rechner untereinander benötigt also ca. 4ms.

Anhand dieser Zahlen erscheint das Konzept auf den ersten Blick nicht besonders sinnvoll, da die für die eigentliche Anwendung benötigte Rechenzeit um drei Größenordnungen unter der für die Synchronisation der Rechner untereinander benötigten Zeit liegt.

Bei der momentanen Implementierung hat jeder Synchronisationspunkt einen hohen zeitlichen Aufwand, der durch zwei Umstände verursacht wird. Zum einen müssen die zu synchronisierenden Daten eine Vielzahl von Schritten durchlaufen, bevor sie tatsächlich synchronisiert werden können – sie werden zuerst an einen Kommunikationsprozeß (dies ist ein Teil der ADES Laufzeitumgebung) übergeben, der sie dann über TCP/IP an die anderen Rechner sendet. In QNX sind alle Protokoll- und Gerätetreiber eigenständige Prozesse – im Falle der Kommunikation über TCP/IP ist der Socket-Prozeß, der Netzwerk-Prozeß und ein Netzwerkinterfacetreiberprozeß involviert. Zum anderen benötigt die Kommunikation über TCP/IP unter QNX sporadisch deutlich mehr Zeit als normal; bei der Dimensionierung der Timeouts, mit denen ein Rechnerausfall erkannt wird, muß aber grundsätzlich der ungünstigste im nicht fehlerhaften Betrieb mögliche Fall berücksichtigt werden. Konkret bedeutet dies, daß die für die Synchronisation notwendige Zeit nicht linear mit der Anzahl der zu synchronisierenden Daten wächst, sondern

primär von der Anzahl der Synchronisationspunkte abhängt. Eine komplexere Anwendung wird daher nur eine unwesentlich längere Reaktionszeit als die Beispielanwendung haben, wenn nur die Menge der synchronisierten Daten erhöht wird, und nicht gleichzeitig auch die Anzahl der Synchronisationspunkte.

Da von der Funktionalität von TCP/IP nur ein Bruchteil benötigt wird, liegt der Gedanke nahe, ein eigenes, wesentlich einfacheres Kommunikationsprotokoll für das Rechner/Rechner-Kommunikationssystem zu entwickeln, mit dem die für die Kommunikation benötigte Zeit reduziert werden kann. Generell aber wird in einem ADES-System – wie auch in allen bisherigen redundanten Systemen, die überwiegend oder ausschließlich per Software synchronisieren – ein Großteil der verfügbaren Systemleistung für die Synchronisation der Kernrechner benötigt werden. Eine deutliche Reduzierung des Synchronisationsoverheads ließe sich nur durch den Einsatz taktsynchroner Hardware mit Hardwarevotern erreichen, was aber dem Ziel, möglichst nur COTS-Komponenten zu verwenden, direkt widerspricht. Der Anwendungsentwickler wird daher darauf achten müssen, seine Anwendung so in einzelne Teilaufgaben aufzuteilen, daß möglichst wenige Synchronisationspunkte benötigt werden.

Ein weiteres grundlegendes Problem ergibt sich bei Regelungsaufgaben wie der Beispielanwendung durch die Atomizität des TTSB-Buszyklus. Sie sorgt dafür, daß die berechneten Stellwerte erst einen Buszyklus nach der Ermittlung der Istwerte übertragen werden können und verursacht so eine äußerst lange Totzeit, die in der Größenordnung der Abtastzeit liegt. Bei Anwendungen, die sehr empfindlich auf Totzeiten reagieren, muß daher entweder stark überabgetastet werden, oder ein Regelalgorithmus mit Beobachter verwendet werden, der die lange Totzeit kompensieren kann.

Gegenüber diesen Einschränkungen stehen aber die Vorteile eines auf COTS Komponenten basierenden Systems. So könnten die Reaktionszeiten des Prototypen allein durch den Einsatz aktueller Rechnertechnik (es sind derzeit Geräte verfügbar, die – bei vergleichbaren Preisen – gegenüber den verwendeten Geräten in etwa die 10-fache Rechenleistung aufweisen) verbessert werden, ohne am sonstigen System etwas ändern zu müssen. Lediglich die verwendeten Timeouts müßten nach unten korrigiert werden.

8.2 Bezug zu Normen

Aufmerksame Leser werden festgestellt haben, daß diese Arbeit keinen Bezug zu den einschlägigen Normen herstellt. Diese „Unterlassung“ ist bewußt geschehen. Ziel des ADES-Projektes war es, zu untersuchen, was technisch mit vertretbarem Aufwand bei der Verwendung von Standardkomponenten an Sicherheit und Verfügbarkeit realisierbar ist, und nicht, ein fertiges System zur normgerechten Umsetzung einer bestimmten Anwendung zu entwickeln. Wird ein reales System nach ADES entworfen, muß der Entwickler daher die für seine Anwendung gültigen Normen studieren und prüfen, ob dieses reale System auch die gestellten Anforderungen erfüllen kann.

Die Einführung der Maschinenrichtlinie der Europäischen Union kommt dem Entwickler hier

insofern entgegen, als die neuen Normen (siehe z. B. EN954-1 „Sicherheit von Maschinen“, [13]) bestimmte Eigenschaften fordern („ein beliebiger Fehler darf nicht zu einer Gefährdung führen“), statt, wie frühere Normen, konkrete technische Vorschriften zu machen („Komponente **X** muß/darf nicht verwendet werden“).

8.3 Ausblick, weitere Schritte

Zentrales Ziel von ADES ist es, die Entwicklung sicherer Systeme zu vereinfachen. Der Entwickler soll sich möglichst auf seine eigentliche Aufgabe, die Entwicklung der Anwendungsfunktionalität, konzentrieren können, ohne sich um die Details der Fehlertoleranz kümmern zu müssen. Dieses hohe Ziel konnte nicht vollständig erreicht werden – während die Synchronisation der Ein- und Ausgabedaten eines Anwendungsprozesses automatisch geschieht, müssen interne Zustände eines Prozesses „von Hand“ identifiziert werden, damit auch sie synchronisiert werden können. Die Synchronisation selber ist hier nicht das Problem, da sie analog zur Synchronisation der Ein- und Ausgabedaten geschehen kann, sondern die Identifizierung der relevanten internen Zustände. Ein hier gemachter Fehler kann sehr schwer zu entdecken sein, da er nicht unbedingt sofort zu einem Ausfall führen muß, aber eine Reintegration nach einem Ausfall verhindern kann.

Abhilfe könnte hier der Einsatz von CASE-Werkzeugen mit automatischer Codeerzeugung bieten. Ein mögliches Werkzeug wäre Matlab/Simulink mit einem speziell für ADES angepaßten Codegenerator. Ein Simulink-Modell besteht aus einer Reihe vorgefertigter Elemente (z. B. ein Integrator), dessen interne Zustände wohl definiert und bekannt sind. Ein an ADES angepaßter Codegenerator könnte daher den Code zur Synchronisierung dieser Zustände gleich mit generieren. Ein solcher Ansatz hat den Vorteil, daß diese Arbeit nur einmal gemacht (bei der Entwicklung des Codegenerators) und vor allem nur einmal getestet und zertifiziert werden muß. Als zusätzlichen „Bonus“ würde eine CASE-basierte Entwicklung durch den höheren Abstraktionsgrad mit hoher Wahrscheinlichkeit auch die Entwicklung der Anwendung an sich vereinfachen und beschleunigen.

Ein weiteres Einsatzgebiet für CASE-Werkzeuge ist die Konfiguration des Peripheriesystems. Hier muß der Entwickler ebenfalls „von Hand“ die Art und Anzahl der realen E/A-Punkte spezifizieren und den Code schreiben, mit denen sie auf virtuelle E/A-Punkte abgebildet werden. Auch diese Arbeit ließe sich mit einem geeigneten Werkzeug automatisieren. Im Gegensatz zum obigen Fall existiert aber kein fertiges Werkzeug, das nur angepaßt werden muß, sondern es müßte von Grund auf neu entwickelt werden.

Literaturverzeichnis

[1] *ASIC-Based Bus-System for a New Generation of Computerised Level Crossing Systems*. FUSE Demonstrator Document, Application Experiment Number 2227. `http://www.fuse-network.com/fuse/demonstration/34_35/2227/2227.pdf`, [24.01.2003].

[2] *CAN Specification 2.0*. Robert Bosch GmbH, 1991.

[3] AS-INTERNATIONAL ASSOCIATION: *As-Interface Safety at Work*. Präsentation der AS-International Association, Oktober 2000. `http://www.as-interface.com/asi/safetygerman.ppt`, [08.10.2002].

[4] *Sicherheitstechnik mit AS-i-Safety At Work*. atp – Automatisierungstechnische Praxis, 44(4):27, 2002.

[5] AVIZIENS, A.: *A design paradigm for fault-tolerant systems*. In *Proc. AIAA/IEEE Digital Avionics Systems Conf.*, Washington, D.C., 1987. American Inst. of Aeronautics and Astronautics, Inc.

[6] BAGINSKI, ALFREDO und MARTIN MÜLLER: *Interbus-S: Grundlagen und Praxis*. Hüthig, Heidelberg, 1994.

[7] BARTH, FRIEDRICH, PAUL MÜHLBAUER, DR. FRIEDRICH NIKOL und KARL WÖRLE (Herausgeber): *Mathematische Formeln und Definitionen*. Bayerischer Schulbuch-Verlag München, 4. Auflage, 1985.

[8] BAUER, GÜNTHER and HERMANN KOPETZ: *Transparent Redundancy in the Time-Triggered Architecture*. In *International Conference on Dependable Systems and Networks*, 2000.

[9] BAUMGARTNER, R.: *Ein fehlertolerantes Multimikrorechnersystem: Entwicklung der E/A-Hardware*. Diplomarbeit, Lehrstuhl für Prozeßrechner, TU München, 1982.

[10] BRIÈRE, DOMINIQUE and PASCAL TRAVERSE: *AIRBUS A320/A330/A340 Electrical Flight Controls – A Family of Fault-Tolerant Systems*. In *International Symposium on Fault-Tolerant Computing*, 1993.

[11] BRONŠTEIN, IL'JA N. und K. A. SEMENDJAJEW: *Taschenbuch der Mathematik*. B. G. Teubner Verlagsgesellschaft, Stuttgart, Leipzig und Verlag Nauka, Moskau, 25. Auflage, 1991.

[12] DANGL, G.: *Ein fehlertolerantes Multimikrorechnersystem: Entwicklung digitaler E/A-Module*. Diplomarbeit, Lehrstuhl für Prozeßrechner, TU München, 1984.

[13] EN954-1: *Sicherheit von Maschinen – Sicherheitsbezogene Teile von Steuerungen. Teil 1: Allgemeine Gestaltungsleitsätze*. Beuth-Verlag, Berlin, 1996.

[14] ENDL, HANS: *Porzeßankopplung in fehlertoleranten Rechnersystemen*. Doktorarbeit, Fakultät für Elektrotechnik und Informationstechnik, TU München, 1988.

[15] ETSCHBERGER, KONRAD und ANDERE: *CAN Controller–Area–Network, Grundlagen, Protokolle, Bausteine, Anwendungen*. Hanser Verlag, 1994.

[16] FRITZSCHE, HANS-THOMAS: *Der Sicherheits-Bus*. Elektronik, Heft 16, 1999.

[17] HEISS, HANS-ULRICH: *Informatik im Cockpit: Pilot contra Computer*. Festvortrag anlässlich der Verabschiedungsfeier der Absolventen des b.i.b. (Bildungszentrum für informationsverarbeitende Berufe) Paderborn, März 1999. `http://kbs.cs.tu-berlin.de/publications/presentations/He260399.pdf`, [02.12.2002].

[18] HERMANN, HANS: *Zuverlässigkeitsverfahren für die Prozeßmeßtechnik*. R. Oldenbourg München Wien, 1972.

[19] INDEFREY, KLAUS und WOLFGANG STRIPF: *Sicherheit im Doppelpack*. Computer und Automation, 1-2, Februar 2000.

[20] INTERBUS CLUB DEUTSCHLAND E.V: *White Paper INTERBUS Safety*. `http://www.phoenixcontact.com/de/information/update/download/whitepaper.pdf`. [08.10.2002].

[21] ISERMANN, ROLF, RALF SCHWARZ, and STEFAN STÖLZL: *Fault-tolerant drive-by-wire systems*. IEEE Control Systems Magazine, 22(5):64–81, October 2002.

[22] KLEINBREUER, WERNER, FRANZ KRUETZKAMPF, KALRHEINZ MEFFERT und DIETMAR REINERT: *BIA Report 6/97: Kategorien für sicherheitsbezogene Steuerungen nach EN 954-1*. Hauptverband der gewerblichen Berufsgenossenschaften (HVBG), 1997.

[23] KOPETZ, H.: *Specification of the TTP/A-Protocol V2.00*. Real-Time Systems Group University of Technology Vienna, September 2002.

[24] LAMPORT, L., R. SHOSTAK, and M. PEASE: *The byzantine generals problem*. ACM Trans. Programming Languages and Systems, Vol. 4(Nr. 3):382–401, Juli 1982.

[25] PARHAMI, BEHROOZ: *Voting algorithms*. IEEE Transactions on Reliability, 43(4):617–629, December 1994.

[26] *Profisafe, Profil für Sicherheitstechnik, prV1.0*. Profibus Nutzerorganisation e. V. , 1999.

[27] POWELL, D., J. ARLAT, L. BEUS-DUKIC, A. BONDAVALLI, P. COPPOLA, A. FANTECHI AMD E. JENN, C. RABÉJAC, and A. WELLINGS: *Guards: A generic upgradable architecture for real-time dependable systems*. IEEE Transactions on Parallel and Distributed Systems, 10(6):pp. 580–599, June 1999.

[28] POWELL, DAVID: *Failure mode assumptions and assumption coverage*. In *Predictablly Dependable Computing Systems*, pages 123–140. Springer Verlag, 1995.

[29] RANNENBERG, KAI und ANDREAS PFITZMANN: *Sicherheit, insbesondere mehrseitige IT-Sicherheit*. Informationstechnik und technische Informatik, 38. Jahrgang(Heft 4), 1996.

[30] REINERT, DIETMAR, MICHAEL SCHAEFER und THOMAS BÖMER: *Regeln für den Entwurf und die Programmierung sicherheitsbezogener Software*. atp – Automatisierungstechnische Praxis, 41(6):21–30, 1999.

[31] RENNELS, DAVID A.: *Fault tolerant computing - concepts and examples*. IEEE Trans. on Computers, Vol. C-33(Nr. 12):1116–1129, 1984.

[32] RUFINO, J., P. VERISSIMO, G. ARROZ, C. ALMEIDA, and L. RODRIGUES: *Fault-tolerant broadcasts in can*. In *Digest of Papers, The 28th IEEE International Symposium on Fault-Tolerant Computing Munich, Germany*, pages 150–159, June 1998.

[33] SCHAEFER, M.: *New concepts for safety-related bus systems*. In *3. Internationales Symposium Programmierbare Systeme für sicherheitsgerichtete Anwendungen*. Berufsgenossenschaftliches Institut für Arbeitssicherheit, May 1998.

[34] SCHRÜFER, ELMAR: *Zuverlässigkeit von Mess- und Automatisierungseinrichtungen*. München; Wien: Hanser, 1984.

[35] SCHUMANN, THILO und HOLGER ZELTWANGER: *Konzepte moderner Sicherheitssysteme: CANopen-Safety*. SPS-Magazin, Heft 3, 2001.

[36] STOLL, JÜRGEN: *Fehlertoleranz in verteilten Realzeitsystemen*. Informatik Fachberichte Nr. 236. Springer Verlag, Berlin Heidelberg, 1990.

[37] SURI, N., C.J. WALTER, and M.M. HUGUE: *Advances In Ultra-Dependable Distributed Systems*, chapter 1. IEEE Computer Society Press, 10662 Los Vaqueros Circle
P.O. Box 3014
Los Alamitos, CA 90720-1264, 1995.

[38] TINDELL, KEN and ALAN BURNS: *Guaranteeing message latencies on controller area network (can)*. In *Proceedings 1st International CAN Conference*, Mainz, Germany, September 1994.

[39] URBAN, GERD: *A survivable avionics system for space applications*. In *International Symposium on Fault-Tolerant Computing*, 1998.

[40] WICKER, STEPHEN B.: *Error control systems for digital communication and storage*. Prentice-Hall, 1995.

[41] WOLF, JACK KEIL and ROBERT D. BLAKENEY II: *An exact evaluation of the probability of undetected error for certain shortened binary crc codes*. In *MILCOM 88, Conference Record*, volume 1, pages 287–292. IEEE, 1988.

Index