Fortschritt-Berichte VDI

VDI

Reihe 10

Informatik/
Kommunikation

Stefan Widmann, M.Sc.,
Freudenberg

Nr. 856

Eine Datenspezifikations-architektur

FernUniversität in Hagen
Schriften zur Informations- und Kommunikationstechnik

Fortschritt-Berichte VDI

Reihe 10

Informatik/
Kommunikation

Stefan Widmann, M.Sc.,
Freudenberg

Nr. 856

Eine Datenspezifikations-architektur

FernUniversität in Hagen
Schriften zur Informations- und Kommunikationstechnik

Widmann, Stefan
Eine Datenspezifikationsarchitektur
Fortschr.-Ber. VDI Reihe 10 Nr. 856. Düsseldorf: VDI Verlag 2017.
328 Seiten, 91 Bilder, 42 Tabellen.
ISBN 978-3-18-385610-7, ISSN 0178-9627,
€ 104,00/VDI-Mitgliederpreis € 93,60.

Für die Dokumentation: Echtzeitsysteme – funktionale Sicherheit – sicherheitsgerichtete Echtzeitsysteme – IEC 61508 – Mikroprozessorarchitekturen – Prozessorarchitekturen – Datentyparchitekturen – Befähigungsarchitekturen – Datenfluss – Datenflussüberwachung

Die vorliegende Arbeit richtet sich an Ingenieure und Wissenschaftler in den Bereichen Mikroprozessorarchitektur und sicherheitsgerichtete Echtzeitsysteme. Sie beginnt mit der Identifikation von 20 datenflussbezogenen Fehler- und Angriffsarten und evaluiert anhand dieser den Stand von Wissenschaft und Technik. Anschließend wird eine neue Prozessorarchitektur, die Datenspezifikationsarchitektur, vorgestellt, welche die in Vergessenheit geratenen Merkmale von Datentyparchitekturen stark erweitert und alle Dateneigenschaften in Form zusätzlicher Kennungen untrennbar mit dem Datenwert verknüpft, überträgt, speichert und verarbeitet. Dies ermöglicht es der neuen Architektur, alle 20 Fehler- und Angriffsarten zu erkennen. Die schlussendliche Gegenüberstellung des Stands von Wissenschaft und Technik und der Datenspezifikationsarchitektur zeigt die Überlegenheit der neuen Architektur und deren hervorragende Eignung für die Realisierung sicherheitsgerichteter Anwendungen.

Bibliographische Information der Deutschen Bibliothek
Die Deutsche Bibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliographie; detaillierte bibliographische Daten sind im Internet unter http://dnb.ddb.de abrufbar.

Bibliographic information published by the Deutsche Bibliothek
(German National Library)
The Deutsche Bibliothek lists this publication in the Deutsche Nationalbibliographie (German National Bibliography); detailed bibliographic data is available via Internet at http://dnb.ddb.de.

Schriften zur Informations- und Kommunikationstechnik
Herausgeber:
Wolfgang A. Halang, Lehrstuhl für Informationstechnik
Herwig Unger, Lehrstuhl für Kommunikationstechnik
FernUniversität in Hagen

ISSN 0178-9627
ISBN 978-3-18-385610-7

Vorwort

Der Bedarf an sicherheitsgerichteten, programmgesteuerten (eingebetteten) Systemen aller Art ist hoch und steigt durch die zunehmende Automatisierung von Prozessen kontinuierlich weiter an. Im Einklang damit wächst auch das gesellschaftliche Sicherheitsbewusstsein. Weil die auf dem Markt vorherrschenden Prozessorarchitekturen kaum Schutz gegen typische Programmierfehler und Malware-Attacken bieten, hatte der Autor des vorliegenden Buches sich zur Aufgabe gemacht, aufbauend auf den allgemeinen Sicherheitsanforderungen gemäß der Norm IEC 61508 auf Maschinenebene eine völlig neue, von Grund auf auf Sicherheit hin ausgelegte Rechnerarchitektur zu entwerfen, die eine Fülle von Programmierfehlern ohne Software-Hilfe erkennen kann und daraufhin die Programmausführung abbricht.

Der Autor des 2014 unter dem Titel „Verfahren zur Kontrollflussüberwachung in sicherheitsgerichteten Rechensystemen“ in dieser Buchreihe mit der Nummer 832 erschienenen Bandes hatte sich bereits sehr eingehend mit der Sicherung des Kontrollflusses in sicherheitsgerichteten Echtzeitsystemen beschäftigt und beeindruckende Ergebnisse vorgelegt. Allerdings stellen Kontrollflussfehler nur einen geringen Anteil aller Programmfehler dar. Der weitaus größere Teil aller Fehler, die in programmgesteuerten Digitalrechnern auftreten, sind Datenflussfehler, deren Überwachung und Verhinderung sich deshalb Herr Widmann an dieser Stelle annimmt.

Datenflussfehler sind in höchstem Maße gefährlich, da sie insbesondere in der von Neumann-Architektur, die die völlig beliebige Interpretation jedes Bitmusters erlaubt, eine Vielzahl der technisch möglichen Reaktionen solcher Rechner auszulösen vermögen. Durch während des Entwurfs eines Programms gemachte oder zur Laufzeit auftretende Datenflussfehler kann sich auch der Kontrollfluss in unerwarteter Weise verändern, so dass Befehle in unvorhergesehener und falscher Reihenfolge, aber auch Daten, in denen keine Befehle codiert sind, als Befehle interpretiert und ausgeführt werden. Durch Fehler in der Gerätetechnik, transiente Störungen, intermittierende Fehler oder permanente Ausfälle kann es dazu kommen, dass die Bitmuster von Daten verändert werden. Sehr oft reicht eine einzige fehlerhafte Bitposition aus, um eine völlig verschiedene Aktion auszuführen.

Die von Herrn Widmann verfolgte Zielstellung ergibt sich unmittelbar aus dem unzureichenden Stand der Technik, und zwar gerätetechnische Fehlervermeidungs- und -erkennungsmöglichkeiten zu schaffen, um damit datenflussbezogene Fehler und Angriffe erkennen und die Einhaltung von Echtzeitbedingungen überwachen zu können. Die vorgestellten Ergebnisse sind in allen Bereichen der elektronischen Datenverarbeitung anwendbar und dort wegen deren geringer Zuverlässigkeit auch dringend erforderlich. Trotzdem ist das Werk aus Sicht der Automatisierungstechnik geschrieben, weil Digitalrechner trotz der Unmöglichkeit, wirklich vertrauenswürdige Sicherheitsnachweise für programmgesteuerte Systeme zu führen, mehr und mehr auch für sicherheitsgerichtete Anwendungen eingesetzt werden und dabei bewährte, oft inhärent sichere gerätetechnische Lösungen ersetzen. Weiterhin sind im Anwendungsgebiet sicherheitskritischer Echtzeitsysteme weder nicht zeitdeterministisch arbeitende Verfahren hinnehmbar, noch dürfen Fehler erst nachträglich korrigiert werden.

Bemerkenswert an Herrn Widmanns wissenschaftlich-technischen Beiträgen ist eine Reihe von Aspekten. In ganzheitlicher Betrachtung von Hardware und Software verfolgt er konzeptionell ein neuartiges Entwurfsparadigma, dass nämlich alle Deskriptoren eines Datenspeicherelementes in untrennbarer Verknüpfung mit diesem gespeichert, verarbeitet und übertragen sowie gerätetechnisch überprüfbar dargestellt werden sollen. Mit Hilfe solcher selbstbeschreibenden Daten können dann in Hardware implementierte Überprüfungen die meisten datenflussbezogenen Fehler in der Datenverarbeitung auch über Grenzen zwischen Systemkomponenten hinweg aufdecken. Er liefert theoretische Beiträge, indem er die datenflussbezogenen Fehler- und Angriffsarten analysiert und identifiziert, die Eigenschaften in sicherheitsgerichteten Echtzeitsystemen gehaltener Daten zusammenstellt und darauf aufbauend seine Datenspezifikationsarchitektur entwirft, die die Dateneigenschaften mit bisher unerreichter Aussagekraft abbildet. Und schließlich arbeitet er konstruktiv-ingenieurmäßig, indem er für jedes angegebene Verfahren geeignete Implementierungsmöglichkeiten vorschlägt und ihre jeweiligen Vor- und Nachteile diskutiert.

Der Preis, den Herr Widmann für die Sicherung des Datenflusses in Digitalrechnern bezahlt, ist erheblich erhöhter Speicherbedarf und dementsprechend größerer Übertragungsaufwand, wohingegen der Umfang zusätzlicher Hardware für die Verarbeitung der Datenkennungen gering ist. In Zeiten enormer Speicherkapazitäten ist deutlich erhöhte Sicherheit diesen Preis jedoch unbestreitbar wert.

Hagen, im August 2017 Wolfgang A. Halang

Inhaltsverzeichnis

1 Einleitung

Der Grad der Automatisierung technischer Prozesse steigt unaufhörlich, und mit ihm auch die Verantwortung für Mensch, Umwelt und Investitionen, welche die dabei eingesetzten Datenverarbeitungssysteme tragen. Gute Beispiele dafür sind die immer stärker in den Fokus rückenden „x-by-wire“-Systeme im Automobil- und Avionikbereich, wobei das „x“ in automobilen Anwendungen z. B. für Lenkung („steer“), Bremsen („brake“) oder Schalten („shift“) stehen kann [89, 113] bzw. in der Avionik für die Realisierung zahlreicher Steuerungsfunktionen in Flugzeugen („fly“) [2]. Zuverlässige, betriebsbewährte mechanische Systeme werden in zunehmendem Maße durch Sensoren und Aktoren ersetzt, wobei mikroprozessorbasierte Datenverarbeitungseinheiten die Sensorsignale verarbeiten und Steuersignale für die Aktoren erzeugen. Die möglichen Folgen wurden beim Nissan Q50 im Jahr 2013 ersichtlich: Nach der Markteinführung des „ersten Serienfahrzeugs mit steer-by-wire Technology“ [113] im August 2013 wurden die Fahrzeuge bereits im November 2013 in die Werkstätten zurückgerufen, da die Lenkung bei tiefen Temperaturen softwarebedingt ausfallen konnte [88].

Ein weiteres Beispiel, das die steigende Verantwortung von Datenverarbeitungssystemen im Automobilbereich illustriert, ist das autonome Fahren, also die Teilnahme eines Kraftfahrzeugs am öffentlichen Verkehr, ohne dabei durch einen Menschen gesteuert zu werden. Zwei Firmen erregen dabei besonderes öffentliches Interesse: Tesla mit dem System „Autopilot“ in den Fahrzeugmodellen Model S und Model X [119] und Google mit dem Google Driverless Car, einem Projekt, das inzwischen unter dem Namen Waymo weitergeführt wird [42].

Zusätzlich zur steigenden Verantwortung der Systeme nimmt die Komplexität der in den Systemen eingesetzten Software stetig zu. Im Jahr 2006 gab Broy in [14] an, dass die Software in einem Fahrzeug bis zu 10 Millionen Zeilen Code umfassen kann. Bereits 2009 – also gerade einmal drei Jahre später – wird Broy in [17] dahingehend zitiert, dass der Umfang nun auf bis zu 100 Millionen Zeilen Code angewachsen sei. Eine weitere Erhöhung auf 200 bis 300 Millionen Codezeilen wurde 2008 in [96] vermutet.

Auch die Komplexität der verwendeten Hardwaresysteme steigt: über 70 untereinander kooperierende eingebettete Systeme wurden nach Broy 2006 in einem Fahrzeug eingesetzt [14], drei Jahre später schon bis zu 100 [17]. Eine ähnliche Entwicklung findet auf der Ebene der integrierten Schaltkreise statt: die Anzahl der Transistoren innerhalb eines Prozessors erreichte im Jahr 2015 1,3 Milliarden [107]. Die fortschreitende Integration von immer größeren Anzahlen von Bauelementen innerhalb integrierter Schaltkreise ist nur durch immer weiter reduzierte Strukturbreiten möglich, wodurch die entstehenden Produkte immer empfindlicher gegenüber Umgebungseinflüssen wie z. B. Strahlung in Form von Neutronen werden, und das sogar zunehmend auf Meereshöhe [8, 90].

All diese Faktoren wirken sich negativ auf die Fehlerwahrscheinlichkeit aus. Die so entstehenden Fehler in Hard- und Software können, wenn sie nicht erkannt und adäquat behandelt werden, zu gefährlichen Ausgabefehlern in sicherheitsgerichteten Systemen führen. Daher ist es notwendig, Fehler weitestgehend zu vermeiden, und trotz aller Vermeidungsmaßnahmen trotzdem auftretende Fehler so frühzeitig wie möglich zu erkennen: idealerweise im Moment ihres Auftretens und nicht erst, wenn ihre Auswirkungen (z. B. durch Vergleich von Ergebnissen oder Zwischenergebnissen) sichtbar werden.

Konventionelle Prozessorarchitekturen sind vor allem auf maximalen Datendurchsatz hin ausgelegt, nicht auf Einfachheit, Fehlervermeidung und -erkennung. Diese – eigentlich ungeeigneten Systeme – kommen meist aus ökonomischen Gründen in den beschriebenen Anwendungen zum Einsatz und werden als „commercial-off-the-shelf (COTS)“ bezeichnet, also als „kommerzielle Produkte aus dem Regal“. Die Datenworte in den Speichern dieser Systeme enthalten neben dem eigentlichen Datenwert keinerlei weiterführende Informationen, die dessen Eigenschaften beschreiben. In solchen Architekturen und Systemen kann den steigenden Anforderungen und Fehlerwahrscheinlichkeiten nur durch weiter steigende Komplexität begegnet werden, z. B. durch den Einsatz von softwarebasierter arithmetischer Kodierung wie Software Encoded Processing (SEP) und Compiler Encoded Processing (CEP) [108].

Gollub hat sich in [41] der Frage gestellt, wie sich der Kontrollfluss innerhalb sicherheitsgerichteter Echtzeitsysteme mit möglichst einfachen Mitteln überwachen lässt. In [118] wird jedoch davon gesprochen, dass 80 - 90 % aller Programmfehler Datenflussfehler sind und nur die verbleibenden 10 - 20 % auf Kontrollflussfehler entfallen. Weiterhin ist eine reine Kontrollflussüberwachung nicht in der Lage, Fehler – z. B. durch Inkompatibilitäten – bei der Verarbeitung von Datenwerten festzustellen, besonders bei komponentenübergreifenden Datenflüssen. Datenflussüberwachungen

können jedoch – unter Verwendung selbstbeschreibender Daten – Fehler in der Datenverarbeitung auch über Systemkomponenten hinweg aufdecken.

In der vorliegenden Arbeit wird eine leistungsfähige Architektur für die Überwachung des Datenflusses bzw. der Datenflüsse innerhalb von sicherheitsgerichteten Echtzeitsystemen vorgestellt.

1.1 Beispiele für Auswirkungen von Fehlern

Die folgenden Beispiele aus der Praxis zeigen die Auswirkungen von Fehlern, die während Spezifikation, Entwurf oder Implementierung von Hard- oder Software in ein System eingebracht wurden. Dabei wurden Beispiele gewählt, anhand derer die Leistungsfähigkeit der in dieser Arbeit vorgestellten Fehlervermeidungs- und -erkennungsmerkmale in der Evaluation besonders deutlich wird.

1.1.1 Selbstzerstörung der Ariane 5

Am Morgen des 4. Juni 1996 wurde nach [79] die Rakete Ariane 5, Flug 501, in Kourou in Französisch-Guayana um 12:34 Uhr UTC gestartet. Nach etwa 37 Sekunden normalen Flugs wich die Rakete vom Kurs ab und explodierte. Bei der Umwandlung einer 64-Bit-Gleitkommazahl in eine vorzeichenbehaftete 16-Bit-Ganzzahl war es zu einer Überschreitung des Wertebereichs gekommen, weil die horizontale Geschwindigkeit der Ariane 5 deutlich höher war als bei der Ariane 4. Der Fehler wurde durch das Inertiale Navigationssystem, engl. „Inertial Reference System SRI“, als Operandenfehler erkannt und ein Diagnose-Bitmuster zusammen mit korrekten Flugdaten an den Bordrechner gesendet, der das Bitmuster fehlerhafterweise als Flugdaten interpretierte. In der Folge steuerte der Bordcomputer die Ablenkdüsen voll aus, wodurch sich die Feststoffzusatztriebwerke von der Hauptstufe lösten und die Selbstzerstörung der Rakete angestoßen wurde.

Bemerkenswert ist, dass sich der Softwarefehler aufgrund fehlender Diversität auf die beiden redundanten Einheiten des Inertialen Navigationssystems identisch auswirkte, worauf diese zeitgleich ausfielen und deshalb keine Umschaltung auf die Reserveeinheit möglich war.

1.1.2 Verlust der NASA-Sonde Mars Climate Orbiter

Der Mars Climate Orbiter, kurz „MCO“, eine Sonde der NASA, sollte als Wettersatellit den Mars umrunden und gleichzeitig als Kommunikationsrelais für die im Dezember 1999 geplante Mars Polar Lander Mission dienen [81]. Durch die Verwendung inkompatibler Einheiten – eine Softwarekomponente rechnete in metrischen SI-Einheiten, die andere in angloamerikanischen Maßeinheiten – war die Trägerrakete zum Zeitpunkt des Absetzens der Sonde im Marsorbit rund 170 km zu tief. Das führte nach Ansicht der Untersuchungskommission dazu, dass die Sonde entweder in der Marsatmosphäre verglühte oder diese wieder verließ und in den Raum abdriftete [81].

1.1.3 Bestrahlungsgerät Therac-25

Das Therac-25 war ein für medizinische Zwecke genutzter linearer Teilchenbeschleuniger für onkologische Bestrahlungen [77]. Das durch eine PDP-11 gesteuerte Gerät konnte einen Elektronenstrahl zur oberflächlichen Behandlung und Röntgenstrahlung zur Behandlung tieferer Gewebeschichten erzeugen. Während seines Einsatzes von 1983 bis 1987 kam es zu sechs bekannt gewordenen, schweren Unfällen durch Strahlungsüberdosen. Das Gerät hatte drei unterschiedliche Betriebsmodi:

- einen Testmodus, mit dessen Hilfe mittels einer Lichtquelle die korrekte Positionierung des Patienten simuliert werden konnte,
- den Elektronenstrahlmodus mit regelbarer Teilchenenergie von 5 bis 25 MeV und einstellbarem Elektronenstrahlstrom, sowie
- den Röntgenmodus mit einer festen Teilchenenergie von 25 MeV und hohem Elektronenstrahlstrom.

Eine drehbare Vorrichtung hatte die Aufgabe, abhängig vom gewählten Betriebsmodus die jeweils notwendige Apparatur in den Strahl einzubringen:

- im Testmodus einen Metallspiegel,
- im Elektronenstrahlmodus Steuermagnete zur Ablenkung des Elektronenstrahls und
- im Röntgenmodus eine Metallvorrichtung, die der Fokussierung des Elektronenstrahls diente.

Im Röntgenmodus musste nach [97] eine gegenüber dem Elektronenstrahlmodus über 100-fache Strahlendosis zur Erreichung der gewünschten Bestrahlungsergebnisse erzeugt werden.

Die Analyse der Unfälle, des Gerätes und der darin verwendeten Software brachte die folgenden Erkenntnisse [77]:

Während das Vorgängergerät Therac-20 noch zusätzliche hardwaretechnische Sicherheitseinrichtungen und mechanische Verriegelungen nutzte, wurden diese Maßnahmen beim Therac-25 durch eine reine Softwarelösung ersetzt – so groß war das Vertrauen in die Software des Geräts.

Die Unfälle wurden dadurch ausgelöst, dass es bei entsprechender Nutzereingabe am Steuermonitor aufgrund unzureichender Synchronisierungsmechanismen zu einer Vermischung vorhergehender und aktueller Betriebsparameter kommen konnte, wodurch sich unzulässige Kombinationen von Bestrahlungsdosen, -zeiten und Einstellung der in den Strahl eingebrachten Apparatur ergaben. Dadurch wurden die betroffenen Patienten massiven Strahlungsüberdosen ausgesetzt, die bei einigen von ihnen sogar zum Tod führten.

1.1.4 Sicherheitslücke Heartbleed

Im April 2014 wurde ein Fehler in der OpenSSL Kryptographiebibliothek bekannt, der zur Folge hatte, dass Millionen Serversysteme umgehend aktualisiert werden mussten [22]. Ein Fehler in einer speziellen Funktion innerhalb der Bibliothek, die für ein zyklisches Lebenszeichen, den sogenannten Herzschlag, engl. „Heartbeat“, genutzt werden sollte, erlaubte es, Speicherinhalte aus dem Speicher des Zielrechners auszulesen, die hochsensible Daten über verschlüsselte Verbindungen enthielten. Vom Namen der eigentlichen Funktion abgeleitet, wurde der Fehler auf den Namen „Heartbleed“ getauft und mit einem entsprechenden Emblem versehen.

Der eigentliche Fehler bestand darin, dass es einem Angreifer möglich war, eine geringe Anzahl an Zusatzbytes an einen Server zu senden und das Zurücksenden von einer deutlich größeren Anzahl an Bytes zurückzufordern. Eine Plausibilitätsprüfung von Anzahl der gesendeten Bytes und der Anzahl der angeforderten Bytes, die zurückgesendet werden sollten, erfolgte nicht. Auf diese Weise konnten Angreifer die bereits erwähnten hochsensiblen Daten von anderen Kommunikationsverbindungen empfangen. Exzellent veranschaulicht werden die Auswirkungen des Fehlers in [134].

Bruce Schneier bezeichnet Heartbleed in [111] als „katastrophal“ und als „11“ auf „einer Skala von 1 bis 10“.

1.2 Der Stand von Wissenschaft und Technik und dessen Nachteile

Der für diese Arbeit relevante Stand von Wissenschaft und Technik soll hier kurz umrissen und seine Nachteile bezogen auf die Erkennung von datenflussbezogenen Fehler- und Angriffsarten vorgestellt werden.

1.2.1 Stand von Wissenschaft und Technik

Als Stand von Wissenschaft und Technik werden in dieser Arbeit die folgenden Architekturen, Architekturmerkmale und Methoden betrachtet:

- die Schutzmechanismen der konventionellen Architekturen x86 im geschützten und 64-Bit-Modus [4, 59], sowie ARM [7, 19, 20],
- auf sicherheitsgerichtete Systeme spezialisierte Prozessoren, die auf konventionellen Architekturen beruhen, wie z. B. der auf der ARM-Architektur basierende TI Hercules [120],
- in Vergessenheit geratene Rechnerarchitekturen wie Datentyp-, Datenstruktur- und Befähigungsarchitekturen [1, 36, 39, 78],
- der Vollständigkeit halber Datenflussarchitekturen [39, 78], da ihr Name eine entsprechende Spezialisierung auf Datenflüsse erahnen lässt,
- die in [125] vorgestellte inhärent sichere Mikroprozessorarchitektur ISMA,
- die Application Data Integrity ADI bzw. das Silicon Secured Memory des SPARC M7 Prozessors,
- arithmetische Kodierung in Form der AN-Kodierung nach Brown [13] und der Erweiterungen zur ANBD-Kodierung durch Forin in [38],
- die auf Signaturen beruhende dynamische Datenflussverifikation DDFV [85] und
- die Datenflussüberwachung in Netzwerken [98, 100, 102] und sicherheitsgerichteten Feldbussen [54].

Die Nachteile der genannten Architekturen und Verfahren werden im folgenden Unterkapitel kurz umrissen.

1.2.2 Nachteile des Stands von Wissenschaft und Technik

Die konventionellen Architekturen x86 und ARM können trotz größter Bemühungen, unter Nutzung komplexester Maßnahmen, einen maximalen Datendurchsatz zu bieten, nur wenige datenflussbezogene Fehler- und Angriffsarten erkennen. Die auf ihnen basierenden, für sicherheitsgerichtete Systeme spezialisierten Prozessoren, wie z. B. der TI Hercules, sind in der Lage, mehr Fehlerarten aufzudecken, basierend auf Redundanz und gewissen Diversitätsarten. Datentyp-, Datenstruktur- und Befähigungsarchitekturen fügen Speicherinhalten Kennungen hinzu, die hardwareverständlich die Inhalte des Speichers beschreiben, womit weitere Fehlerarten erkennbar werden. Datenflussarchitekturen sind zwar auf die Bearbeitung von Datenflüssen spezialisiert, sind aber nicht in der Lage, datenflussbezogene Fehler- und Angriffsarten aufzudecken.

Das Fehlererkennungsmerkmal Application Data Integrity ADI des Oracle SPARC M7 Prozessors, das später in Silicon Secured Memory SSM umbenannt wurde, fügt Blöcken von 64 Byte Größe eine Versionskennung hinzu und nutzt Teile von Zeigern, um dort eine erwartete Version zu hinterlegen. Dadurch können Fehler zwischen der erwarteten und der tatsächlichen Version aufgedeckt werden. Das Verfahren weist mehrere Nachteile auf: Versionen können nur für Datenblöcke, nicht jedoch für einzelne Datenspeicherelemente vergeben werden, die erwarteten Versionen in den Zeigern sind absolute Versionen und müssen durch die Software gesetzt werden und die Versionskennungen werden nur zwischen Prozessor und Speicher verwendet, nicht jedoch kommuniziert.

Die dynamische Datenflussprüfung DDFV erlaubt die signaturbasierte Prüfung der Datenflüsse auf Registerebene, ist aber nicht in der Lage, in diese Prüfungen den oder die Speicher einzubeziehen, geschweige denn systemweite Datenflüsse zu überwachen. Zudem kann eine Abweichung vom vorgesehenen Datenfluss erst am Ende eines Überwachungsblocks und nicht im Moment des Auftretens erkannt werden.

Die arithmetische AN-Kodierung kann zusammen mit den Erweiterungen zur ANBD-Kodierung einige wichtige Datenflussfehler erkennen, ist jedoch nur für bestimmte Operationen und Datentypen geeignet und verursacht erhöhten Laufzeitbedarf. Weiterhin sind die kodierten Datenspeicherelemente nicht mehr ohne weitere Aufbereitung menschenlesbar, was die Fehlersuche erschwert.

Die Datenflussüberwachung in Netzwerken weist nur wenige Fehlererkennungsmerkmale auf. Die Protokolle für sicherheitsgerichtete Feldbusse hingegen nutzen viele Fehlererkennungsmaßnahmen, welche die Erkennung verschiedener Fehlerarten ermöglichen. Allerdings wird nur die erfolgreiche Übertragung der Daten über die verschiedenen Kommunikationsstrecken geprüft, nicht jedoch deren Weiterverarbeitung innerhalb der Datenverarbeitungseinheiten.

Allgemein fehlt eine systemweite, ganzheitliche Betrachtung von Daten, ihrer Eigenschaften und ihrer Wege durch ein System. Selbst wenn bestimmte Dateneigenschaften von der Software auf einer Systemkomponente lokal zur Laufzeit betrachtet oder sogar überwacht werden, so werden diese getrennt von den eigentlichen Daten verwaltet und die Information geht bei der Übertragung der Daten zwischen verschiedenen Softwareprogrammen innerhalb der Systemkomponente, spätestens jedoch bei der Übertragung der Daten an andere Systemkomponenten verloren. Die Inkompatibilität von Operanden bezogen auf deren Datentyp kann durch Datentyp-, Datenstruktur- und Befähigungsarchitekturen erkannt werden. Allerdings wird zur Laufzeit und vor allem über die Grenzen der jeweiligen Komponente hinweg keine Prüfung der Einheiten der Datenwerte vorgenommen. Die Hauptverantwortung für die Fehlererkennung trägt meist die Software, was deren Komplexität und Fehlerwahrscheinlichkeit weiter erhöht.

1.3 Ziel der Arbeit

Das Ziel dieser Arbeit ist die Identifikation relevanter datenflussbezogener Fehler- und Angriffsarten und der Eigenschaften von Daten im Anwendungsbereich sicherheitsgerichteter Echtzeitsysteme, um

- einfache Fehlervermeidungs- und -erkennungsmöglichkeiten auf Hardwareebene,
- einfache Überwachungsmöglichkeiten von Echtzeitbedingungen durch die Hardware selbst,
- ein Optimum an Fehlervermeidung und Erkennbarkeit verbleibender Fehler und
- eine ganzheitliche Betrachtung eines gesamten Systems inklusive aller Hard- und Software

zu erreichen, ohne dem Trend zur unnötigen weiteren Erhöhung der Komplexität der eingesetzten Entwicklungswerkzeuge (z. B. Übersetzer) oder der entstehenden Software zu folgen. Datentyp-, Datenstruktur- und Befähigungsarchitekturen haben sehr leistungsfähige Fehlererkennungsmerkmale hervorgebracht, und dies mit einfachsten Mitteln. Es gilt, deren Merkmale zu nutzen und zu erweitern.

Feustel zitiert Iliffe in [37] dahingehend, dass die Eigenschaften von Datenfeldern untrennbar von den eigentlichen Daten in den Feldern selbst unterzubringen seien, anstatt in den auf die Daten der Felder zugreifenden Algorithmen. Diese Anforderung bezog sich zunächst nur auf die in einem Datenfeld enthaltenen Datentypen und die Anzahl der Elemente innerhalb des Felds. Diese Anforderung soll jedoch für diese Arbeit zu folgendem Entwurfsparadigma erweitert werden:

> *Alle ein Datenspeicherelement beschreibenden Eigenschaften sollen untrennbar mit diesem verknüpft, gespeichert, übertragen, verarbeitet und in einer hardwareverständlichen und -überprüfbaren Form dargestellt werden.*

Basierend auf diesem Ziel lassen sich die meisten datenflussbezogenen Fehler auf einfache Weise durch hardwarebasierte Überprüfungen aufdecken.

1.4 Ergebnisse der Arbeit

Im Zuge dieser Arbeit ist eine neue Prozessorarchitektur entstanden, die aufgrund der umfassenden, hardwareverständlichen Beschreibung von Dateneigenschaften als Datenspezifikationsarchitektur bezeichnet wird. Die Beiträge der Arbeit zum Stand von Wissenschaft und Technik sind dabei:

- die Identifikation von insgesamt 20 datenflussbezogenen Fehler- und Angriffsarten,
- eine umfassende Sammlung der Eigenschaften von Daten in sicherheitsgerichteten Echtzeitsystemen und
- die Vorstellung der auf Basis dieser Ergebnisse entwickelten Datenspezifikationsarchitektur DSA, welche die identifizierten Dateneigenschaften in Form von hardwareverständlichen Kennungen weit umfangreicher darstellt, als dies bei bisherigen Architekturen der Fall war.

Die Neuheiten der Fehlererkennungsmerkmale der Datenspezifikationsarchitektur DSA gegenüber dem Stand von Wissenschaft und Technik sind:

- die Definition von Messwertdatentypen in Form eines Werteintervalls zur Darstellung fehlerbehaftete Werte, um die Fortpflanzung dieser Fehler bei der Werteverarbeitung durch Intervallarithmetik verfolgen und eventuelle Genauigkeitsprobleme zu erkennen, zusammen mit speziellen Befehlen zur Prüfung der Genauigkeit,

- eine Wertebereichskennung [131], die es erlaubt, einerseits eine Plausibilitätsprüfung beim Lesen eines Datenspeicherelements durchzuführen, indem geprüft wird, ob der Datenwert innerhalb des spezifizierten Wertebereichs liegt, andererseits beim Schreiben in ein Datenspeicherelement, das eine Wertebereichsvorgabe enthält, um eine sofortige Prüfung des zu schreibenden Datenwerts durch die Hardware zu ermöglichen,

- die Erweiterung der von Datentyparchitekturen bekannten Datentypkennungen [127] um von den nativen Datentypen abgeleitete Datentypen mit Spezifikation der gestatteten Operationen, deren Eigenschaften von der Hardware überwacht werden,

- eine Einheitenkennung [126], die die Einheit des Datenwerts eines Datenspeicherelements in Form von Potenzen der sieben SI-Basiseinheiten beschreibt und umfassende Kompatibilitätsprüfungen bei der Nutzung von Operanden gestattet; bei Multiplikationen und Divisionen wird die Einheit des Ergebnisses durch die Hardware automatisch auf Basis der Potenzgesetze berechnet,

- eine Verarbeitungswegkennung [130], die beschreibt, wer die Daten erzeugt hat, welche Stationen die Daten auf ihrem Weg von Datenquelle bis -senke verarbeiten dürfen und wer die Daten schlussendlich entgegennehmen darf,

- eine Zeitschrittkennung [132], die – ähnlich einer Sequenznummer – beschreibt, zu welchem diskreten Zeitpunkt der betroffene Datenwert generiert worden ist, zusammen mit einer Erweiterung des Befehlssatzes um eine Kennung, die die erwartete temporale Beziehung der Operanden einer Operation beschreibt, welche durch die Hardware überprüft wird,

- eine Fristkennung [128], die es gestattet, den Gültigkeitszeitraum der Daten einzugrenzen und die Verwendung von Daten nach Ablauf der Frist als Fehler zu erkennen,

- eine Zykluszeitkennung [133], die beschreibt, innerhalb welcher zeitlicher Grenzen eine aktualisierte Version eines Datums erwartet wird, wodurch das Ausbleiben einer Werteaktualisierung ebenso wie zu frühe Aktualisierung von Werten als Fehler erkannt werden können, und

- eine Signaturkennung [129] für besonders anspruchsvolle Anwendungen wie Chipkarten, bei denen der hohe Aufwand der kryptographischen Signatur jedes einzelnen Datenspeicherelements zur Sicherstellung der Schutzziele Integrität und Authentizität zu rechtfertigen ist,

- Datenportale in Form von Dateneingangs- und -ausgangsportalen, die es ermöglichen, Daten mit Einbeziehung der Adresse in die Integritätsprüfung bzw. Signatur zwischen Systemkomponenten zu übertragen; bei Nutzung einer kryptographischen Signatur der Daten übernehmen die Dateneingangsportale zusätzlich die Aufgabe der Prüfung der Signatur des Absenders und der Umsignierung mit dem eigenen geheimen Schlüssel und

- die Vorstellung einer Realisierungsmöglichkeit der Merkmale einer Datenspezifikationsarchitektur DSA in Datenflussarchitekturen durch Erweiterung der Verarbeitungseinheiten.

Mit Hilfe dieser und weiterer Merkmale, die dem Stand von Wissenschaft und Technik entsprechen, ist es der Datenspezifikationsarchitektur möglich, alle identifizierten Fehler- und Angriffsarten zur Laufzeit zu erkennen und entsprechende Fehlerbehandlungsmaßnahmen einzuleiten, ohne dabei die Komplexität und den Laufzeitbedarf der Software signifikant zu erhöhen. Durch den Zwang, sich zum Zeitpunkt der Spezifikation bzw. des Entwurfs hinreichend tief mit den Eigenschaften der in einem System entstehenden und verarbeiteten Daten auseinanderzusetzen, wird ein hohes Maß an Fehlervermeidung erreicht.

1.5 Aufbau der Arbeit

Zunächst erfolgt in Kapitel 2 eine detaillierte Vorstellung von Fehlern und deren Entstehungsmechanismen, zusammen mit der Vorstellung von 20 identifizierten datenflussbezogenen Fehler- und Angriffsarten, anhand derer der Stand von Wissenschaft und Technik, sowie die Ergebnisse dieser Arbeit bewertet werden. Einige Beispiele für die Auswirkungen derartiger Fehler wurden bereits in Kapitel 1.1 vorgestellt.

In Kapitel 3 werden die für diese Arbeit relevanten existierenden Architekturen und Fehlererkennungsverfahren detailliert vorgestellt und auf Basis der datenflussbezogenen Fehler- und Angriffsarten evaluiert. Jede einzelne Fehlerart wird dabei in einer Tabelle dahingehend bewertet, ob und in welchem Umfang sie durch die Merkmale und Verfahren des jeweiligen Stands von Wissenschaft und Technik aufzudecken ist. Dabei wird die Erkennbarkeit wie folgt bewertet:

- „nein“ bedeutet, dass die betroffene Fehler- bzw. Angriffsart nicht erkannt werden kann,
- „begrenzt“ bedeutet, dass die betroffene Fehler- bzw. Angriffsart nur erkannt werden kann, wenn besondere Bedingungen erfüllt werden, die im Text zur Tabelle näher beschrieben werden,
- „(ja)“ bedeutet, dass die betroffene Fehler- bzw. Angriffsart zwar häufig erkannt werden kann, aber Einschränkungen existieren, die die Leistungsfähigkeit der Erkennung oder Verarbeitung der Daten oder die Wahrscheinlichkeit der Erkennung beschränken, wobei die Einschränkungen ebenfalls im Text zur Tabelle beschrieben werden, während
- „ja“ bedeutet, dass die Fehler- bzw. Angriffsart ohne Einschränkungen durch Merkmale oder Verfahren des Stands von Wissenschaft und Technik aufgedeckt werden können.

Im Anschluss werden in Kapitel 4 die Eigenschaften von Daten in sicherheitsgerichten Echtzeitsystemen gesammelt und eine Datenspezifikationsarchitektur vorgestellt, die den Daten diese Eigenschaften in hardwarelesbaren Kennungen hinzufügt. Die einzelnen Merkmale der Architektur werden detailliert vorgestellt und – analog zum Stand von Wissenschaft und Technik – in Bezug auf die Erkennbarkeit der datenflussbezogenen Fehler- und Angriffsarten evaluiert.

Die Evaluation der Datenspezifikationsarchitektur erfolgt in Kapitel 5, wobei die folgenden Gesichtspunkte getrennt betrachtet werden:

- die Art der Abbildung von Daten in einer Datenspezifikationsarchitektur im Vergleich zu existierenden Lösungen,
- die Einordnung der Architekturart bezogen auf Datentyp-, Datenstruktur- und Befähigungsarchitekturen,
- die Erkennbarkeit der in Kapitel 2 identifizierten datenflussbezogenen Fehler- und Angriffsarten,

- die Erkennbarkeit der Fehler, die zu den Auswirkungen der in Kapitel 1 vorgestellten Fehlerbeispiele aus der Praxis führten, und
- die Speicherausnutzung der Architektur.

Den Abschluss bilden eine Zusammenfassung der Beiträge dieser Arbeit und eine Übersicht über die Weiterführungsmöglichkeiten in Kapitel 6.

1.6 Darstellung von Zahlen und Speichergrößen in der Arbeit

In dieser Arbeit werden Zahlen in unterschiedlichen Darstellungsformen verwendet. Dezimalzahlen werden dabei ohne zusätzliche Markierungen dargestellt, während Zahlen in hexadezimaler Schreibweise – der Darstellungsform der Hochsprache C folgend – mit dem Präfix „0x“ versehen werden. Bei binären Darstellungen kommt nach dem Vorbild des Microsoft Assemblers MASM das Postfix „b“ zum Einsatz. Als Beispiel sei hier die Zahl 23 in dezimaler, hexadezimaler und binärer Darstellung gezeigt:

$$23 = 0\text{x}17 = 10111\text{b}$$

Speichergrößen werden in dieser Arbeit – der Norm IEC 80000-13:2008 [61] folgend – mit den IEC-Präfixen zur Bezeichnung von Zahlen zur Basis 2 dargestellt:

$$1\,\text{KiB} = 2^{10}\,\text{Byte} = 1024^{1}\,\text{Byte} = 1024\,\text{Byte}$$
$$1\,\text{MiB} = 2^{20}\,\text{Byte} = 1024^{2}\,\text{Byte} = 1048576\,\text{Byte}$$
$$1\,\text{GiB} = 2^{30}\,\text{Byte} = 1024^{3}\,\text{Byte} = 1073741824\,\text{Byte}$$

2 Fehlerarten, -ursachen, -auswirkungen und -behandlung

In diesem Kapitel werden zunächst die Kategorisierung von Fehlern in Hard- und Software, deren Ursachen und typische Fehlerdichten in offener und proprietärer Software vorgestellt. Anschließend werden typische datenflussbezogene Fehler- und Angriffsarten identifiziert, die im Betrieb eines sicherheitsgerichteten Echtzeitsystems auftreten können. Den Abschluss des Kapitels bilden die Beschreibung möglicher Auswirkungen von Fehlern, sowie die Erkennung und Behandlung auftretender Fehler.

2.1 Fehlerkategorien

Hardwarefehler werden nach [68, 112] anhand der Art und Dauer ihres Auftretens wie folgt kategorisiert:

- Permanente Fehler, engl. „permanent errors“ bzw. „hard errors“, sind Fehler, die nach ihrem erstmaligen Auftreten bestehen bleiben, bzw. bereits von Anfang an vorhanden sind. Dies sind meist Beschädigungen der Hardware, z. B. durch Elektromigration, aber auch Spezifikations-, Entwurfs- und Implementierungsfehler.

- Intermittierende Fehler, engl. „intermittent errors“, sind Fehler, die in unregelmäßigen Abständen auftreten. Ursachen für derartige Fehler kann eine instabile oder grenzwertig ausgelegte Hardware sein, die auf bestimmte Betriebsbedingungen fehlerhaft reagiert.

- Transiente Fehler, engl „transient errors“ bzw. „soft errors“, sind Fehler, die zufällig verteilt für einen bestimmten Zeitraum auftreten und z. B. durch Strahlungseinflüsse ausgelöst werden [8, 90].

Im Bereich der Software existieren keine transienten oder intermittierenden Fehler, sie unterliegt keiner Alterung oder anderen von der Hardware bekannten Fehlermechanismen. Daher sind alle Fehler, die sie enthält, permanenter Natur [44].

2.2 Fehlerquellen in Soft- und Hardware

In Abbildung 2.1, die aus [44] stammt, wird gezeigt, welche Fehlerursachen zu Fehlern führen können, die sich dann in Ausgaben des Systems auswirken. Der Mensch kann dabei neben Bedienungsfehlern vor allem Software- und Hardwarefehler durch Fehler in den Entwicklungsphasen Spezifikation, Entwurf und Implementierung verursachen. Weiterhin hat der Mensch Einfluss auf die Betriebsbedingungen des Systems. Diese können zusammen mit den Naturgesetzen Prozesse im System selbst oder dessen Umwelt bewirken. In der Folge kann es zu Bauelementausfällen oder Störsignalen kommen, die sich in Form von Hardwarefehlern manifestieren.

Fehler werden weder in der Software, noch in der Hardware eines Systems jemals komplett vermieden werden können, da die Fehlerquellen so vielfältig sind. Da diese in Abbildung 2.1 nicht in vollem Umfang ersichtlich sind, werden sie in Abbildung 2.2 nochmals detaillierter dargestellt.

Im oberen Teil der Abbildung sind die typischen Entwicklungsphasen einer Software zu sehen, während der untere Teil die Entwicklung der Hardware darstellt. Die Mitte der Abbildung bilden die Programmierung und Konfiguration, sowie der Betrieb des aus Hard- und Software bestehenden Systems. Die möglichen Fehlerquellen werden durch entsprechende Symbole bzw. Zeichenketten gekennzeichnet. Nicht alle im Bild enthaltenen Zwischenschritte bzw. Werkzeuge werden im jeweiligen Projekt bei dessen Realisierung zum Einsatz kommen.

Zunächst soll die Entwicklung der Software detaillierter betrachtet werden: während den Entwicklungsphasen Spezifikation, Entwurf und Implementierung werden Fehler allein durch den Menschen verursacht, angedeutet durch ein Menschensymbol. Bei der anschließenden Übersetzung des entstandenen Quellcodes kommt wieder der Faktor Mensch ins Spiel, allerdings nur indirekt über die Fehler, die durch die gleiche Fehlerkette in den Übersetzer eingebracht wurden, gekennzeichnet durch ein Menschensymbol und die Zeichenkette „SEIÜBL“, die für die Fehlerkette Spezifikation - Entwurf - Implementierung - Übersetzer - Binder - Lader steht. Das Gleiche gilt für den anschließenden Einsatz des Binders, engl. „Linker“, der das ausführbare Programm erzeugt und dabei ggf. Betriebssystem-, Bibliotheks- oder Laufzeitkomponenten in dieses einbindet. Zu beachten ist, dass Fehler der Hardware, auf der

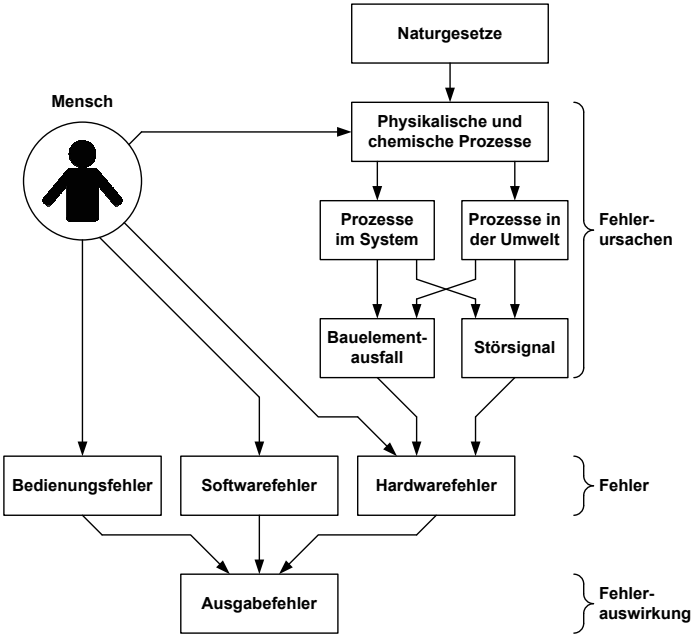


Abbildung 2.1: Fehlerursachen nach [44]

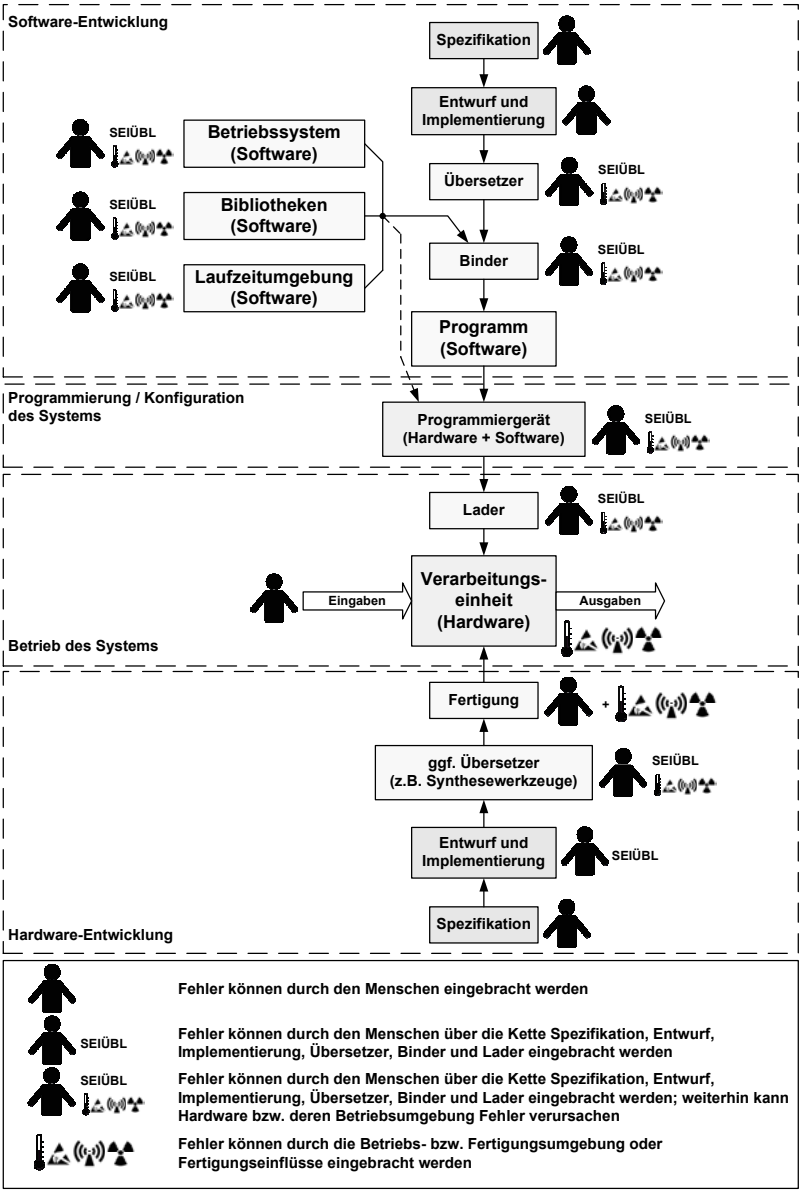


Abbildung 2.2: Kette von Fehlerquellen

das jeweilige Werkzeug zum Einsatz kommt, sich auch in Fehlern der entstehenden Software auswirken können, was durch die Symbole unterhalb der Zeichenkette angedeutet wird. Wie in [44] angedeutet, können Fehler, die durch die ein Entwicklungswerkzeug ausführende Hardwareplattform in die erzeugte Software eingebracht werden, durch eine mehrfach hintereinander ausgeführte Übersetzung mit Vergleich der Ergebnisse aufgedeckt werden.

Ähnlich sieht es bei der Entwicklung der Hardware aus: je nachdem, ob, und wenn ja, welche Entwicklungswerkzeuge zum Einsatz kommen, können Fehler direkt oder indirekt vom Menschen in der jeweiligen Entwicklungsphase verursacht werden. Denn auch Hardwareentwurfs- und -synthesewerkzeuge sind Softwareprogramme von oft sehr hoher Komplexität. Neben menschlichen Einflüssen können in der Fertigung der Hardware weitere Fehler durch Fertigungs- und Umwelteinflüsse entstehen, angedeutet durch die verschiedenen Symbole.

Die Software wird ggf. im Zuge eines Programmier- oder Konfigurationsvorgangs unter Nutzung eines Programmiergeräts auf der Zielhardware gespeichert. Dieses Programmiergerät ist wiederum selbst ein System bestehend aus Hardware und ggf. Software und kann somit ebenfalls Fehler enthalten, die eine fehlerhafte Programmierung zur Folge haben können. Zwar kann ein Verifikationsschritt, bei dem das Programm nach der Programmierung aus dem Speicher ausgelesen und mit dem originalen Programm verglichen wird, grobe Programmierfehler aufdecken. Was dabei allerdings nicht erkannt werden kann, sind nicht hinreichend programmierte Speicherzellen, die ihre Programmierung kurz- oder mittelfristig verlieren.

Auf der entstandenen Hardware wird das erzeugte Programm zur Ausführung gebracht. Gegebenenfalls kommt hierbei eine weitere Software zum Einsatz: der Lader, engl. „Loader“, der das Programm beispielsweise aus einem Festwertspeicher in einen Ausführungsspeicher überträgt und womöglich notwendige Abhängigkeiten zu Laufzeitbibliotheken auflöst. Diese letzte verbleibende Softwarefehlerquelle ist auch der Grund dafür, dass bei der diversitären Rückwärtsanalyse statt des übersetzten und gebundenen Programms ein Speicherabbild analysiert wird, um eventuelle Fehler des Laders ebenfalls erkennen zu können [70]. Neben dem eigentlichen Programm können auch weitere Softwarepakete auf der Hardware ausgeführt werden, wie beispielsweise bestimmte Laufzeitumgebungen der eingesetzten Programmiersprache oder ein Betriebssystem. Beide Beispiele sind ebenfalls der bereits erwähnten Fehlerkette SEIÜBL unterworfen. Eine weitere, häufig unterschätzte Fehlerquelle kann die gegenseitige Beeinflussung von verschiedenen Softwarepaketen sein, die auf der Hardware zum Ablauf gebracht werden, vor allem dann, wenn diese um nur begrenzt verfügbare Ressourcen wie Speicher und Prozessor konkurrieren.

Bei der Ausführung des Programms unterliegt dieses keinem Verschleiß – wohl aber die Hardware, die es interpretiert. Umwelteinflüsse wie Temperatur, elektrostatische und elektromagnetische Einflüsse und verschiedene Arten von Strahlung können die Inhalte von Speichern und Registern der Hardware verändern oder die Hardware auch dauerhaft beschädigen. Die immer immer weiter reduzierten Strukturbreiten in integrierten Schaltkreisen machen diese empfindlicher gegenüber derartigen Einwirkungen von außen [8, 90].

Schlussendlich kann der Mensch durch Eingabe- oder Bedienungsfehler direkt Ausgabefehler verursachen, was idealerweise vom System erkannt werden sollte, ohne gefährliche Ausgaben zu verursachen.

Die Übersetzer und die weiteren am Entwicklungsprozess von Software beteiligten Werkzeuge werden von Programmierern oft als unfehlbar angenommen. Dabei ist meist das Gegenteil der Fall, was an folgendem Beispiel verdeutlicht werden soll:

Die GNU Compiler Collection GCC – eine quelloffene Übersetzersammlung, die besonders im Linux-Umfeld zum Einsatz kommt – bestand 2015 aus über 14 Millionen Codezeilen [74]. Bei einer Überprüfung der Übersetzersammlung wurde nach [18] durch statische Codeanalyse die sehr geringe Fehlerquote von 0,202 Fehlern pro 1000 Zeilen Code ermittelt. Bei ungefähr 700000 überprüften Zeilen Code konnten 140 Fehler aufgedeckt werden – die hoffentlich danach auch behoben wurden. Die tatsächliche Anzahl an Fehlern dürfte natürlich noch höher liegen, da auch das Prüfungsprogramm nur bestimmten Teil der Fehler identifizieren konnte.

Dabei ist zu beachten: selbst ein einziger Fehler kann sich in einem Übersetzer oder Binder derart auswirken, dass gefährliche Fehler im entstehenden Programm erzeugt werden. Auch kleine Fehler in der Software können gravierendste Auswirkungen haben, was in [44] als „Unstetigkeit von Software“ bezeichnet wird.

2.3 Fehlerdichte in Software

Regelmäßig wird von der Firma Coverity durch Auswertung der Ergebnisse des hauseigenen Werkzeugs zur statischen Codeanalyse ermittelt, wie hoch die durchschnittliche Fehlerdichte der untersuchten Software ist. Für das Jahr 2013 ergaben sich dabei die in Tabelle 2.1 gezeigten Anzahlen von Fehlern pro 1000 Zeilen Code für Programme, die in den Hochsprachen C oder C++ erstellt wurden [24].

Tabelle 2.1: Fehlerdichte in Fehlern pro 1000 Zeilen Code [24]

Softwareart	Durchschnittliche Fehlerdichte
Quelloffene Software	0,59
Proprietäre Software	0,72

Anhand dieser Ergebnisse lässt sich schlussfolgern, dass quelloffene Software mit 0,59 Fehlern pro tausend Zeilen Quellcode – zumindest innerhalb der zugrundeliegenden Stichprobenmenge – im Vergleich zu proprietärer Software mit 0,72 Fehlern pro tausend Zeilen Quellcode als signifikant fehlerärmer angesehen werden kann. Hier scheint das Entwicklungsparadigma der Offenlegung des Quellcodes Früchte zu tragen.

2.4 Datenflussbezogene Fehler- und Angriffsarten

Zur Evaluation des Stands von Wissenschaft und Technik, der in Kapitel 3 vorgestellt werden wird, sollen typische, den Datenfluss in sicherheitsgerichteten Echtzeitsystemen betreffende Fehlerbilder dienen, die es zu erkennen gilt. Nach [118] betreffen immerhin 80 - 90 % aller Fehler den Datenfluss und nur die verbleibenden 10 - 20 % den Kontrollfluss innerhalb eines Systems. Die identifizierten Fehlerarten werden in die folgenden Kategorien eingeteilt:

- Inkompatibilität von Operanden
- Wertebereichsverletzungen und Genauigkeitsprobleme
- Fehlerhafte Operationen
- Verletzung von Echtzeitbedingungen
- Allgemeine Datenflussfehler
- Datenverfälschung durch Fehler oder Störungen
- Fehlerhafter Zugriff auf Daten

Da technische Prozesse auch in immer zunehmendem Maße das Ziel von Hackerangriffen werden, wie z. B. der Stuxnet-Wurm [72] gezeigt hat, sollen mögliche

- Hackerangriffe

die Betrachtung der datenflussbezogenen Problemfälle vervollständigen.

2.4.1 Inkompatibilität von Operanden

Eine typische Fehlerquelle bei der Softwareentwicklung – besonders bei Zusammenschluss mehrerer Systemkomponenten – ist die Inkompatibilität von Operanden bezogen auf die Art der Repräsentation der Datenwerte innerhalb der Datenspeicherelemente. In [103] wird darum gefordert, alle Funktionsparameter und Eingaben an Schnittstellen daraufhin zu prüfen, ob die übergebenen Datenwerte den erwarteten Datentyp besitzen. Entsprechende Fehler werden in [118] auf Systemebene als „Konfigurationsfehler“ bezeichnet. Auch der Versuch, Code als Daten zu interpretieren, ist als Inkompatibilität der Operanden zu werten.

Weiterhin kann es vorkommen, dass Operanden zur Verarbeitung herangezogen werden, die nicht zueinander passende Einheiten besitzen. Die möglichen Auswirkungen eines solchen Fehlers wurden anhand des Verlusts des Mars Climate Orbiter in Kapitel 1.1.2 vorgestellt [81].

2.4.2 Wertebereichsverletzungen und Genauigkeitsprobleme

In vielen Applikationen dürfen Datenwerte innerhalb eines Datenspeicherelements nicht im gesamten durch den zugrunde liegenden Datentyp definierten Wertebereich liegen, sondern nur in einem Teilbereich. In [103] wird daher verlangt, dass alle Eingaben – in Form von Funktionsparametern oder Eingabedaten auf Schnittstellen – darauf geprüft werden, ob sie innerhalb eines erwarteten, gültigen Wertebereichs liegen. Der Versuch, außerhalb des zulässigen Bereichs liegende Datenwerte in ein Datenspeicherelement zu schreiben, wird in [118] als „Wertzuweisungsfehler“ bezeichnet. Zum Fehlerbild der Wertbereichsverletzungen sind auch Unter- und Überläufe bei der Durchführung arithmetischer Operationen zu rechnen, die in [86] auf Platz 24 der 25 gefährlichsten Softwarefehler geführt werden.

Die möglichen Auswirkungen derartiger Fehler wurden bei der Selbstzerstörung der Ariane 5 (siehe Kapitel 1.1.1) ersichtlich, die die schlussendliche Konsequenz einer Reihe von Fehlern war und mit einem Überlauf aufgrund zu großer Eingabewerte begann.

In technischen Prozessen werden Prozessgrößen durch Sensoren erfasst und umgewandelt. Die so gewonnenen Messwerte unterliegen nach [117] stets einer gewissen Messunsicherheit. Liegt die Genauigkeit von Messwerten oder deren Verarbeitungsergebnissen nicht innerhalb spezifizierter Grenzen, können Probleme auftreten.

Die folgenden Fehlerszenarien können zu Genauigkeitsproblemen führen:

- Sensoren erzeugen ggf. in bestimmten Abschnitten ihres Messbereichs Messwerte mit unterschiedlicher Genauigkeit. Wird dies beim Systementwurf nicht entsprechend berücksichtigt, können ungenaue Messwerte zu Fehlern führen.
- Durch die Verarbeitung fehlerbehafteter Datenwerte ergeben sich durch Fehlerfortpflanzung Ergebnisse mit nicht tolerierbarer Ungenauigkeit.
- In einer Überlastungssituation können im Sinne der allmählichen Leistungsabsenkung [45] alternative, ungenauere Ergebnisse liefernde Berechnungen angewandt werden. So könnten aufgrund einer massiven Überlastsituation mehrere hintereinander ausgeführte Alternativberechnungen Ergebnisse mit inakzeptabler Genauigkeit produzieren.
- Bei einem Geräteaustausch könnte ein Ersatzgerät ungenauere Messwerte erzeugen.
- Bei Umrüstungen könnten Sensoren in Messbereichen benutzt werden, in denen sie größere Grenzfehler aufweisen, als für die Werteverarbeitung zulässig wäre. Ein Beispiel für eine derartige Umrüstung oder Weiterverwendung in anderen Systemen war die Nutzung bestimmter Systemkomponenten aus der Ariane 4 in der Ariane 5, wodurch es zu Wertebereichsverletzungen kam [79]. Als Fehlerfall wären hier auch Genauigkeitsproblematiken denkbar.
- Ein Sensor könnte durch Alterung im Verlauf seiner Nutzung immer ungenauere Messwerte erzeugen.

2.4.3 Fehlerhafte Operationen

Bei der Verarbeitung der Daten in den arithmetisch-logischen Einheiten ALE der Datenverarbeitungseinheiten kann es zu Berechnungsfehlern kommen. Diese müssen rechtzeitig erkannt werden, bevor sie zu gefährlichen Ausgaben führen. Forin beschreibt dabei in [38] die folgenden möglichen Fehlerarten:

- Die Operation wird mit einem oder mehreren falschen Operanden durchgeführt, wodurch ein fehlerhaftes Ergebnis entsteht.
- Die Operation wird mit den korrekten Operanden, aber einem falschen Operator durchgeführt, wodurch ein – bezogen auf den falschen Operator korrektes, jedoch auf die durchzuführende Operation – falsches Ergebnis entsteht.

- Die Berechnung wird mit den korrekten Operanden und dem korrekten Operator durchgeführt, liefert jedoch ein inkorrektes Ergebnis.

2.4.4 Verletzung von Echtzeitbedingungen

Eine wesentliches Ziel bei der Datenverarbeitung in Echtzeitsystemen ist die Rechtzeitigkeit, also die Reaktion auf Ereignisse innerhalb eines fest vorgegebenen Zeitraums. Werden Daten nicht innerhalb dieses Zeitraums verarbeitet, so werden sie und alle aus ihnen abgeleiteten Ergebnisse nutzlos und ggf. kann ihre Verwendung zu gefährlichen Ausgaben führen. Idealerweise wird ein Echtzeitsystem zeitgesteuert realisiert [52, 53], Daten werden daher zyklisch erfasst und verarbeitet. Entsprechend sollen sie in konstanten Zeitabständen innerhalb der im System vorhandenen Datenverarbeitungseinheiten verarbeitet werden. Bleiben diese Daten aus, kann es zu Fehlfunktionen des Systems kommen, besonders dann, wenn statt der aktualisierten Daten unerkannt alte Datenstände verwendet werden.

In [27] werden die folgenden Zeitbedingungen innerhalb von Echtzeitsystemen identifiziert:

- eine maximale Bearbeitungszeit nach Auftreten eines Ereignisses,
- eine minimale Zeit zwischen zwei Ereignissen und
- eine maximale Zeit zwischen zwei Ereignissen.

Während meist nur die maximale Reaktionszeit auf ein Ereignis im Fokus der Entwickler eines Echtzeitsystems steht, kann ein fehlerhafter Kommunikationsteilnehmer, der zu häufig Daten versendet, schwierig zu beherrschende Überlastsituationen hervorrufen und ggf. Kommunikationsverbindungen blockieren.

2.4.5 Allgemeine Datenflussfehler

Unter dem Begriff „Allgemeine Datenflussfehler“ sollen einige Fehlerarten zusammengefasst werden, die den Weg der Daten durch das System und innerhalb der Quellen, Datenverarbeitungseinheiten und Senken betreffen.

2.4.5.1 Verlorengegangene Datenaktualisierungen

Wenn Schreibzugriffe auf Datenspeicherelemente fehlschlagen, z. B. durch Adressierungsfehler, so enthalten diese nach dem fehlerhaften Schreibzugriff nicht den erwarteten Datenstand. Diese Fehlerart wird von Forin als „lost update“, also „verlorene Aktualisierung“ bezeichnet [38].

2.4.5.2 Synchronisationsfehler und unvollständige Datenübertragungen

Bei nebenläufigem Zugriff auf Daten besteht bei unzureichender Synchronisation die Gefahr von Dateninkonsistenzen. Diese Art von Fehler wird in [118] als „Serialisierungsfehler“ bezeichnet. Ein gutes Beispiel für die Auswirkungen eines solchen Fehlers stellt das in Kapitel 1.1.3 vorgestellte medizinische Bestrahlungsgerät Therac-25 dar. Durch Synchronisationsfehler kam es hierbei zu inkonsistenten Behandlungsparametern [77].

Ähnliche Auswirkungen haben unvollständige Datenübertragungen, bei denen ein Teil eines Puffers mit neuen Daten gefüllt wird, die restlichen Datenspeicherelemente jedoch aufgrund eines Fehlers ihren letzten Stand behalten.

Da beide Fehler zu inkonsistenten Datenständen führen, deren Interpretation gefährliche Ausgaben verursachen kann, werden sie zusammengefasst.

2.4.5.3 Pufferunter- oder -überläufe

Die Unter- oder Überschreitung von Feld- und Puffergrenzen stellt einen häufigen Datenflussfehler [66, 86, 103] dar, der unbeabsichtigte und schwer aufzudeckende Veränderungen von Speicherinhalten mit schwerwiegenden Folgen nach sich ziehen kann und daher in [86] auf Platz 3 der 25 gefährlichsten Programmierfehler eingeordnet wird. Die sich daraus ergebenden Probleme können teilweise so ausgenutzt werden, dass Sicherheitslücken entstehen, die die informationstechnischen Schutzziele gefährden. Ein gutes Beispiel einer solchen Sicherheitslücke ist der Heartbleed-Fehler [22], der in Kapitel 1.1.4 detailliert vorgestellt wurde.

2.4.5.4 Fehlerhafter Datenfluss

Nicht immer werden Daten in einem System dem Pfad folgen, der in der Spezifikation für sie vorgesehen war. Ein gutes Beispiel für einen fehlerhaften Datenfluss ist der

Fehler der Ariane 5, der bereits in Kapitel 1.1.1 vorgestellt wurde: Diagnosedaten wurden durch den Bordcomputer als Flugdaten interpretiert, die eigentlich durch ein entsprechendes Fehlerbehandlungsprogramm bearbeitet werden sollten [79].

2.4.5.5 Duplizierte Daten

Durch Fehler im Datenfluss können einem System identische Datenstände mehr als nur einmalig zur Verarbeitung angeboten werden. Diese Art von Fehler ist vor allem aus dem Bereich der Kommunikationstechnik bekannt und wird z. B. in der Norm IEC 61784-3 [54] für sicherheitsgerichtete Feldbuskommunikation als Fehlerart identifiziert.

2.4.6 Datenverfälschung durch Fehler oder Störungen

Daten können auf ihrem Weg durch das System, aber auch innerhalb der Datenquellen, Datenverarbeitungseinheiten und Senken durch auftretende transiente oder permanente Fehler in Hardware, durch Störungen und auch Softwarefehler unbeabsichtigt verändert werden. Daher ist es sinnvoll, derartige Fehler durch entsprechende Fehlererkennungsmaßnahmen aufzudecken. In der IEC 61508 [51–53] werden entsprechende Fehlererkennungsmaßnahmen in Form von polynomialen Codes oder Hamming-Kodierung gefordert. Diese Fehlerart ist streng von gezielt verfälschten Daten durch Angreifer zu unterscheiden, bei der die Daten gezielt verändert werden. Eine entsprechende Angriffsart wird in Kapitel 2.4.8 vorgestellt.

2.4.7 Fehlerhafter Zugriff auf Daten

Bei einer weiteren Art von Fehler wird auf Daten nicht in der Form zugegriffen, die für sie vorgesehen war. Dazu zählen

- Zugriffe auf die falschen Daten, die nicht für die Verwendung durch die aktuell ablaufende Programminstanz vorgesehen waren,
- fehlerhafte Schreibzugriffe auf Daten, die zwar zur aktuell ablaufenden Programminstanz gehören, aber nicht hätten geschrieben werden dürfen, in [118] als „Zuweisungsfehler bei schreibgeschützten Daten“ bezeichnet, und
- lesende Zugriffe auf Daten, die keine nutzbaren Werte enthalten, also z. B. nicht initialisierte Variablen, was nach [118] einer der häufigsten Datenflussfehler ist.

Die beiden ersten Fehlerarten werden unter dem Begriff „Fehlerhafter Datenzugriff“ zusammengefasst, die dritte wird von diesen getrennt betrachtet.

2.4.8 Hackerangriffe

Im Juni 2010 wurde der Stuxnet-Wurm [72] entdeckt, der Siemens-Steuerungen befiel und vermutlich das iranische Atomprogramm angreifen sollte, um die Urananreichungszentrifugen durch falsche Betriebsparameter zu zerstören. Spätestens seit diesem Zeitpunkt ist klar, dass technische Prozesse Angriffsziele von Hackern werden können, z. B. im Zuge eines kriegerischen Akts. Daher gilt es, verschiedene Angriffsarten zu identifizieren und technische Prozesse dagegen zu schützen.

Das Hauptaugenmerk liegt auf gesicherten Kommunikationsverbindungen und der damit verbundenen Datenübertragungssoftware unter Nutzung sicherer Authentifizierungs- und Verschlüsselungsmaßnahmen. Diese sind nicht Teil dieser Arbeit und wurden z. B. in [109] bereits umfassend betrachtet.

Im Zuge dieser Arbeit sollen jedoch zwei Angriffsarten betrachtet werden, die trotz gesicherter Kommunikationsverbindungen für Angreifer interessant sein können: die gezielte Verfälschung von Daten und Wiedereinspielungsattacken.

2.4.8.1 Gezielte Verfälschung von Daten

Im Gegensatz zur Verfälschung von Daten durch auftretende Fehler im System kann ein Angreifer versuchen, Daten gezielt zu manipulieren, um bestimmte Reaktionen des Systems auf diese Daten zu provozieren. Darunter sind auch manipulierte Nachrichten zu verstehen, bei denen ein Angreifer versucht, sich als gültiger Kommunikationspartner auszugeben. Diese Art von Angriffen kann unterschiedlichen Zielen dienen:

- der Ausspähung von Daten,
- dem Stören von Betriebsabläufen bis hin zur
- Auslösung von Fehlfunktionen.

Dass diese Art von Angriff eine reale Gefahr für die Anlage, bzw. ein ernstzunehmendes Ziel von Angreifern darstellt, wurde durch den bereits erwähnten Stuxnet-Wurm hinreichend bewiesen [72].

2.4.8.2 Wiedereinspielungsattacken

Eine weitere zu berücksichtigende Angriffsart sind Wiedereinspielungsattacken, bei denen ein Angreifer gültigen, ggf. verschlüsselten oder signierten Datenverkehr aufzeichnet, um ihn zu gegebener Zeit den Datenverarbeitungseinheiten als aktuelle Daten zur Verarbeitung „anzubieten“. Dies erfolgt mit dem Ziel, den technischen Prozess zu stören oder zu beeinflussen.

Ein entsprechendes Szenario könnte z. B. ein chemischer Prozess sein, bei dem eine Temperatur erfasst und zu einer Stellgröße für eine Heizung verarbeitet wird. Würden hier gültige Datenpakete des Sensors mit niedriger Temperatur aufgezeichnet und zu einem späteren Zeitpunkt an die Datenverarbeitungseinheit als aktuelle Sensordaten anstelle der echten Temperaturmesswerte gesendet, so könnte eine Überhitzung des Prozesses provoziert werden.

2.4.9 Zusammenfassung der identifizierten datenflussbezogenen Fehler- und Angriffsarten

Die 20 identifizierten datenflussbezogenen Fehler- und Angriffsarten werden in Tabelle 2.2 nochmals übersichtlich zusammengefasst. Anhand dieser Sammlung wird die Leistungsfähigkeit bzgl. der Fehlererkennung bekannter Architekturen und Verfahren und der in dieser Arbeit vorgestellten Datenspezifikationsarchitektur evaluiert.

2.5 Auswirkungen von Fehlern

In Kapitel 1.1 wurden bereits einige Auswirkungen von in der Praxis aufgetretenen Fehlern in Form von Sach- oder Personenschäden vorgestellt. Doch nicht jeder Fehler wirkt sich derart stark aus. In Abbildung 2.3, die auf [67] basiert und in [75] erweitert wurde, werden die möglichen Auswirkungen von Fehlern dargestellt.

Ausgehend von einem fehlerfreien Zustand eines Prozessautomatisierungssystems können ungefährliche Ausfälle auftreten, bei denen beispielsweise ein Anzeigeelement ausfällt, die jedoch keine gefährlichen Auswirkungen haben. Fällt jedoch eine Automatisierungsfunktion aus, so wird von einem sicherheitsbezogenen Ausfall gesprochen, da die Sicherheit des automatisierten technischen Prozesses gefährdet sein kann.

Tabelle 2.2: Sammlung datenflussbezogener Fehler- und Angriffsarten

Fehlerkategorie	**Fehlerart**
Inkompatibilität von Operanden	Inkompatible Datentypen
	Inkompatible Einheiten
Wertebereichsverletzungen und Genauigkeitsprobleme	Wertebereichsunter- bzw. -überschreitung
	Genauigkeitsproblem
Fehlerhafte Operationen	Falsche Operandenauswahl
	Falsche Operatorauswahl
	Fehlerhaftes Operationsergebnis
Verletzung von Echtzeitbedingungen	Fristüberschreitung
	Zyklusunterschreitung
	Zyklusüberschreitung
Allgemeine Datenflussfehler	Verlorengegangene Datenaktualisierung
	Synchronisationsfehler oder unvollständige Datenübertragung
	Pufferunter- oder -überläufe
	Fehlerhafter Datenfluss (falsche Adressaten, ...)
	Duplizierte Daten
Datenverfälschung durch Fehler oder Störungen	Durch Fehler oder Störungen verfälschte Daten
Fehlerhafter Zugriff auf Daten	Fehlerhafter Datenzugriff (fehlende Zugriffsrechte)
	Nutzung nicht initialisierter Daten
Angriffskategorie	**Angriffsart**
Hackerangriffe	Gezielt verfälschte Daten
	Wiedereinspielungsattacke

Abbildung 2.3: Auswirkung von Fehlern (nach [67] und [75])

Werden durch die fehlerhafte Automatisierungsfunktion keine falschen Steuersignale ausgegeben, so handelt es sich um eine nicht sicherheitsbezogene Fehlfunktion. Werden jedoch falsche Steuersignale in Folge des Ausfalls generiert, so handelt es sich um eine sicherheitsbezogene Fehlfunktion. Hier ist zu unterscheiden, in welcher Form sich die fehlerhaften Steuersignale auf den Prozess auswirken. In [75] wird als Beispiel die Steuerung der Schranken an einem Bahnübergang herangezogen. Bewirken die falschen Steuersignale ein Schließen der Schranken, obwohl derzeit kein Zug den Bahnübergang passiert, liegt ein ungefährlicher Prozesszustand vor. Würden sich die Schranken jedoch trotz eines nahenden Zuges aufgrund fehlerhafter Steuersignale nicht schließen, würde ein gefährlicher Prozesszustand entstehen.

Ein gefährlicher Prozesszustand muss jedoch nicht immer einen Unfall zur Folge haben. Wenn kein Verkehrsteilnehmer die Schienen im Bereich des Bahnübergangs in dem Moment kreuzt, wenn der Zug diesen passiert, so wird kein Unfall eintreten können. Kommt es jedoch zu einer Kollision, so können die Auswirkungen geringer Art sein, z. B. in Form geringer Sachschäden, aber es können ebenso Personen verletzt oder sogar getötet werden. Neben Sach- und Personenschäden sind auch negative Auswirkungen auf die Umwelt möglich, wie beispielsweise bei auslaufenden Kraft- oder Giftstoffen.

Da nach [75] beim Einsatz von Computern nie bestimmt werden kann, wie sich ein Ausfall auswirkt, sind alle Ausfall- und Fehlerarten als sicherheitsbezogen anzusehen. Wie bereits erwähnt, können bei Software selbst kleine Fehler extreme Auswirkungen haben, was in [44] als „Unstetigkeit von Software“ bezeichnet wird.

2.6 Fehlererkennung und -behandlung

Um auftretende Fehler behandeln zu können, müssen diese zunächst erkannt werden. Dabei erfolgt die Fehlererkennung nach [44] durch die Prüfung mindestens zweier Werte auf die Einhaltung der für diese Werte vorgesehenen Zusammenhänge. Werden Abweichungen festgestellt, so ist von einem Fehler auszugehen. Auf erkannte Fehler kann mit verschiedenen Maßnahmen zur Fehlerbehandlung reagiert werden, von denen einige für diese Arbeit relevante Verfahren in den folgenden Unterkapiteln beschrieben werden.

2.6.1 Einnehmen und Halten eines sicheren Zustands

Eine der wichtigsten Maßnahmen zur Behandlung auftretender Fehler ist das Einnehmen und Halten eines sicheren Zustands, sofern für den jeweiligen technischen Prozess ein solcher existiert. Typische Beispiele für technische Prozesse mit einem sicheren Zustand sind nach [75]

- das Kraftfahrzeug, bei welchem der sichere Zustand der Stillstand ist; das Fahrzeug wird also durch Abbremsen in diesen Zustand überführt,
- ein Kernreaktor, bei dem die Steuerstäbe in den Kern eingefahren werden, um die Kettenreaktion zum Erliegen zu bringen.

Dagegen ist das Flugzeug während des Flugs ein Beispiel für einen technischen Prozess, der keinen sicheren Zustand besitzt.

2.6.2 Anwendung von Redundanzmaßnahmen

Die Sicherheit technischer Prozesse ohne sicheren Zustand kann nach [75] nur dadurch gewährleistet werden, dass diese eine sehr hohe Zuverlässigkeit aufweisen. Dies wird meist dadurch realisiert, dass kritische Einheiten mehrfach, also redundant vorhanden sind, so dass der Ausfall einzelner Einheiten toleriert und der sichere Betrieb des jeweiligen Prozesses unter Nutzung der verbleibenden Einheiten aufrechterhalten werden kann. Idealerweise sind die redundanten Einheiten diversitär ausgelegt, um einen Ausfall mehrerer oder sogar aller Einheiten aufgrund derselben Fehlerursache zu vermeiden. Als Beispiel für den Ausfall aller redundanten Einheiten eines Systems aufgrund fehlender Diversität wurde in Kapitel 1.1.1 die Selbstzerstörung der Ariane 5 beschrieben.

Ein Vergleicher vergleicht die Ausgaben der durch ihn zu überwachenden Einheiten und führt bei Differenzen einen Mehrheitsentscheid durch. Die überstimmte Einheit bzw. die überstimmten Einheiten werden dann als fehlerhaft angenommen.

2.6.3 Allmähliche Leistungsabsenkung

Stellt die Software während der Datenverarbeitung fest, dass es ihr nicht möglich ist, die erforderlichen Ergebnisse innerhalb spezifizierter Zeitgrenzen zu berechnen, so kann sie den Übergang in und das Halten eines sicheren Zustands auslösen,

sofern der jeweilige technische Prozess einen solchen besitzt. Eine andere Möglichkeit der Reaktion auf Überlastsituationen ist nach [45] der Versuch, die gegebenen Zeitgrenzen trotz der Überlastung durch die Suspendierung nicht unbedingt erforderlicher Tasks oder die Ausführung alternativer Berechnungsmethoden mit ggf. geringerer Genauigkeit der Ergebnisse einzuhalten. Die Anwendung der allmählichen Leistungsabsenkung ist daher besonders für die Systeme sinnvoll, die entweder keinen sicheren Zustand besitzen oder anderweitige hohe Anforderungen an ihre Verfügbarkeit aufweisen.

3 Stand von Wissenschaft und Technik

Die folgenden Methoden, Techniken und Architekturen werden mit ihren Merkmalen zur Erkennung von datenflussbezogenen Fehlern in diesem Kapitel detailliert vorgestellt:

- die Schutzmechanismen konventioneller Architekturen am Beispiel der x86-Architektur im geschützten und 64-Bit-Modus, sowie der ARM-Architektur,
- auf sicherheitsgerichtete Systeme spezialisierte Prozessoren, die auf konventionellen Architekturen beruhen, wie z. B. der TI Hercules,
- in Vergessenheit geratene Rechnerarchitekturen wie Datentyp-, Datenstruktur- und Befähigungsarchitekturen, darunter auch moderne Entwicklungen wie PUMP als Teil des SAFE-Projekts,
- der Vollständigkeit halber Datenflussarchitekturen, da ihr Name eine entsprechende Spezialisierung auf Datenflüsse erahnen lässt,
- die inhärent sichere Mikroprozessorarchitektur ISMA,
- die als ADI bzw. SSM bezeichnete Versionskennung des Oracle SPARC M7 Prozessors,
- arithmetische Kodierung in Form der ANBD-Kodierung, sowie
- die Datenflussüberwachung in Netzwerk- und sicherheitsgerichteten Feldbusprotokollen.

Die jeweiligen Merkmale und Verfahren zur Fehlererkennung werden anhand der Erkennbarkeit der 20 in Kapitel 2.4 vorgestellten Fehler- und Angriffsarten evaluiert.

3.1 Konventionelle Architekturen

Als Stand der Technik sollen zunächst konventionelle Architekturen vorgestellt werden, also Architekturen, die millionenfach eingesetzt werden, aber keine oder nur wenige spezielle Merkmale zur Fehlervermeidung und -erkennung aufweisen. Als Beispiele für derartige Architekturen werden die sich am meisten im Einsatz befindlichen Architekturen x86 und ARM [50] detailliert betrachtet.

3.1.1 Die x86-Architektur

Die x86-Familie, die ihren Siegeszug mit dem 16-Bit Mikroprozessor 8086 im Jahr 1978 begonnen hat, ist mit ihrer von-Neumann-Architektur bis heute eine der meistgenutzten Mikroprozessorfamilien [50]. Beginnend mit ersten Erweiterungen beim 80286 kam der geschützte Modus, engl. „Protected Mode“, mit dem 80386 in vollem Umfang zum Einsatz. Der 80386 brachte nach [59] eine Erweiterung der allgemein verwendbaren Register von 16 auf 32 Bit mit sich, ebenso eine Erweiterung des Adressraums auf 32 Bit, daher also 4 GiB. Zusätzlich wurde eine Seitenverwaltung im 80386 implementiert, die so die Realisierung von virtuellem Speicher ermöglichte. Neben Intel stellten und stellen auch Wettbewerber x86-kompatible Prozessoren her, allen voran die Firma AMD. Diese war auch der Vorreiter bei der Einführung von 64 Bit breiten Registern und eines entsprechenden Betriebsmodus [4]. Die Sicherheitsmerkmale des geschützten Modus und die Nachteile der Architektur sollen in den folgenden Unterkapiteln näher erläutert werden.

3.1.1.1 Der geschützte Modus

Im Gegensatz zum Real-Modus, engl. „Real Mode“, in dem nahezu keine Sicherheitsmerkmale durch die Prozessorhardware unterstützt werden, sind im geschützten Modus, engl. „Protected Mode“, mehrere dieser Merkmale verfügbar. Nach [59] ist zunächst zwischen Segmentierung und Seitenverwaltung zu unterscheiden. Adressen innerhalb eines Segments werden nach [84] als virtuelle Adressen bezeichnet. Bei der Segmentierung werden Deskriptoren angelegt, die neben der Basisadresse eines Segments im Speicher auch dessen Größe und die Eigenschaften der im Segment enthaltenen Befehle oder Daten beschreiben. Unterschieden werden dabei Befehls- und Datendeskriptoren. Während die Inhalte eines Befehlssegments grundsätzlich nicht beschreibbar sind, sind die Inhalte eines Datensegments grundsätzlich nicht ausführbar. Für Befehlssegmente kann im Deskriptor bestimmt werden, ob die enthaltenen

Befehle lesbar sind, analog dazu kann für Datensegmente ausgewählt werden, ob die enthaltenen Daten beschreibbar sind. Neben diesen Mitteln zur Beschreibung der Lage, der Größe und der Rechte beim Zugriff auf ein Segment, enthält der Deskriptor auch die Angabe einer Privilegierungsstufe, die minimal notwendig ist, um auf die Inhalte eines Segments zugreifen zu dürfen. Diese Privilegierungsstufe wird durch einen Wert zwischen null und drei angegeben, wobei die Null der höchsten Stufe und dem größten Satz an Rechten entspricht. Dabei wird häufig die Darstellung in Form von Ringen gewählt, wie in Abbildung 3.1 nach [84] zu sehen. Als Anwendungsbeispiele, wie die Privilegierungsstufen oder -ringe den einzelnen Bestandteilen einer Softwareumgebung zuzuordnen sind, wird in [59] vorgeschlagen, das Betriebssystem mit den höchsten Rechten in Stufe 0, hardwarenahe Treiber in Stufe 1, Schnittstellen von Treibern zu Anwendungen und Betriebssystemdienste in Stufe 2 und Anwendungsprogramme in Stufe 3 einzuordnen. In der Praxis werden jedoch oft nur die Privilegierungsstufen 0 und 3 genutzt, wie z. B. beim Betriebssystem Linux [10]. Neben dem Zugriff auf entsprechend privilegierte Segmente wurde ab dem 80386 auch der Befehlssatz in verschiedene Privilegierungsgruppen unterteilt, um zu verhindern, dass Code eines Segments mit geringer Privilegierungsstufe sich einfach selbst höhere Rechte erteilt. Wird eine solche privilegierte Operation von Code mit geringer Privilegierungsstufe ausgeführt, wird ein Ausnahmefehler generiert.

Die Seitenverwaltung liegt hierarchisch gesehen unter der Segmentierung. Sie sorgt dafür, dass die Adressen, die sich innerhalb eines Segments ergeben (Basis des Segments + Adressabstand eines bestimmten Elements innerhalb des Segments) zunächst als logische Adressen interpretiert werden, die mit der physikalischen Lage eines Elements im Speicher nicht mehr direkt in Verbindung zu bringen sind. Die Seitenverwaltung übernimmt die Übersetzung der logischen Adresse in eine physikalische Adresse, wie in Abbildung 3.2 gezeigt. Bei der Seitenverwaltung existieren ebenfalls Privilegierungs- und Rechtebeschreibungen für die jeweiligen Seiten. Dabei sind, im Gegensatz zur Segmentierung, nur noch zwei Privilegierungsstufen vorhanden: eine privilegierte und eine Benutzerstufe. Code, der in einem Ring-3-Segment ausgeführt wird, kann nur auf Seiten mit Benutzerstufe zugreifen. Bei den Zugriffsrechten kann eine Seite als nur-lesbar markiert werden, nach [59] sind jedoch alle Seiten zunächst lesbar und enthaltene Speicherworte sind als ausführbarer Code interpretierbar. Bis zur Einführung des sog. NX-Bits war es nicht möglich, Speicherseiten in der Seitenverwaltung als nicht-ausführbar zu deklarieren, es gab also keine Möglichkeit, den Versuch, Daten als Instruktionen zu interpretieren, mittels der Seitenverwaltung als Fehler zu erkennen.

Obwohl mit der Segmentierung starke Sicherheitsmerkmale zur Verfügung gestellt werden, wird in der Praxis in den dominantesten Betriebssystemen Microsoft Windows [84] und Linux [10] ein flaches Speichermodell eingesetzt, wie es auch schon in [59] vorgeschlagen wurde. Dabei werden Befehls- und Datensegmente mit derselben Basisadresse und maximalen Limits angelegt, üblicherweise auch mit maximalen Rechten, wodurch der gesamte Speicher und die gleichen Speicherinhalte über die Befehls- und Datensegmentdeskriptoren angesprochen werden können. In Abbildung 3.3 werden die vier dafür angelegten Segmentdeskriptoren im Betriebssystem Linux nach [10] dargestellt. Für das Betriebssystem hat ein solches flaches Speichermodell den Vorteil, dass viel einfacher auf die Inhalte des Speichers eines Anwendungsprozesses zugegriffen werden kann und z. B. ein Ladeprogramm die Befehle der Anwendung einfach als Daten in den Speicher der Anwendung schreibt, die dann als Befehle interpretiert werden. Andererseits werden die Sicherheitsmechanismen der Segmentierung dadurch fast komplett ausgehebelt, es bleiben nur noch die Sicherheitsmerkmale der Seitenverwaltung bestehen.

Zusätzlich zu den bislang beschriebenen Mechanismen können die Rechte für die Ein- und Ausgabe über E/A-Anschlüsse, die über die dedizierten Befehle IN und OUT angesprochen werden, für Anwendungscode eingeschränkt werden.

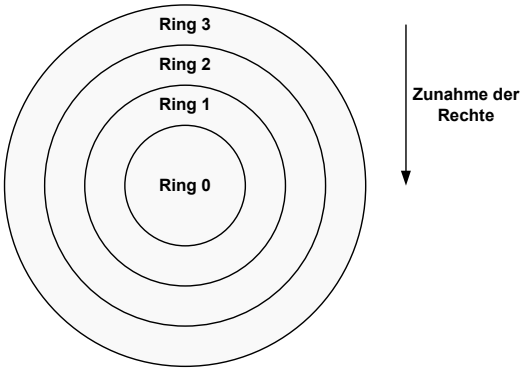


Abbildung 3.1: Die vier Privilegierungsstufen in Ringdarstellung

Abbildung 3.2: Umsetzung einer virtuellen zu einer physikalischen Adresse

Kernsegmente

Basis:	**0x00000000**
Limit:	**4 GiB**
Typ:	**Code + lesbar**
Privilegierungsstufe (DPL):	**0**
Basis:	**0x00000000**
Limit:	**4 GiB**
Typ:	**Daten + schreibbar**
Privilegierungsstufe (DPL):	**0**

Anwendungssegmente

Basis:	**0x00000000**
Limit:	**4 GiB**
Typ:	**Code + lesbar**
Privilegierungsstufe (DPL):	**3**
Basis:	**0x00000000**
Limit:	**4 GiB**
Typ:	**Daten + schreibbar**
Privilegierungsstufe (DPL):	**3**

Abbildung 3.3: Die vier Segmente für Betriebssystem und Anwendungen bei Linux

3.1.1.2 Der 64-Bit-Betriebsmodus

Der von AMD eingeführte 64-Bit-Betriebsmodus, engl. „Long Mode“ genannt, ermöglicht die native Nutzung von 64 Bit breiten Registern und eines entsprechend großen Adressraums zusammen mit neuen, auf die Registerbreite und Adressraumgröße abgestimmten Instruktionen, dem AMD64-Befehlssatz [4]. Zusätzlich wurde die Anzahl der allgemeinen Register von 8 auf 16 erhöht. Die Segmentierung und die mit ihr verbundenen Sicherheitsmerkmale, die im geschützten Modus angeboten werden, sind im 64-Bit-Betriebsmodus nicht vorhanden, es ist daher nur ein flaches Speichermodell möglich. Es bleiben somit nur die Sicherheitsmerkmale der Seitenverwaltung übrig.

3.1.1.3 Moderne sicherheitstechnische Erweiterungen der x86-Architektur

Neben der bereits erwähnten Einführung des NX-Bits in der Seitenverwaltung, das es erlaubt, Speicherseiten als nicht ausführbar zu kennzeichnen, wurden von Intel jüngst zwei weitere Ansätze zur Erhöhung der Systemsicherheit eingeführt, die hier kurz vorgestellt werden sollen:

- die Memory Protection Extensions (kurz „MPX“) und
- die Control-flow Enforcement Technology (kurz „CET“).

Während MPX die Fehlererkennung der x86-Architektur in Bezug auf die Erkennung von Pufferüber- und -unterläufen verbessert, dient CET einzig der Aufdeckung von Kontrollflussanomalien, wie sie bei gezielten Angriffen zur Ausnutzung von Sicherheitslücken in Software auftreten. CET ist daher unerheblich für den Stand von Wissenschaft und Technik für diese Arbeit, wird jedoch trotzdem kurz umrissen, da damit gezeigt werden kann, dass weiterhin erheblicher Bedarf an neuen Sicherheitsmechanismen in x86-Architekturen besteht, um Sicherheitslücken durch typische Softwarefehler zu erkennen.

3.1.1.3.1 Memory Protection Extensions MPX

Die Memory Protection Extensions MPX von Intel dienen – wie der Name bereits andeutet – der Erweiterung der existierenden Speicherschutzmechanismen der x86-Architektur [47]. Dabei werden zusätzliche Instruktionen und Register zur Bereichsprüfung durch den Prozessor zur Verfügung gestellt, um eine häufige und gefährliche Art von Softwarefehlern [66, 86, 103] aufdecken zu können: Pufferüber-

bzw. -unterläufe, also das Lesen oder Schreiben über die unteren oder oberen Grenzen eines Puffers hinaus. Da viele Sicherheitslücken in Softwareprogrammen auf diese Art von Fehlern zurückzuführen sind, werden diese in [86] auf Platz 3 der 25 gefährlichsten Softwarefehler geführt.

Der Aufwand zur Nutzung der neuen Funktionalitäten ist nicht unerheblich: für den Programmierer besteht der Aufwand zwar nur darin, den Übersetzer dazu anzuweisen, MPX zu nutzen, auf Maschinenbefehlsebene ist der Zusatzaufwand allerdings beträchtlich:

In [47] wird eine Funktion einmal mit und einmal ohne MPX-Unterstützung übersetzt und die sich dabei ergebenden Maschinenbefehle verglichen (die beiden folgenden Auflistungen sind direkt [47] entnommen und in Hinsicht auf die Lesbarkeit bearbeitet worden).

```
00000000004006e0 <dog_letter>:
  4006e0: 48 63 ff                movslq %edi,%rdi
  4006e3: 0f b6 87 43 10 60 00    movzbl 0x601043(%rdi),%eax
  4006ea: c3                      retq
```

Maschinencode des Beispiels ohne Nutzung von MPX

```
0000000000400750 <dog_letter>:
  400750: 66 0f 1a 05 f8 08 20 00 bndmov 0x2008f8(%rip),%bnd0
  400758: 48 63 ff                movslq %edi,%rdi
  40075b: 48 8d 87 67 10 60 00    lea    0x601067(%rdi),%rax
  400762: f3 0f 1a 00             bndcl  (%rax),%bnd0
  400766: 66 0f 1a 0d e2 08 20 00 bndmov 0x2008e2(%rip),%bnd1
  40076e: f2 0f 1a 08             bndcu  (%rax),%bnd1
  400772: 0f b6 87 67 10 60 00    movzbl 0x601067(%rdi),%eax
  400779: f2 c3                   bnd retq
```

Maschinencode des Beispiels mit Nutzung von MPX

Mit den beiden `bndmov`-Anweisungen werden die Speicherbereichsgrenzen des Datenfelds in ein Bereichsregister geladen und mittels `bndcl` bzw. `bndcu` wird der Index vor dem eigentlichen Zugriff auf das Datenfeld daraufhin überprüft, ob er sich innerhalb der Feldgrenzen befindet. Es lässt sich feststellen, dass die Funktion

ohne Nutzung von MPX aus gerade einmal 3 Instruktionen mit einer Gesamtgröße von 11 Byte besteht, während die Variante mit Nutzung von MPX bereits 8 Instruktionen verwendet und beachtliche 42 Byte belegt.

3.1.1.3.2 Control-flow Enforcement Technology CET

Die Erweiterung Control-flow Enforcement Technology CET von Intel soll die Ausnutzung von Sicherheitslücken erschweren, die durch Programmierfehler entstehen und eine Veränderung des vorhergesehenen Kontrollflusses des fehlerhaften Programms zulassen [60].

Dazu kommen nach [60] zwei neue Merkmale zum Einsatz:

- ein Schattenstapelspeicher, auf dem die Rücksprungadressen bei Funktionsaufrufen zusätzlich abgelegt werden und bei Rückkehr zur aufrufenden Funktion mit jenen auf dem eigentlichen Stapelspeicher verglichen werden und
- die Prüfung, ob indirekte Sprung- und Unterprogrammaufrufbefehle als Ziel ein vorhergesehenes Sprung- bzw. Aufrufziel haben, indem dort eine neue Pseudoinstruktion erwartet wird, die „Endbranch“ genannt wird.

Während die Prüfung der Sprung- und Unterprogrammaufrufziele sehr einfach zu realisieren ist, ist die Realisierung des Schattenstapelspeichers deutlich aufwendiger. So wurde z. B. ein neues Bit in der Seitenverwaltung eingeführt, um zu erreichen, dass die Speicherbereiche, die Schattenstapelspeicher enthalten, vor Zugriffen durch – möglicherweise bösartigen – Nutzercode geschützt werden. Dies ist notwendig, da ansonsten entsprechender Schadcode neben der Manipulation des eigentlichen Stapelspeichers lediglich noch die Inhalte des Schattenstapelspeichers entsprechend anpassen müsste, um unerkannt sein Ziel zu erreichen.

3.1.2 Die ARM-Architektur

Im Gegensatz zu x86-Prozessoren stellt das Unternehmen ARM Ltd. keine eigenen Prozessoren her, sondern vergibt Fertigungslizenzen an Chiphersteller, die dann die vorgefertigten Prozessorentwürfe in ihre Produkte integrieren [20]. Die Prozessoren sind auf besonders niedrige Leistungsaufnahme hin optimiert und kommen z. B. bei eingebetteten Systemen und Mobiltelefonen in großer Zahl zum Einsatz [50].

Ein ARM-Prozessor bietet bis zu 9 verschiedene Betriebsmodi, von denen die Modi Supervisor (für Betriebssysteme), System (für privilegierte Anwendungen) und User (für Anwendungen) relevant sind [7, 12].

Nach [7] gibt es bei ARM-Prozessoren keine Segmentierung, wodurch nur ein flaches Speichermodell implementierbar ist. Die Schutzmechanismen basieren daher allein auf der Seitenverwaltung, wobei seit der Prozessorgeneration ARMv6 ein dem NX-Bit der x86-Prozessoren äquivalentes XN-Bit, engl. „eXecute Never“, zur Verhinderung der Ausführung von Daten implementiert ist. Interessant ist der Hinweis in [7], dass nicht alle ARM-Prozessoren eine Division durch Null als Ausnahmefehler melden, sondern als Ergebnis eine Null zurückgeben. ARM-Prozessoren haben, je nach Typ [19], eine der von-Neumann- oder Harvard-Architektur entsprechende Speicheranbindung, also einen gemeinsamen Bus für Befehle und Daten oder zwei getrennte Busse.

3.1.3 Integritätsprüfung durch ECC

Im Serverumfeld und gelegentlich auch in Mikroprozessoren für den Einsatz in eingebetteten Systemen kommen Integritätsprüfungsmechanismen für Daten zum Einsatz, anhand derer geprüft werden kann, ob die in einem Datenspeicherelement enthaltenen Daten korrekt sind. Dabei werden die Daten bei Schreibvorgängen mit einer Fehlererkennungs- und ggf. sogar -korrekturkodierung versehen und dann im Arbeitsspeicher abgelegt. Beim Lesen wird die Integrität der Speicherinhalte anhand dieser Kodierung geprüft und im Fehlerfall

- bei reinen Fehlererkennungskodierungen ein Ausnahmefehler generiert bzw.
- bei Fehlerkorrekturkodierung eine Korrektur der Daten vorgenommen oder – falls dies nicht möglich ist – ein Ausnahmefehler generiert.

Bei Servern, die auf der x86-Architektur basieren, handelt es sich um eine Implementierung eines Erweiterten-(72,64)-Hamming-Codes [68], die „ECC“ genannt wird und für „Error Correction Code“ steht.

Die Integrität von Datenspeicherelementen wird dabei nur zwischen Speichercontroller und Arbeitsspeicher geprüft, wie in Abbildung 3.4 dargestellt. Ein Speichercontroller errechnet die entsprechenden Integritätsprüfungsbits, fügt sie den Daten hinzu, überträgt die Daten mitsamt den Prüfbits in den Arbeitsspeicher, liest diese zum Zeitpunkt der Verwendung wieder aus dem Arbeitsspeicher, prüft die Inhalte auf Integrität unter Zuhilfenahme der Prüfbits und reicht die Inhalte dann an den Prozessor weiter, jedoch ohne die Fehlererkennungs- oder -korrekturkodierung. Eine durchgängige Integritätsprüfung, die auch die Verbindung zwischen Speichercontroller und Prozessor und innerhalb des Prozessors die Verarbeitung in dessen

Registern einschließt – wie sie bei später in dieser Arbeit vorgestellten Architekturen zum Einsatz kommt – wird nicht angewendet.

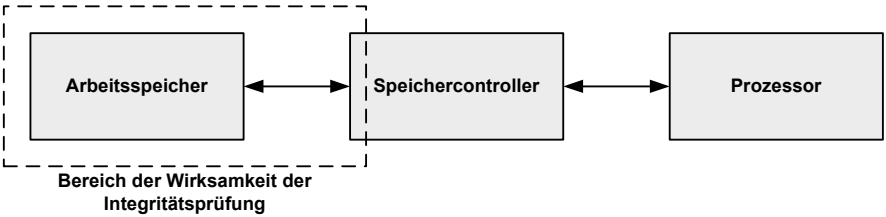


Abbildung 3.4: Abdeckungsbereich der Integritätsprüfung in konventionellen Architekturen

3.1.4 Evaluation konventioneller Architekturen

Der sog. „von-Neumann-Flaschenhals“ schränkt die Leistungsfähigkeit der x86-Architektur ein, weil sequentiell auf Befehle und die dazugehörigen Daten zugegriffen werden muss. Um die Leistungsfähigkeit zu steigern, wurden hochkomplexe Mechanismen eingeführt, wie zum Beispiel mehrere Ebenen von Verdecktspeichern, engl. „Caches“, um Speicherzugriffe zu beschleunigen und Sprungvorhersagen oder die Ausführung von Anweisungen in einer vom Maschinencode abweichenden Reihenfolge, engl. „Out of Order Execution“, um Pipeline-Blockaden zu vermeiden und den Durchsatz zu optimieren [9, 84]. Der Beweis der Korrektheit dieser Mechanismen ist aufgrund ihrer hohen Komplexität sehr aufwendig. Der Einsatz der Verdecktspeicher und die Nutzung von virtuellem Speicher beeinträchtigen die Vorhersagbarkeit des zeitlichen Verhaltens eines x86-Systems.

In konventionellen Architekturen werden Daten nur in Form ihres Datenwerts dargestellt und alle weiteren Eigenschaften des Datenwerts werden implizit bei der Verarbeitung durch die Anwendung entsprechender Befehle angenommen. Zugriffsrechte werden losgelöst von den eigentlichen Daten über Sicherheitsmechanismen wie Segmentierung und Seitenverwaltung mit den Datenspeicherelementen verknüpft.

Die Segmentierung bringt ein hohes Maß an Komplexität in der Verwaltung durch das Betriebssystem mit sich, weshalb auf sie in flachen Speichermodellen verzichtet wird. Dies stellt, neben der fehlenden Unterscheidung von Code und Daten

durch den Befehlsprozessor selbst, durch die Möglichkeit, Daten in flachen Speichermodellen als Befehle auszuführen, ein großes Problem dar. So bleiben nur die Sicherheitsmerkmale der Seitenverwaltung übrig, die bis zur Einführung des NX-Bits, einem englischen Akronym für „No eXecute“, in der Seitenverwaltung nicht erlaubten, Seiten als nicht ausführbar zu kennzeichnen. Der 64-Bit-Modus verzichtet auf die ohnehin nicht oder nur ansatzweise genutzten Sicherheitsmerkmale der Segmentierung und bietet nur ein flaches Speichermodell und die Sicherheitsmerkmale der Seitenverwaltung an.

Die Nachteile der ARM-Architektur sind im Wesentlichen mit denen der x86-Architektur identisch. Bei ARM-Modellen mit einer Speicheranbindung nach der Harvard-Architektur gibt es natürlich keine Geschwindigkeitseinbußen beim Zugriff auf Befehle und Daten und die Gefahr, Daten als Befehle zu interpretieren, ist dadurch nicht gegeben.

Die Hersteller derartiger Prozessoren versuchen sich bei maximalen Taktfrequenzen, Anzahl der Kerne und Angaben über die pro Sekunde ausgeführten Instruktionen (z. B. MIPS – „million instructions per second“) bzw. Gleitkommaoperationen (z. B. FLOPS – „floating point operations per second“) gegenseitig zu übertrumpfen. Für dieses Muskelspiel wurde – wie bereits oben angedeutet – eine Vielzahl von Verfahren zur Erhöhung der genannten Kennzahlen entwickelt, statt sich auf die Entwicklung einfacher, sinnvoller und lesitungsfähiger Fehlervermeidungs- und -erkennungsmaßnahmen zu konzentrieren. Entsprechend können nur die wenigsten Fehlerarten erkannt werden, wie sich anhand der Erkennbarkeit der 20 in Kapitel 2.4 vorgestellten Fehler- und Angriffsarten in Tabelle 3.1 klar erkennen lässt.

3.1.4.1 Evaluation der x86-Architektur

Die x86-Architektur zeigt sich anhand Tabelle 3.1 wenig leistungsfähig bezogen auf die erkennbaren Fehlerarten. Während im geschützten Modus noch die Instruktion `bound` verfügbar war, um Wertebereichsunter- bzw. -überschreitungen durch zusätzliche Softwareprüfungen zu ermöglichen, wurde dieser im modernen 64-Bit-Modus entfernt. Damit kann diese Fehlerart nur als begrenzt erkennbar bewertet werden.

Pufferunter- und -überläufe können mit dem neuen MPX-Schutzmerkmal zuverlässig erkannt werden, aber die Kosten in Form von Codegröße und Ausführungszeit sind hoch.

Werden Daten durch Störungen verfälscht, kann dies beim Einsatz von ECC-Speichern erkannt werden. Allerdings werden nur die Inhalte des Arbeitsspeichers

Tabelle 3.1: Fehlererkennung durch x86 und ARM

Fehlerart	**x86**	**ARM**
Inkompatible Datentypen	nein	nein
Inkompatible Einheiten	nein	nein
Wertebereichsunter- bzw. -überschreitung	begrenzt (bound)	nein
Genauigkeitsproblem	nein	nein
Falsche Operandenauswahl	nein	nein
Falsche Operatorauswahl	nein	nein
Fehlerhaftes Operationsergebnis	nein	nein
Fristüberschreitung	nein	nein
Zyklusunterschreitung	nein	nein
Zyklusüberschreitung	nein	nein
Verlorengegangene Datenaktualisierung	nein	nein
Synchronisationsfehler oder unvollständige Datenübertragung	nein	nein
Pufferunter- oder -überläufe	(ja) (MPX)	nein
Fehlerhafter Datenfluss (falsche Adressaten, ...)	nein	nein
Duplizierte Daten	nein	nein
Durch Fehler oder Störungen verfälschte Daten	begrenzt (ECC)	begrenzt (ECC)
Fehlerhafter Datenzugriff (fehlende Zugriffsrechte)	begrenzt	begrenzt
Nutzung nicht initialisierter Daten	nein	nein
Angriffsart		
Gezielt verfälschte Daten	nein	nein
Wiedereinspielungsattacke	nein	nein

und die Datenverbindung zum Speichercontroller dadurch geprüft, Fehler in Prozessorregistern würden dadurch z. B. nicht aufgedeckt. Da ECC-Speicher zudem nicht in allen Systemen eingesetzt werden kann, kann diese Fehlerart ebenfalls nur als begrenzt erkennbar bewertet werden.

Fehlerhafte Datenzugriffe können nur im Rahmen der groben Segmentierungs- und Seitenverwaltungsmechanismen erkannt werden. Auch hier ist nur eine begrenzte Fehlererkennungsmöglichkeit gegeben.

3.1.4.2 Evaluation der ARM-Architektur

Die ARM-Architektur schneidet bei der Betrachtung der erkennbaren Fehler sogar noch schlechter als die x86-Architektur ab, wie in Tabelle 3.1 ersichtlich.

Beim Einsatz von ECC-Speichern können Datenverfälschungen in Speichern und auf dem Weg bis zur Paritätsprüfung aufgedeckt werden. Wie bei der x86-Architektur werden aber auch hier wesentliche Teile der Datenverarbeitung innerhalb des Prozessors nicht in die Prüfung einbezogen. Fehlerhafte Datenzugriffe können durch die Seitenverwaltung nur in grober Granularität aufgedeckt werden. Beide Fehlerarten können daher nur als begrenzt erkennbar bewertet werden.

3.2 Prozessoren für sicherheitsgerichtete Anwendungen

Neben den bereits vorgestellten konventionellen Architekturen x86 und ARM gibt es Mikroprozessoren, die zwar auf konventionellen Architekturen basieren, jedoch speziell für den Einsatz in sicherheitsgerichteten Anwendungen vorgesehen sind. Da Aufbau, Funktionsumfang und Sicherheitsmerkmale dieser Prozessoren deutlich vom bisher vorgestellten Stand der Technik abweichen, werden die Prozessoren für sicherheitsgerichtete Anwendungen hier getrennt vorgestellt und evaluiert.

3.2.1 Aufbau der Prozessoren für sicherheitsgerichtete Anwendungen

Als Beispiele für auf konventionellen Architekturen basierende Prozessoren für sicherheitsgerichtete Anwendungen sollen drei wichtige Vertreter vorgestellt werden:

- TI Hercules (basierend auf ARM Cortex-R) [120]
- NXP MPC (basierend auf PowerPC e200) [91]
- Infineon AURIX (basierend auf TriCore) [58]

Betrachtet man die Prozessoren im Detail, so stellt man fest, dass sie alle sehr ähnliche Fehlererkennungsmerkmale aufweisen, obwohl die Kerne drei unterschiedlichen Architekturen entstammen. Diese Merkmale – auch wenn sie nicht alle in jedem der Prozessoren zur Anwendung kommen – sind:

- Redundanz: Im sog. „Lockstep-Modus“ arbeitet ein redundanter Prozessorkern basierend auf den gleichen Eingabedaten wie ein Hauptkern und die Ausgaben beider Kerne werden verglichen
- Räumliche Diversität: Der redundante Kern ist auf dem Prozessor möglichst weit vom Hauptkern entfernt
- Räumliche Diversität: Der redundante Kern ist gegenüber dem ersten Kern gedreht
- Zeitliche Diversität: Der redundante Kern arbeitet n Takte versetzt gegenüber dem ersten Kern
- Entwurfsdiversität: Infineon wirbt in [58] damit, dass der zweite Kern diversitär gegenüber dem ersten entworfen worden sein soll
- Prüfung der Datenintegrität: Nutzung einer Integritätsprüfung in Form eines ECC in Festwert- und Arbeitsspeichern, sowie bei Datenübertragung über die internen Datenbusse
- Adressierungsfehler: Die Adresse von Daten oder Instruktionen ist Teil der Datenintegritätsprüfung, wie in der IEC 61508-2 [51] gefordert

In Abbildung 3.5 wird der Lockstep-Modus der beiden ARM-Cortex-R-Kerne des TI Hercules mit räumlicher und zeitlicher Diversität dargestellt.

3.2.2 Evaluation der Prozessoren für sicherheitsgerichtete Anwendungen

Im Vergleich zu den nicht sicherheitsgerichteten konventionellen Architekturen x86 und ARM sind die auf sicherheitsgerichtete Anwendungen spezialisierten Prozessoren in der Lage, weitere Arten der 20 in Kapitel 2.4 aufgelisteten Fehler- und

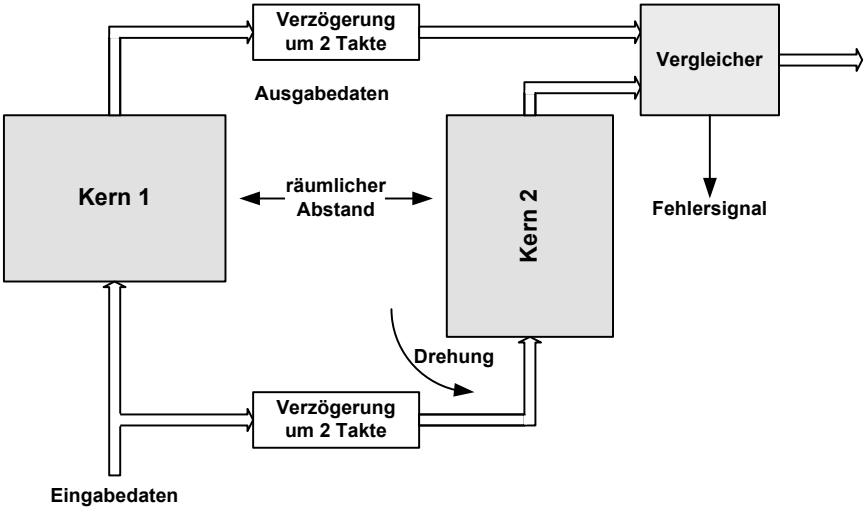


Abbildung 3.5: Zwei Kerne im Lockstep-Modus des ARM Cortex-R (nach [120])

Angriffsarten zu erkennen und entsprechende Reaktionen einer sicherheitsgerichteten Anwendung auszulösen. Die Erkennbarkeit der Fehler- und Angriffsarten wird in Tabelle 3.2 gezeigt.

Durch in den Prozessoren integrierte Zeitüberwachungseinheiten – engl. „Watchdogs“ – können in begrenztem Umfang Verletzungen zeitlicher Bedingungen erkannt werden, wenn auch nur in begrenztem Umfang, da sie für gesamte Verarbeitungszyklen und nicht auf feingranularer Basis einzelner Datenworte definierbar sind.

Die Datenintegritätsprüfung durch ECC wurde gegenüber den nicht sicherheitsgerichteten Prozessorvarianten deutlich umfangreicher realisiert. Neben den verschiedenen Speichern werden nun auch die Datenbusse in die Prüfung mit einbezogen. Allerdings bleiben die Register von dieser Prüfung ausgenommen, wodurch Datenverfälschungen auf Registerebene durch die Redundanz aufgedeckt werden müssen. Wie auch bei den nicht sicherheitsgerichteten Prozessorvarianten sind Daten, die durch den Prozessor von außen eingelesen oder nach außen ausgegeben werden, nicht mit einer Integritätsprüfung versehen, dies muss durch die Software realisiert werden. Durch die Einbeziehung der Adressen in die Integritätsprüfung – wie in der IEC 61508-2 [51] gefordert – können auch Adressierungsfehler innerhalb des Prozessors aufgedeckt werden.

Tabelle 3.2: Fehlererkennung durch Prozessoren für sicherheitsgerichtete Anwendungen

Fehlerart	**Erkennbarkeit**
Inkompatible Datentypen	nein
Inkompatible Einheiten	nein
Wertebereichsunter- bzw. -überschreitung	nein
Genauigkeitsproblem	nein
Falsche Operandenauswahl	ja
Falsche Operatorauswahl	ja
Fehlerhaftes Operationsergebnis	ja
Fristüberschreitung	begrenzt (Watchdog)
Zyklusunterschreitung	nein
Zyklusüberschreitung	nein
Verlorengegangene Datenaktualisierung	nein
Synchronisationsfehler oder unvollständige Datenübertragung	nein
Pufferunter- oder -überläufe	nein
Fehlerhafter Datenfluss (falsche Adressaten, ...)	begrenzt
Duplizierte Daten	nein
Durch Fehler oder Störungen verfälschte Daten	(ja) (ECC)
Fehlerhafter Datenzugriff (fehlende Zugriffsrechte)	begrenzt
Nutzung nicht initialisierter Daten	nein
Angriffsart	
Gezielt verfälschte Daten	nein
Wiedereinspielungsattacke	nein

Die Stärken der redundanten Prozessorkonfiguration zeigen sich im Bereich der Erkennung fehlerhafter Datenverarbeitungsergebnisse. Durch den Vergleich der Ausgaben beider Prozessorkerne können falsche Operanden, falsche Operatoren und Fehler des Ergebnisses der Datenverarbeitung sicher aufgedeckt werden. Es gilt jedoch zu bedenken, dass beide Kerne die identische Software ausführen, wodurch im Falle eines Softwarefehlers die Redundanzmaßnahmen wirkungslos werden, da beide Kerne das gleiche, jedoch falsche Ergebnis generieren würden. Durch entsprechende Gestaltung der eingesetzten Software, z. B. unter Nutzung diversitärer Tasks, kann diesem Problem begegnet werden. Beim TI Hercules sind die beiden Prozessorkerne identisch aufgebaut, also nicht diversitär realisiert [120]. Im Falle eines Entwurfs- oder Implementierungsfehlers würden beide Prozessoren die identischen fehlerhaften Ergebnisse liefern, die dann durch den Vergleicher nicht aufgedeckt werden könnten. Dies ist beim Infineon AURIX laut [58] nicht der Fall, da hier auf Entwurfsebene der zweite Kern diversitär entwickelt worden sein soll.

3.3 Datentyparchitekturen

Feustel beschreibt in [36] die Vorteile einer Rechnerarchitektur, bei der Speicherworte zusätzliche Informationsbits, sogenannte Kennungen oder Kennungsbits, zur Beschreibung ihrer Inhalte mit sich führen und daher als engl. „Tagged Architectures“ bezeichnet werden, was man grob übersetzt als „Architektur mit Beschriftungen“ verstehen könnte. Diese Art von Architektur wird nach Giloi [39] im Deutschen Datentyparchitektur genannt. Die Vorteile solcher Architekturen liegen auf der Hand [36, 87]:

- Daten sind selbstbeschreibend.
- Befehle und Daten weisen sich als solche aus, wodurch es z. B. nicht möglich ist, Daten versehentlich als Code zu interpretieren.
- Fehlerhafte Verwendung von Daten, also z. B. Operanden mit inkompatiblen Datentypen, kann durch die Hardware erkannt werden.
- Befehle können ein auf die verwendeten Datentypen der Operanden angepasstes Verhalten aufweisen, wodurch dedizierte Befehle, die beschreiben, ob die Operanden z. B. als Ganz- oder Gleitkommazahlen zu interpretieren sind, überflüssig werden.

- In gewissem Umfang können Übersetzer einfacher und fehlerärmer werden, da ihnen nicht mehr die Auswahl der zu den jeweiligen Datentypen passenden Instruktionen obliegt.

Den genannten Vorteilen steht der erhöhte Speicherverbrauch gegenüber, da die Beschreibungsbits zusammen mit den eigentlichen Daten im Speicher abgelegt werden müssen. Ein weiterer Nachteil besteht darin, dass konventionelle Architekturen nicht darauf ausgelegt sind, Datentypkennungen zu nutzen und zu prüfen. Nach [39] ist es dennoch möglich, in konventionellen Architekturen eine Typenkennung zu implementieren, indem man das BEBOP-Verfahren anwendet. Dieses basiert auf der Einteilung des Speichers in n Teilbereiche für n verschiedene Datentypen. Ein Teil der Adresse eines Objekts wird daher einen Teilbereich des Speichers und damit den Datentyp identifizieren. Großer Nachteil des Verfahrens ist jedoch der verschwenderische Umgang mit dem vorhandenen Speicher. Zudem ist die Datentypkennung damit nur auf dem impliziten Weg des Ablageorts im Speicher mit dem Datenwert verbunden, wodurch die Information z. B. beim Datenaustausch mit anderen Programmen oder Systemkomponenten verloren geht und getrennt vom Datenwort übermittelt und verwaltet werden muss.

3.3.1 Beispiele von Datentyparchitekturen

Als Vertreter reiner Datentyparchitekturen werden hier der Großrechner TR 4 der Firma Telefunken und, als exotisches Beispiel, der Experimentierrechner CP1 der Firma Kosmos vorgestellt.

3.3.1.1 TR 4 von Telefunken

Kommerziell, historisch

Der Großrechner TR 4 von Telefunken, hergestellt nach [104] in den Jahren 1962 bis 1968, arbeitete nach [1] mit Lochstreifen, Lochkarten und Magnetbandspeichern und implementierte einige wirkungsvolle Sicherheitsmerkmale. Jedes Speicherwort umfasste 52 Bit, wovon 2 Bit den Inhalt der restlichen 50 Bit so ergänzten, dass der Inhalt jedes Speicherworts durch drei teilbar wurde. Diese Dreierprobenbits waren für den Programmierer nicht zugänglich. Vor Verwendung eines Speicherworts wurde die Prüfung auf Teilbarkeit durch Drei durch das sog. Dreierproben-Prüfwerk

durchgeführt. Bei einem Fehler wurde ein Dreierprobenfehler (DP-Alarm) generiert und somit eine Integritätsprüfung der Speicherinhalte und von Teilen der Datenverarbeitungseinheit vorgenommen. Des Weiteren besaß jedes Speicherwort eine Typenkennung, die durch 2 weitere Bits realisiert wurde. Dadurch konnten vier verschiedene Arten von Speicherworten unterschieden werden:

- 0 - Gleitkommazahlen
- 1 - Festkommazahlen
- 2 - Einadressbefehle
- 3 - alphanumerische Zeichen

Stieß der Befehlsprozessor auf ein Speicherwort, das er ausführen sollte, mit einer Typenkennung ungleich zwei, sollten Instruktionsworte als Daten interpretiert werden oder wurden Befehle auf die falsche Art von Daten angewandt, so wurde nach [1] ein sog. Typenkennungsalarm ausgelöst. Ab 1968 wurde der TR 4 nach [104] durch seinen deutlich leistungsfähigeren Nachfolger, den TR 440, abgelöst, der die gleichen Sicherheitsmerkmale wie sein Vorgänger aufwies.

3.3.1.2 Experimentiercomputer CP1 von Kosmos

Kommerziell, wenn auch nur als Lernspielzeug und nicht im industriellen Rahmen, historisch

Der Kosmos Computer Praxis CP1, gezeigt in Abbildung 3.6, war ein in den frühen 1980er Jahren hergestellter Experimentiercomputer, um nach [69] Kindern, Jugendlichen und Erwachsenen Mikroelektronik und Computertechnik spielerisch näher zu bringen. Auf Basis eines Intel 8049 Mikrocontrollers aus der MCS-48-Familie wurde eine virtuelle Maschine realisiert, die auf einer assemblernahen Sprache durch 24 verschiedene Instruktionen 01.000 bis 24.00x programmiert werden konnte. Der CP1 wird hier als Beispiel im Umfeld von Großrechnern und leistungsfähigen konventionellen Architekturen vorgestellt, da er ebenfalls eine einfache und doch hochfunktionale Art der Typenkennung implementierte, die Befehle von Daten unterschied: Nach [69] begann jeder Befehl, wie bereits erwähnt, mit einer Befehlskennung zwischen 01 und 24, wohingegen Daten immer die Kennung 00 besaßen. Der Versuch, Daten als Befehle auszuführen, führte zur Generierung des Ausnahmefehlers F 002. Ebenso wurde der Versuch, Befehle durch Datenverarbeitungsbefehle als Daten zu behandeln, durch den CP1 erkannt und der Ausnahmefehler F 005 erzeugt. Traten

bei den arithmetischen Operationen Addition und Subtraktion Über- bzw. Unterläufe auf, führten diese zum Programmabbruch aufgrund des Fehlers F 006.

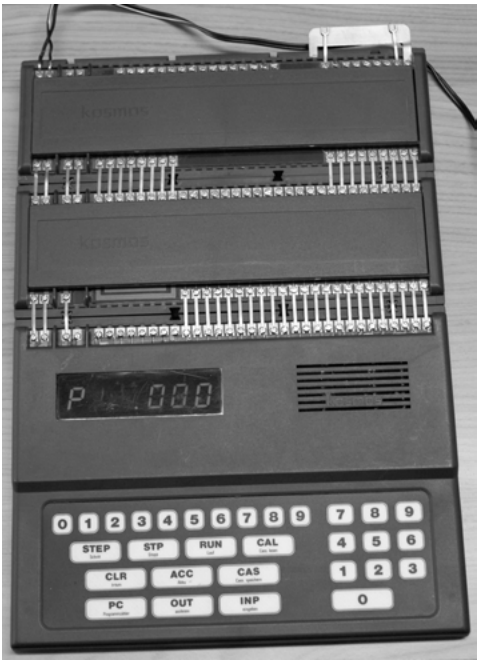


Abbildung 3.6: Kosmos CP1 mit Erweiterungsmodulen

3.3.2 Evaluation der Datentyparchitekturen

Die Fehlererkennbarkeit von Datentyparchitekturen basierend auf den in Kapitel 2.4 vorgestellten 20 Fehler- und Angriffsarten wird in Tabelle 3.3 dargestellt.

Datentyparchitekturen ermöglichen die Erkennung von Inkompatibilitäten der Datentypen von Operanden durch die explizite, hardwareverständliche Spezifikation der Datentypen innerhalb der Datenspeicherelemente selbst. Die Prüfung der Kompatibilität erfolgt in der Hardware parallel zur Ausführung der eigentlichen Operati-

Tabelle 3.3: Fehlererkennung durch Datentyparchitekturen

Fehlerart	**Erkennbarkeit**
Inkompatible Datentypen	ja
Inkompatible Einheiten	nein
Wertebereichsunter- bzw. -überschreitung	nein
Genauigkeitsproblem	nein
Falsche Operandenauswahl	nein
Falsche Operatorauswahl	nein
Fehlerhaftes Operationsergebnis	nein
Fristüberschreitung	nein
Zyklusunterschreitung	nein
Zyklusüberschreitung	nein
Verlorengegangene Datenaktualisierung	nein
Synchronisationsfehler oder unvollständige Datenübertragung	nein
Pufferunter- oder -überläufe	nein
Fehlerhafter Datenfluss (falsche Adressaten, ...)	nein
Duplizierte Daten	nein
Durch Fehler oder Störungen verfälschte Daten	ja (IP)
Fehlerhafter Datenzugriff (fehlende Zugriffsrechte)	nein
Nutzung nicht initialisierter Daten	begrenzt
Angriffsart	
Gezielt verfälschte Daten	nein
Wiedereinspielungsattacke	nein

IP: Integritätsprüfung

on und verursacht damit keine zusätzlichen Kosten bezogen auf den Laufzeitbedarf des Codes.

Einige Realisierungen von Datentyparchitekturen bieten Integritätsprüfungen von Speicherelementen an, meist in Form von Paritätsbits. Im Gegensatz zu konventionellen Architekturen erfolgt die Integritätsprüfung dabei nicht durch den Speichercontroller, sondern bei der Nutzung der Speicherelemente im Prozessorkern parallel zur Ausführung der Operation. Daher kann diese Fehlerart als voll erkennbar angegeben werden, sofern eine Integritätsprüfung IP durch die Architektur angeboten wird.

Lesender Zugriff auf Datenspeicherelemente, die keine gültigen Daten enthalten, können dann durch Datentyparchitekturen aufgedeckt werden, wenn die Datentypkennung des nicht initialisierten Datenspeicherelements nicht mit der erwarteten Datentypkennung übereinstimmt. Daher ist diese Fehlerart als begrenzt erkennbar zu bewerten.

3.4 Datenstruktur- bzw. Deskriptorarchitekturen

Können neben elementaren Datentypen auch Datenstrukturen durch zusätzliche Beschreibungsinformationen für die Hardware lesbar markiert und identifiziert werden, spricht man nach [39] von einer Datenstrukturarchitektur. Zur Beschreibung von Datenstrukturen definiert Giloi in [39] das sog. DRAMA-Prinzip, wobei DRAMA für „Descriptor Referenced Autonomous Memory Access“ steht. In Form von Vektordeskriptoren werden Datenstrukturen charakterisiert, wobei ein Vektordeskriptor durch das 3-Tupel

Vektordeskriptor := (Typkennung, Basisadresse, Elementanzahl)

dargestellt wird, wodurch der Datentyp der in der Struktur enthaltenen Elemente, die Basisadresse der Struktur im Speicher und die Anzahl der Elemente innerhalb der Struktur festgelegt werden. Architekturen, die Deskriptoren auf Ebene der Hardware unterstützen, werden in [78] als Deskriptorarchitekturen bezeichnet.

3.4.1 Beispiele von Datenstruktur- bzw. Deskriptorarchitekturen

Als Beispiel für Datenstruktur- oder Deskriptorarchitekturen wird hier der Burroughs B5000 vorgestellt.

Kommerziell, historisch

Der Großrechner B5000, produziert von Burroughs ab 1961, hatte nach [82] 48 Bit breite Datenworte mit einem Datentypbit, engl. „tag bit“. Dieses Bit spezifizierte, ob ein Speicherwort Daten oder Befehle enthielt. Ab dem Nachfolger B6000 wurden drei Datentypbits in 51 Bit breiten Datenworten eingesetzt. Neben dieser Datentypkennung kamen Deskriptoren für Datenfelder zum Einsatz, die es erlaubten, sichere Felder mit Bereichsprüfung durch die Hardware zu implementieren. Ein Paritätsbit diente der Integritätsprüfung der Speicherworte.

3.4.2 Evaluation der Datenstrukturarchitekturen

Die Erkennbarkeit der 20 in Kapitel 2.4 vorgestellten Fehler- und Angriffsarten durch die Fehlererkennungsmerkmale von Datenstrukturarchitekturen wird in Tabelle 3.4 gezeigt.

Gegenüber den bei Datentyparchitekturen gezeigten Fehlererkennungsmöglichkeiten bieten Datenstrukturarchitekturen die zusätzliche Fähigkeit, durch die hardwareverständliche Beschreibung von Datenstrukturen wie z. B. Datenfeldern sichere Feldzugriffe zu gestatten und dabei auftretende Fehler sofort zu erkennen. Die dazu notwendigen Prüfungen können durch die Hardware parallel zur Ausführung des eigentlichen Datenzugriffs ausgeführt werden. Dies erlaubt die zuverlässige Erkennbarkeit von Pufferunter- und -überläufen.

3.5 Befähigungsarchitekturen

Um Objekte im Speicher vor unberechtigtem Zugriff zu schützen, können nach [39, 78] so genannte Befähigungen, engl. „Capabilities“, eingesetzt werden. In [39] werden Elemente, die auf ein bestimmtes Objekt zugreifen möchten, also z. B. Rechenprozesse, als Subjekte bezeichnet. Die Zugriffsrechte eines jeden Subjekts auf die vorhandenen Objekte können nach [39] in Form einer Zugriffsrechte- oder Objektschutzmatrix dargestellt werden, siehe Abbildung 3.7. Die Zeilen der Matrix werden dabei als Befähigungslisten, engl. „capability lists“, und die Spalten als Zugriffslisten, engl. „access lists“ bezeichnet.

Während Giloi [39] eine Befähigung als 2-Tupel der Form

Befähigung := (Objektidentifikator, Zugriffrechte)

Tabelle 3.4: Fehlererkennung durch Datenstrukturarchitekturen

Fehlerart	**Erkennbarkeit**
Inkompatible Datentypen	ja
Inkompatible Einheiten	nein
Wertebereichsunter- bzw. -überschreitung	nein
Genauigkeitsproblem	nein
Falsche Operandenauswahl	nein
Falsche Operatorauswahl	nein
Fehlerhaftes Operationsergebnis	nein
Fristüberschreitung	nein
Zyklusunterschreitung	nein
Zyklusüberschreitung	nein
Verlorengegangene Datenaktualisierung	nein
Synchronisationsfehler oder unvollständige Datenübertragung	nein
Pufferunter- oder -überläufe	ja
Fehlerhafter Datenfluss (falsche Adressaten, ...)	nein
Duplizierte Daten	nein
Durch Fehler oder Störungen verfälschte Daten	ja (IP)
Fehlerhafter Datenzugriff (fehlende Zugriffsrechte)	nein
Nutzung nicht initialisierter Daten	begrenzt
Angriffsart	
Gezielt verfälschte Daten	nein
Wiedereinspielungsattacke	nein

IP: Integritätsprüfung

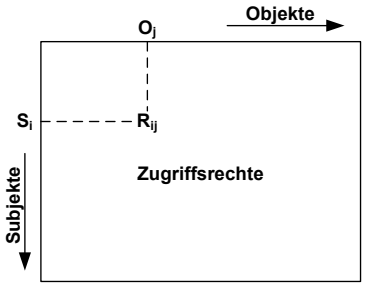


Abbildung 3.7: Zugriffsrechtematrix zum Schutz von Objekten (nach [39])

darstellt, wählt Levy [78] eine graphische Repräsentation, wie in Abbildung 3.8 gezeigt. Der Objektidentifikator ist dabei entweder ein Zeiger auf das Objekt oder ein eindeutiger Bezeichner des Objekts.

Architekturen, die Befähigungen auf Ebene der Hardware unterstützen, werden von Levy [78] Befähigungsarchitekturen oder auch befähigungsbasierte Systeme genannt. Diese Befähigungen ersetzen ungeschützte Zeiger auf die eigentlichen Objekte und verbinden diese mit der Erteilung bestimmter Zugriffsberechtigungen an ein Subjekt [39]:

- keinerlei Zugriff,
- nur lesenden Zugriff,
- nur schreibenden Zugriff oder
- lesenden und schreibenden Zugriff.

Auch die Ausführbarkeit kann nach [78] ein zusätzliches Zugriffsrecht darstellen. Neben der in Abbildung 3.8 gezeigten Möglichkeit, Befähigungen als eigenständige Elemente im Speicher zu hinterlegen, besteht nach [39] auch die Möglichkeit, diese direkt im Objekt zu hinterlegen.

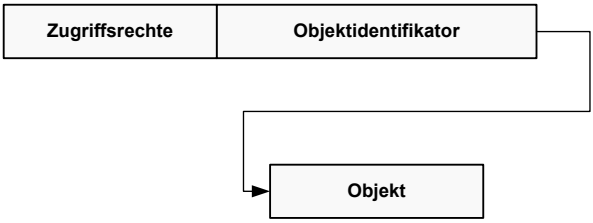


Abbildung 3.8: Aufbau einer Befähigung (nach [78])

Befähigungen dürfen nur durch das Objekt selbst oder eine privilegierte Instanz, z. B. das Betriebssystem, vergeben und verwaltet werden, damit sich Subjekte nicht selbst Zugriff auf Objekte verschaffen können [39, 78, 87]. Dazu werden die Befähigungen entweder in privilegierten Segmenten oder, wenn Datentypkennungen zur Verfügung stehen, entsprechend gekennzeichnet, wodurch die Hardware bei Zugriff auf die Befähigungen die Berechtigung des Zugriffs prüfen kann [39].

3.5.1 Beispiele historischer Befähigungsarchitekturen

Als Beispiele für historische Befähigungsarchitekturen sollen hier drei verschiedene Systeme vorgestellt werden. Während das System/360 von IBM recht rudimentäre und sehr einfache Schutzmechanismen bot, war die Komplexität der hierarchischen Befähigungsstrukturen des CAP Computers und des Intel iAPX 432 hoch bzw. sogar sehr hoch.

3.5.1.1 System/360 von IBM

Kommerziell, historisch

Das System/360, eine Großrechnerfamilie von IBM, hergestellt in den Jahren 1965 bis 1977, diente nach [49] zur Bearbeitung kommerzieller, wissenschaftlicher, kommunikationstechnischer und steuerungstechnischer Aufgaben. Zur Integritätsprüfung wurde nach [49] jedem Byte ein Paritätsbit mit ungerader Parität zugeordnet, das nicht durch die laufenden Programme beeinflussbar war. Bei einem Paritätsfehler wurde ein entsprechender Alarm ausgegeben. Überläufe von arithmetischen Operationen wurden nach [49] durch einen Alarm gemeldet. Nach [92] implementierten manche Modelle einen Isolationsmechanismus durch Bereitstellung eines 4 Bit breiten Schutzschlüssels im Programmstatuswort PSW und der Einteilung des Hauptspeichers in 2 KiB große Blöcke. Jedem Block konnte ein bestimmter Schlüssel zugeordnet werden. Beim Zugriff durch ein Programm wurde der Schlüssel des angesprochenen Speicherblocks mit dem Schutzschlüssel im PSW verglichen. Bei Nichtübereinstimmung wurde ein Ausnahmefehler generiert. Einer der Nachfolger des System/360, das System/38, wird von [78] als erster bedeutender kommerzieller Vertreter der Befähigungsarchitekturen genannt.

3.5.1.2 CAP Computer

Akademisch, historisch

Nach [78] konnte nach 6 Jahren der Entwicklung 1976 der CAP Computer an der Cambridge University in Betrieb genommen werden. Jeder Prozess in CAP führt eine Liste mit Befähigungen mit sich, mit deren Hilfe auf die damit verknüpften Objekte zugegriffen werden kann. Zur Manipulation der Befähigungen standen dedizierte Befehle zur Verfügung. Sowohl Daten als auch Befehle wurden in Objekten gekapselt. Der Zugriff auf die eigentlichen Objekte konnte nur nach Überwindung von mehreren Indirektionsstufen stattfinden. Um diesem Nachteil entgegenzuwirken,

wurden Verdecktspeicher, engl. „Caches“, implementiert, die z. B. bereits ausgewertete Befähigungen zwischenspeicherten [78].

3.5.1.3 Intel iAPX 432

Kommerziell, historisch

Die Firma Intel brachte 1981 nach [39, 78, 87] den objektorientierten iAPX 432 Mikroprozessor auf den Markt, dessen herausragendes Merkmal die Unterstützung bestimmter Betriebssystemfunktionen wie

- Prozessverwaltung,
- Interprozesskommunikation,
- Betriebsmittelverwaltung und
- Verwaltung der Laufzeitumgebung

durch die Hardware darstellte, die durch Befähigungen geschützt werden. Dazu wurde nach [39] eine Hierarchie von Zugriffsbereichen bzw. Objekten definiert, die in Abbildung 3.9 dargestellt werden.

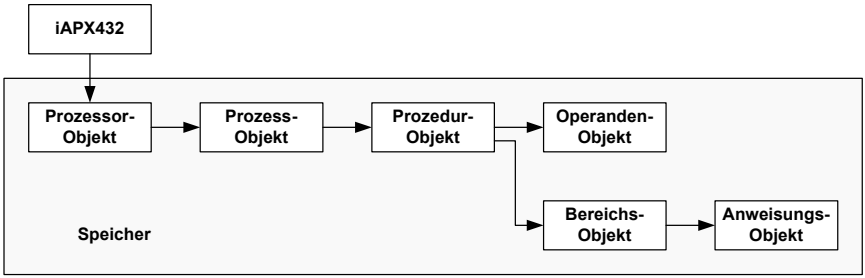


Abbildung 3.9: Vom Intel iAPX 432 durch die Hardware unterstützte Objekte (nach [39])

Nach [39] stand dem kommerziellen Erfolg des iAPX 432 trotz seiner umfassenden Leistungsmerkmale seine ineffiziente Rechenleistung im Weg, da der Zugriff auf jedes Speicherobjekt über mindestens 8 Indirektionsstufen erfolgte. In [23] werden die Auswirkungen der Komplexität der iAPX-432-Architektur auf seine Rechenleistung umfassend diskutiert.

3.5.2 Beispiele moderner Befähigungsarchitekturen

Die Befähigungsarchitekturen erleben – nachdem sie wie Datentyp- und Datenstrukturarchitekturen in Vergessenheit geraten waren – in den letzten 10 Jahren eine Art Renaissance, die Leistungsfähigkeit ihrer Fehlererkennungsmerkmale wurde „wiederentdeckt“. Dabei konzentriert sich die Forschung auf die Datensicherheit, engl. „IT security“, also darauf, wer in welcher Form auf bestimmte Daten zugreifen darf. Einige wichtige Vertreter dieser modernen Befähigungsarchitekturen sollen hier in unterschiedlichem Detailgrad vorgestellt werden.

3.5.2.1 The Loki tagged memory architecture

Akademisch, modern

In [135], einer Veröffentlichung aus dem Jahr 2008, wird Loki beschrieben, eine Befähigungsarchitektur, bei der zum Zweck der Isolation verschiedener virtueller Maschinen im Speicher ein Sicherheitsmonitor als Zwischenschicht zwischen den Betriebssystemen der virtuellen Maschinen und dem Speicher eingefügt wird. Jedem 32 Bit breiten Speicherwort wird ein ebenfalls 32 Bit breiter Beschreiber zugewiesen. Das System wurde mit Hilfe eines synthetisierbaren SPARC-Kerns auf einem FPGA implementiert und das UNIX-ähnliche Betriebssystem HiStar auf die Architektur portiert.

3.5.2.2 Capability Hardware Enhanced RISC Instructions CHERI

Akademisch, modern

Das CHERI-Projekt der Universität Cambridge [123] ist dabei, eine Architektur zur Isolation von Anwendungen zu entwickeln, die feingranulare Isolation von Objekten im Prozessadressraum anbieten soll. Die Architektur wird in Form eines 64-Bit MIPS-Kerns auf einem FPGA evaluiert, als Betriebssystem kommt u.a. FreeBSD zum Einsatz.

3.5.2.3 lowRISC

Akademisch, modern

Das lowRISC-Projekt [11] der Universität Cambridge wurde 2014 ins Leben gerufen, mit dem Ziel, eine Architektur zu etablieren, die eine hohe Immunität gegen

Fehler bzw. Fremdeinflüsse bezüglich des Kontrollflusses aufweist. Um dieses Ziel zu erreichen, sollen Datentypkennungen verwendet werden, die es der Hardware erlauben, bei Zugriff auf ein Datenwort zu prüfen, ob der jeweilige Zugriff zulässig ist. Die zwei Bits, die für diese Kennungen vorgesehen sind, teilen dem Prozessor die in Tabelle 3.5 dargestellten Zugriffsrechte mit.

Tabelle 3.5: Datentypkennungen von lowRISC

Kennungsbits	Bedeutung
00b	jede Art von Zugriff ist gestattet
01b	Generierung eines Ausnahmefehlers bei lesendem Zugriff
10b	Generierung eines Ausnahmefehlers bei schreibendem Zugriff
11b	Generierung eines Ausnahmefehlers bei lesendem oder schreibendem Zugriff

Eine der wichtigsten Funktionen ist dabei nach [11] der Schutz bestimmter Datenstrukturen auf dem Stapelspeicher, die bei typischen Pufferüberläufen unbemerkt überschrieben würden, z. B. die Rücksprungadresse. Diese sollen daher durch entsprechende Kennungen als weder les- noch schreibbar oder zumindest nur-lesbar deklariert werden, wodurch die Hardware fehlerhafte Zugriffe als solche erkennen und eine entsprechende Reaktion auslösen kann. Bei den in lowRISC eingesetzten Kennungsbits handelt es sich nicht um eine Datentypbeschreibung der in den Speicherworten enthaltenen Daten. Stattdessen werden die Zugriffsrechte auf feinstgranularer Ebene pro Speicherwort festgelegt.

Obwohl bei lowRISC keine eigenständigen Befähigungsobjekte im Speicher angelegt und verwaltet werden, so ist die Architektur trotzdem als Beispiel für eine Befähigungsarchitektur zu erachten, da die Zugriffsrechte auf feinstgranularer Ebene für jedes Speicherelement – und damit Objekt – im Speicher durch die zwei Kennungsbits hardwareverständlich spezifiziert und durch die Hardware überwacht werden.

3.5.2.4 SAFE

Akademisch / kommerziell / militärisch, modern

Das SAFE-Projekt [106] ist ein durch die für militärische Forschungsprojekte verantwortliche US-amerikanische Behörde Defense Advanced Research Projects Agency,

kurz DARPA, gefördertes Projekt, das 2010 gestartet wurde und zum Ziel hat, eine auf Datensicherheit spezialisierte Architektur unter Nutzung der Merkmale von Befähigungsarchitekturen zu entwickeln [28].

3.5.2.4.1 Aufbau der Speicherelemente in SAFE

SAFE verwendet eine einheitliche Speicherelementgröße in Speichern, Verdecktspeichern („Caches“) und Registern von 128 Bit für Befehle und Daten. Der Aufbau dieser Elemente wird in [21] beschrieben und in Abbildung 3.10 dargestellt.

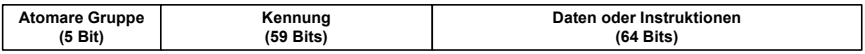


Abbildung 3.10: Aufbau der Speicherelemente in SAFE (nach [21])

Die Speicherelemente werden – da die Kennungen „untrennbar“ mit den eigentlichen Daten oder Befehlen verbunden sind – als „Atome“ bezeichnet. Die obersten 5 Bit identifizieren die „atomic group“, die eine Datentypkennung darstellt, wie sie von Datentyparchitekturen her bekannt ist. Neben verschiedenen Zeigerarten können durch die Datentypkennung Gleitkommazahlen, Ganzzahlen, uninitialisierte Daten und Befehle spezifiziert werden [32]. Es folgt eine 59 Bit breite Kennung, die verschiedenste Überprüfungen erlaubt, die durch die programmierbare Metadatenverarbeitungseinheit, engl. „Programmable Unit for Metadata Processing PUMP“, verarbeitet werden [31]. Die verbleibenden 64 Bit entfallen auf die eigentlichen im Speicherelement enthaltenen Daten oder Instruktionen. Befähigungen werden durch „fat pointers“, also zu Deutsch „fette Zeiger“, realisiert, die neben der Adresse, auf die sie verweisen, noch Angaben über die Basis und die Grenzen des Speicherbereichs auf bzw. in den sie zeigen, beinhalten [21].

3.5.2.4.2 Funktionsweise von PUMP

Die programmierbare Metadatenverarbeitungseinheit, engl. „Programmable Unit for Metadata Processing PUMP“, erlaubt es der Software eines SAFE-Systems, die 59 Bit breite Kennung innerhalb aller Speicherelemente für die Definition beliebiger Regeln zu verwenden, die bei der Verwendung der betroffenen Speicherelemente geprüft werden sollen. Um die Begrenzung der Kennungen auf 59 Bit aufzuheben, ist es möglich, Zeiger auf erweiterte Kennungsfelder im Systemspeicher mit beliebiger

Länge in der Kennung eines Speicherelements unterzubringen. Wichtig ist, dass die besagte Kennung nicht durch die Hardware interpretiert wird, sondern die Hardware diese auf der Suche nach der für die Kennung anzuwendenden Regel als Eingabe nutzt.

Die softwaredefinierten Regeln werden als Funktion

$$\begin{aligned}\text{Regel} := (&\text{K[Programmzählerregister]},\\ &\text{K[aktuelle Instruktion]},\\ &\text{K[Operand in Register 1]},\\ &\text{K[Operand in Register 2]},\\ &\text{K[gelesener Speicherinhalt]})\\ &\mapsto\\ &(\text{K[Programmzählerregister}_{\text{neu}}\text{]},\\ &\text{K[Zieloperand in Register 3 bzw. ein Zielspeicherelement]},\\ &\text{boolesche Aussage über Gestattung der Operation})\end{aligned}$$

festgelegt, wobei K[Komponentenname] jeweils die Kennung einer an der Ausführung einer Operation beteiligten Komponente zurückliefert. Als Ergebnisse liefert die Funktion die neue Kennung für das Programmzählerregister, einen Wert, der – abhängig von der durchzuführenden Operation – in ein Ergebnisregister oder ein Zielspeicherelement geschrieben werden soll, und eine boolesche Aussage darüber, ob die Operation an sich durchgeführt werden darf, zurück. Wird die Ausführung durch die Regel abgelehnt, so wird ein Ausnahmefehler generiert, wodurch die Software eine entsprechende Fehlerbehandlungsroutine ausführen kann. In [33] wird vorgeschlagen, z. B. den betroffenen Softwareprozesses zu beenden oder sichere Werte zurückzugeben.

Die Regeln werden zur beschleunigten Bearbeitung in einem Verdecktspeicher zur Speicherung der PUMP-Regeln – engl. „PUMP rule cache“ – zwischengespeichert. Die Hardware versucht nun bei der Ausführung jeder Instruktion in besagtem Verdecktspeicher eine Regel zu finden, die den Eingabedaten der Funktion in Form des Tupels

$$\begin{aligned}\text{Eingabedaten} := (&\text{K[Programmzählerregister]},\\ &\text{K[aktuelle Instruktion]},\\ &\text{K[Operand in Register 1]},\\ &\text{K[Operand in Register 2]},\\ &\text{K[gelesener Speicherinhalt]})\end{aligned}$$

entspricht. Wird eine entsprechende Funktion im Verdecktspeicher gefunden, so werden ihre Ausgabedaten

Ausgabedaten := (K[Programmzählerregister$_{neu}$],
K[Zieloperand in Register 3 bzw. ein Zielspeicherelement],
boolesche Aussage über die Gestattung der Operation)

direkt aus dem Verdecktspeicher übernommen, ohne nochmals eventuell durch die Kennungen indizierte erweiterte Kennungen im Systemspeicher auszuwerten. Wird keine zum Eingabedatentupel passende Funktion gefunden, so ruft der Prozessor eine Softwareunterbrechungroutine auf, die dann ihrerseits alle Eingabekennungen und die ggf. durch die Kennungen indizierten erweiterten Kennungen im Systemspeicher evaluiert und die Ausgaben der Funktion setzt [33]. Die Eingabedaten werden zusammen mit den Ausgabedaten anschließend von der Prozessorhardware in den Verdecktspeicher übernommen, um beim nächsten Auftreten des identischen Eingabedatentupels die Ausgaben ohne die Notwendigkeit des Aufrufs der Softwareunterbrechungroutine nutzen zu können.

3.5.2.4.3 Einsatzszenarien von PUMP

Als Einsatzszenarien von PUMP werden

- Datentypkennungen nach dem Vorbild typischer Datentyp- bzw. -strukturarchitekturen [31, 33],
- abgeleitete Datentypen [31, 32],
- Befähigungen zur Regelung von Zugriffen auf Speicherobjekte, nach Vorbild von Befähigungsarchitekturen [31],
- Speicherschutz und -isolation [31, 33],
- Kontrollflussüberwachung [31, 33] und
- Färbungen [31, 33], engl. „Taints“, mit deren Hilfe sich ungeprüfte und damit potenziell gefährliche Nutzereingaben markieren lassen,

vorgeschlagen, wobei durch den flexiblen Aufbau der PUMP-Kennungen auch beliebige Kombinationen dieser Merkmale möglich sind.

3.5.2.4.4 Evaluation von SAFE

SAFE ist aufgrund der Verwendung der „fetten Zeiger“ als Befähigungarchitektur zu klassifizieren. Die Verwendung der Kennungen ist – im Gegensatz zu anderen modernen Befähigungsarchitekturen – durchgängig: sie sind in allen Speichern, Verdecktspeichern und Registern vorhanden und nicht nur in Teilen der datenspeichernden und -verarbeitenden Instanzen.

PUMP ist – trotz seiner großen Flexibilität, also der Möglichkeit, verschiedenste Kennungsarten zu realisieren – als softwarebasiertes Kennungssystem von erhöhter Komplexität zu verstehen, da die Hardware hier nur die Zwischenspeicherung der Regelfunktionen und die Anwendung der Ergebnisse dieser Funktionen vornimmt, die Software jedoch die Kennungen interpretieren und eine entsprechende Regelfunktion generieren muss. Weiterhin schwächt die Notwendigkeit des Aufrufs einer Softwarefunktion beim Antreffen einer bislang nicht zwischengespeicherten Regelfunktion die Vorhersagbarkeit des zeitlichen Verhaltens des Systems. Auch wenn nach einer gewissen Zeit nach Start eines Systems alle bislang benötigten Regelfunktionen im Regelverdecktspeicher zwischengespeichert sein sollten, ist davon auszugehen, dass

- mit steigender Anzahl an zwischenzuspeichernden Regelfunktionen der Verdecktspeicher zu einem bestimmten Zeitpunkt voll ist und dann selten genutzte Regelfunktionen aus diesem entfernt werden müssen, um Platz für die neuen Regelfunktionen zu schaffen und
- selten auftretende Kombinationen aus Kennungen im entscheidenden Moment nicht zwischengespeichert sein werden.

Durch die Möglichkeit, beliebig große erweiterte Kennungen im Speicher abzulegen, können zwar viele verschiedene Kennungsarten realisiert und neue Kennungsarten ohne Hardwareänderungen hinzugefügt werden, es ist jedoch zu berücksichtigen, dass die Kombination mehrerer verschiedener Kennungsarten zu einer beträchtlichen Anzahl verschiedener Regelfunktionen führt. Weiterhin können mit PUMP keine Kennungen realisiert werden, die dynamische Komponenten wie die Zeit einbeziehen, also z. B. Fristen.

Dadurch, dass die Hardware nur die Kennungen mit den ggf. darin enthaltenen Zeigern auf erweiterte Kennungsstrukturen im Speicher zum Auffinden der gültigen Regelfunktion nutzt, nicht jedoch die erweiterten Kennungen im Speicher einbezieht, kann es zu Inkonsistenzen kommen, wenn die erweiterten Kennungen verändert werden, im Regelverdecktspeicher aber noch die nun veralteten Regelfunktionen gespeichert sind und angewendet werden. Bei jeder Änderung der erweiterten Kennungen

wäre es also notwendig, alle betroffenen Regelfunktionen aus dem Verdecktspeicher zu löschen bzw. den gesamten Regelverdecktspeicher zu löschen, mit entsprechenden zeitlichen Auswirkungen für die erneute Evaluation der betroffenen bzw. aller Kennungen durch die Software, bis die benötigten Regeln wieder im Verdecktspeicher zwischengespeichert sind.

Noch größer ist die Gefahr von Inkonsistenzen, wenn Daten zwischen Systemen oder Systemkomponenten übertragen werden, da

- zusätzlich zum eigentlichen Datenspeicherelement die an anderen Stellen im Speicher abgelegten erweiterten Kennungen ebenfalls übertragen, im Speicher der Zielkomponente abgelegt und die Zeiger innerhalb der Kennungen der Datenspeicherelemente dann entsprechend angepasst werden müssten und
- die Interpretation der verschiedenen Kennungsarten durch die Software in allen beteiligten Komponenten identisch sein muss, da in der Hardware der Komponenten kein Wissen über die Bedeutung der Kennungen vorliegt. Entsprechend können auch Fehler bzgl. der Deutung der Kennungsinhalte durch die Hardware nicht erkannt werden.

3.5.3 Evaluation der Befähigungsarchitekturen

Wie auch bei den vorhergehenden Architekturarten werden die Befähigungsarchitekturen auf Basis der 20 in Kapitel 2.4 identifizierten Fehler- und Angriffsarten bewertet. In Tabelle 3.6 wird das Ergebnis dieser Bewertung dargestellt.

Zusätzlich zu den Fehlererkennungsmöglichkeiten der Datenstrukturarchitekturen bieten Befähigungsarchitekturen die Möglichkeit, durch die hardwareverständliche Beschreibung von Zugriffsrechten von Subjekten auf Objekte eine feingranulare und effektive Prüfung von festgelegten Zugriffsrechten durchzuführen. Die dazu notwendigen Prüfungen können durch die Hardware parallel zur Ausführung des eigentlichen Datenzugriffs ausgeführt werden. Dies erlaubt die Erkennung fehlerhafter Datenzugriffe und im begrenzten Umfang auch die Aufdeckung falscher Datenflüsse im System, falls die Zugriffsrechte dabei mit übertragen werden.

3.6 Datenflussarchitekturen

Datenflussarchitekturen unterscheiden sich grundlegend von konventionellen kontrollflussorientierten Architekturen und lassen durch ihre Bezeichnung vermuten,

Tabelle 3.6: Fehlererkennung durch Befähigungsarchitekturen

Fehlerart	**Erkennbarkeit**
Inkompatible Datentypen	ja
Inkompatible Einheiten	nein
Wertebereichsunter- bzw. -überschreitung	nein
Genauigkeitsproblem	nein
Falsche Operandenauswahl	nein
Falsche Operatorauswahl	nein
Fehlerhaftes Operationsergebnis	nein
Fristüberschreitung	nein
Zyklusunterschreitung	nein
Zyklusüberschreitung	nein
Verlorengegangene Datenaktualisierung	nein
Synchronisationsfehler oder unvollständige Datenübertragung	nein
Pufferunter- oder -überläufe	ja
Fehlerhafter Datenfluss (falsche Adressaten, ...)	begrenzt
Duplizierte Daten	nein
Durch Fehler oder Störungen verfälschte Daten	ja (IP)
Fehlerhafter Datenzugriff (fehlende Zugriffsrechte)	ja
Nutzung nicht initialisierter Daten	begrenzt
Angriffsart	
Gezielt verfälschte Daten	nein
Wiedereinspielungsattacke	nein

IP: Integritätsprüfung

neben der Spezialisierung auf Datenflüsse auch entsprechende Fehlererkennungsmaßnahmen zu bieten. Darum soll diese Form der Architektur an dieser Stelle vorgestellt werden.

3.6.1 Funktionsweise von Datenflussarchitekturen

Datenflussarchitekturen weichen in ihrem Aufbau grundsätzlich von kontrollflussorientierten Architekturen ab: sie besitzen keinen Befehlszeiger und keinen programm- oder komponentenweiten Arbeitsspeicher. Sie bilden den Aufbau von Datenflussgraphen in natürlicher Weise als vernetzte Funktionsblöcke ab, wie dies z. B. von Funktionsplänen speicherprogrammierbarer Steuerungen bekannt ist. Jeder Funktionsblock weist eine bestimmte Anzahl von Ein- und Ausgängen auf. Eine Neuberechnung der Ausgabewerte erfolgt nach [39], wenn an jedem Eingang eine Marke, engl. „Token“, und damit ein zu verarbeitender Operand anliegt.

In Abbildung 3.11 wird ein Datenflussgraph gezeigt, der das Ergebnis z der Gleichung

$$z = (w + v) \cdot (x - y)$$

berechnet.

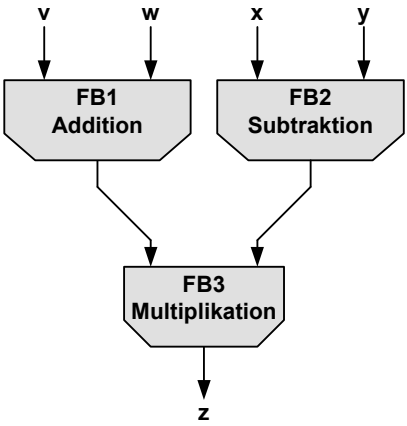


Abbildung 3.11: Beispiel eines Datenflussgraphen mit 3 Funktionsblöcken

Bei der Berechnung des Ergebnisses z kommen drei Funktionsblöcke zum Einsatz: in Funktionsblock FB 1 wird die Addition der anliegenden Operanden realisiert, in FB 2 die Subtraktion und in FB 3 die Multiplikation. Während eine kontrollflussorientierte Architektur die Addition und die Subtraktion sequenziell, also nacheinander ausführen muss, kann hier eine datenflussorientierte Architektur die Berechnungen in den Funktionsblöcken FB 1 und FB 2 parallelisiert ausführen und somit früher das Ergebnis z berechnen.

Grundsätzlich werden nach [39] zwei Arten von Datenflussarchitekturen unterschieden:

- statische Datenflussarchitekturen, bei denen die vorhandenen Prozessoren bzw. Verarbeitungseinheiten bestimmten Operationen fest zugeordnet sind und
- dynamische Datenflussarchitekturen, bei denen die Marken gesammelt werden und von einer Steuerungseinheit auf die vorhandenen Prozessoren bzw. Verarbeitungseinheiten verteilt werden, die dann die gewünschte Operation ausführen.

Eine Datenflussarchitektur setzt nicht zwingend die Verwendung spezialisierter Prozessoren voraus, die die Merkmale einer Datenflussarchitektur nativ unterstützen. Es können konventionelle Mikroprozessoren eingesetzt werden, die dann die Datenflüsse zwischen den Funktionsblöcken steuern und die Berechnungen innerhalb der Funktionsblöcke in gewohnter Weise sequenziell abarbeiten. Ein Beispiel dafür ist der Einsatz des Intel 8088 – einer aus Kostengründen nur mit einem 8-Bit-Datenbus ausgestatteten Variante des Intel 8086 – in [76].

3.6.2 Evaluation von Datenflussarchitekturen

Datenflussarchitekturen haben den Vorteil eines problemorientierten Architekturaufbaus, d. h. sie sind auf die Aufgabe spezialisiert, Daten entgegenzunehmen, diese zu verarbeiten und die Ergebnisse auszugeben, ohne dabei den Umweg über kontrollflussorientierte Datenverarbeitungskonstrukte gehen zu müssen. Weiterhin erlaubt die feingranulare Unterteilung der Datenverarbeitungsaufgaben einen hohen Grad an Parallelisierung, da theoretisch alle parallel ausführbaren Datenverarbeitungsschritte auch tatsächlich zeitlich simultan ausgeführt werden können.

Allerdings bieten Datenflussarchitekturen keine spezialisierten Fehlererkennungsmaßnahmen, die Fehler im Datenfluss erkennen könnten. Daher ergibt sich keine Erkennbarkeit der in Kapitel 2.4 vorgestellten 20 Fehler- und Angriffsarten, wie in

Tabelle 3.7: Fehlererkennung durch Datenflussarchitekturen

Fehlerart	**Erkennbarkeit**
Inkompatible Datentypen	nein
Inkompatible Einheiten	nein
Wertebereichsunter- bzw. -überschreitung	nein
Genauigkeitsproblem	nein
Falsche Operandenauswahl	nein
Falsche Operatorauswahl	nein
Fehlerhaftes Operationsergebnis	nein
Fristüberschreitung	nein
Zyklusunterschreitung	nein
Zyklusüberschreitung	nein
Verlorengegangene Datenaktualisierung	nein
Synchronisationsfehler oder unvollständige Datenübertragung	nein
Pufferunter- oder -überläufe	nein
Fehlerhafter Datenfluss (falsche Adressaten, ...)	nein
Duplizierte Daten	nein
Durch Fehler oder Störungen verfälschte Daten	nein
Fehlerhafter Datenzugriff (fehlende Zugriffsrechte)	nein
Nutzung nicht initialisierter Daten	nein
Angriffsart	
Gezielt verfälschte Daten	nein
Wiedereinspielungsattacke	nein

Tabelle 3.7 gezeigt wird. Bei statischen Datenflussarchitekturen können einige der Fehlerarten theoretisch gar nicht auftreten, als Beispiel sei hier die Verwendung nicht initialisierter Variablen genannt. Dies könnte entsprechend in der Tabelle vermerkt werden. Allerdings ist davon auszugehen, dass bei dynamischen Datenflussarchitekturen durch das Sammeln der Marken und der dynamischen Verteilung von auszuführenden Operationen an die universellen Verarbeitungseinheiten wieder alle genannten Fehlerarten auftreten können. Daher sollen alle Fehler- und Angriffsarten weiterhin als nicht erkennbar klassifiziert werden.

Ein Vorschlag für die Gestaltung einer Datenflussarchitektur, die die genannten Fehler- und Angriffsarten in großem Umfang erkennen kann, wird in Kapitel 4.10 unterbreitet.

3.7 Die inhärent sichere Mikroprozessorarchitektur ISMA

In [125] wurden auf Basis von Normanforderungen aus der IEC 61508 [51–53] und verschiedenen Quellen zu häufigen Ursachen von Softwarefehlern 23 Anforderungen gewonnen, die an eine inhärent sichere Mikroprozessorarchitektur zu stellen sind. Zur Erfüllung dieser Anforderungen wurde auf Basis der bereits beschriebenen, in Vergessenheit geratenen Architekturarten Datentyp-, Datenstruktur- und Befähigungsarchitekturen eine neue Architektur vorgestellt, die weit über bekannte Ansätze hinausgeht. Sie ist konventionellen Architekturen in Bezug auf die Erfüllung der genannten Anforderungen deutlich überlegen, wie in Abbildung 3.12 gut zu erkennen ist.

Die x86-Architektur erfüllt nur 5 der 23 Anforderungen komplett, die ARM-Architektur sogar nur 4, während ISMA 22 der Anforderungen voll erfüllt.

3.7.1 Aufbau der Datenspeicherelemente in ISMA

Bei ISMA besitzen alle Befehls- und Datenspeicherelemente, sowie alle Registerinhalte den in Abbildung 3.13 gezeigten identischen Grundaufbau. Die Speicherelemente haben eine einheitliche Breite von 128 Bit, wobei 64 Bit für Sicherungs- und Verwaltungsdaten und 64 Bit auf die eigentlichen Daten bzw. Operandenadressen entfallen.

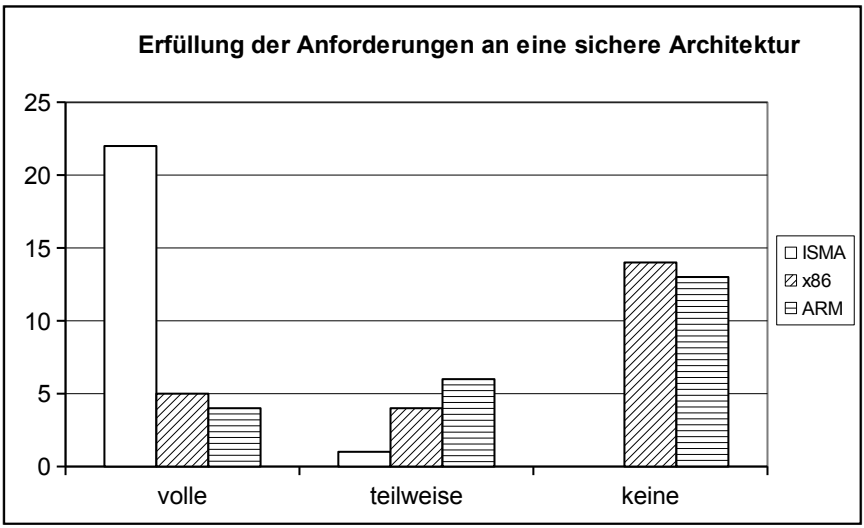


Abbildung 3.12: Gegenüberstellung von ISMA, x86 und ARM

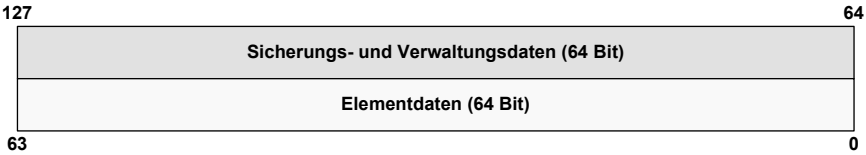
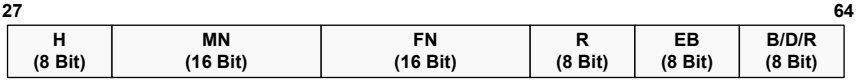


Abbildung 3.13: Aufbau aller Speicherelemente und Registerinhalte bei ISMA

Die Sicherungs- und Verwaltungsdaten, detailliert dargestellt in Abbildung 3.14, enthalten

- einen Erweiterten-(128,120)-Hamming-Code H zur Integritätsprüfung,
- die Angabe der Modul- und Funktionsnummer, MN und FN, der die Daten oder Instruktionen innerhalb der Software zugeordnet sind,
- den Elementbeschreiber EB, der den genauen Typ und weitere Eigenschaften des Speicherelements identifiziert und
- basierend auf dem jeweiligen Speicherelementtyp entweder einen Befehlscode, eine Datentyp- oder eine Registerkennung.

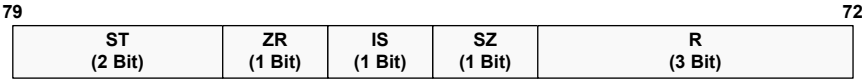


H	**Erweiterter (128, 120)-Hamming-Code**
MN	**Modulnummer**
FN	**Funktionsnummer**
R	**reserviert für zukünftige Erweiterungen, alle Bits sind auf Null zu setzen**
EB	**Elementbeschreiber**
B/D/R	**Befehlscode / Datentypkennung / Registerkennung**

Abbildung 3.14: Aufbau der Sicherungs- und Verwaltungsdaten

Der Elementbeschreiber EB spezifiziert weitere Eigenschaften des Speicherelements, sowie der darin enthaltenen Daten. Sein Aufbau wird in Abbildung 3.15 gezeigt. Der Speicherelementtyp ST gibt an, ob es sich bei dem vorliegenden Speicherelement um ein Befehls-, ein Datenspeicherelement oder einen Registerinhalt handelt. Das Zugriffsrechtebit ZR legt fest, ob schreibend auf die enthaltenen Daten zugegriffen werden darf. Der Initialisierungsstatusbeschreiber IS gibt an, ob ein Speicherelement gültige Daten enthält, die gelesen werden können. Nur für Befehlsspeicherelemente wird die Sprungzielmarkierung SZ genutzt, die angibt, ob das jeweilige Speicherelement ein gültiges Ziel eines Sprungs ist.

ISMA bietet keine Merkmale zur Nutzung von Unterbrechungen an, um ein Höchstmaß an zeitlicher Vorhersagbarkeit zu bieten. Zur Verwaltung auftretender externer Ereignisse und generierter interner Zeitereignisse, sowie zur Überwachung der Bearbeitungsfristen der jeweiligen Ereignisse nutzt ISMA eine vom Hauptprozessor getrennte Ereignisverwaltungseinheit EVE, dargestellt in Abbildung 3.16.



ST	**Speicherelementtyp**
ZR	**Zugriffsrechte**
IS	**Initialisierungsstatus**
SZ	**Sprungzielmarkierung**
R	**reserviert für zukünftige Erweiterungen, alle Bits sind auf Null zu setzen**

Abbildung 3.15: Aufbau des Elementbeschreibers EB

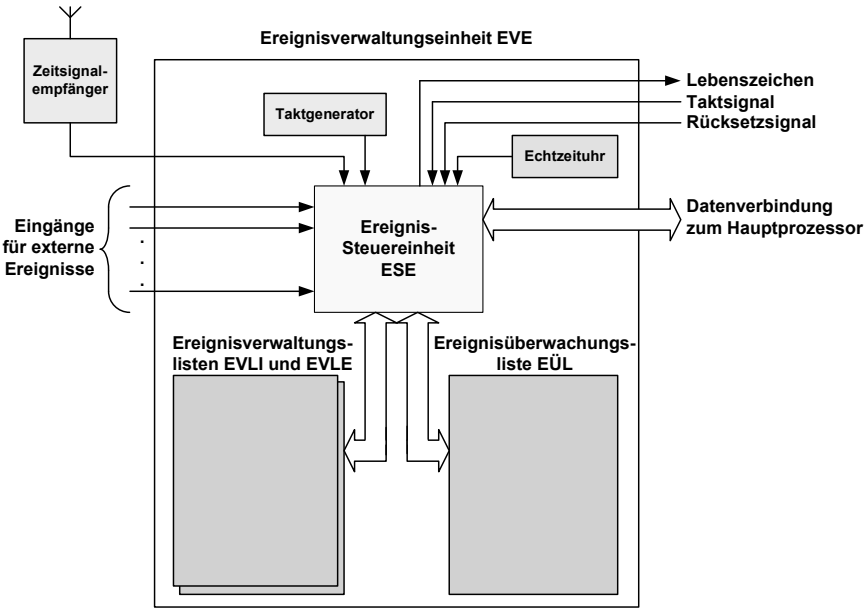


Abbildung 3.16: Aufbau der Ereignisverwaltungseinheit EVE

Diese nutzt verschiedene Listen zur Verwaltung der internen und externen Ereignisse, sowie zur Überwachung der Einhaltung von deren Fristen nach deren Eintreten.

Der Hauptprozessor fragt in regelmäßigen Abständen bei der EVE an, ob Ereignisse seit der letzten Abfrage aufgetreten sind und erhält ggf. den Ereignisidentifikator des Ereignisses mit der kürzesten verbleibenden Frist zusammen mit der Angabe der verbleibenden Bearbeitungszeit.

3.7.2 Evaluation von ISMA

Da ISMA nicht speziell für die Datenflussüberwachung entworfen wurde, kann sie nur 5 der 20 in Kapitel 2.4 vorgestellten Fehler- und Angriffsarten erkennen, wie in Tabelle 3.8 dargestellt. Die Inkompatibilität der Datentypen von Operanden kann durch die Datentypkennung DT erkannt werden.

Fristüberschreitungen bei der Bearbeitung aufgetretener interner und externer Ereignisse werden durch die EVE zwar erkannt, es werden jedoch keine Übertragungs- und Verarbeitungszeiten innerhalb der Sensoren oder Aktoren betrachtet, sondern nur die Bearbeitungszeit innerhalb von ISMA selbst, weshalb diese Fehlerart als nur begrenzt erkennbar aufgeführt wird. Pufferunter- und -überläufe – also Zugriffe auf Datenfelder mit Indizes, die außerhalb der Feldgrenzen liegen – werden durch die Nutzung von Felddatentypen mit dedizierten Feldzugriffsbefehlen durch die Hardware bei Prüfung des Feldindex erkannt. Fehlerhafter Datenfluss, z. B. durch Adressierungsfehler bei Datenübertragungen oder Speicherzugriffen kann durch ISMA nur dann erkannt werden, wenn die Modul- und Funktionsnummern MN und FN, die Datentypkennungen DT oder das Zugriffsrechtebit ZR den Fehler aufdecken, daher wird diese Fehlerart als begrenzt erkennbar angegeben. Durch Störungen verfälschte Daten können innerhalb der Fehlererkennungsgrenzen des (128,120)-Hamming-Codes H zuverlässig erkannt werden. Zugriffe auf Daten, für die einem Softwaremodul die notwendigen Zugriffsrechte fehlen, kann ISMA durch die Prüfung der in den Daten angegebenen Modul- und Funktionsnummern MN und FN, sowie das Zugriffsrechtebit ZR erkennen. Versuche, lesend auf Datenspeicherelemente zuzugreifen, die keine gültigen Daten enthalten, werden durch das Initialisierungsstatusbit IS aufgedeckt.

Tabelle 3.8: Fehlererkennung durch ISMA

Fehlerart	**Erkennbarkeit**
Inkompatible Datentypen	ja
Inkompatible Einheiten	nein
Wertebereichsunter- bzw. -überschreitung	nein
Genauigkeitsproblem	nein
Falsche Operandenauswahl	nein
Falsche Operatorauswahl	nein
Fehlerhaftes Operationsergebnis	nein
Fristüberschreitung	begrenzt
Zyklusunterschreitung	nein
Zyklusüberschreitung	nein
Verlorengegangene Datenaktualisierung	nein
Synchronisationsfehler oder unvollständige Datenübertragung	nein
Pufferunter- und -überläufe	ja
Fehlerhafter Datenfluss (falsche Adressaten, ...)	begrenzt
Duplizierte Daten	nein
Durch Fehler oder Störungen verfälschte Daten	ja
Fehlerhafter Datenzugriff (fehlende Zugriffsrechte)	ja
Nutzung nicht initialisierter Daten	ja
Angriffsart	
Gezielt verfälschte Daten	nein
Wiedereinspielungsattacke	nein

3.8 Application Data Integrity ADI bzw. Silicon Secured Memory SSM

Oracle stellte 2014 eine neue Generation der SPARC-Prozessoren vor, den SPARC M7 [95]. Dieser beinhaltet als besonderes neues Merkmal die Application Data Integrity, kurz ADI [93], bei der Datenblöcke mit Versionsnummernkennungen versehen werden, die durch die Hardware überprüft werden können. Es handelt sich daher beim SPARC-M7-Prozessor um eine „Tagged Memory“ Architektur, also eine Architektur mit Kennungen. Da die Versionskennung keine Datentypkennung darstellt, passt der deutsche Begriff „Datentyparchitektur“ hier nicht richtig, weshalb der SPARC M7 hier getrennt von den Datentyparchitekturen vorgestellt wird. ADI wurde 2015 in Silicon Secured Memory, kurz SSM, umbenannt [43]. In dieser Arbeit sollen jedoch beide Bezeichnungen erwähnt werden, da ADI in bestehender Literatur erwähnt wird, so z. B. in [114].

3.8.1 Funktion von ADI bzw. SSM

Bei ADI bzw. SSM kann die Software für Datenblöcke mit einer Größe von 64 Byte im Speicher sogenannte „Versionsnummern“ festlegen, also eine Kennung, die den Versionsstand der Daten angibt [93]. Die Kennung hat dabei eine Breite von 4 Bit [116]. In den zum Speicherzugriff genutzten Zeigern wird der erwartete Versionsstand der Daten spezifiziert. Bei lesenden oder schreibenden Zugriffen auf die Datenspeicherelemente über diese Zeiger prüft die Hardware, wie in Abbildung 3.17 dargestellt, ob die in der Versionskennung der Daten angegebene Versionsnummer mit der in den zum Datenzugriff verwendeten Zeigern spezifizierten Versionsnummer übereinstimmt. Ist dies nicht der Fall, so wird ein Ausnahmefehler generiert.

3.8.2 Evaluation von ADI bzw. SSM

Durch das Hinzufügen von Versionskennungen zu Datenblöcken von 64 Byte Größe kann ADI bzw. SSM gegenüber konventionellen Architekturen zusätzliche Fehler aufdecken. In Tabelle 3.9 wird die Erkennbarkeit der 20 in Kapitel 2.4 identifizierten Fehler- und Angriffsarten durch ADI bzw. SSM dargestellt.

ADI bzw. SSM kann durch die Versionskennungen in den Datenblöcken

- Synchronisierungsfehler und unvollständige Datenübertragungen,

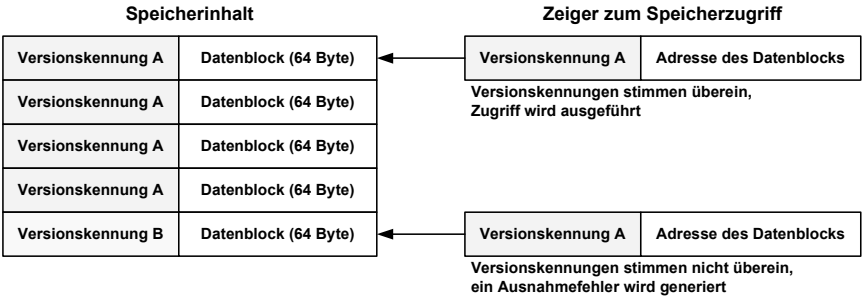


Abbildung 3.17: Funktionsweise von ADI bzw. SSM (nach [93])

Tabelle 3.9: Fehlererkennung durch ADI bzw. SSM

Fehlerart	**Erkennbarkeit**
Inkompatible Datentypen	nein
Inkompatible Einheiten	nein
Wertebereichsunter- bzw. -überschreitung	nein
Genauigkeitsproblem	nein
Falsche Operandenauswahl	nein
Falsche Operatorauswahl	nein
Fehlerhaftes Operationsergebnis	nein
Fristüberschreitung	nein
Zyklusunterschreitung	nein
Zyklusüberschreitung	nein
Verlorengegangene Datenaktualisierung	(ja)
Synchronisationsfehler oder unvollständige Datenübertragung	(ja)
Pufferunter- oder -überläufe	(ja)
Fehlerhafter Datenfluss (falsche Adressaten, ...)	nein
Duplizierte Daten	(ja)
Durch Fehler oder Störungen verfälschte Daten	nein
Fehlerhafter Datenzugriff (fehlende Zugriffsrechte)	nein
Nutzung nicht initialisierter Daten	(ja)
Angriffsart	
Gezielt verfälschte Daten	nein
Wiedereinspielungsattacke	nein

- Pufferunter- und -überläufe,
- duplizierte Daten und
- die Nutzung nicht initialisierter Daten

als Fehler erkennen. Da die Prüfung der Versionsangabe der Daten mit der im Zeiger spezifizierten erwarteten Version parallel zur Ausführung des Lese- oder Schreibbefehls erfolgt, entsteht kein oder nur ein kleiner zusätzlicher Laufzeitbedarf beim Einsatz von ADI bzw. SSM. In [43] wird die Erkennbarkeit des Heartbleed-Fehlers durch ADI bzw. SSM beworben, der in dieser Arbeit in Kapitel 1.1.4 vorgestellt wurde. Ein Nachteil von ADI bzw. SSM ist die Notwendigkeit, die Versionsstände in den Daten durch Software zu setzen. Durch die Angabe des absoluten erwarteten Versionsstands in den Zeigern ist es nicht direkt möglich, relative temporale Beziehungen zwischen Datenspeicherelementen zu prüfen. Deswegen muss die Software bei jeder Änderung des erwarteten Versionsstands diesen in allen verwendeten Zeigern neu setzen. Ein weiterer Nachteil ist die Spezifikation der Version für 64 Byte lange Datenblöcke [93], wodurch es nicht möglich ist, kleineren Dateneinheiten, also z. B. einzelnen Datenworten eine Versionskennung zuzuweisen. Aufgrund dieser Nachteile wurden die entsprechenden Fehlerarten als erkennbar, aber mit Einschränkungen bewertet.

3.9 Dynamic Dataflow Verification DDFV

Die dynamische Datenflussprüfung, engl. „Dynamic Dataflow Verification DDFV“, ist ein hardwareunterstütztes signaturbasiertes Fehlererkennungsverfahren, dass 2007 in [85] vorgestellt wurde.

3.9.1 Funktion der dynamischen Datenflussprüfung

Bei der dynamischen Datenflussprüfung wird ein Programm in kleine Einheiten zerlegt, innerhalb derer der Datenfluss durch die Register überwacht wird. Am Anfang eines jeden Datenblocks steht dabei eine spezielle Instruktion, die dem Prozessor den Abschluss der bisherigen überwachten Programmeinheit und den Beginn der neuen anzeigt. Gleichzeitig führt dieser Befehl die erwartete Datenflusssignatur für die nun beginnende zu überwachende Programmeinheit mit sich. Diese erwartete Signatur wird am Ende der Programmeinheit, also beim Antreffen der nächsten Spezialinstruktion, durch den Prozessor mit der während der Ausführung berechneten

Signatur verglichen. Bei fehlender Übereinstimmung liegt ein Fehler im Datenfluss vor und ein Ausnahmefehler wird generiert.

In die Signatur fließen die Identifikatoren der ausgeführten Instruktionen sowie der dabei verwendeten Register ein. Somit kann dieses Verfahren fälschlicherweise ausgeführte Instruktionen und fehlerhafte Registernutzung aufdecken.

3.9.2 Evaluation der dynamischen Datenflussprüfung

In [85] wird gezeigt, dass die Implementierung der dynamischen Datenflussprüfung nur moderate Anpassungen am Prozessorkern nötig macht. Die Programmausführung wird nur um wenige Prozent gegenüber der Ausführungszeit ohne Datenflussüberwachung verlangsamt. Durch die feingranulare Prüfung des Datenflusses werden in einem Großteil des Prozessors auftretende Fehler aufgedeckt. Die Bewertung anhand der 20 in Kapitel 2.4 vorgestellten Fehler- und Angriffsarten wird in Tabelle 3.10 dargestellt.

DDFV erlaubt die zuverlässige Erkennung der Nutzung falscher Operatoren und Registerinhalte in arithmetischen Operationen. Allerdings fehlt die Einbeziehung von Arbeitsspeicherinhalten und der Datenübermittlung über Systemteile hinweg. Das Verfahren kann z. B. einen Fehler, bei dem ein Schreibvorgang in den Arbeitsspeicher fehlschlägt, also den Verlust der Aktualisierung eines Datenspeicherelements, nicht feststellen, sondern nur, ob in einem überwachten Codefragment die richtigen Operationen auf die richtigen – in Registern liegenden – Operanden angewendet wurde. Daher können diese Fehlerarten nur als begrenzt erkennbar gewertet werden. Verzichtet man bei der Realisierung einer Prozessorarchitektur auf die Nutzung arithmetischer Register, wie es in [115] zur Vereinfachung der Übersetzer gefordert wurde, so erweist sich DDFV als nutzlos, da es nicht über die Registerebene hinaus anwendbar ist. Weiterhin hat die DDFV auch den Nachteil, den Gollub in [41] bei Signaturverfahren zur Kontrollflussüberwachung identifizierte: erst am Ende eines überwachten Instruktionsblocks kann ein aufgetretener Fehler erkannt werden und damit ggf. erst einige Takte später.

3.10 Fehlererkennung durch AN(BD)-Kodierung

Eine sehr leistungsfähige Kodierung zur Erkennung von Fehlern bei der Verarbeitung von Daten ist die ANBD-Kodierung. Dabei handelt es sich um eine arithmetische Kodierung basierend auf der Arbeit von Brown [13], die als AN-Kodierung

Tabelle 3.10: Fehlererkennung durch DDFV

Fehlerart	**Erkennbarkeit**
Inkompatible Datentypen	nein
Inkompatible Einheiten	nein
Wertebereichsunter- bzw. -überschreitung	nein
Genauigkeitsproblem	nein
Falsche Operandenauswahl	begrenzt
Falsche Operatorauswahl	ja
Fehlerhaftes Operationsergebnis	nein
Fristüberschreitung	nein
Zyklusunterschreitung	nein
Zyklusüberschreitung	nein
Verlorengegangene Datenaktualisierung	nein
Synchronisationsfehler oder unvollständige Datenübertragung	nein
Pufferunter- oder -überläufe	nein
Fehlerhafter Datenfluss (falsche Adressaten, ...)	begrenzt
Duplizierte Daten	nein
Durch Fehler oder Störungen verfälschte Daten	nein
Fehlerhafter Datenzugriff (fehlende Zugriffsrechte)	nein
Nutzung nicht initialisierter Daten	nein
Angriffsart	
Gezielt verfälschte Daten	nein
Wiedereinspielungsattacke	nein

bekannt ist. Forin erweiterte diese Kodierung in [38] für den Vital Coded Processor VCP um die Adressprüfung B und die Aktualitätsprüfung D zur ANBD-Kodierung.

3.10.1 AN-Kodierung zur Integritätsprüfung von Datenspeicherelementen und arithmetischen Operationen

Bei der AN-Kodierung werden Variablenwerte mit dem Faktor A multipliziert, wodurch das kodierte Ergebnis in der Regel die doppelte Bitbreite der Variable aufweist. Von der Wahl von A hängt der mit der AN-Kodierung erreichbare minimale Hamming-Abstand MHD ab [108]. Die Kodierung ist mathematisch sehr einfach und wird durchgeführt, indem das gewählte A auf den Variablenwert x multipliziert wird, wodurch man die kodierte Variable x_c erhält.

$$x_c = A \cdot x$$

Ist ein kodiertes Wort korrekt, wurde also nicht verfälscht, so ist der Rest der Division des kodierten Werts durch das gewählte A Null.

$$x_c = A \cdot x \equiv 0 \mod A$$

Ist diese Bedingung nicht erfüllt, so liegt eine Verfälschung des kodierten Worts vor oder ein Fehler in der arithmetischen Einheit, die zur Prüfung herangezogen wurde.

AN-Kodierung von arithmetischen Operationen

Neben der reinen Integritätsprüfung von kodierten Worten ist es mittels der AN-Kodierung weiterhin möglich, die Korrektheit von arithmetischen Operationen zu verifizieren. Dabei werden alle Operanden einer arithmetischen Operation entsprechend kodiert und nach Durchführung der Operation kann das Ergebnis analog zur Prüfung der Integrität verifiziert werden, indem der Rest der Division des Ergebnisses durch A geprüft wird.

AN-Kodierung von Additionen und Subtraktionen

Die Durchführung von Additionen und Subtraktionen erfolgt, indem die kodierten Operanden summiert bzw. voneinander subtrahiert werden. Wenn die Operation ohne Fehler durchgeführt wurde, dann ergibt das Ergebnis modulo A Null.

$$x_c \pm y_c = A \cdot x \pm A \cdot y = A(x \pm y) \equiv 0 \mod A$$

AN-Kodierung von Multiplikationen

Auch bei der Multiplikation werden einfach die kodierten Operanden multipliziert und das Ergebnis kann ebenfalls durch Prüfung der Bedingung, ob es modulo A Null ergibt, überprüft werden.

$$x_c \cdot y_c = A \cdot x \cdot A \cdot y = A^2 \cdot x \cdot y \equiv 0 \mod A$$

Alternativ kann auf die Kodierung des zweiten Operanden verzichtet werden, um die Bitbreite des Ergebnisses klein zu halten, ohne dabei die Prüfbarkeit des Ergebnisses einzuschränken. Allerdings ist dann natürlich keine Integritätsprüfung des betroffenen Operanden vor der Multiplikation mehr möglich.

$$x_c \cdot y = A \cdot x \cdot y \equiv 0 \mod A$$

AN-Kodierung von Divisionen

Die Division von AN-kodierten Operanden weicht von den bisherigen Beispielen ab: Wenn beide Operanden kodiert sind, wird bei der Durchführung der Division die Integritätsprüfung A abdividiert und das Ergebnis liegt somit unkodiert vor. Weder Korrektheit des Ergebnisses, noch dessen Integrität können geprüft werden.

$$\frac{x_c}{y_c} = \frac{A \cdot x}{A \cdot y} = \frac{x}{y} \not\equiv 0 \mod A$$

Um die Verifikation des Ergebnisses zu ermöglichen, kann z. B. nur ein Operand kodiert werden.

$$\frac{x_c}{y} = \frac{A \cdot x}{y} = A \cdot \frac{x}{y} \equiv 0 \mod A$$

3.10.2 ANB-Kodierung: Hinzufügen der Adressprüfung B

Um zu prüfen, ob die richtigen Operanden für eine arithmetische Operation herangezogen wurden, schlug Forin in [38] vor, die Adresse bzw. eine Signatur B des Operanden dem kodierten Datenwert in Form einer Addition hinzuzufügen. Ein entsprechendes Merkmal kommt im Vital Coded Processor VCP zum Einsatz.

$$x_{cANB} = A \cdot x + B_x$$

Die Prüfung, ob das vorhergehende Ablegen eines Datenworts und das Lesen desselben zu einem späteren Zeitpunkt korrekt funktioniert haben, also z. B. keine Adressierungsfehler aufgetreten sind, erfolgt, indem vor Division durch A die erwartete Adresse bzw. Signatur B_x von x_c abgezogen wird.

$$x_{cANB} - B_x = (A \cdot x + B_x) - B_x = A \cdot x \equiv 0 \mod A$$

Alternativ kann natürlich auch geprüft werden, ob die Bedingung

$$x_{cANB} \equiv B_x \mod A$$

erfüllt ist.

Sollte durch einen Adressierungsfehler ein falscher Operand, in nachfolgender Gleichung z, geladen worden sein, so wird dieser Fehler durch

$$z_{cANB} - B_x = (A \cdot z + B_z) - B_x = A \cdot x + B_z - B_x \not\equiv 0 \mod A$$

oder alternativ

$$z_{cANB} \equiv B_z \neq B_x \mod A$$

aufgedeckt.

ANB-Kodierung von arithmetischen Operationen

Auch die ANB-Kodierung kann dazu verwendet werden, die Korrektheit von arithmetischen Operationen zu überprüfen. Im Gegensatz zur reinen AN-Kodierung müssen dabei ggf. – je nach durchzuführender Operation – nach der Durchführung der Operationen Korrekturen vorgenommen werden, um das Ergebnis so aufzubereiten, dass die typischen Prüfmöglichkeiten bestehen.

ANB-Kodierung von Additionen und Subtraktionen

Bei Additionen und Subtraktionen entsteht ein kodiertes Ergebnis, das als Signatur B die Summe bzw. die Differenz der beiden Signaturen der Operanden trägt.

$$z_{cANB} = x_{cANB} \pm y_{cANB} = (A \cdot x + B_x) \pm (A \cdot y + B_y) = A(x \pm y) + B_x \pm B_y$$

An dieser Stelle tritt eine Schwierigkeit bzgl. der ANB-Kodierung zu Tage: Soll der B-Anteil des Ergebnisses z_{cANB} wirklich der Adresse des Zieldatenspeicherelements entsprechen, so muss $B_x \pm B_y$ durch eine Korrektur in B_z umgewandelt werden. Wird $B_x \pm B_y$ dagegen als Signatur betrachtet, die nicht der Adresse des Datenspeicherelements entsprechend muss, so muss die erwartete Signatur im Programm in irgendeiner Weise gespeichert werden, da sie sich nicht aus z_{cANB} oder seiner Position im Speicher ableiten lässt.

ANB-Kodierung von Multiplikationen

Bei der Multiplikation werden Korrekturen des Ergebnisses notwendig, da sich

$$\begin{aligned} z_{cANB} &= x_{cANB} \cdot y_{cANB} = (A \cdot x + B_x) \cdot (A \cdot y + B_y) \\ &= A^2 \cdot x \cdot y + A \cdot x \cdot B_y + A \cdot y \cdot B_x + B_x \cdot B_y \end{aligned}$$

statt der erwarteten Form

$$z_{cANB} = A \cdot x \cdot y + B_x \cdot B_y$$

ergibt.

ANB-Kodierung von Divisionen

Für ANB-kodierte Operanden sind für die Division keine sinnvollen Lösungen bekannt, außer der Verwendung einer Schleife, innerhalb derer die Division durch eine Reihe kodierter Subtraktionen abgebildet wird [108]. Allgemein gelten Divisionen von ANB-kodierten Operanden als nicht durchführbar [108, 122]. Deswegen wird in manchen Implementierungen einer der Operanden vor Durchführung der Division auf die AN-kodierte Form reduziert [108].

3.10.3 ANBD-Kodierung: Hinzufügen der Aktualitätsprüfung D

Ein weiteres Merkmal, welches Forin in [38] als Merkmal des Vital Coded Processor VCP vorstellte, war das Hinzufügen eines Zeitstempels D_t zur kodierten Zahl, wodurch sich als kodierte Form von x

$$x_{cANBD} = A \cdot x + B_x + D_{xt}$$

ergibt.

Durch die Prüfung der Gleichung

$$x_{cANB} - B_x - D_{xt} = (A \cdot x + B_x + D_{xt}) - B_x - D_{xt} = A \cdot x \equiv 0 \mod A$$

bzw.

$$x_{cANBD} = A \cdot x + B_x + D_{xt} \equiv B_x + D_{xt} \mod A$$

können die Integrität des Datenworts, das Heranziehen des richtigen Datenworts mit der Adresse bzw. Signatur B_x und die Version des Datenworts mittels D_{xt} überprüft werden.

ANBD-Kodierung von arithmetischen Operationen

Für die Verwendung ANBD-kodierter Operanden in arithmetischen Operationen gelten die bereits bei der ANB-Kodierung vorgestellten Bedingungen inklusive der Notwendigkeit der Korrektur der Ergebnisse und der Schwierigkeiten bei der Durchführung von Divisionen.

3.10.4 Realisierung der AN(BD)-Kodierung

Die Realisierung der AN(BD)-Kodierung ist in verschiedenen Ausprägungen möglich. Neben der manuellen Implementierung aller notwendigen Schritte im Quellcode eines Programms – von der Kodierung der Operanden über die Durchführung der kodierten arithmetischen Operationen bis hin zur Prüfung der Ergebnisse – wurden von Schiffel in [108] zwei automatisierte Nutzungsverfahren der ANB(D)-Kodierung vorgestellt:

- Software Encoded Processing SEP, bei dem ein bereits übersetztes Programm, welches keine arithmetische Kodierung verwendet, durch einen Interpreter ausgeführt wird, der die Operanden arithmetischer Operationen vorab ANB-kodiert und die Ergebnisse überprüft und
- Compiler Encoded Processing CEP, bei dem die ANBD-Kodierung durch den Übersetzer während der Übersetzung des Quellcodes in das entstehende Programm eingebracht wird.

Bei SEP sorgt alleine die indirekte Ausführung des Programms durch einen Interpreter dafür, dass sich die benötigte Ausführungszeit auf mindestens das 900-Fache erhöht, teilweise sogar deutlich mehr [108]. Die Nutzung der ANB-Kodierung durch den Interpreter verlangsamt die Bearbeitung nochmals um den Faktor 2 bis 25 [108].

Wird die Kodierung nicht zur Laufzeit durch den Interpreter in ein Programm eingebracht, sondern bei CEP bereits durch den Übersetzer zur Übersetzungszeit in das Programm eingefügt, fällt der zusätzliche Laufzeitbedarf deutlich geringer aus. Je nach Stimulus beträgt die Ausführungszeit das Doppelte bis hin zum über 500-Fachen der Laufzeit des unkodierten Programms [108]. Dabei nutzt CEP Instruktionen, die speziell für die angewendete Form der Kodierung erweitert wurden.

3.10.5 Evaluation der AN(BD)-Kodierung

Ein Vorteil der AN(BD)-Kodierung ist die Möglichkeit des Einsatzes auf gewöhnlicher Hardware, ohne spezielle Hardwaremerkmale zu erfordern. Diesem Vorteil stehen jedoch deutliche Nachteile gegenüber:

- Bei der Wahl der Werte B und D besteht die Einschränkung, dass $B + D < A$ gelten muss, um die beschriebenen Prüfungsvorschriften nicht zu beeinflussen.

- Die kodierten Werte sind nicht mehr ohne Umrechnung menschenlesbar, was die Fehlersuche erschwert.
- Werden keine automatisierten Verfahren wie SEP oder CEP eingesetzt, so entsteht ein hoher Aufwand bei der Implementierung und dem Testen der Kodierung, der in jedem Projekt erneut zu betreiben ist.
- Beim Einsatz des Compiler Encoded Processing CEP erzeugt der Übersetzer schwer verständlichen Maschinencode, wodurch die diversitäre Rückwärtsanalyse erschwert wird.
- Die Kodierung verursacht deutlich erhöhten Laufzeitbedarf, z. B. durch die teilweise notwendigen Korrekturen der Ergebnisse nach der Verarbeitung [108].
- Die Kodierung ist nur für vorzeichenbehaftete und vorzeichenlose Ganzzahldatentypen geeignet und nicht für Gleitkommazahlen [108].
- Die Kodierung ist nur für bestimmte Operationen geeignet [108, 122]: Additionen, Subtraktionen, Multiplikationen und logische Operationen. Divisionen sind nur bei AN-Kodierung verwendbar. Schiffel hat in [108] Vorschläge unterbreitet, auch bisher nicht oder nur mit Einschränkungen nutzbare Operationen – teilweise unter Schwächung der Fehlererkennungsmöglichkeiten – kodiert auszuführen, so z. B. die Division.
- Die B- bzw. D-Signaturen müssen – sofern sie sich nicht vom Operanden selbst ableiten lassen, was bei der B-Signatur ggf. die Adresse des Operanden sein kann – zusätzlich zu den kodierten Daten gespeichert und verwaltet werden, um die Integrität der Daten und die Richtigkeit von Operationsergebnissen prüfen zu können.

In Tabelle 3.11 wird die Erkennbarkeit der 20 in Kapitel 2.4 vorgestellten Fehler- und Angriffsarten durch die AN(BD)-Kodierung gezeigt.

Aufgrund der beschriebenen Nachteile werden alle erkennbaren Fehlerarten mit einem „(ja)“ bewertet, also als erkennbar, jedoch mit Einschränkungen. Die detaillierte Beschreibung der Erkennbarkeit der verschiedenen Fehler- und Angriffsarten der AN-, ANB- und ANBD-Kodierung erfolgt in den folgenden Unterkapiteln.

3.10.5.1 Evaluation der AN-Kodierung

Die beiden durch die AN-Kodierung erkennbaren Fehlerarten – mit den bereits erwähnten Einschränkungen – sind fehlerhafte Operationsergebnisse und Verfäl-

Tabelle 3.11: Fehlererkennung durch AN(BD)-Kodierung

Fehlerart	**AN**	**ANB**	**ANBD**
Inkompatible Datentypen	nein	nein	nein
Inkompatible Einheiten	nein	nein	nein
Wertebereichsunter- bzw. -überschreitung	nein	nein	nein
Genauigkeitsproblem	nein	nein	nein
Falsche Operandenauswahl	nein	(ja)	(ja)
Falsche Operatorauswahl	nein	(ja)	(ja)
Fehlerhaftes Operationsergebnis	(ja)	(ja)	(ja)
Fristüberschreitung	nein	nein	nein
Zyklusunterschreitung	nein	nein	nein
Zyklusüberschreitung	nein	nein	nein
Verlorengegangene Datenaktualisierung	nein	nein	(ja)
Synchronisationsfehler oder unvollständige Datenübertragung	nein	nein	(ja)
Pufferunter- oder -überläufe	nein	begrenzt	(ja)
Fehlerhafter Datenfluss (falsche Adressaten, ...)	nein	nein	nein
Duplizierte Daten	nein	nein	nein
Durch Fehler oder Störungen verfälschte Daten	(ja)	(ja)	(ja)
Fehlerhafter Datenzugriff (fehlende Zugriffsrechte)	nein	nein	nein
Nutzung nicht initialisierter Daten	begrenzt	(ja)	(ja)
Angriffsart			
Gezielt verfälschte Daten	nein	nein	nein
Wiedereinspielungsattacke	nein	nein	nein

schungen der Daten durch Störungen. Die Erkennung der Nutzung nicht initialisierter Daten ist bei AN-Kodierung nur möglich, wenn die keine nutzbaren Daten enthaltenden Datenspeicherelemente mit einem Datenwert gefüllt werden, der kein Vielfaches von A ist. Mit Nullen vorbelegte Werte werden nicht als ungültig erkannt, da

$$0 \equiv 0 \mod A$$

die Bedingungen eines gültigen Datenspeicherelements erfüllt.

3.10.5.2 Evaluation der ANB-Kodierung

Zusätzlich zur Erkennung fehlerhafter Operationsergebnisse und der Verfälschung von Daten durch Störungen, kann die ANB-Kodierung die Auswahl falscher Operanden oder Operatoren aufdecken. Zudem wird es einfacher, Datenspeicherelemente ohne gültige Datenwerte so vorzubelegen, dass die Bedingung

$$x_c \equiv B_x \mod A$$

nicht erfüllt wird, um bei der Verarbeitung der ungültigen Daten diese als solche zu erkennen. Rein theoretisch könnten auch Pufferunter- oder -überläufe durch die ANB-Kodierung erkannt werden, wenn für verschiedene angrenzende Felder oder Puffer unterschiedliche B-Signaturen zum Einsatz kommen, darum wird diese Fehlerart als begrenzt erkennbar bewertet.

3.10.5.3 Evaluation der ANBD-Kodierung

Die ANBD-Kodierung erweitert die Fehlererkennung der ANB-Kodierung um die Erkennung verlorengegangener Datenaktualisierungen, Synchronisationsfehler und unvollständiger Datenübertragungen. Werden falsche Daten im Zuge eines Pufferunter- oder -überlaufs gelesen, so kann dieser Fehler dann erkannt werden, wenn die Adresssignatur B oder die Zeitstempelsignatur D nicht den erwarteten Werten entsprechen. Bei entsprechenden fehlerhaften Schreibzugriffen außerhalb von Feld- oder Puffergrenzen kann erst bei einem später erfolgenden Lesen der überschriebenen Werte der Fehler aufgedeckt werden.

3.11 Datenflussüberwachung in Netzwerken und sicherheitsgerichteten Feldbussen

Die Datenflüsse innerhalb eines Echtzeitsystems haben Ähnlichkeit mit jenen in Netzwerken. Datenübertragungsprotokolle, die zur Kommunikation über Netzwerke genutzt werden, sind darauf spezialisiert, Daten von einem Teilnehmer zu einem anderen zu übermitteln und dabei – je nach dabei verwendetem Protokoll – eine bestimmte Menge an Fehlererkennungs- und -toleranzmaßnahmen anzuwenden. Daher liegt es nahe, einige Netzwerkprotokolle und deren Methoden zur Datenflussüberwachung genauer zu betrachten.

Vorgestellt werden in diesem Kapitel als Stand der Technik

- das konventionelle Netzwerkprotokoll IP und das darauf aufsetzende verbindungsorientierte Protokoll TCP und
- Netzwerkprotokolle, die für sicherheitsgerichtete Feldbuskommunikation entwickelt wurden.

3.11.1 Netzwerkprotokolle TCP/IP

Netzwerkprotokolle werden nach dem OSI-Referenzmodell („Open System Interconnection Model“) [63] entsprechend ihrer Funktion im Kommunikationsablauf klassifiziert. Dieses OSI-Modell wird in Abbildung 3.18 dargestellt und definiert 7 Protokollschichten, von denen nicht alle innerhalb einer Kommunikationssitzung zum Einsatz kommen müssen.

Zwei wichtige Protokolle sollen hier vorgestellt werden:

- das Internetprotokoll IP in Version 4, definiert in RFC 791 [98], das der Vermittlungsschicht zuzuordnen ist und
- das verbindungsorientierte Transportkontrollprotokoll TCP, definiert in RFC 793 [100], dass die Transportschicht in der Kommunikation bildet und somit oberhalb des IP-Protokolls angesiedelt ist.

Die für diese Arbeit relevanten, der Datenflussüberwachung dienenden Merkmale der IP- und TCP-Protokolle sind

- Prüfsummen in den verschiedenen Protokollschichten,
- Überwachung der Paketlebenszeit,

Abbildung 3.18: OSI-Schichtenmodell (nach [63])

- Spezifikation der Quell- und Zieladressen der Kommunikationsteilnehmer und
- Sequenznummern,

die in den folgenden Unterkapiteln detaillierter vorgestellt werden.

3.11.1.1 Prüfsummen

Zur Verifikation der Integrität der übertragenen Daten werden in Netzwerken die Datenpakete in den verschiedenen Protokollschichten mit Prüfsummen versehen. Teilweise werden sogar mehrere unabhängige Prüfsummen für verschiedene Teile eines Datenpakets verwendet, wie z. B. die Kopfprüfsumme innerhalb des in Abbildung 3.19 dargestellten Paketkopfs des Internetprotokolls Version 4 [98].

3.11.1.2 Paketlebenszeit

Im Internetprotokoll IP Version 4, kurz IPv4, wird jedem Datenpaket, das über ein Netzwerk gesendet wird, eine Paketlebenszeit, engl. „Time To Live TTL“, in dem entsprechenden Feld TTL im Paketkopf zugeordnet, wie in Abbildung 3.19 dargestellt. Nach [98] ist es dabei vorgesehen, dass jede Station, die ein Paket annimmt und weiterreicht, die Anzahl der für diesen Vorgang benötigten Sekunden vom TTL-Wert abzieht, minimal jedoch den Wert 1, auch wenn die Bearbeitung des Pakets eine kürzere Zeitspanne in Anspruch nahm. Erreicht TTL den Wert 0, so wird das Datenpaket verworfen und eine entsprechende Fehlermeldung über das

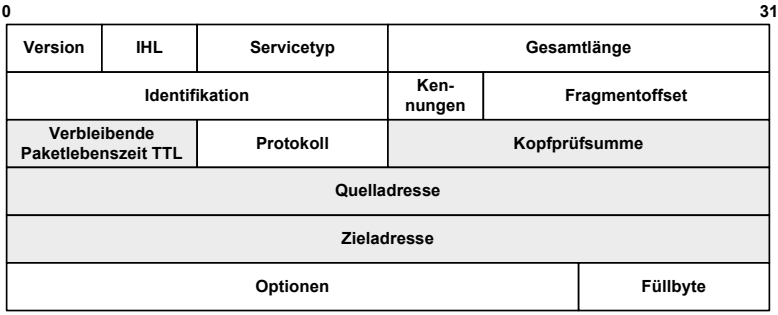


Abbildung 3.19: Paketkopf des Internetprotokolls Version 4 (nach [98])

ICMP-Protokoll [99] an den Absender des Pakets gesendet, wodurch dieser Kenntnis über die Zeitüberschreitung erhält. Die Abbildung 3.20 zeigt den Verlauf des Wertes im Feld TTL während der Übermittlung eines Paketes in einem Netzwerk, einmal ohne und einmal mit Fehlerfall. Nach [101] liegt der aktuell empfohlene Wert, der in das TTL-Feld eingetragen werden soll, bei 64.

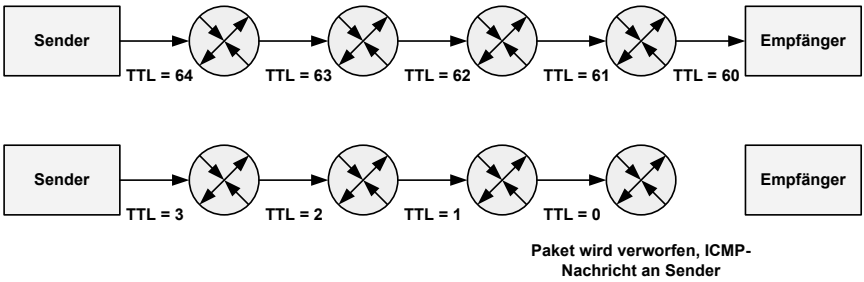


Abbildung 3.20: Verlauf des TTL-Werts, oben ohne und unten mit Fehlerfall

Durch die geringe zeitliche Auflösung des Werts im Feld TTL in ganzen Sekunden und die Vorgabe, immer mindestens eine Sekunde als Bearbeitungszeit abzuziehen, gibt der vom Absender in TTL eingetragene Wert nach [98] lediglich eine maximale Übermittlungszeit für ein Paket vor. In der Praxis ist diese Art der Laufzeitüberwachung kaum zur Realisierung einer Frist im Sinne der Echtzeitverarbeitung nutzbar. In der Nachfolgeversion, dem Internetprotokoll Version 6, kurz IPv6, wurde das Feld aus diesem Grund auch in „Hop Limit“, also übersetzt in etwa „Begrenzung der

Anzahl an Zwischenstationen“ umbenannt [102], was die auch bereits in Version 4 realisierte Funktionalität wesentlich treffender beschreibt.

3.11.1.3 Quell- und Zieladresse

In jedem Datenpaket, das über ein Netzwerk gesendet werden soll, werden Quell- und Zieladresse, also die Adressen von Absender und Empfänger des Datenpakets angegeben, wie in Abbildung 3.19 ersichtlich ist.

3.11.1.4 Sequenznummern

Das Transport Control Protocol, kurz TCP, ist ein verbindungsorientiertes Protokoll, das zur Überwachung des Datenflusses Sequenznummern in den Protokollköpfen einsetzt [100], wie in Abbildung 3.21 gezeigt. Dadurch wird es dem Empfänger ermöglicht, eintreffende Pakete in die vom Sender beabsichtigte Reihenfolge zu bringen und – in Bezug auf diese Arbeit wesentlich wichtiger – zu erkennen, ob Pakete verlorengegangen sind.

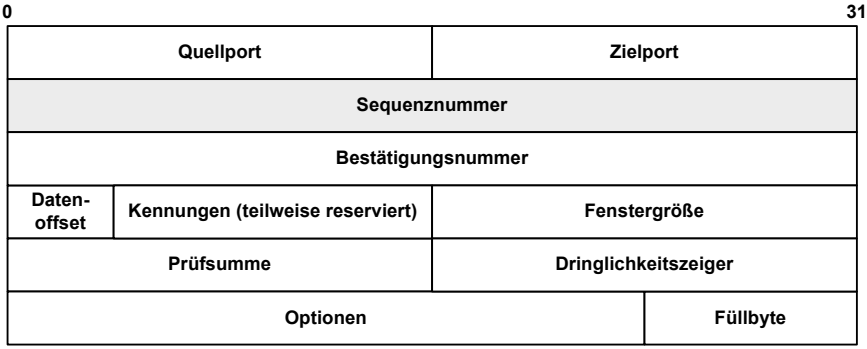


Abbildung 3.21: Sequenznummer im Paketkopf des Transportkontrollprotokolls (nach [100])

3.11.2 Sicherheitsgerichtete Feldbusprotokolle

Die internationale Norm IEC 61784-3 [54] beschreibt die generellen Anforderungen an die Protokolle sicherheitsgerichteter Feldbusse bzgl. Fehlererkennung und -behandlung. Die einzelnen standardisierten sicherheitsgerichteten Feldbussysteme werden in Unternormen der IEC 61784-3 im Detail beschreiben. Diese sogenannten „Profile“ werden in Tabelle 3.12 aufgelistet.

Tabelle 3.12: Sicherheitsgerichtete Feldbusprofile nach IEC 61784-3 [54]

Profil	**Feldbusbezeichnung**	**Unternorm**
Profilfamilie 1	FOUNDATION Feldbus	IEC 61784-3-1
Profilfamilie 2	CIP	IEC 61784-3-2
Profilfamilie 3	PROFIBUS, PROFINET (PROFIsafe)	IEC 61784-3-3
Profilfamilie 6	INTERBUS	IEC 61784-3-6
Profilfamilie 8	CC-Link	IEC 61784-3-8
Profilfamilie 12	EtherCAT	IEC 61784-3-12
Profilfamilie 13	Ethernet POWERLINK	IEC 61784-3-13
Profilfamilie 14	EPA	IEC 61784-3-14
Profilfamilie 17	RAPIEnet	(IEC 61784-3-17 in Arbeit)
Profilfamilie 18	SafetyNET p Feldbus	IEC 61784-3-18

Einer der wichtigsten Inhalte der IEC 61784-3 ist die Beschreibung typischer Fehlerfälle, die ein sicherheitsgerichteter Feldbus aufdecken und behandeln können muss. Diese sind:

- Datenverfälschung durch Störungen
- Wiederholung von Nachrichten
- Veränderung der Nachrichtenreihenfolge
- Verlust von Nachrichten
- Verzögerung von Nachrichten über ein vertretbares Maß hinaus, also unter Verletzung der Echtzeitbedingungen
- Einfügung von Nachrichten von unbekannten Quellen

- Maskerade von Nachrichten, bei der eine nicht sicherheitsgerichtete Nachricht durch Fehler als sicherheitsgerichtete Nachricht erscheint
- Adressierungsfehler, d. h. Nachrichten werden vom falschen Empfänger empfangen und ggf. verarbeitet

Weiterhin beschreibt die Norm einige Erkennungsmethoden, die vorgeschlagen werden, um die genannten Fehler aufzudecken. Diese Methoden sind:

- Nutzung einer Sequenznummer
- Nutzung von Zeitstempeln
- Überwachung der Zeit zwischen dem Eintreffen zweier Nachrichten; bei Überschreitung dieser Zeit ist von einem Fehler auszugehen
- Gegenseitige Authentifizierung von Sender und Empfänger über eine eindeutige logische Adresse
- Bestätigungsnachrichten zur Bestätigung des korrekten Empfangs von Nachrichten
- Nutzung von Integritätsprüfungen zur Aufdeckung von Datenverfälschungen
- Gezielte Mehrfachversendung der identischen Nachricht unter Nutzung verschiedener Integritätsprüfungsmethoden, ggf. unter Nutzung redundanter Kommunikationskanäle
- Nutzung verschiedener Integritätsprüfungsmethoden, wenn über einen Bus sicherheitsgerichtete und nicht-sicherheitsgerichtete Daten übermittelt werden

Welche Fehlerarten die einzelnen Erkennungsmethoden nach [54] erkennen können, wird in Tabelle 3.13 gezeigt.

Diese vorgeschlagenen Erkennungsmerkmale werden in den einzelnen Profilfamilien unterschiedlich eingesetzt, meist kommt nur eine Untermenge der vorgeschlagenen Merkmale zum Einsatz, ggf. auch in modifizierter Form. Die detaillierte Beschreibung, welche der Fehlererkennungsmerkmale zum Einsatz kommen und wie sie realisiert werden, erfolgt in der jeweiligen Unternorm, die die betreffende Protokollfamilie beschreibt.

Im Detail sollen nun zwei sicherheitsgerichtete Feldbussysteme vorgestellt werden, die unterschiedliche Fehlererkennungsmerkmale nutzen: PROFIsafe, das mit Telegrammen mit Sequenznummern arbeitet, und CIP Safety, welches Zeitstempel einsetzt.

Tabelle 3.13: Fehleraufdeckung durch die vorgestellten Methoden (nach [54])

	Sequenznummer	Zeitstempel	Zeitbedingungen	Authentifizierung	Bestätigung	Integritätsprüfung	Redundante Nachrichten	Verschiedene Integritäts-prüfungen
Datenverfälschung					x	x		
Nachrichtenduplikat	x	x					x	
Sequenzfehler	x	x					x	
Nachrichtenverlust	x				x		x	
Verzögerung		x	x					
Einfügung von Nachrichten	x	x		x	x		x	
Maskerade				x	x			x
Adressierungsfehler				x				

3.11.2.1 PROFIsafe

PROFIsafe ist ein Kommunikationsprofil für PROFIBUS und PROFINET, welches in der Unternorm IEC 61784-3-3 [56] beschrieben wird. Darin wird u.a. festgelegt, welche der in der IEC 61784-3 [54] beschriebenen Fehlererkennungsmechanismen angewendet und wie diese im Detail realisiert werden. Aktoren und Sensoren werden als „F-Device“ und Datenverarbeitungseinheiten als „F-Host“ bezeichnet. Ein entsprechendes sicherheitsgerichtetes System ist in Abbildung 3.22 dargestellt.

Sensoren und Aktoren werden bei PROFIsafe als „F-Devices“ bezeichnet, Datenverarbeitungseinheiten, die die Sensorsignale verarbeiten und Stellgrößen für die Aktoren generieren, als „F-Hosts“.

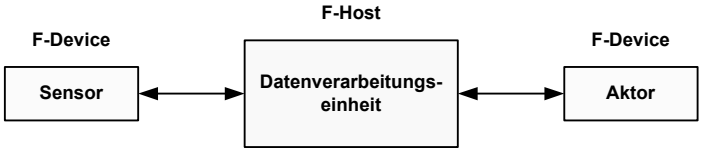


Abbildung 3.22: Systemaufbau mit PROFIsafe

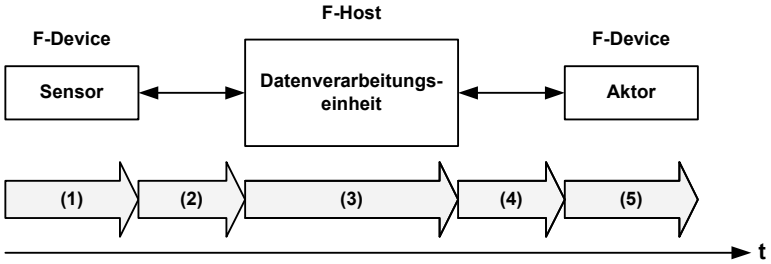
Die im PROFIsafe-Protokoll eingesetzten Fehlererkennungsmerkmale werden in Tabelle 3.14 dargestellt.

Tabelle 3.14: Fehlererkennungsmerkmale von PROFIsafe (nach [56])

	Sequenznummer	Zeitstempel	Zeitbedingungen	Authentifizierung	Bestätigung	Integritätsprüfung	Redundante Nachrichten	Verschiedene Integritäts-prüfungen
Datenverfälschung						x		
Nachrichtenduplikat	x							
Sequenzfehler	x							
Nachrichtenverlust	x		x		x			
Verzögerung			x		x			
Einfügung von Nachrichten	x		x	x	x			
Maskerade			x	x	x	x		
Adressierungsfehler				x				
Speicherfehler in Netzwerkgeräten	x							

Eine Sequenznummer mit einer Breite von 24 Bit erlaubt es dem Empfänger, zu erkennen, ob er die Nachrichten in der korrekten Reihenfolge erhalten hat, eine Nachricht dupliziert wurde oder verlorengegangen ist. Dabei wird diese Sequenznummer nicht übertragen, sondern in jedem der Teilnehmer selbst verwaltet und daher als „virtuelle Sequenznummer VCN“ bezeichnet. Lediglich ein Bit, das bei jeder Nachricht den Zustand wechselt, wird mitgesendet. Zur Überprüfung, ob eine Nachricht vom Sender mit der vom Empfänger erwarteten Sequenznummer versehen wurde, beziehen beide Teilnehmer die VCN in die Berechnung der Prüfsummen mit ein. Eine Abweichung der Sequenznummer des Senders von der erwarteten Sequenznummer des Empfängers wird dadurch bei der Überprüfung der Prüfsumme eines Datenpakets aufgedeckt. In jedem Kommunikationsteilnehmer läuft ein Überwachungszeitgeber, engl. „watchdog“, für die Überwachung der Übertragungszeiten, der mit einer maximalen Frist konfiguriert wird. Diese Frist wird als „F_WD_Time“

bezeichnet. Beim Eintreffen einer gültigen Nachricht mit einer neueren Sequenznummer werden diese Zeitgeber zurückgesetzt und die Frist somit erneuert. Läuft die Frist eines dieser Überwachungszeitgeber ab, so wechselt die betroffene Einheit in einen vorkonfigurierten sicheren Zustand. Weitere Überwachungszeitgeber kommen in den Teilnehmern zum Einsatz, um die Datenverarbeitung innerhalb der Geräte zeitlich zu überwachen. Die einzelnen zum Einsatz kommenden Fristen sind in Abbildung 3.23 dargestellt. Für die verschiedenen Geräte werden dabei jeweils maximale Bearbeitungszeiten und zwischen den Geräten maximale Übertragungszeiten festgelegt. Aus der Summe der gezeigten Zeiten und der zeitlichen Auslösung der Überwachungszeitgeber ergibt sich die Sicherheitsreaktionszeit, innerhalb derer auf einen Fehler reagiert werden kann.



(1): Maximal zulässige Bearbeitungszeit des Sensors
(2): Maximal zulässige Übertragungszeit zwischen Sensor und Datenverarbeitungseinheit
(3): Maximal zulässige Bearbeitungszeit der Datenverarbeitungseinheit
(4): Maximal zulässige Übertragungszeit zwischen Datenverarbeitungseinheit und Aktor
(5): Maximal zulässige Bearbeitungszeit des Aktors

Abbildung 3.23: Fristen der Überwachungszeitgeber (nach [56])

Allen Teilnehmern werden spezielle Adressen zugewiesen, die „F-Adressen“ genannt werden und die nur innerhalb der sicherheitsgerichteten Kommunikation Anwendung finden. Sie dienen der Authentifizierung der Kommunikationsteilnehmer, die dadurch feststellen können, ob empfangene Daten vom erwarteten Absender stammen. Wie bei der virtuellen Sequenznummer werden die F-Adressen dabei nicht in den Datenpaketen eingebettet, sondern gehen ebenfalls nur in die Prüfsummenberechnung ein, wodurch ein falscher Adressat anhand der Prüfsummen die fehlgeleiteten Daten als solche identifizieren kann. Verschiedene Prüfsummenverfahren erlauben es, nicht sicherheitsgerichtete und sicherheitsgerichtete Daten zu unterscheiden und Verfälschungen der Daten aufzudecken.

3.11.2.2 CIP Safety

CIP Safety ist ein sicherheitsgerichtetes Feldbusprotokoll, welches auf dem Common Industrial Protocol CIP basiert und in der Unternorm IEC 61784-3-2 [55] spezifiziert wird. Wie auch bei PROFIsafe definiert die Unternorm unter anderem, welche der in der IEC 61784-3 [54] vorgeschlagenen Fehlererkennungsmerkmale in CIP Safety zur Anwendung kommen sollen und wie diese genutzt werden. Diese werden in Tabelle 3.15 gezeigt.

Tabelle 3.15: Fehlererkennungsmerkmale von CIP Safety (nach [55])

	Sequenznummer	Zeitstempel	Zeitbedingungen	Authentifizierung	Bestätigung	Integritätsprüfung	Redundante Nachrichten	Verschiedene Integritäts-prüfungen
Datenverfälschung						x	x	
Nachrichtenduplikat		x				x	x	
Sequenzfehler		x				x	x	
Nachrichtenverlust			x			x	x	
Verzögerung		x	x					
Einfügung von Nachrichten		x		x		x	x	
Maskerade		x		x		x	x	x
Adressierungsfehler				x		x		

Jede Nachricht, nicht jedoch jedes Datenspeicherelement innerhalb der Nachricht, wird von einer Datenquelle mit einem Zeitstempel versehen, der die Entstehungszeit der Nachricht angibt. Da der Empfänger der Nachricht ein maximales Alter einer Nachricht erwartet, können entsprechende Fehlerarten wie z. B. Nachrichtenverzögerung und Sequenzfehler aufgedeckt werden. Zur Abstimmung der Zeitbasen zwischen den Kommunikationsteilnehmern sieht CIP Safety entsprechende Synchronisationsmechanismen vor, um regelmäßig Abweichungen der in den Geräten eingesetzten Taktgeneratoren zu kompensieren. Zur Sicherstellung, dass Daten von den erwarteten Absendern stammen, wird in die Prüfsummen der Datenpakete ein

Identifikator des Absenders einberechnet. Ein Empfänger, der ein fehlgeleitetes Datenpaket erhält, wird hier den falschen Identifikator zur Prüfsummenberechnung heranziehen, wodurch die Prüfsumme als ungültig identifiziert wird. Über Konfigurationsdateien wird den Kommunikationsteilnehmern mitgeteilt, welche Daten versendet oder empfangen werden sollen.

3.11.3 Evaluation der Datenflussüberwachung in Netzwerken und sicherheitsgerichteten Feldbussen

Die Erkennbarkeit der 20 in Kapitel 2.4 vorgestellten Fehler- und Angriffsarten durch TCP/IP- und die ausgewählten sicherheitsgerichteten Feldbusprotokolle wird in Tabelle 3.16 gezeigt.

Ein grundlegendes Problem der Fehlererkennungsmechanismen von Kommunikationsprotokollen ist der Wirkungsbereich der jeweiligen Fehlererkennungsmethoden, da sie nur der Prüfung der fehlerfreien Datenübertragung über die Kommunikationsstrecke dienen, wie in Abbildung 3.24 dargestellt. Die entsprechenden zusätzlichen Informationen werden vor Versand der Daten an diese angehängt und nach Empfang und Prüfung wieder entfernt.

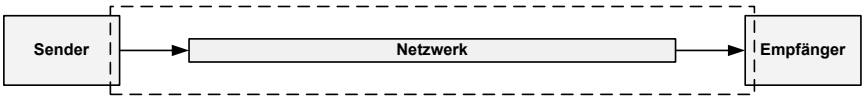


Abbildung 3.24: Prüfungsbereich des Datenflusses in Netzwerken

In den folgenden Unterkapiteln werden die TCP/IP- und sicherheitsgerichteten Feldbusprotokolle getrennt evaluiert.

3.11.3.1 Evaluation der Netzwerkprotokolle TCP/IP

Die Paketlebenszeit in Form der zulässigen Anzahl von Zwischenstationen ermöglicht keine feingranulare Prüfung von Echtzeitbedingungen. Weiterhin wird der Empfänger nicht über den Zeitablauf und damit den Paketverlust informiert, sondern nur der Sender, weshalb keine der echtzeitbezogenen Fehlerarten als erkennbar eingestuft werden kann. Die im TCP-Protokoll eingesetzte Sequenznummer erlaubt

Tabelle 3.16: Fehlererkennung durch TCP/IP- und sicherheitsgerichtete Feldbusprotokolle

Fehlerart	**TCP/IP**	**PROFIsafe**	**CIP Safety**
Inkompatible Datentypen	nein	nein	nein
Inkompatible Einheiten	nein	nein	nein
Wertebereichsunter- bzw. -überschreitung	nein	nein	nein
Genauigkeitsproblem	nein	nein	nein
Falsche Operandenauswahl	nein	nein	nein
Falsche Operatorauswahl	nein	nein	nein
Fehlerhaftes Operationsergebnis	nein	nein	nein
Fristüberschreitung	nein	(ja)	(ja)
Zyklusunterschreitung	nein	nein	nein
Zyklusüberschreitung	nein	(ja)	(ja)
Verlorengegangene Datenaktualisierung	(ja)	(ja)	(ja)
Synchronisationsfehler oder unvollständige Datenübertragung	(ja)	(ja)	(ja)
Pufferunter- oder -überläufe	nein	nein	nein
Fehlerhafter Datenfluss (falsche Adressaten, ...)	(ja)	(ja)	(ja)
Duplizierte Daten	(ja)	(ja)	(ja)
Durch Störungen oder Fehler verfälschte Daten	ja	ja	ja
Fehlerhafter Datenzugriff (fehlende Zugriffsrechte)	nein	nein	nein
Nutzung nicht initialisierter Daten	nein	nein	nein
Angriffsart			
Gezielt verfälschte Daten	nein	begrenzt	begrenzt
Wiedereinspielungsattacke	nein	begrenzt	begrenzt

das Aufdecken verlorengegangener und duplizierter Datenpakete, jedoch ist keine Prüfung möglich, ob die Daten innerhalb der Quelle auch wirklich aktualisiert wurden. Unvollständige Übertragungen können durch die Längenangaben in den Paketköpfen und die Prüfsummen erkannt werden. Die Quell- und Zieladressen erlauben keine Prüfung des Wegs der Daten durch ein gesamtes System, sondern nur auf den einzelnen Kommunikationsstrecken zwischen den Systembestandteilen. So könnten Daten, die zwar korrekt von einem Sensor zu einer Datenverarbeitungseinheit transportiert wurden, innerhalb dieser nach Verarbeitung zu einer Stellgröße jedoch durch einen Fehler nicht an den richtigen Aktor gesendet wurden, nicht als fehlgeleitet erkannt werden. Die Kommunikationssoftware würde den falschen Aktor durch seine entsprechende Zieladresse als gültigen Adressaten ausweisen. Durch Störungen verfälschte Daten können durch die Verwendung der verschiedenen Prüfsummen erkannt werden.

3.11.3.2 Evaluation der sicherheitsgerichteten Feldbusprotokolle

Die beiden sicherheitsgerichteten Feldbusprotokolle PROFIsafe und CIP Safety haben trotz ihrer unterschiedlichen Fehlererkennungsmechanismen – PROFIsafe verwendet Sequenznummern und Überwachungszeitgeber, CIP Safety dagegen Zeitstempel und maximales Alter von Nachrichten – einen sehr ähnlichen Umfang an erkennbaren Fehlerarten.

PROFIsafe kann durch die verschiedenen Überwachungszeitgeber die Verletzung von Fristen erkennen, ebenso die Überschreitung einer definierten maximalen Zykluszeit. Verlorengegangene Datenaktualisierungen und Duplizierung von Daten werden durch die verwendeten virtuellen Sequenznummern aufgedeckt. Unvollständige Datenübertragungen können durch die eingesetzten Prüfsummen erkannt werden. Fehlgeleitete Daten werden anhand der Nichtübereinstimmung der F-Adressen identifiziert. Die verwendeten Prüfsummen sorgen dafür, dass durch Störungen verfälschte Daten als solche erkannt werden können. Durch Einsatz der virtuellen Sequenznummer und der F-Adressen, die in die Berechnung der Prüfsummen eingehen, jedoch nicht in den Datenpaketen eingebettet werden, wird es Angreifern erschwert, die Daten zu verfälschen, weil dazu die betreffenden Parameter bekannt sein müssten. Für Wiedereinspielungsattacken wäre es zusätzlich notwendig, die korrekten virtuellen Sequenznummern aufzuzeichnen. In [3] wird jedoch gezeigt, dass es trotzdem möglich ist, die über PROFIsafe übermittelten sicherheitsgerichteten Daten unbemerkt zu verändern. Daher werden beide Attacken als begrenzt erkennbar klassifiziert.

Bei CIP Safety können Fristüberschreitungen dadurch erkannt werden, dass der Empfänger das Alter einer Nachricht anhand ihres Zeitstempels ermittelt und es mit einem maximal zulässigen Alter vergleicht. Nicht rechtzeitig eintreffende und verlorengegangene Datenpakete werden dadurch erkannt, dass das Alter der letzten erfolgreich empfangenen Nachricht eine vorab festgelegte Grenze überschreitet. In begrenztem Umfang lassen sich dadurch auch Duplikate bereits empfangener Nachrichten erkennen, da diese einen entsprechend alten Zeitstempel tragen, wodurch bei entsprechendem Nachrichtenalter die maximale Altersgrenze der Daten überschritten wird. Fehlgeleitete Datenpakete können daran erkannt werden, dass die in die Prüfsummenberechnung einbezogenen Identifikatoren von Sender und Empfänger die Überprüfung der Prüfsumme fehlschlagen lassen. Durch Störungen verfälschte Daten können durch die Prüfsummen bis auf eine geringe Restfehlerwahrscheinlichkeit aufgedeckt werden. Die Einbeziehung verschiedener Faktoren in die Berechnung der Prüfsummen der Pakete erlaubt eine begrenzte Sicherheit von CIP Safety gegenüber gezielter Veränderung von Paketinhalten, da ein Angreifer zunächst alle in die Prüfsummenberechnung einbezogenen Daten in Erfahrung bringen muss, bevor er Pakete erfolgreich manipulieren kann. Kann er die Prüfsummen nicht selbst berechnen, so sind auch Widereinspielungsattacken ausgeschlossen, da die Empfänger die aufgezeichneten und wieder eingespielten Datenpakete anhand der Zeitstempel als veraltet erkennen würden. Da die Faktoren für einen Angreifer jedoch ermittelbar sein dürften, können beide Attacken nur als begrenzt erkennbar gewertet werden.

3.12 Zusammenfassung des Stands von Wissenschaft und Technik

Der vorgestellte Stand von Wissenschaft und Technik soll nun noch einmal zusammenfassend betrachtet werden, um dessen Grenzen und Nachteile bzgl. der Erkennung von Fehlern und Angriffen aufzuzeigen.

3.12.1 Zusammenfassung der Fehlererkennungsmöglichkeiten

In Tabelle 3.17 wird noch einmal zusammengefasst dargestellt, welche der 20 identifizierten Fehler- und Angriffsarten durch die vorgestellten Verfahren und Architekturen erkannt werden können. Details zu den Bewertungen sind den jeweiligen ausführlichen Beschreibungen zu entnehmen.

Tabelle 3.17: Zusammenfassung der Fehlererkennung durch die vorgestellten Verfahren und Architekturen

Fehler- bzw. Angriffsart	x86, ARM	SGP	DT, DS, BA	DFA	ISMA	ADI SSM	ANBD	DDFV	TCP/IP	PROFIsafe, CIP Safety
Inkompatible Datentypen	-	-	+	-	+	-	-	-	-	-
Inkompatible Einheiten	-	-	-	-	-	-	-	-	-	-
Wertebereichsverletzung	◯ x86	-	-	-	-	-	-	-	-	-
Genauigkeitsproblem	-	-	-	-	-	-	-	-	-	-
Falsche Operanden	-	+	-	-	-	-	(+) BD	◯	-	-
Falsche Operatoren	-	+	-	-	-	-	(+) BD	+	-	-
Fehlerhafte Operation	-	+	-	-	-	-	(+) BD	-	-	-
Fristüberschreitung	-	◯	-	-	◯	-	-	-	-	(+)
Zyklusunterschreitung	-	-	-	-	-	-	-	-	-	-
Zyklusüberschreitung	-	-	-	-	-	-	-	-	-	(+)
Verlorengeg. Aktualisier.	-	-	-	-	-	(+)	(+) D	-	(+)	(+)
Synchronisationsfehler und unvollst. Übertragungen	-	-	-	-	-	(+)	(+) D	-	(+)	(+)
Pufferunter- oder -überlauf	(+) x86	-	+ DS	-	+	(+)	(+) D	-	-	-
Fehlerh. Datenfluss	-	◯	◯ BA	-	◯	-	-	◯	(+)	(+)
Duplizierte Daten	-	-	-	-	-	(+)	-	-	(+)	(+)
Durch Störungen oder Fehler verfälschte Daten	◯	(+)	+	-	+	-	(+) AN	-	+	+
Fehlerh. Datenzugriff (fehlende Zugriffsrechte)	◯	◯	+ BA	-	+	-	-	-	-	-
Nicht initialisierte Daten	-	-	◯	-	+	(+)	(+) BD	-	-	-
Gezielte Verfälschung	-	-	-	-	-	-	-	-	-	◯
Wiedereinspielungsattacke	-	-	-	-	-	-	-	-	-	◯

Fehlererkennung: - nicht mögl., ◯ begrenzt mögl., (+) mit Einschränkungen mögl., + mögl.; SGP: Prozessoren für sicherheitsgerichtete Anwendungen, DT, DS, BA: Datentyp-, -struktur-, Befähigungsarchitekturen, DFA: Datenflussarchitekturen

Die konventionellen Architekturen x86 und ARM nutzen komplexeste Maßnahmen zur Erzielung eines maximalen Datendurchsatzes, können jedoch nur wenige der identifizierten Fehler- und Angriffsarten aufdecken. Die vorhandenen Fehlererkennungsmaßnahmen wie Segmentierung und Seitenverwaltung können Zugriffsrechte nur für größere Dateneinheiten wie Seiten oder ganze Segmente vergeben. Die Zugriffsrechte werden zudem unabhängig von den durch sie geschützten Daten verwaltet. Die auf konventionellen Architekturen basierenden Prozessoren für den Einsatz in sicherheitsgerichteten Systemen, wie z. B. der TI Hercules, können durch den Einsatz verschiedener Redundanz- und Diversitätsarten zwar mehr Fehlerarten im Bereich fehlerhafter Operationen und Datenverfälschungen erkennen, jedoch reichen diese Maßnahmen nicht aus, was durch die Anzahl der nicht aufdeckbaren Fehlerarten ersichtlich wird.

Datentyp-, -struktur- und Befähigungsarchitekturen fügen Speicherinhalten Kennungen hinzu, die hardwareverständlich die Inhalte des Speichers beschreiben, womit weitere Fehlerarten wie z. B. inkompatible Datentypen und die Verletzung von Zugriffsrechten auf feinstgranularer Ebene erkennbar werden. Allerdings werden nur wenige Dateneigenschaften in Kennungen abgebildet, wodurch weiterhin viele Fehlerarten nicht aufgedeckt werden können.

Die programmierbare Metadatenverarbeitungseinheit PUMP des SAFE-Projekts, welches den Befähigungsarchitekturen zuzuordnen ist, ist in der Lage, verschiedenste Kennungsarten mit nahezu beliebiger Länge zu verwenden. Diese Kennungen sind nur teilweise untrennbar mit dem jeweiligen Datenspeicherelement verbunden, erweiterte Kennungen können getrennt von den Daten im Speicher abgelegt sein. Die Hardware übernimmt bei PUMP nur die Zwischenspeicherung und das Suchen der zutreffenden Kennungsregeln im Zwischenspeicher, sowie die Anwendung derselben. Bislang nicht zwischengespeicherte Kennungsregeln müssen von der Software auf der Basis von Unterbrechungen evaluiert und an die Hardware übergeben werden. Die Hardware wendet diese Regeln ohne Wissen um deren Bedeutung an und es besteht die Gefahr von Inkonsistenzen bei der Übertragung der Datenspeicherelemente und bei Änderung der erweiterten Kennungen im Speicher. Dynamische Einflüsse wie die Zeit können nicht in die Kennungen einbezogen werden, weshalb z. B. Fristen in PUMP nicht realisiert werden können.

Die auf die Bearbeitung von Datenflüssen spezialisierten Datenflussarchitekturen enthalten keine spezialisierten Fehlererkennungsmerkmale und sind dadurch nicht in der Lage, entsprechende Fehler oder Angriffe zu erkennen.

Die inhärent sichere Mikroprozessorarchitektur ISMA kann mehr Fehlerarten als konventionelle Architekturen erkennen, ist jedoch nicht auf die Erkennung daten-

flussbezogener Fehler spezialisiert. Der Fokus von ISMA liegt auf der strikten Isolation von Datentypen, der Anwendung von umfassenden Prüfungen bei der Typumwandlung, der Verwendung von Kontrollflussprüfungen und dem nicht durch die Software zugreifbaren Stapelspeicher.

Das durch Oracle im SPARC M7 Prozessor eingeführte Fehlererkennungsmerkmal Application Data Integrity ADI, später umbenannt in Silicon Secured Memory SSM, fügt Datenblöcken von 64 Byte Größe und Zeigern eine Versionskennung hinzu, die beim Datenzugriff miteinander verglichen werden. Obwohl das Verfahren mehrere Fehlerarten, darunter die Aufdeckung verlorengegangener Datenaktualisierungen und Synchronisationsfehler, aufdecken kann, weist es wesentliche Nachteile auf: die Versionskennungen können nur für Datenblöcke, nicht jedoch für einzelne Datenspeicherelemente vergeben werden und die in den Zeigern hinterlegten Versionen sind absolute Angaben und müssen durch die Software gesetzt und bei jeder Versionsänderung durch diese aktualisiert werden.

Die arithmetische AN-Kodierung kann zusammen mit den Erweiterungen zur ANBD-Kodierung einige wichtige Datenflussfehler wie z. B. falsche Operanden, falsche Operationen und fehlerhafte Operationsergebnisse sowie verlorengegangene Datenaktualisierungen erkennen, ist jedoch nur für bestimmte Operationen und Datentypen geeignet und verursacht erhöhten Laufzeitbedarf. Weiterhin sind die kodierten Datenspeicherelemente nicht mehr ohne weitere Aufbereitung menschenlesbar, was die Fehlersuche erschwert.

Die dynamische Datenflussprüfung DDFV erlaubt die signaturbasierte Prüfung der Datenflüsse auf Registerebene, bezieht aber keine Speicherinhalte in die Prüfung mit ein und kann keine Datenübertragungen überprüfen.

Die Kommunikationsprotokolle TCP/IP weisen nur wenige Merkmale zur Erkennung von Datenflussfehlern auf, die sich speziell auf die Integrität des Datenflusses beziehen, nicht jedoch auf dessen Inhalte. Die Protokolle für sicherheitsgerichtete Feldbusse wenden mehr Fehlererkennungsmaßnahmen an, besonders im Bereich der Zeitüberwachung. Allerdings wird jeweils nur sichergestellt, dass die Inhalte erfolgreich und zeitgerecht über die jeweilige Kommunikationsstrecke übertragen werden. Dazu werden den zu übertragenden Daten zusätzliche Informationen hinzugefügt, die nach der Übertragung wieder von diesen getrennt werden. Die beiden sicherheitsgerichteten Feldbusprotokolle PROFIsafe und CIP Safety weisen gewisse Vorkehrungen gegen die gezielte Manipulation von Dateninhalten und Wiedereinspielungsattacken auf. In beiden Protokollen werden verschiedenste Faktoren in die Bildung der Paketprüfsummen einbezogen, die dem Angreifer nicht bekannt sein sollten, wodurch die Manipulation der Dateninhalte erschwert wird. Die Sequenznummer

bei PROFIsafe und die Zeitstempel bei CIP Safety verhindern Wiedereinspielungsattacken, da die Datenpakete von den Empfängern als veraltet identifiziert werden können. Allerdings wurde in [3] erläutert, wie sich durch Aufzeichnung des Netzwerkverkehrs die unbekannten Prüfsummenbestandteile von PROFIsafe ermitteln lassen. Dadurch ist es einem Angreifer möglich, die Daten unbemerkt zu verändern. Für CIP Safety gelten ähnliche Bedingungen.

3.12.2 Zusammenfassende Kritik am Stand von Wissenschaft und Technik

Obwohl einige interessante Fehlererkennungsverfahren bekannt sind, fehlt eine systemweite, ganzheitliche Betrachtung von Daten, ihrer Eigenschaften und ihrer Wege durch ein System. Selbst wenn bestimmte Dateneigenschaften von der Software auf einer Systemkomponente lokal zur Laufzeit betrachtet oder sogar überwacht werden, so werden diese getrennt von den eigentlichen Daten verwaltet und die Information geht bei der Übertragung der Daten zwischen verschiedenen Softwareprogrammen innerhalb der Systemkomponente, spätestens jedoch bei der Übertragung der Daten an andere Systemkomponenten verloren. Die Hauptverantwortung für die Fehlererkennung trägt meist die Software, was deren Komplexität und Fehlerwahrscheinlichkeit weiter erhöht. Zudem müssen derartige, in Software realisierte Fehlererkennungsmaßnahmen in jedem Projekt erneut spezifiziert, entworfen, implementiert und getestet werden.

Bei der AN(BD)-Kodierung werden die Fehlererkennungsmaßnahmen zwar direkt mit den Daten verbunden und gehen bei deren Übertragung nicht verloren. Allerdings ist sie nur auf bestimmte Datentypen und Operationen anwendbar und erfordert teils großen Laufzeitaufwand.

Datentyp-, -struktur- und Befähigungsarchitekturen bilden einige wenige Dateneigenschaften in Kennungen in hardwareles- und -überprüfbarer Form zusätzlich zum Datenwert ab und erlauben somit einfache und die Laufzeit kaum verlängernde Prüfungen. Diese Kennungen sind untrennbar mit den Datenwerten verbunden und werden mit diesen gespeichert, übertragen und verarbeitet. Allerdings bilden die genannten Architekturen zu wenige Dateneigenschaften in Kennungen ab, wie sich anhand der nicht erkennbaren Fehler- und Angriffsarten erkennen lässt.

Die Realisierung von Fehlererkennungsmerkmalen in Hardware in Form von Kennungen weist einige Vorteile gegenüber Softwareimplementierungen auf:

- Im Gegensatz zu Softwareimplementierungen müssen hardwarebasierte Fehlererkennungsmaßnahmen nur einmalig spezifiziert, entworfen, implementiert und getestet werden.
- Die Prüfung von in den Kennungen spezifizierten Eigenschaften erfolgt im Idealfall ohne zusätzlichen Zeitaufwand durch parallelisierte Prüfungsstrukturen innerhalb der Prozessorhardware: die Prüfungen werden vom Zeitbereich in die Hardware verschoben, wie in Abbildung 3.25 angedeutet.
- Da alle Eigenschaften eines Datenspeicherelements in diesem selbst beschrieben werden, ist inhärente Atomarität gegeben, d. h. es können sich keine Inkonsistenzen innerhalb von Datenspeicherelementen z. B. als Folge von Unterbrechungen ergeben.
- Die Eigenschaften der Daten gehen beim Datenaustausch mit anderen Programmen oder Systemkomponenten nicht verloren oder müssen getrennt von den Daten übermittelt werden, d. h. es können sich auch hier keine Inkonsistenzen ergeben.

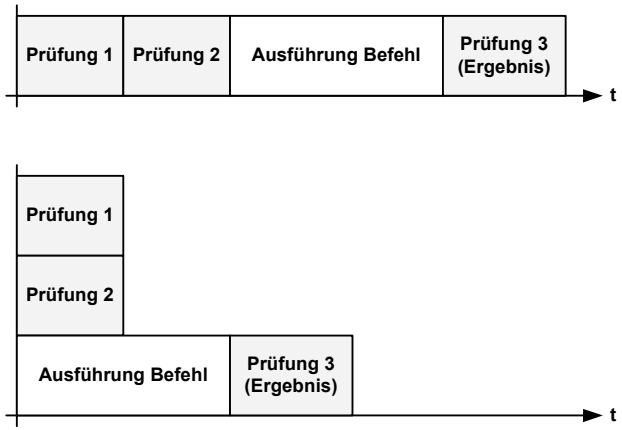


Abbildung 3.25: Verschiebung sequentieller Prüfungen in die Hardware

Der wesentliche Nachteil der Nutzung von Kennungen in den Datenspeicherelementen ist der erhöhte Speicherbedarf, der sich durch die größeren Datenspeicherelemente ergibt. In der Vergangenheit mag dieser „verschwenderische Umgang mit

Speicher“ aufgrund der kleinen verfügbaren Speichergrößen und des hohen Preises noch Relevanz gehabt haben. Jedoch kann ein solches Argument angesichts der heute verfügbaren Speicherkapazitäten zu immer geringeren Preisen keinen Bestand mehr haben. Der Vorteil, Fehler

- mit einfachsten Mitteln,
- zum frühestmöglichen Zeitpunkt, idealerweise bei deren Entstehung,
- in großem Umfang auch in einkanaligen Systemen

erkennen zu können, wiegt den Zusatzbedarf an Speicher auf.

4 Eine Datenspezifikationsarchitektur

In diesem Kapitel wird eine neuartige Mikroprozessorarchitektur vorgestellt, die aufgrund ihrer umfassenden hardwareverständlichen Spezifikation von Dateneigenschaften als

Datenspezifikationsarchitektur

bezeichnet wird und datenflussbezogene Fehlervermeidung und -erkennung in einem deutlich höheren Ausmaß als bisher bekannte Architekturen bietet.

In diesem Kapitel wird zunächst der Systemaufbau eines auf den Merkmalen einer Datenspezifikationsarchitektur DSA basierenden Systems vorgestellt. Dazu wird die grundlegende Struktur technischer Prozesse mit konventionellen und intelligenten Sensoren betrachtet und darauf aufbauend ein DSA-System ohne und mit Einsatz von Redundanz beschrieben. Es folgt die Sammlung von Eigenschaften von in sicherheitsgerichteten Echtzeitsystemen erzeugten, verarbeiteten und genutzten Daten. Anschließend wird erläutert, wie diese in Form von Kennungen den Daten untrennbar hinzugefügt werden und welche Fehler- und Angriffsarten damit erkannt werden können. Die vorgestellten Kennungen werden dann nochmals übersichtlich zusammengefasst. Die Pseudocodeauflistung einer einfachen Instruktion zeigt den Umfang der durch die Hardware durchgeführten Prüfungen. Weitere Inhalte des Kapitels sind die Beschreibung von Anforderungen, die an die einzelnen Komponenten eines DSA-Systems gestellt werden, die Konfiguration der Systemkomponenten und die Vorstellung von Anforderungen an Begutachtungen und Audits, z. B. im Zuge von Zertifizierungsmaßnahmen.

4.1 Systemaufbau und Fehlerbehandlung

In diesem Unterkapitel wird zunächst der grundlegende Systemaufbau eines technischen Prozesses gezeigt, um anschließend den Aufbau eines die Merkmale einer Datenspezifikationsarchitektur DSA nutzenden Systems vorzustellen. Dabei wird ebenfalls erläutert, wie in den entsprechenden Systemen Fehler erkannt und behandelt werden können.

4.1.1 Grundlegender Systemaufbau technischer Prozesse

Zum Verständnis der Darstellungen und Erläuterungen in dieser Arbeit soll an dieser Stelle noch einmal ein technischer Prozess – gleich welcher Art, ob chemischer Prozess oder Steuerung der Bremse in einem Fahrzeug – ganz allgemein vorgestellt werden. In Abbildung 4.1 wird die Grundstruktur eines solchen Prozesses gezeigt.

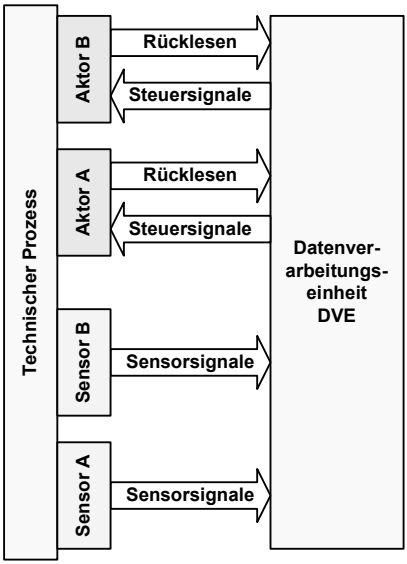


Abbildung 4.1: Technischer Prozess mit herkömmlichen Sensoren

Auf der linken Seite ist der eigentliche technische Prozess symbolisch dargestellt. In diesen sind Sensoren eingebracht, die bestimmte Größen des Prozesses erfassen, in ein analoges Spannungs- oder Stromsignal umwandeln und an die Datenverarbeitungseinheit oder mehrere dieser Einheiten weiterleiten. Bei der Verarbeitung der Sensordaten werden Stellgrößen berechnet, die anschließend an die Aktoren weitergeleitet werden, die dadurch den technischen Prozess beeinflussen. In sicherheitsgerichteten Anwendungen werden die Aktorausgaben – wie in der Abbildung gezeigt – zurückgelesen, um sie mit der vorgegebenen Stellgröße zu vergleichen und somit Fehlfunktionen der Aktoren aufdecken zu können.

Heutzutage kommen immer häufiger sogenannte intelligente Sensoren zum Einsatz [105]. Diese beinhalten neben der Messgrößenerfassung und -umwandlung gleichzeitig eine Mikroprozessoreinheit, die eine Vorverarbeitung der Daten vornimmt und diese über störsichere digitale Kommunikationsverbindungen wie z. B. Feldbusse an die Datenverarbeitungseinheiten übermittelt [105]. Damit erübrigen sich die meisten störanfälligen analogen Signalleitungen und es ergibt sich der in Abbildung 4.2 dargestellte Systemaufbau. Die erfassten Sensorsignale, die Stellgrößen für die Aktoren und ggf. auch die zurückgelesenen Aktorausgaben werden dabei über eine einzige Feldbusleitung übertragen.

Im Allgemeinen werden die Prozessgrößen zyklisch erfasst und ausgewertet, ebenso werden die Stellgrößen zyklisch an die Aktoren übermittelt. Während konventionelle Sensoren ihre analogen Ausgangsgrößen kontinuierlich den Datenverarbeitungseinheiten oder externer Peripherie zur Verfügung stellen und diese dann zyklisch eine Analog-Digitalwandlung vornehmen, können intelligente Sensoren ihre Messgrößen direkt zyklisch erfassen und z. B. über einen Feldbus den Datenverarbeitungseinheiten zur Verfügung stellen.

4.1.2 Aufbau eines auf einer Datenspezifikationsarchitektur basierenden Systems

Je nachdem, ob ein technischer Prozess einen sicheren Zustand besitzt, und den jeweiligen Anforderungen an die Verfügbarkeit des Systems, kann dessen Auslegung ein- oder mehrkanalig erfolgen, z. B. durch Anwendung geeigneter Redundanzmaßnahmen. Für beide Fälle werden in den beiden folgenden Unterkapiteln exemplarische Systemaufbauten vorgestellt.

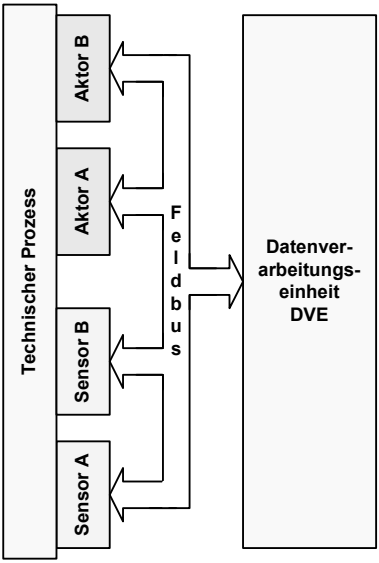


Abbildung 4.2: Technischer Prozess mit intelligenten Sensoren

4.1.2.1 Systemaufbau ohne Redundanz

In Abbildung 4.3 wird der Aufbau eines auf den Merkmalen einer Datenspezifikationsarchitektur DSA aufbauenden Systems gezeigt.

An den technischen Prozess – links im Bild – sind diverse intelligente Sensoren und Aktoren angeschlossen. Die Sensoren sind mit der Datenverarbeitungseinheit DVE verbunden und senden ihre Sensorsignale an diese. Zeitgleich übermitteln sie ein Lebenszeichen LZ an die DVE, anhand dessen diese die fehlerfreie Funktion des jeweiligen Sensors erkennen kann. Stellt ein Sensor einen Fehler fest, so stellt er umgehend die Generierung des Lebenszeichens ein. Die DVE erkennt diesen Zustand und kann entsprechend reagieren, indem sie den Betrieb basierend auf den verbleibenden Sensoren fortsetzt oder den Übergang in einen sicheren Zustand auslöst. Die DVE verarbeitet die empfangenen Sensorsignale zu Steuersignalen für die Aktoren und reicht diese an die ausfallsicherheitsgerichtet gestaltete Systemüberwachungseinheit SÜE weiter, die diese an die Aktoren weiterleitet und die von den Aktoren zurückgelesenen Steuersignale auswertet. Die SÜE empfängt und überwacht weiterhin Lebenszeichen von der DVE und den Aktoren. Stellt sie einen Fehlerzustand innerhalb des Systems fest, da

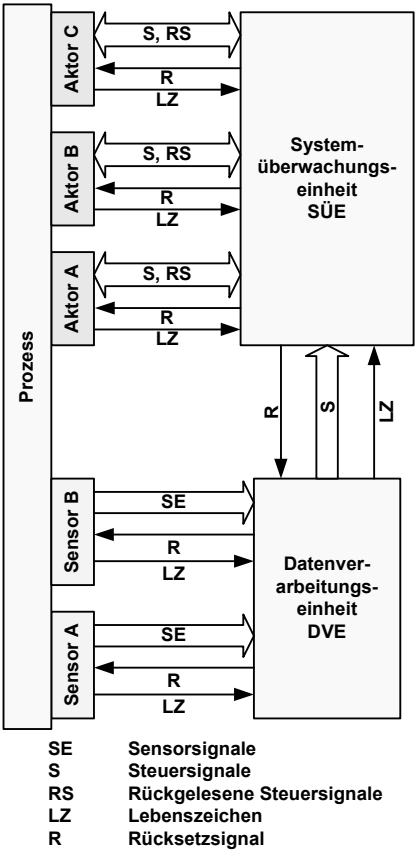


Abbildung 4.3: Aufbau eines DSA-Systems ohne Redundanz

- ein rückgelesenes Steuersignal nicht mit dem gesendeten übereinstimmt oder
- ein Aktor bzw. die Datenverarbeitungseinheit kein Lebenszeichen mehr generiert,

so kann die SÜE den Übergang in und das Halten eines sicheren Zustands auslösen, indem sie entsprechende Steuersignale an die Aktoren sendet.

Über Rücksetzsignale R kann die SÜE die Aktoren und die DVE beim Systemstart oder im Fehlerfall zurücksetzen. Die DVE gibt dieses Signal an die angeschlossenen Sensoren weiter.

4.1.2.2 Systemaufbau mit Redundanz

Besitzt ein technischer Prozess keinen sicheren Zustand oder es bestehen anderweitige erhöhte Anforderungen an die Verfügbarkeit des jeweiligen Systems, so ist dieses mit hinreichenden Redundanzmaßnahmen auszulegen. Ein entsprechendes System mit n-facher Redundanz, also mit n Datenverarbeitungseinheiten DVE 1 bis n, wird in Abbildung 4.4 gezeigt. Dabei ist zu beachten, dass ohne den Einsatz sinnvoller Diversitätsarten ein Ausfall mehrerer Datenverarbeitungseinheiten aufgrund der gleichen Fehlerursache droht, wie es z. B. bei der Ariane 5 der Fall war, wie in Kapitel 1.1.1 beschrieben.

In dem dargestellten redundanten System vergleicht die Systemüberwachungseinheit SÜE die von den Datenverarbeitungseinheiten DVE 1 bis DVE n gelieferten Steuersignale S miteinander und kann im Falle von Abweichungen entweder – sofern vorhanden – den Wechsel in und das Halten eines sicheren Zustands auslösen oder den Betrieb auf Basis eines Mehrheitsentscheides fortsetzen. Weiterhin überwacht die SÜE die Lebenszeichen LZ der einzelnen DVE, um bei deren Ausbleiben ähnliche Schritte wie bei Abweichungen der Steuersignale einzuleiten. Die überstimmte Einheit oder überstimmten Einheiten bzw. die Einheit oder die Einheiten, die kein LZ mehr generieren, da sie einen Fehler festgestellt haben, können u. U. mit Hilfe des Rücksetzsignals R neu gestartet werden, um die vollständige Funktionalität des Systems wiederherzustellen.

4.1.3 Fehlerbehandlung in einer Datenspezifikationsarchitektur

Die Behandlung erkannter Fehler in einer Datenspezifikationsarchitektur wird in diesem Unterkapitel erläutert. Dabei wird zwischen Fehlern, die bei der Verarbeitung der Daten erkannt werden, und typischen Übertragungsfehlern unterschieden,

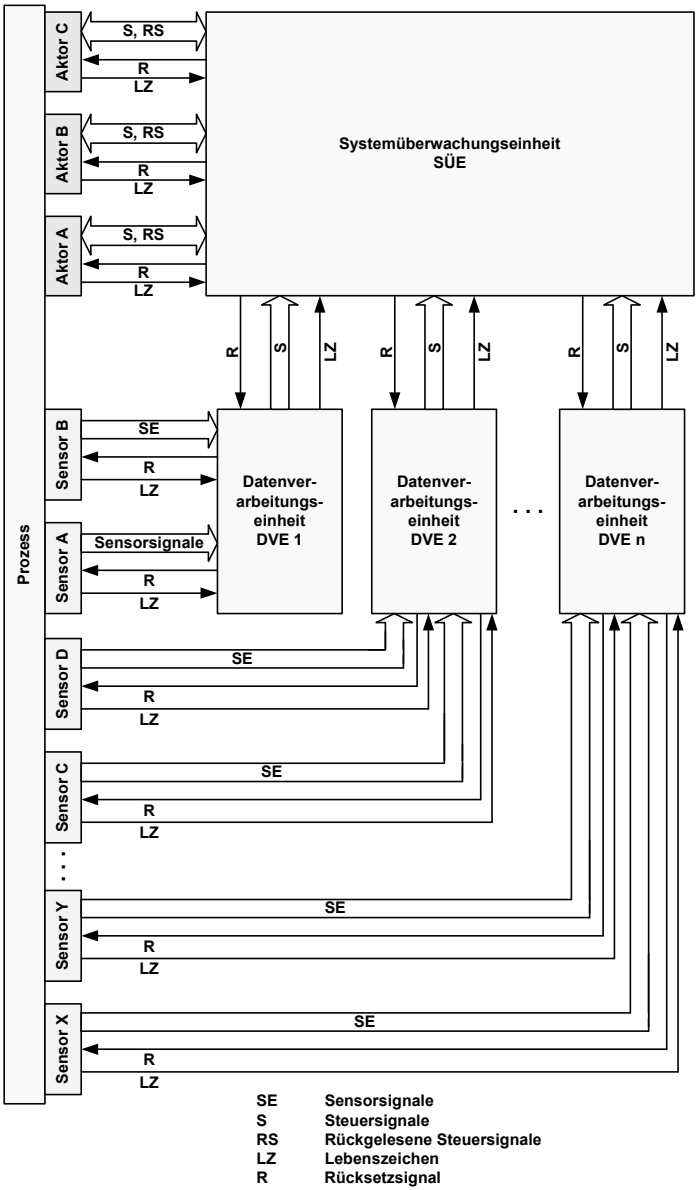


Abbildung 4.4: Aufbau eines DSA-Systems mit n-facher Redundanz

da letztere vergleichsweise häufig auftreten und entsprechend toleriert werden müssen.

4.1.3.1 Behandlung von bei der Datenverarbeitung erkannten Fehlern

Wird ein Fehler innerhalb einer Systemkomponente erkannt, so wird ein Ausnahmefehler generiert. In konventionellen Architekturen wird daraufhin in der Regel eine Softwarebehandlungsroutine für die betreffende Fehlerart aufgerufen und die Behandlung des Fehlers damit der Software überlassen. Bei der in Kapitel 3.7 vorgestellten inhärent sicheren Mikroprozessorarchitektur ISMA [125] wird bei der Generierung eines Ausnahmefehlers hingegen die Programmausführung abgebrochen und die Erzeugung des Lebenszeichensignals eingestellt. Die übergeordnete Systemüberwachungseinheit erkennt durch Beobachtung des Lebenszeichens oder der Lebenszeichen Ausfälle der überwachten Datenverarbeitungseinheiten und kann dann – je nach Systemaufbau – den Wechsel in und das Halten eines sicheren Zustands herbeiführen oder den Betrieb des Systems basierend auf den eventuell verbleibenden redundanten Datenverarbeitungseinheiten aufrechterhalten.

Eine entsprechende Reaktion auf erkannte Fehler soll auch ein auf den Merkmalen einer Datenspezifikationsarchitektur basierendes System aufweisen. Dabei stellen die Datenverarbeitungseinheiten die Erzeugung ihres Lebenszeichensignals neben der Erkennung von Fehlern auch dann ein, wenn das Lebenszeichen eines der angeschlossenen Sensoren ausbleibt.

Bei der Beschreibung der Fehlererkennungsmechanismen der in dieser Arbeit vorgestellten Datenspezifikationsarchitektur wird jeweils angegeben, unter welchen Umständen ein Ausnahmefehler generiert wird, der die beschriebene Fehlerbehandlung nach sich zieht.

4.1.3.2 Behandlung von Übertragungsfehlern

Bei der Übertragung von Daten können diese durch Fehler und Störungseinflüsse verfälscht werden. Ein Merkmal für die Qualität der Übertragung bzw. der Übertragungsstrecke ist dabei die sogenannte Bitfehlerrate – engl. „bit error rate BER“ –, die sich aus der Anzahl der fehlerhaften Bits einer übertragenen Menge an Bits nach

$$\mathrm{BER} = \frac{\text{Anzahl fehlerhaft übertragener Bits}}{\text{Gesamtzahl der übertragenen Bits}}$$

berechnet [64]. Auf durch entsprechende Fehlererkennungsmaßnahmen erkannte Datenübertragungsfehler auf den Kommunikationsstrecken

- zwischen den Sensoren und den Datenverarbeitungseinheiten DVE,
- zwischen verschiedenen Datenverarbeitungseinheiten DVE,
- zwischen Datenverarbeitungseinheiten DVE und der Systemüberwachungseinheit SÜE und
- zwischen der Systemüberwachungseinheit SÜE und den Aktoren

könnte ebenfalls mit der Generierung eines Ausnahmefehlers reagiert und das System – sofern vorhanden – in einen sicheren Zustand gebracht werden. Da Bitfehler bei der Übertragung von Daten über Kommunikationsleitungen jedoch verhältnismäßig häufig auftreten, wie sich an den aus [64] entnommenen Angaben in Tabelle 4.1 erkennen lässt, hätte dies äußerst ungünstige Auswirkungen auf die Verfügbarkeit des betroffenen Systems. Sinnvoller ist es daher, dem Sender der als fehlerhaft identifizierten empfangenen Daten die fehlerhafte Übertragung mitzuteilen und ein erneutes Senden der betroffenen Daten anzufordern.

Tabelle 4.1: Typische Bitfehlerraten nach [64]

Übertragungsmedium	**Typische Bitfehlerrate**
Analoge Fernsprechleitungen	10^{-5}
Digitale Übertragungsleitungen	10^{-6} bis 10^{-7}
Koaxialkabel	10^{-9}
Glasfaserkabel	10^{-12}

4.2 Sammlung relevanter Dateneigenschaften

In den gängigen Architekturen werden Daten meist einzig durch den Datenwert repräsentiert. Alle weiteren Eigenschaften der Datenwerte ergeben sich nur implizit durch die Art des Zugriffs und der Verwendung. Eine Ausnahme bilden hier die

großteils in Vergessenheit geratenen Datentyp-, -struktur- und Befähigungsarchitekturen, die weitere Dateneigenschaften untrennbar mit den Daten verbinden und auf diese Weise sehr einfache und leistungsfähige hardwarebasierte Fehlererkennung ermöglichen. Aber selbst diese Architekturen spezifizieren nur wenige Eigenschaften der Daten, wie Datentyp und Zugriffsrechte. Entsprechend wenige datenflussbezogene Fehler- und Angriffsarten lassen sich mit dem Stand von Wissenschaft und Technik erkennen, wie die Auswertung in Kapitel 3.12 gezeigt hat.

Dabei gibt es wesentlich mehr Dateneigenschaften, die es zu berücksichtigen gilt. Tritt man einen Schritt zurück und löst sich von den bekannten Ansätzen und Mechanismen, dann lassen sich die folgenden Eigenschaften von Datenwerten identifizieren, die im Besonderen bei sicherheitsgerichteten Echtzeitsystemen eine wichtige Rolle spielen:

- der Datenwert W,
- die Genauigkeit GE des Datenwerts W, da jeder Messwert einer entsprechenden Messunsicherheit unterliegt,
- der Wertebereich WB, innerhalb dessen sich der Datenwert W befinden darf,
- der Datentyp DT des Datenwerts, der das Format bzw. die Darstellung von W und WB spezifiziert,
- die Einheit EI des Datenwerts W, angegeben in vorzeichenbehafteten Potenzen der SI-Basiseinheiten,
- die Zugriffsrechte ZR auf das Datenspeicherelement, die Lesbarkeit, Schreibbarkeit und Rechtebesitzer identifizieren,
- der Initialisierungsstatus IS, anhand dessen geprüft werden kann, ob das Datenspeicherelement lesbare Daten enthält, also initialisiert ist,
- die Herkunft bzw. Quelle Q eines Datenwerts, also die Aussage darüber, wer die Daten erzeugt hat, z. B. welcher Sensor,
- der Verarbeitungsweg VW, der spezifiziert, welche Verarbeitungseinheiten oder Programmteile ein Datenwert durchlaufen darf,
- das Ziel oder die Ziele Z, also die Angabe, welche Senke oder welche Senken die Datenwerte nach ihrer Verarbeitung schlussendlich entgegennehmen werden, z. B. ein Aktor oder mehrere Aktoren,
- den diskreten Entstehungszeitpunkt bzw. Zeitschritt ZS, z. B. die Nummer eines diskreten Abtastzeitpunkts,

- die Frist FR, also der Zeitpunkt, bis zu der die Daten zur Verarbeitung verwendet werden dürfen,
- die Zykluszeit ZY, bestehend aus einem frühesten und einem spätesten zulässigen Zeitpunkt, innerhalb derer zyklisch erzeugte Daten des nächsten diskreten Zeitschritts in der Verarbeitungseinheit oder Senke zur Verfügung stehen müssen,
- die Adresse oder ein Identifikator der Daten AD zur Erkennung von Adressierungsfehlern,
- die Integritätsprüfung IP, anhand derer geprüft werden kann, ob die Daten korrekt übertragen und verarbeitet wurden, z. B. durch entsprechende Fehlererkennungskodierung und
- die kryptographische Signatur S, die die Prüfung der Authentizität des Datenspeicherelements erlaubt.

Wie bereits im Einführungskapitel erwähnt, wird Iliffe in [37] von Feustel dahingehend zitiert, dass die Eigenschaften von Datenfeldern explizit und untrennbar von den eigentlichen Daten in den Feldern selbst enthalten sein sollen, anstatt durch die Art des Zugriffs auf die Felder und die Verwendung der enthaltenen Daten impliziert zu werden. Auch wenn sich diese Anforderung zunächst nur auf die Datentypen und die Anzahl der Elemente innerhalb von Datenfeldern bezog, soll sie jedoch zu dem folgenden zentralen Entwurfsparadigma für eine Datenspezifikationsarchitektur verallgemeinert werden:

> *Alle ein Datenspeicherelement beschreibenden Eigenschaften sollen untrennbar mit diesem verknüpft, gespeichert, übertragen, verarbeitet und in einer hardwareverständlichen und -überprüfbaren Form dargestellt werden.*

Diesem Ansatz folgend, ergibt sich somit für ein Datenspeicherelement, welches in einer Datenspezifikationsarchitektur DSA neben dem Datenwert W auch Angaben zu den genannten weiteren Dateneigenschaften enthält, die folgende Tupeldarstellung:

$$D := (W, GE, WB, DT, EI, ZR, IS, Q, VW, Z, ZS, FR, ZY, AD, IP, S)$$

Im Verlauf dieser Arbeit werden an einigen Stellen Optimierungen der Dateneigenschaften vorgenommen und z. B. bestimmte Eigenschaften zusammengefasst. Ein

Beispiel dafür ist die in der IEC 61508-2 [51] vorgeschlagene Integration der Datenadresse AD in die Integritätsprüfung IP, wodurch dedizierte AD-Kennungen unnötig werden und Adressierungsfehler trotzdem zuverlässig aufdeckbar sind.

4.3 Realisierung der Datenflussüberwachung

Im Folgenden soll aufgezeigt werden, auf welche Weise eine entsprechend gestaltete Datenspezifikationsarchitektur den Datenfluss innerhalb eines gesamten Systems anhand der identifizierten Dateneigenschaften überwachen kann. Besonders wichtig ist es dabei, Systemkomponenten nicht als unabhängige Subsysteme zu betrachten und isoliert zu spezifizieren, sondern das System als große Einheit zu sehen, innerhalb derer die Datenflüsse einen vorher festgelegten Pfad – z. B. von den Sensoren durch eine oder mehrere Datenverarbeitungseinheiten bis hin zu den Aktoren – durchlaufen sollen. Wieso sollte nicht bereits ein intelligenter Sensor einen von ihm gelieferten Datenwert mit einer entsprechenden Markierung versehen, die angibt, welcher „Endkunde“ – also z. B. ein Aktor – die Daten schlussendlich nach ihrer Verarbeitung entgegennehmen wird?

Bevor die Einbettung der einzelnen Dateneigenschaften in den Datenspeicherelementen vorgestellt wird, soll zunächst erläutert werden, wie diese im Einzelnen beschrieben werden.

4.3.1 Einleitende Erläuterungen

Zum besseren Verständnis der folgenden Unterkapitel sollen an dieser Stelle zunächst die Gliederung der Vorstellung der einzelnen Dateneigenschaften und die Darstellung der Kennungen und ggf. vorhandener Teilkennungen in den Daten- und Befehlsspeicherelementen erläutert werden. Anschließend wird erklärt, wie die Pseudocodeauflistungen aufgebaut sind, die zur Veranschaulichung von Prüfungen durch die Hardware und der Funktionalität von Befehlen genutzt wird. Weiterhin wird der in ISMA [125] geprägte Begriff der Formatierung von Datenspeicherelementen und dessen Relevanz im Rahmen dieser Arbeit umrissen.

4.3.1.1 Gliederung der Vorstellungen

Die identifizierten Eigenschaften der Daten in sicherheitsgerichteten Echtzeitsystemen, die in den Datenspeicherelementen explizit in hardwareles-, -prüf- und

-verarbeitbaren Kennungen abgelegt werden sollen, werden in den folgenden Unterkapiteln detailliert vorgestellt. Einige Eigenschaften werden dabei in einer Kennung zusammengefasst. Die Erläuterungen gliedern sich in

- eine Kurzbeschreibung der Dateneigenschaft bzw. der Dateneigenschaften,
- die Abbildung der Eigenschaft oder der Eigenschaften in einer Kennung innerhalb eines Datenspeicherelements und der eventuell dazugehörigen Kennung innerhalb eines Befehlsspeicherelements und speziellen Registern,
- die unter Nutzung der jeweiligen Kennung durchgeführten Prüfungen zur Fehlererkennung,
- die Erläuterung, wie die Inhalte der vorgestellten Kennung von Ergebnissen von Operationen ermittelt und gesetzt werden,
- die zur Verwaltung der entsprechenden Kennung vorgesehenen Befehle,
- die Präsentation von Anwendungsbeispielen zur Veranschaulichung der Verwendung der Kennung zur Fehlererkennung,
- die Vorstellung existierender Möglichkeiten oder alternativer bzw. neuer Vorschläge zur Spezifikation der Dateneigenschaft oder Dateneigenschaften in Hochsprachen und
- die Evaluation der auf Basis der Kennung und der vorgestellten Prüfungen erkennbaren Fehler- und Angriffsarten aus Kapitel 2.4,

wobei nicht alle aufgeführten Gliederungspunkte bei allen Kennungen vorhanden sind.

4.3.1.2 Darstellung der Kennungen in Daten- und Befehlsspeicherelementen

Eine Kennung wird in den Erläuterungen in dem in Abbildung 4.5 gezeigten Format vorgestellt. Dabei wird der Kennungsname – im Beispiel AB – innerhalb der Kennung angegeben.

Einige der Kennungen sind in mehrere Teilkennungen untergliedert, wie in Abbildung 4.6 dargestellt. Im Falle des Beispiels wären das die Teilkennungen AB, CD und EF. Der Name der ersten Teilkennung stimmt absichtlich mit dem Namen der Gesamtkennung überein, da dies bei manchen Kennungen vorkommt.

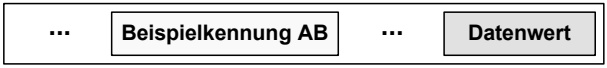


Abbildung 4.5: Beispielkennung AB in einem Datenspeicherelement

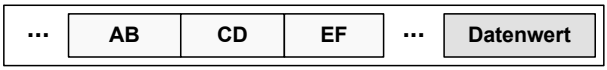


Abbildung 4.6: Aufbau der Beispielkennung AB im Datenspeicherelement

Die Kennungen innerhalb von Befehlsspeicherelementen werden identisch zu jenen der Datenspeicherelemente dargestellt, wie in Abbildung 4.7 gezeigt. Im Beispiel wurde der zur Beispielkennung AB im Datenspeicherelement identische Name AB für die Beispielkennung gewählt, da beide Kennungen zusammenhängen.

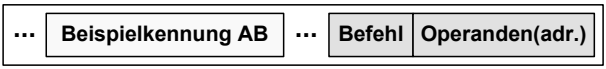


Abbildung 4.7: Beispielkennung AB in einem Befehlsspeicherelement

Auch die Kennungen innerhalb von Befehlsspeicherelementen können in Teilkennungen unterteilt sein. Im in Abbildung 4.8 dargestellten Beispiel sind dies die Teilkennungen XY und ZA.

Im Text wird die Kennung mit ihrem Namen gefolgt von ihrer Abkürzung genannt. In formalen Darstellungen wird nur die Abkürzung genutzt.

4.3.1.3 Nutzung von Pseudocode zur Veranschaulichung der Funktion von Prüfungen und Befehlen

Die Vorstellungen der durchgeführten Prüfungen und der Befehle zur Verwaltung der Kennungen werden zum besseren Verständnis durch Auflistungen in Pseudocode ergänzt. Prüfungen, die die Hardware bei Zugriffen auf Datenspeicherelemente durchführt, werden mit dem Namen der Prüfung eingeleitet, wie in der folgenden Auflistung gezeigt.

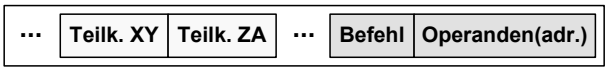


Abbildung 4.8: Aufbau der Beispielkennung AB im Befehlsspeicherelement

```
Prüfung_<Name der Prüfung> :=

<Auflistung der durchgeführten Prüfungen>
```

Werden Kennungen des Ergebnisses einer Operation durch diese auf bestimmte Werte gesetzt, so kommen zur Veranschaulichung ebenfalls Pseudocodeauflistungen zum Einsatz.

```
Setzen_<Name der Kennung>_in_Ergebnis :=

<Erläuterung des Setzens der entsprechenden Kennung>
```

Die Beschreibung von Befehlen beginnt mit der Abkürzung des Befehls und einer Auflistung der Namen seiner Operanden.

```
<Befehlsname> <Operandennamen> :=

<Erläuterung der Funktionalität>
```

Die bei einer Prüfung einbezogenen Datenspeicherelemente werden – je nach Art des Zugriffs – als „Quelle“ oder „Ergebnis“ bezeichnet, während bei der Beschreibung eines Befehls die jeweiligen Operandenbezeichnungen verwendet werden. Weiterhin bezeichnet „Ergebnis“ ebenfalls die Ergebnisse von durchgeführten Operationen, die die arithmetisch-logische Einheit ALE aus den Operanden berechnet hat.

Zur einfachen Beschreibung der durchgeführten Prüfungen und Befehle werden verschiedene Funktionen definiert. Diese sind:

- `W([<Operandenname>])` liefert den Datenwert W des Operanden zurück,
- `<Kennungsname>([<Operandenname>])` liefert den Inhalt der angegebenen Kennung des Operanden zurück,

- `<Kennungsname>.<Teilkennungsname>([<Operandenname>])` liefert den Inhalt der angegebenen Teilkennung des Operanden zurück und
- `[<Operandenname>]` liefert den gesamten Inhalt eines Datenspeicherelements zurück, was z. B. im Zuge der Integritätsprüfung genutzt wird.

Die beschriebenen Funktionen können auch zum Referenzieren der Kennungen bzw. deren Teilkennungen innerhalb der Befehlsspeicherelemente verwendet werden. Dies wird durch

- `<Kennungsname>([Befehl])` bzw.
- `<Kennungsname>.<Teilkennungsname>([Befehl])`

dargestellt. Neben Kennungen in Daten- und Befehlsspeicherelementen kommen zur Fehlererkennung bei der Abbildung von Dateneigenschaften auch spezielle Register zum Einsatz. Diese werden im Pseudocode analog zu den Kennungen verwendet, wobei

- `[<Registername>]` den gesamten Inhalt des referenzierten Registers und
- `[<Registername>.<Teilregistername>]` den Inhalt des spezifizierten Teilregisters

zurückliefern.

Zuweisungen werden durch die Nutzung des Zuweisungsoperators „:=“ gekennzeichnet:

- `W([<Operandenname>]) := x` setzt den Datenwert W des angegebenen Operanden auf den Wert x,
- `<Kennungsname>([<Operandenname>]) := y` setzt den Inhalt der angegebenen Kennung auf den Wert y,
- `<Kennungsname>.<Teilkennungsname>([<Operandenname>]) := z` setzt den Inhalt der angegebenen Teilkennung auf den Wert z,
- `[<Operandenname_1>] := [<Operandenname_2>]` überträgt sämtliche Inhalte von einem in den anderen Operanden,
- `[<Registername>] := a` weist dem angegebenen Register den Wert a zu und
- `[<Registername>.<Teilregistername>] := b` legt den Wert b im spezifizierten Teilregister ab.

Die beschriebene Art der Darstellung von Prüfungen und Befehlen mit der Nutzung der Kennungen von Operanden und Befehlen werden in den zwei folgenden Auflistungen gezeigt. Stellt die Hardware bei der Ausführung einer Prüfung oder eines Befehls einen Fehler fest, so wird ein Ausnahmefehler generiert, in den Auflistungen dargestellt durch „Generierung_Ausnahmefehler“.

Bei der Prüfung Prüfung_Beispiel wird die Einhaltung verschiedener Bedingungen verifiziert. Die Inhalte der Teilkennungen AB.AB, AB.CD und AB.EF des Quelldatenspeicherelements müssen in vorgegebenen Beziehungen zueinander stehen. Stellt die Hardware fest, dass eine der Bedingungen nicht eingehalten wird, so wird ein Ausnahmefehler generiert.

```
Prüfung_Beispiel :=

WENN AB.AB([Quelle]) ≠ AB.CD([Quelle]) DANN
  Generierung_Ausnahmefehler;
SONST
  WENN AB.CD([Quelle]) > AB.EF([Quelle]) DANN
    Generierung_Ausnahmefehler;
  ENDEWENN
ENDEWENN
```

Der Befehl XYZ besitzt die drei Operanden A, B und C und die Kennung AB, die die Teilkennungen XY und ZA enthält.

```
XYZ A, B, C :=

WENN AB.AB([A]) < AB.CD([B]) ∨ AB.EF([B]) > AB.XY([Befehl]) DANN
  Generierung_Ausnahmefehler;
SONST
  W([C]) := W([A]) + W([B]);
  AB.AB([C]) := AB.AB([A]) + AB.AB([B]);
ENDEWENN
```

Vor der Ausführung der eigentlichen Funktion des Befehls XYZ, werden verschiedene Prüfungen der Inhalte der Teilkennungen der drei Operanden und einer Teilkennung des Befehls vorgenommen. Schlägt eine der Prüfungen fehl, so wird ein Ausnahmefehler generiert. Ansonsten wird dem Datenwert von C die Summe der Datenwerte

von A und B zugewiesen. Zusätzlich wird die Teilkennung AB.AB von C auf die Summe der Inhalte der Teilkennungen AB.AB der Operanden A und B gesetzt.

Neben gängigen mathematischen Operatoren kommen in den Pseudocodeauflistungen auch die logischen Operationen

- bitweise Disjunktion, dargestellt durch „ODER“,
- bitweise Konjunktion, dargestellt durch „UND“ und
- bitweise Antivalenz, dargestellt durch „EXODER“

vor.

4.3.1.4 Begriff der Formatierung von Speicherelementen

ISMA [125] versieht alle Datenspeicherelemente, die durch ein Programm genutzt werden sollen, beim Programmstart mit einer Ausgangskonfiguration, bei der z. B. der Datentyp DT auf „undefiniert“ und der Initialisierungsstatusbeschreiber IS auf 0, also „nicht initialisiert“, gesetzt werden. Damit kann sichergestellt werden, dass die Datenspeicherelemente zu jedem Zeitpunkt verwertbare Informationen enthalten und nicht z. B. nach einem Neustart Werte des letzten Programmablaufs oder gar zufällige Werte enthalten. Dieser Vorgang des Setzens einer Ausgangskonfiguration wird bei ISMA **Formatierung** genannt.

Im Gegensatz zu konventionellen Architekturen, bei denen Zieldatenspeicherelemente keinerlei Prüfungen vor der Wertzuweisung unterzogen werden – es werden höchstens die Zugriffsberechtigungen anhand der Adresse durch Segmentierung oder Seitenverwaltung geprüft –, können bei formatierten Zieldatenspeicherelementen zusätzliche Prüfungen durch die Hardware einer Datenspezifikationsarchitektur vorgenommen werden. Ein gutes Beispiel dafür ist die Prüfung, ob das Ergebnis einer Operation innerhalb des Wertebereichs des Zieldatenspeicherelements liegt.

Diese Möglichkeit der Durchführung weiterer Prüfungen im Rahmen von Zuweisungen soll auch innerhalb der Datenspezifikationsarchitektur verwendet werden.

4.3.2 Datenwert und dessen Genauigkeit

Die in technischen Prozessen erfassten Prozessgrößen unterliegen stets einer gewissen Messunsicherheit [117]. Daher ist davon auszugehen, dass die von Sensoren zur

Verfügung gestellten Datenwerte W eine bestimmte Genauigkeit GE aufweisen und die Genauigkeit eine grundlegende Eigenschaft dieser Datenwerte darstellt.

Nach [117] wird ein Messwert x durch

$$x = x_{\text{Best}} \pm \delta x$$

dargestellt, wobei x_{Best} als „Bestwert“ oder „bester Schätzwert“ bezeichnet wird und δx den absoluten Fehler der Messung spezifiziert. Der Fehler der Messung kann auch in relativer Form als

$$\frac{\delta x}{|x_{\text{Best}}|}$$

angegeben werden.

4.3.2.1 Realisierung des Datenwerts W

Statt – wie bei konventionellen Architekturen üblich – einen Datenwert W als einzelne Zahl anzugeben, soll dieser in einer Datenspezifikationsarchitektur DSA als Intervall formuliert werden, welches die Genauigkeit des Werts durch die Intervallgrenzen abbildet. Diese ergeben sich durch das Addieren bzw. Subtrahieren des absoluten Fehlers δx vom Bestwert x_{Best}.

Damit ergibt sich für den Datenwert W die Darstellung

$$W := (W_{min}, W_{max}),$$

wobei W_{min} und W_{max} die Intervallgrenzen definieren und somit die Genauigkeit des Datenwerts wiedergeben. In den Datenspeicherelementen wird der Datenwert im Datenwertfeld W spezifiziert, wie in Abbildung 4.9 dargestellt. Zusätzlich ist die Datentypkennung angedeutet, die den Aufbau und den Datentyp der im Datenwertfeld W enthaltenen Datenwerte angibt. Details zur Datentypkennung DT werden im Kapitel 4.3.4 erläutert. An dieser Stelle reicht jedoch das Wissen über deren grobe Bedeutung aus.

Konventionelle Datentypen, die keiner Messunsicherheit unterliegen und deren Datenwert nicht als Intervall, sondern als einzelner Datenwert gespeichert werden, nutzen nur die unteren 64 Bit des Datenwertfelds W, wie in Abbildung 4.10. Die oberen 64 Bit sind reserviert und sollen – dem Vorbild von ISMA [125] folgend – mit einem alternierenden Bitmuster gefüllt werden.

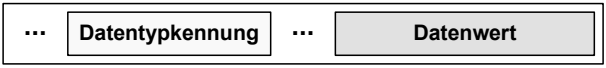


Abbildung 4.9: Datenwertfeld W in Datenspeicherelementen

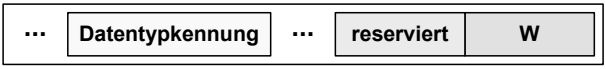


Abbildung 4.10: Aufbau des Datenwertfelds W bei konventionellen Datentypen

Bei Messdatentypen wird der Datenwert in Form eines Intervalls angegeben, wofür im Datenwertfeld W die beiden Intervallgrenzen W_{min} und W_{max} spezifiziert werden. Abbildung 4.11 veranschaulicht den sich dadurch ergebenden Aufbau des Datenwertfelds W.

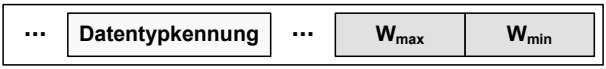


Abbildung 4.11: Aufbau des Datenwertfelds W bei Messwertdatentypen

In Tabelle 4.2 werden die vorzeichenlosen und -behafteten Ganzzahldatentypen, sowie die Gleitkommadatentypen mit einfacher und doppelter Genauigkeit von ISMA [125] gezeigt. Aus Platzgründen werden dabei die verschiedenen Bitbreiten einer Datentypart in einer Zeile zusammengefasst. In der letzten Spalte sind die Kürzel der zugehörigen Messwertdatentypen aufgeführt. So ist z. B. der zum konventionellen Ganzzahldatentyp GZ8 mit 8 Bit Breite äquivalente Messwertdatentyp dabei GZM8, wobei die beiden Intervallgrenzen jeweils im Wertebereich zwischen 0 und 2^8-1 liegen können.

4.3.2.2 Arithmetische Operationen mit Messwertdatentypen

Operationen von Operanden, die nicht aus der Menge der Messwertdatentypen stammen und somit keiner Messunsicherheit unterworfen sind, werden auf gewohnte Weise berechnet. Sind an einer Operation jedoch Messwertdatentypen beteiligt, gelten die Verarbeitungsregeln der Intervallarithmetik, die in den folgenden Unterkapiteln kurz vorgestellt werden und auf [65] basieren. Details zu Optimierungs-

Tabelle 4.2: Konventionelle und Messwertdatentypen

Datentyp	**Bitbreite**	**Erläuterung**	**zugehöriger Messwertdaten-typ**
GZ{8,16,32,64}	{8,16,32,64}	Ganzzahl zwischen 0 und $2^{\{8,16,32,64\}}$-1	GZM{8,16,32,64}
VGZ{8,16,32,64}	{8,16,32,64}	Vorzeichenbehaftete Ganzzahl zwischen $-2^{\{7,15,31,63\}}$ und $2^{\{7,15,31,63\}}$-1	VGZM{8,16,32,64}
GKZ{8,16,32,64}	{32,64}	Gleitkommazahl mit {einfacher,doppelter} Genauigkeit zwischen $\pm 2^{\{-126,-1022\}}$ und $\pm(1\text{-}2^{\{-24,-53\}})\cdot 2^{\{128,1024\}}$	GKZM{8,16,32,64}

möglichkeiten, Rundungen und ein Vorschlag zur Hardwarerealisierung können [65] bzw. der IEEE-Norm 1788 [57] für Intervallarithmetik entnommen werden.

4.3.2.2.1 Addition

Wird zu einem Operanden mit Messwertdatentyp ein Operand ohne Messunsicherheit addiert, so wird dieser jeweils auf die untere bzw. die obere Intervallgrenze addiert, um die Intervallgrenzen des Ergebnisses zu erhalten. Aus der Berechnung ist auch ersichtlich, dass das Ergebnis der Berechnung wiederum einen Messwertdatentyp besitzt.

$$W_{\text{min,Erg}} = W_{\text{min,Quelle\_1}} + W_{\text{Quelle\_2}}$$
$$W_{\text{max,Erg}} = W_{\text{max,Quelle\_1}} + W_{\text{Quelle\_2}}$$

Bei der Addition zweier Operanden, die beide Messwertdatentypen besitzen, werden jeweils die Minima und Maxima addiert, um den minimalen und den maximalen Datenwert für das Werteintervall des Ergebnisses zu berechnen.

$$W_{\text{min,Erg}} = W_{\text{min,Quelle\_1}} + W_{\text{min,Quelle\_2}}$$
$$W_{\text{max,Erg}} = W_{\text{max,Quelle\_1}} + W_{\text{max,Quelle\_2}}$$

4.3.2.2.2 Subtraktion

Ist an einer Subtraktion ein Operand beteiligt, der keinen Messwertdatentyp besitzt, so sind zwei Fälle zu unterscheiden, je nachdem, ob Subtrahend oder Minuend den Messwertdatentypen entstammen. Besitzt der Minuend einen Messwertdatentyp, so wird der Datenwert des Subtrahenden jeweils von der unteren bzw. oberen Intervallgrenze abgezogen, um die Intervallgrenzen des Ergebnisses zu erhalten. Wie auch bei der Addition besitzt das Ergebnis der Subtraktion einen Messwertdatentyp.

$$\begin{aligned} \mathrm{W}_{\mathrm{min,Erg}} &= \mathrm{W}_{\mathrm{min,Quelle\_1}} - \mathrm{W}_{\mathrm{Quelle\_2}} \\ \mathrm{W}_{\mathrm{max,Erg}} &= \mathrm{W}_{\mathrm{max,Quelle\_1}} - \mathrm{W}_{\mathrm{Quelle\_2}} \end{aligned}$$

Besitzt der Subtrahend einen Messwertdatentyp, so errechnet sich die untere Intervallgrenze des Ergebnisses, indem vom Datenwert des Minuenden die obere Intervallgrenze des Subtrahenden abgezogen wird. Die obere Intervallgrenze des Ergebnisses entsteht durch Subtraktion der unteren Intervallgrenze des Subtrahenden vom Datenwert des Minuenden.

$$\begin{aligned} \mathrm{W}_{\mathrm{min,Erg}} &= \mathrm{W}_{\mathrm{Quelle\_1}} - \mathrm{W}_{\mathrm{max,Quelle\_2}} \\ \mathrm{W}_{\mathrm{max,Erg}} &= \mathrm{W}_{\mathrm{Quelle\_1}} - \mathrm{W}_{\mathrm{min,Quelle\_2}} \end{aligned}$$

Bei der Subtraktion von Operanden, die beide Messwertdatentypen besitzen, wird zur Berechnung des minimalen Datenwerts des Ergebnisses die obere Grenze des Subtrahenden von der unteren Intervallgrenze des Minuenden abgezogen. Der maximale Datenwert des Ergebnisses errechnet sich durch Subtraktion der unteren Grenze des Subtrahenden von der oberen Grenze des Minuenden.

$$\begin{aligned} \mathrm{W}_{\mathrm{min,Erg}} &= \mathrm{W}_{\mathrm{min,Quelle\_1}} - \mathrm{W}_{\mathrm{max,Quelle\_2}} \\ \mathrm{W}_{\mathrm{max,Erg}} &= \mathrm{W}_{\mathrm{max,Quelle\_1}} - \mathrm{W}_{\mathrm{min,Quelle\_2}} \end{aligned}$$

4.3.2.2.3 Multiplikation

Bei Multiplikationen wird unterschieden, ob nur einer oder beide Operanden einen Messwertdatentyp und damit eine Intervalldarstellung besitzen, da im ersten Fall weniger Berechnungen notwendig sind. Bei einer Multiplikation, bei der einer der Operanden einen Messwertdatentyp besitzt, werden die Produkte der Intervallgrenzen dieses Operanden mit dem Datenwert des zweiten Operanden multipliziert. Das

niedrigere Ergebnis wird als untere, der höhere als obere Intervallgrenze des Ergebnisses genutzt.

$$\begin{aligned} \mathrm{W}_{\mathrm{min,Erg}} &= \min(\mathrm{W}_{\mathrm{min,Quelle\_1}} \cdot \mathrm{W}_{\mathrm{Quelle\_2}}, \mathrm{W}_{\mathrm{max,Quelle\_1}} \cdot \mathrm{W}_{\mathrm{Quelle\_2}}) \\ \mathrm{W}_{\mathrm{max,Erg}} &= \max(\mathrm{W}_{\mathrm{min,Quelle\_1}} \cdot \mathrm{W}_{\mathrm{Quelle\_2}}, \mathrm{W}_{\mathrm{max,Quelle\_1}} \cdot \mathrm{W}_{\mathrm{Quelle\_2}}) \end{aligned}$$

Bei der Multiplikation zweier Operanden, die beide Messwertdatentypen besitzen, werden die Permutationen der Intervallgrenzen der Quelloperanden gebildet und das Minimum und das Maximum ihrer Multiplikationsergebnisse als Intervallgrenzen des Ergebnisses genutzt.

$$\begin{aligned} \mathrm{W}_{\mathrm{min,Erg}} = \min(&\mathrm{W}_{\mathrm{min,Quelle\_1}} \cdot \mathrm{W}_{\mathrm{min,Quelle\_2}}, \mathrm{W}_{\mathrm{min,Quelle\_1}} \cdot \mathrm{W}_{\mathrm{max,Quelle\_2}} \\ &\mathrm{W}_{\mathrm{max,Quelle\_1}} \cdot \mathrm{W}_{\mathrm{min,Quelle\_2}}, \mathrm{W}_{\mathrm{max,Quelle\_1}} \cdot \mathrm{W}_{\mathrm{max,Quelle\_2}}) \\ \mathrm{W}_{\mathrm{max,Erg}} = \max(&\mathrm{W}_{\mathrm{min,Quelle\_1}} \cdot \mathrm{W}_{\mathrm{min,Quelle\_2}}, \mathrm{W}_{\mathrm{min,Quelle\_1}} \cdot \mathrm{W}_{\mathrm{max,Quelle\_2}} \\ &\mathrm{W}_{\mathrm{max,Quelle\_1}} \cdot \mathrm{W}_{\mathrm{min,Quelle\_2}}, \mathrm{W}_{\mathrm{max,Quelle\_1}} \cdot \mathrm{W}_{\mathrm{max,Quelle\_2}}) \end{aligned}$$

4.3.2.2.4 Division

Auch bei der Division ist zu unterscheiden, ob nur einer der Operanden oder beide einen Messwertdatentyp besitzen, da auch hier weniger Berechnungden notwendig sind, wenn ein Operand keine Intervalldarstellung besitzt. Besitzt der Dividend einen Messwertdatentyp, so werden die Divisionen der unteren bzw. oberen Intervallgrenze des Dividenden durch den Datenwert des Divisors berechnet. Das niedrigere Ergebnis wird als untere, das höhere als obere Intervallgrenze des Ergebnisses verwendet.

$$\begin{aligned} \mathrm{W}_{\mathrm{min,Erg}} &= \min\left(\frac{\mathrm{W}_{\mathrm{min,Quelle\_1}}}{\mathrm{W}_{\mathrm{Quelle\_2}}}, \frac{\mathrm{W}_{\mathrm{max,Quelle\_1}}}{\mathrm{W}_{\mathrm{Quelle\_2}}}\right) \\ \mathrm{W}_{\mathrm{max,Erg}} &= \max\left(\frac{\mathrm{W}_{\mathrm{min,Quelle\_1}}}{\mathrm{W}_{\mathrm{Quelle\_2}}}, \frac{\mathrm{W}_{\mathrm{max,Quelle\_1}}}{\mathrm{W}_{\mathrm{Quelle\_2}}}\right) \end{aligned}$$

Besitzt nur der Divisor einen Messwertdatentyp, so erfolgt die Berechnung analog zu den obigen Gleichungen.

$$W_{\text{min,Erg}} = \min\left(\frac{W_{\text{Quelle\_1}}}{W_{\text{min,Quelle\_2}}}, \frac{W_{\text{Quelle\_1}}}{W_{\text{max,Quelle\_2}}}\right)$$
$$W_{\text{max,Erg}} = \max\left(\frac{W_{\text{Quelle\_1}}}{W_{\text{min,Quelle\_2}}}, \frac{W_{\text{Quelle\_1}}}{W_{\text{max,Quelle\_2}}}\right)$$

Wie auch bei der Multiplikation von Messwertdatentypen werden bei der Division von zwei Messwertdatentypen besitzenden Operanden die Permutationen der Intervallgrenzen der Quelloperanden gebildet und das Minimum und das Maximum ihrer Divisionen als Intervallgrenzen des Divisionsergebnisses genutzt.

$$W_{\text{min,Erg}} = \min\left(\frac{W_{\text{min,Quelle\_1}}}{W_{\text{min,Quelle\_2}}}, \frac{W_{\text{min,Quelle\_1}}}{W_{\text{max,Quelle\_2}}}, \frac{W_{\text{max,Quelle\_1}}}{W_{\text{min,Quelle\_2}}}, \frac{W_{\text{max,Quelle\_1}}}{W_{\text{max,Quelle\_2}}}\right)$$
$$W_{\text{max,Erg}} = \max\left(\frac{W_{\text{min,Quelle\_1}}}{W_{\text{min,Quelle\_2}}}, \frac{W_{\text{min,Quelle\_1}}}{W_{\text{max,Quelle\_2}}}, \frac{W_{\text{max,Quelle\_1}}}{W_{\text{min,Quelle\_2}}}, \frac{W_{\text{max,Quelle\_1}}}{W_{\text{max,Quelle\_2}}}\right)$$

Ein grundsätzliches Problem stellt der Fall dar, dass der Divisor einen Messwertdatentyp aufweist und im spezifizierten Werteintervall die Null enthält. Auch Vergleichsoperationen erweisen sich als kompliziert. Vorschläge zur Behandlung dieser Fälle würden den Rahmen dieser Arbeit sprengen, weshalb hier nur auf [65] und [57] verwiesen wird.

4.3.2.3 Befehle zur Verwaltung des Datenwerts W

Neben dem Setzen des Datenwerts durch Zuweisungen ermöglicht die Angabe des durch die Genauigkeit des Datenwerts definierten Intervalls des Datenwerts die Prüfung, ob der Datenwert bestimmten Anforderungen an seine Genauigkeit gerecht wird.

Zur Prüfung, ob ein Datenwert vom Datentyp Messwert einen maximalen vorgegebenen relativen Grenzfehler aufweist, kann der Befehl Prüfe Relative Genauigkeit PRG genutzt werden. Zunächst wird geprüft, ob das durch B indizierte zu prüfende Datenspeicherelement einen Messwertdatentyp besitzt. Ist dies nicht der Fall, so gilt die zu prüfende Bedingung automatisch als erfüllt. Handelt es sich um einen Messwert in Intervalldarstellung, so wird unterschieden, ob die Intervallgrenzen als Gleitkomma- oder Ganzzahl formuliert sind. Bei einem gleitkommabasierten Messwertdatentyp wird die obere Intervallgrenze mit dem durch A indizierten Gleitkommafaktor multipliziert. Bei ganzzahlbasierten Messwertdatentypen wird die obere Intervallgrenze

mit dem durch B indizierten Faktor in Festkommadarstellung multipliziert, wobei dieser das Format Q0.(Bitbreite des Faktordatentyps) aufweist, also 0 Bit für den Vorkommaanteil und sämtliche Bits des Festkommawerts für den Nachkommateil genutzt werden. Das Ergebnis der jeweiligen Multiplikation definiert die kleinste zulässige untere Intervallgrenze und wird mit der unteren Intervallgrenze verglichen. Liegt diese unterhalb des Ergebnisses, wird ein Ausnahmefehler ausgelöst. Der Befehl führt den Faktor also in zwei Operanden in zwei verschiedenen Darstellungen mit sich, um die Prüfung von gleitkomma- und ganzzahlbasierten Messwertdatentypen ohne Datentypumwandlungen zu gestatten.

In den folgenden Pseudocodeauflistungen wird durch `DT.DT()` der Datentyp eines Operanden zurückgeliefert. Die Funktion $\omega()$ liefert die Bitbreite eines Datentyps zurück, was für die Korrektur des Ergebnisses bei der Festkommaarithmetik notwendig ist.

```
PRG A, B, C :=

WENN DT.DT([C]) ∈ Messwertdatentypen DANN
  WENN DT.DT([A]) ∉ Gleitkommadatentypen DANN
    Generierung_Ausnahmefehler;
  ENDEWENN

  WENN DT.DT([B]) ∉ Ganzzahldatentypen DANN
    Generierung_Ausnahmefehler;
  ENDEWENN

  WENN DT.DT([C]) ∈ gleitkommabasierte_Messwertdatentypen DANN
    WENN W_max([C]) · W([A]) > W_min([C]) DANN
      Generierung_Ausnahmefehler;
    ENDEWENN
  SONST
    WENN W_max([C]) · W([B]) SHR ω(DT.DT([B])) > W_min([C]) DANN
      Generierung_Ausnahmefehler;
    ENDEWENN
  ENDEWENN
ENDEWENN
```

Während der Befehl Prüfe Relative Genauigkeit PRG in den meisten Anwendungsfällen bevorzugt werden wird, soll ein weiterer Befehl die Verifikation eines

maximalen absoluten Grenzfehlers erlauben. Dieser Befehl wird Prüfe Absolute Genauigkeit PAG genannt und prüft – analog zu PRG – zunächst, ob es sich beim durch B indizierten Zieldatenspeicherelement um einen gleitkommabasierten Messwertdatentyp handelt. Entsprechend findet ein Vergleich mit dem durch A indizierten Gleitkomma- bzw. dem durch B indizierten Ganzzahlvergleichswert statt.

```
PAG A, B, C :=

WENN DT.DT([C]) ∈ Messwertdatentypen DANN
  WENN DT.DT([A]) ∉ Gleitkommadatentypen DANN
    Generierung_Ausnahmefehler;
  ENDEWENN

  WENN DT.DT([B]) ∉ Ganzzahldatentypen DANN
    Generierung_Ausnahmefehler;
  ENDEWENN

  WENN DT.DT([C]) ∈ gleitkommabasierte_Messwertdatentypen DANN
    WENN W_max([C]) - W_min([C]) > 2 · W([A]) DANN
      Generierung_Ausnahmefehler;
    ENDEWENN
  SONST
    WENN W_max([C]) - W_min([C]) > 2 · W([B]) DANN
      Generierung_Ausnahmefehler;
    ENDEWENN
  ENDEWENN
ENDEWENN
```

4.3.2.4 Prüfung der Wertegenauigkeit in Hochsprachen

Zur Nutzung der vorgestellten Befehle zur Prüfung der Genauigkeit wird für die Hochsprache C die Einführung zweier intrinsischer Funktionen vorgeschlagen. Die erste intrinsische Funktion,

`__assert_absolute_accuracy(<Variablenname>,<absoluter Fehler>)`,

dient der Prüfung der Genauigkeit der angegebenen Variable anhand des spezifizierten absoluten Fehlers, um den der ideale Messwert nach oben oder unten abweichen

darf. Der Übersetzer wandelt den Funktionsaufruf in die Nutzung des Befehls Prüfe Absolute Genauigkeit PAG um.

Zur Prüfung der Genauigkeit einer Variable unter Nutzung der Angabe eines zulässigen relativen Fehlers wird die intrinsische Funktion

```
__assert_relative_accuracy(<Variablenname>,<relativer Fehler>)
```

vorgeschlagen, die vom Übersetzer in den Befehl Prüfe Relative Genauigkeit PRG überführt wird. Dabei rechnet dieser den angegebenen relativen Fehler in einen Faktor – abhängig vom Datentyp der Variable – in Festkomma- bzw. Gleitkommadarstellung um, der von der Hardware zur Multiplikation mit der oberen Wertgrenze genutzt werden kann.

Ein Einsatzbeispiel beider Funktionen wird in der folgenden Auflistung gezeigt. Im Beispiel darf `var_a` einen maximalen absoluten Grenzfehler von $\pm$ 7 aufweisen, während `var_b` einen maximalen relativen Grenzfehler von $\pm$ 10 % aufweisen darf.

```
__assert_absolute_accuracy(var_1, 7);
__assert_relative_accuracy(var_2, 10);
```

4.3.2.5 Spezifikation von fehlerbehafteten Werten in Hochsprachen

Die Spezifikation fehlerbehafteter Werte wird zunächst am Beispiel der Hochsprache NewSpeak [25] gezeigt. Anschließend wird ein Vorschlag für die Hochsprache C vorgestellt.

4.3.2.5.1 Fehlerbehaftete Werte in NewSpeak

Die Hochsprache NewSpeak [25] erlaubt die Deklaration von auf Gleitkommadatentypen basierenden fehlerbehafteten Variablen. Dabei kann sowohl ein absoluter, als auch ein relativer Fehler spezifiziert werden. Das gezeigte Beispiel wurde der genannten Quelle entnommen und zeigt die Deklaration der Variable x, die einen relativen Fehler von $\pm$ 10^{-8} aufweist.

```
x: Float {0..100}±{Rel 10^-8}
```

4.3.2.5.2 Vorschlag zur Spezifikation fehlerbehafteter Werte in C

Für die Hochsprache C wird eine Erweiterung der Variablendeklaration um das Schlüsselwort `measurement` vorgeschlagen, um dem Übersetzter anzuzeigen, dass eine Variable einen Messwertdatentyp besitzen soll. Die Wertzuweisung erfolgt durch die neuen intrinsischen Funktionen

- `__value_intervall(<untere Grenze` x_{min}`>,<obere Grenze` x_{max}`>)`, durch die ein Werteintervall direkt spezifiziert werden kann,
- `__value_absolute_error(<`x_{Best}`>,<` δ`x>)`, welche das zu setzende Werteintervall indirekt auf Basis des besten Schätzwertes x_{Best} und des absoluten Fehlers δx beschreibt, sowie
- `__value_relative_error(<`x_{Best}`>,<` $\frac{\delta x}{|x_{Best}|}$ `>)`, mittels derer das Werteintervall indirekt durch den besten Schätzwert x_{Best} und die Angabe des relativen Fehlers $\frac{\delta x}{|x_{Best}|}$ festgelegt wird.

Während der Übersetzer die durch `__value_intervall()` spezifizierten Intervallgrenzen direkt in eine Variable des entsprechenden Datentyps eintragen kann, muss er bei den beiden indirekten Festlegungen die Intervallgrenzen zunächst berechnen. Das folgende Beispiel verdeutlicht die Nutzung der intrinsischen Funktionen.

```
measurement unsigned int x = __value_intervall(0,2);
measurement unsigned int y = __value_absolute_error(200,7);
measurement float z = __value_relative_error(300.0,3.2);
```

In der ersten Zeile des Beispiels wird eine Variable x als vorzeichenloser ganzzahlbasierter Messwertdatentyp deklariert und ihre Intervallgrenzen werden auf $W_{min} = 0$ und $W_{max} = 2$ gesetzt. Die Variable y soll den gleichen Datentyp haben und der Übersetzer berechnet aus den beiden Angaben die Intervallgrenzen $W_{min} = 193$ und $W_{max} = 207$. Die Variable z soll einen gleitkommabasierten Messwertdatentyp besitzen und der Übersetzer weist ihr die Intervallgrenzen $W_{min} = 290.4$ und $W_{max} = 309.6$ zu.

Die vorgestellten intrinsischen Funktionen können auch im Rahmen von Termen genutzt werden, wie das folgende Beispiel zeigt.

```
x = y + (measurement unsigned int) __value_absolute_error(30,2);
```

Dabei wird x die Summe von y und dem Wert 30±2 zugewiesen. Damit ergeben sich für x nach der Zuweisung des Ergebnisses unter Nutzung des im obigen Beispiel für y festgelegten Werteintervalls der fehlerbehaftete Wert von 230±9 mit den Intervallgrenzen $W_{min} = 221$ und $W_{max} = 239$.

4.3.2.6 Evaluation des Datenwerts W

Der Einsatz der vorgestellten Intervallarithmetik erlaubt zusammen mit den vorgeschlagenen Befehlen zur Prüfung erwarteter Genauigkeiten die Erkennung von Genauigkeitsproblemen, wie es in Tabelle 4.3 gezeigt wird.

Tabelle 4.3: Fehlererkennung durch Intervallarithmetik

Fehlerart	**Erkennbarkeit**
Inkompatible Datentypen	nein
Inkompatible Einheiten	nein
Wertebereichsunter- bzw. -überschreitung	nein
Genauigkeitsproblem	ja
Falsche Operandenauswahl	nein
Falsche Operatorauswahl	nein
Fehlerhaftes Operationsergebnis	nein
Fristüberschreitung	nein
Zyklusunterschreitung	nein
Zyklusüberschreitung	nein
Verlorengegangene Datenaktualisierung	nein
Synchronisationsfehler oder unvollständige Datenübertragung	nein
Pufferunter- oder -überläufe	nein
Fehlerhafter Datenfluss (falsche Adressaten, ...)	nein
Duplizierte Daten	nein
Durch Fehler oder Störungen verfälschte Daten	nein
Fehlerhafter Datenzugriff (fehlende Zugriffsrechte)	nein
Nutzung nicht initialisierter Daten	nein
Angriffsart	
Gezielt verfälschte Daten	nein
Wiedereinspielungsattacke	nein

4.3.3 Wertebereich

Nicht immer darf ein Datenwert alle Werte annehmen, die sich im Wertebereich des zugrundeliegenden Datentyps befinden, was im Besonderen an Schnittstellen zwischen Systemkomponenten und bei Funktionsparametern wichtig ist. Die Selbstzerstörung der Ariane 5 (siehe Kapitel 1.1.1) nahm ihren Anfang, weil es zu einem Überlauf bei einer Berechnung aufgrund zu großer Werte kam. Deshalb überwacht z. B. die Hochsprache NewSpeak [25] zur Übersetzungszeit die Wertebereiche von Variablen, um Wertebereichsverletzungen aufdecken zu können. Falls derartige Prüfungen auch zur Laufzeit vorgesehen werden, müssen diese durch die Software vorgenommen werden.

4.3.3.1 Realisierung der Wertebereichskennung WB

Die Einführung der Wertebereichskennung WB gestattet es verarbeitenden Instanzen, zu erkennen, ob der Datenwert eines Datenspeicherelements innerhalb spezifizierter Grenzen liegt. Die Wertebereichskennung WB wird dazu einem Datenspeicherelement hinzugefügt, wie in Abbildung 4.12 dargestellt.

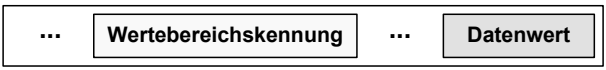


Abbildung 4.12: Datenspeicherelement mit Wertebereichskennung WB und Datenwert W

Dabei wird das zulässige Wertebereichsintervall durch das Tupel

$$WB := (WB_{unten}, WB_{oben})$$

beschrieben, wobei WB_{unten} die zulässige Untergrenze und WB_{oben} die Obergrenze des Datenwerts W des jeweiligen Datenspeicherelements angibt, wodurch sich der in Abbildung 4.13 gezeigte Aufbau der Wertebereichskennung WB ergibt.

Abbildung 4.13: Aufbau der Wertebereichskennung WB

Der Datentyp beider Grenzwerte entspricht dem des Datenwerts W, der in der Datentypkennung DT definiert wird. Dadurch wird klar, dass für beide Grenzwerte jeweils die Bitbreite des Datentyps mit der größten Bitbreite vorzusehen ist, bei 64 Bit breiten Ganzzahlen wäre für die Wertebereichskennung eine Bitbreite von $2 \cdot 64$ Bit, also 128 Bit zu reservieren.

Der zulässige Wertebereich kann durch den Übersetzer teilweise automatisch gesetzt werden, z. B. für Enumerationen, die nur einen festgelegten Wertebereich besitzen.

4.3.3.2 Prüfung auf Wertebereichsverletzungen anhand der Wertebereichskennung WB

Anhand der Wertebereichskennung kann die Hardware einer Datenspezifikationsarchitektur DSA die Gültigkeit eines Datenwerts überprüfen. Während beim lesenden Zugriff lediglich der im Datenspeicherelement enthaltene Datenwert W auf Plausibilität geprüft werden kann, kann die Hardware beim schreibenden Zugriff anhand des im Zieldatenspeicherelement als zulässig definierten Wertebereichs prüfen, ob der zu schreibende Datenwert gültig ist, also innerhalb des Wertebereichs liegt. Dies kann z. B. für die Prüfung von Funktionsparametern, Rückgabewerten und Stellgrößen genutzt werden. Es muss daher die Bedingung

$$W \in [[WB_{unten}];[WB_{oben}]]$$

erfüllt sein. Wichtig ist weiterhin, dass der in der Wertebereichskennung WB spezifizierte Wertebereich des Datenwerts eine Teilmenge des Wertebereichs des zugrundeliegenden Datentyps sein muss, also dass

$$[WB_{unten};WB_{oben}] \subseteq WB(Datentyp)$$

erfüllt ist. Trifft mindestens eine der beiden Bedingungen nicht zu, so generiert die Hardware einen Ausnahmefehler.

```
Prüfung_WB_Lesezugriff :=

WENN DT.DT([Quelle]) ∈ Messwertdatentypen DANN
  WENN [Wmin([Quelle]);Wmax([Quelle])] ⊈ WB([Quelle]) DANN
    Generierung_Ausnahmefehler;
  ENDEWENN
SONST
  WENN W([Quelle]) ∉ WB([Quelle]) DANN
    Generierung_Ausnahmefehler;
```

```
  ENDEWENN
ENDEWENN

WENN WB([Quelle]) ⊈ WB(DT.DT([Quelle])) DANN
  Generierung_Ausnahmefehler;
ENDEWENN
```

```
Prüfung_WB_Schreibzugriff :=

WENN DT.DT([Ergebnis]) ∈ Messwertdatentypen DANN
  WENN [W_min([Ergebnis]);W_max([Ergebnis])] ⊈ WB([Ziel]) DANN
    Generierung_Ausnahmefehler;
  ENDEWENN
SONST
  WENN W([Ergebnis]) ∉ WB([Ziel]) DANN
    Generierung_Ausnahmefehler;
  ENDEWENN
ENDEWENN

WENN WB([Ziel]) ⊈ WB(DT.DT([Ziel])) DANN
  Generierung_Ausnahmefehler;
ENDEWENN
```

4.3.3.3 Befehle zur Verwaltung der Wertebereichskennung WB

Zum Setzen des Wertebereichs eines Datenspeicherelements wird der Befehl Setze Wertebereich SWB definiert. Dabei stellt die Hardware zunächst sicher, dass die durch A und B indizierten Werte ein gültiges Intervall spezifizieren. Eine weitere Prüfung stellt fest, ob der zu setzende Wertebereich des Datenspeicherelements eine Teilmenge des Wertebereichs des zugrundeliegenden Datentyps des durch C indizierten Datenspeicherelements darstellt. Schlägt eine dieser Prüfungen fehl, wird ein Ausnahmefehler generiert. Andernfalls wird der Wertebereich in die Felder WB_{unten} und WB_{oben} der Wertebereichskennung WB des Zieldatenspeicherelements eingetragen. An dieser Stelle wird im Pseudocode auch kurz vorgegriffen: DT.DT([Operand]) liefert den Datentyp des Operanden zurück.

```
SWB A, B, C :=

WENN W([A]) ≤ W([B]) DANN
  WENN [W([A]);W([B])] ⊆ WB(DT.DT([C])) DANN
    WB.WB_unten([C]) := W([A]);
    WB.WB_oben ([C]) := W([B]);
  SONST
    Generierung_Ausnahmefehler;
  ENDEWENN
SONST
  Generierung_Ausnahmefehler;
ENDEWENN
```

Weiterhin wird der Befehl Prüfe Einen Operanden PEO eingeführt, der keine Änderung am Operanden vornimmt, sondern nur lesend auf diesen zugreift und dabei verschiedene mit dem Lesen des Operanden verknüpfte Prüfungen seiner Kennungsinhalte vornimmt. Damit hat der Befehl die Funktion einer Assertion, also eines reinen Prüfbefehls. Er wird bei verschiedenen Kennungen erwähnt und die durchgeführten Prüfungen – jeweils mit dem Fokus auf die aktuell vorgestellte Kennung – erläutert. Die Durchführung weiterer Prüfungen wird durch „...“ angedeutet. Für die Wertebereichskennung WB wird für den durch A indizierten Operanden sichergestellt, dass der Datenwert innerhalb des durch die Kennung spezifizierten Wertebereichs liegt und dieser Wertebereich eine Teilmenge des Basisdatentyps darstellt.

```
PEO A :=

...
WENN DT.DT([A]) ∈ Messwertdatentypen DANN
  WENN [W_min([A]);W_max([A])] ⊈ WB([A]) DANN
    Generierung_Ausnahmefehler;
  ENDEWENN
SONST
  WENN W([A]) ∉ WB([A]) DANN
    Generierung_Ausnahmefehler;
  ENDEWENN
ENDEWENN
```

```
WENN WB([A]) ⊈ WB(DT.DT([A])) DANN
  Generierung_Ausnahmefehler;
ENDEWENN
...
```

4.3.3.4 Spezifikation von Wertebereichen in Hochsprachen

Die Spezifikation zulässiger Wertebereiche für Variablen, Funktionsparameter und -rückgabewerte oder Datentypen wird zunächst anhand der Hochsprachenbeispiele Ada, Pascal und NewSpeak gezeigt. Im Anschluss werden entsprechende Vorschläge für die Hochsprache C unterbreitet. Neben der Überführung der zulässigen Wertebereiche in die Wertebereichskennungen WB der betroffenen Variablen sollen die Übersetzer selbstverständlich auch bereits zur Übersetzungszeit auf Basis der Wertebereichsspezifikationen mögliche Wertebereichsverletzungen erkennen und die Erstellung eines entsprechend fehlerhaften Programms abbrechen.

4.3.3.4.1 Wertebereichsdefinition in Ada

Die Hochsprache Ada erlaubt die Definition von Datentypen mit einem eingeschränkten Wertebereich [121].

```
type column is range 1 .. 72;
```

Eine solche Begrenzung des zulässigen Wertebereichs kann der Übersetzer mittels des Befehls Setze Wertebereich SWB in die Wertebereichkennungen WB der Variablen, die auf dem jeweiligen Datentyp basieren, übertragen.

4.3.3.4.2 Wertebereichsdefinition in Pascal

In Pascal können Wertebereichseinschränkungen durch sogenannte „Subrange“-Datentypen vorgenommen werden. Ein entsprechendes Beispiel wird in [40] gezeigt.

```
TYPE Month = 1 .. 12;
```

In [80] werden neben der Vorstellung von Sprachmitteln zur Spezifikation von Einheiten von Variablen auch Vorschläge bzgl. der Begrenzung des gültigen Wertebereichs von einheitenbehafteten Datentypen unterbreitet. Der genannten Quelle entstammt auch das unten gezeigte Beispiel.

```
TYPE Energy       = 0 MeV .. 1e3 MeV;
```

Ein für eine Datenspezifikationsarchitektur DSA angepasster Übersetzer kann diese Wertebereichsdefinitionen in die Wertebereichskennung WB unter Nutzung des Befehls Setze Wertebereich SWB eintragen.

4.3.3.4.3 Wertebereichsdefinition in NewSpeak

In der Sprache NewSpeak [25] kann bei der Definition von Typen und Variablen ein Wertebereich festgelegt werden. Dies geschieht durch Angabe des zugrundeliegenden Datentyps gefolgt von der Spezifikation des zulässigen Wertebereichs in geschweiften Klammern, wie in der folgenden Abbildung gezeigt.

```
Type BOOL = Bit{0..1}
Type CHAR = Byte{0..255}
```

Auf Basis dieser Definition können die Variablen durch die Anwendung des Befehls Setze Wertebereich SWB mit einer entsprechenden Wertebereichskennung WB versehen werden.

4.3.3.4.4 Vorschlag zur Wertebereichsdefinition in C

Zur Definition zulässiger Wertebereiche wird für die Hochsprache C ein Attribut definiert, welches

`__range_of_values(<unterer_Wert>,<oberer_Wert>)`

genannt wird. Die Festlegung gültiger Wertebereiche könnte bei der Definition von

- Variablen,
- Funktionsparametern und
- Funktionsrückgabewerten

angewendet werden. Die zwei folgenden Auflistungen in C zeigen exemplarisch die Anwendung der neuen Funktion. Im ersten Beispiel wird eine vorzeichenbehaftete Variable definiert, deren Wertebereich auf das Intervall [5;231] begrenzt werden soll.

```
void funktion_y(void)
{
  int __range_of_values(5,231) variable_x;
}
```

In der zweiten Auflistung wird eine Funktion definiert, deren vorzeichenbehafteter Parameter im Intervall [9;16] liegen muss und einen vorzeichenbehafteten Rückgabewert im Intervall [-3;4] zurückgeben soll.

```
int __range_of_values(-3,4) f_x(int __range_of_values(9,16) par_x)
{
  ...
}
```

Weiterhin kann ein Übersetzer bei Enumerationen – bei C über das Schlüsselwort `enum` – automatisch den Wertebereich der Variablen des Enumerationstyps begrenzen.

Der Übersetzer kann eine Variable mit eingeschränktem Wertebereich dadurch realisieren, dass er die Wertebereichskennung WB des mit der Variablen verknüpften Datenspeicherelements mit Hilfe des Befehls Setze Wertebereich SWB vor deren Nutzung setzt. Eine automatische Prüfung des Wertebereichs von Übergabeparametern kann dadurch realisiert werden, dass diese direkt in formatierte Datenspeicherelemente mit gesetzter Wertebereichskennung WB gespeichert oder in solche übertragen werden. Analog dazu können die Rückgabewerte in formatierte Datenspeicherelemente mit entsprechender Wertebereichseinschränkung gespeichert werden. Bei den entsprechenden Zuweisungen kann die Hardware automatisch sicherstellen, dass der zu speichernde Wert innerhalb des spezifizierten Wertebereichs liegt.

4.3.3.5 Evaluation der Wertebereichskennung WB

Die durch die Nutzung einer Wertebereichskennung aufdeckbaren Fehlermechanismen der 20 in Kapitel 2.4 vorgestellten Fehler- und Angriffsarten werden in Tabelle 4.4 gezeigt.

Tabelle 4.4: Fehlererkennung durch die WB-Kennung

Fehlerart	**Erkennbarkeit**
Inkompatible Datentypen	nein
Inkompatible Einheiten	nein
Wertebereichsunter- bzw. -überschreitung	ja
Genauigkeitsproblem	begrenzt
Falsche Operandenauswahl	begrenzt
Falsche Operatorauswahl	begrenzt
Fehlerhaftes Operationsergebnis	begrenzt
Fristüberschreitung	nein
Zyklusunterschreitung	nein
Zyklusüberschreitung	nein
Verlorengegangene Datenaktualisierung	nein
Synchronisationsfehler oder unvollständige Datenübertragung	nein
Pufferunter- oder -überläufe	begrenzt
Fehlerhafter Datenfluss (falsche Adressaten, ...)	nein
Duplizierte Daten	nein
Durch Fehler oder Störungen verfälschte Daten	begrenzt
Fehlerhafter Datenzugriff (fehlende Zugriffsrechte)	nein
Nutzung nicht initialisierter Daten	(ja)
Angriffsart	
Gezielt verfälschte Daten	nein
Wiedereinspielungsattacke	nein

Die Wertebereichskennung erlaubt bei lesendem und schreibendem Zugriff die Prüfung, ob der im Datenspeicherelement enthaltene Datenwert W bzw. der in das Datenspeicherelement zu schreibende Datenwert innerhalb des durch die Wertebereichskennung definierten gültigen Intervalls liegt. Ist dies nicht der Fall, wird ein Ausnahmefehler generiert. Ungenaue Datenwerte können als Fehler erkannt werden, wenn mindestens eine der Intervallgrenzen des Datenwerts außerhalb des spezifi-

zierten zulässigen Wertebereichs liegt. In begrenztem Umfang können mittels der Wertebereichskennung Fehler aufgedeckt werden, die zur Nutzung falscher Operanden, falscher Operatoren oder zu gänzlich falschen Operationsergebnissen führen, allerdings nur dann, wenn die Ergebnisse dabei die Wertebereichsgrenzen der jeweiligen Zieldatenspeicherelemente verletzen. Pufferunter- oder -überläufe können auf die gleiche Weise erkannt werden, falls Wertebereichsverletzungen auftreten. Wird der Datenwert W innerhalb eines Datenspeicherelements durch eine Störung derart verändert, dass er außerhalb des gültigen Wertebereichs liegt, kann dieser Fehler bei lesendem Zugriff aufgedeckt werden. In gewissen Grenzen kann auch der lesende Zugriff auf Datenspeicherelemente, die keine gültigen Werte enthalten, also nicht initialisiert sind, dadurch erkannt werden, dass hier explizit ein Datenwert außerhalb des Wertebereichs vorbelegt wird. Ein Lesezugriff würde dann zur Generierung eines Ausnahmefehlers führen.

4.3.4 Datentyp

Die explizite Angabe des Datentyps des in einem Datenspeicherelement enthaltenen Datenwerts erlaubt es, Inkompatibilitäten bei der Verwendung von Daten zu erkennen, die durch verschiedenste Fehlerursachen hervorgerufen werden können. Diese Ursachen reichen von Spezifizierungs- bis hin zu Hardwarefehlern. Die explizite und hardwareverständliche Angabe des Datentyps DT des Datenwerts W ist das grundlegende, namensgebende Merkmal der in Kapitel 3.3 vorgestellten Datentyparchitekturen.

4.3.4.1 Typische Datentypen in verschiedenen Architekturen

In der Vergangenheit wurden durch Datentyp-, Datenstruktur- und Befähigungsarchitekturen verschiedene Datentypen nativ unterstützt und in den jeweiligen Datentypkennungen unterschieden. Dazu wird den Datenwerten in den Datenspeicherelementen eine entsprechende Datentypkennung DT hinzugefügt, wie in Abbildung 4.14 dargestellt.

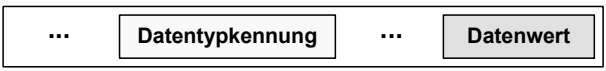


Abbildung 4.14: Datenspeicherelement mit Datentypkennung DT und Datenwert W

Die inhärent sichere Mikroprozessorarchitektur ISMA [125] gestattet neben der generellen Unterscheidung von Code und Daten die Spezifikation von

- Bitfeldern,
- vorzeichenlosen und -behafteten Ganzzahlen,
- Gleitkommazahlen,
- Datenfelddeskriptoren, Feldelementen und Feldindizes,
- Zeichen und
- relativen bzw. absoluten Zeitangaben,

je nach Datentyp in verschiedenen Bitbreiten zwischen 1 und 64 Bit. Weitere native Datentypen diverser Datentyparchitekturen waren nach [36] z. B.

- komplexe Zahlen,
- Vektoren,
- einfach und doppelt verkettete Listen,
- Stapel,
- Ereignisse und
- Semaphoren.

Einige Befähigungsarchitekturen verwendeten einen dedizierten Datentyp zur Definition von Befähigungsobjekten im Speicher, um diese vor unberechtigtem Zugriff zu schützen und nur privilegierten Programminstanzen wie z. B. einem Betriebssystem zugänglich zu machen.

In Kapitel 4.3.2 wurden die Messwertdatentypen zur Spezifikation fehlerbehafteter Messwerte vorgestellt, welche die hier beschriebenen Datentyparten ergänzen.

4.3.4.2 Hardwarebasierte Unterstützung abgeleiteter Datentypen

Als Erweiterung gegenüber bestehenden Datentyparchitekturen soll es in einer Datenspezifikationsarchitektur ermöglicht werden, durch die Hardware überwachte abgeleitete Datentypen auf Grundlage der bekannten Basisdatentypen zu definieren. Dies erlaubt eine erweiterte Isolation ausgewählter Daten und eine noch weiter verbesserte Fehlererkennung durch die Hardware der Datenspezifikationsarchitektur.

Die Ableitung neuer Datentypen auf Basis bestehender Datentypen wird auf Programmiersprachenebene z. B. in [118] beschrieben, inklusive der Möglichkeiten der Einschränkung des Wertebereichs. In einer Veröffentlichung bzgl. PUMP als Teil des SAFE-Projekts [32] wird vorgeschlagen, abgeleitete Datentypen in Kennungen zu beschreiben, aber die Realisiserung wird nicht detailliert beschrieben. Als Anwendungsbeispiel wird dabei in [32] die Isolation der zwei ganzzahlbasierten abgeleiteten Datentypen „Kontonummer“ und „Datum“ genannt.

In dieser Arbeit wird für eine Datenspezifikationsarchitektur DSA jedoch eine konkrete Realisierung und zeitgleich eine Erweiterung dieser Ansätze vorgeschlagen, indem für die abgeleiteten Datentypen Einschränkungen bzgl.

- Datentypumwandlungen,
- arithmetischen Operationen,
- Schiebebefehlen und
- bitweisen logischen Operationen

definiert werden können, die auf Datenspeicherelementen mit einem entsprechenden abgeleiteten Datentyp ausgeführt werden dürfen. Dabei können die zulässigen Operationen gegenüber dem zugrundeliegenden Basisdatentyp nur eingeschränkt, aber nicht erweitert werden. Es können also keine Rechte zu einem abgeleiteten Datentyp hinzugefügt werden, die der Basisdatentyp nicht gewährt. Wird versucht, gegen diese Regelung bei der Definition eines abgeleiteten Datentyps zu verstoßen, wird dies durch die Hardware als Fehler erkannt und die Programmausführung mit der Generierung eines Ausnahmefehlers abgebrochen. Ein solcher Fehler in der Datentypdefinition läge z. B. bei Anwendung der in ISMA [125] vorgestellten Regeln bzgl. der Handhabung von Datentypen vor, wenn versucht würde, arithmetische Operationen auf Bitfeldern oder logische Operationen auf Ganzzahl- oder Gleitkommadatentypen zu gestatten.

4.3.4.3 Realisierung der hardwarebasierten abgeleiteten Datentypen

Die Realisierung würde – wie in Abbildung 4.15 dargestellt – dadurch erfolgen, dass zusätzlich zur bekannten Datentypkennung DT eine Subdatentypkennung SDT und eine typbezogene Berechtigungskennung TB zu jedem Datenspeicherelement hinzugefügt werden. Wird in der SDT-Kennung eine Null angegeben, wird kein abgeleiteter Datentyp definiert und es gelten die datentypbezogenen Regeln des Basisdatentyps. Wird die SDT-Kennung auf einen Wert ungleich Null gesetzt, so gelten die datentypbezogenen Regeln des Basisdatentyps mit den Einschränkungen, die in der TB-Kennung spezifiziert werden.

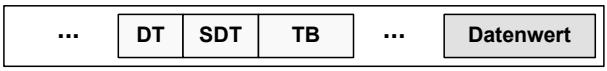


Abbildung 4.15: Aufbau der erweiterten Datentypkennung DT

4.3.4.4 Prüfung der Datentypkompatibilität anhand der Datentypkennung DT

Bei der Durchführung von Operationen stellt die Hardware der Datenspezifikationsarchitektur zunächst die allgemeine Kompatibilität der Datentypen der an der Operation beteiligten Operanden sicher, indem sie die Basisdatentypen DT der Datentypkennung DT auf Gleichheit überprüft. Auch die Subdatentypen SDT der Operanden müssen identisch sein. In den Typberechtigungen TB beider Operanden muss die durchzuführende Operation als zulässig ausgewiesen sein. Aus Platzgründen werden im folgenden Pseudocodebeispiel die Operation als Op, die Operanden als Q_1 und Q_2 und die Bitmasken als Bitm abgekürzt.

```
Prüfung_Operation :=

WENN DT.DT([Q_1]) ≠ DT.DT([Q_2]) DANN
  Generierung_Ausnahmefehler;
ENDEWENN
```

```
WENN DT.SDT([Q_1]) ≠ DT.SDT([Q_2]) DANN
  Generierung_Ausnahmefehler;
ENDEWENN

WENN DT.TB([Q_1]) UND DT.TB([Q_2]) UND Bitm(Op) ≠ Bitm(Op) DANN
  Generierung_Ausnahmefehler;
ENDEWENN
```

4.3.4.5 Befehle zur Verwaltung der Datentypkennung DT

Der Befehl Setze Typ ST zur Festlegung des Datentyps eines Datenspeicherelements ist von der inhärent sicheren Mikroprozessorarchitektur ISMA [125] her bekannt. Da die Datentypkennung gegenüber ISMA zur Definition abgeleiteter Datentypen und der erlaubten Operationen erweitert wurde – ISMA benutzte hier nur die Spezifikation des Datentyps ST –, muss auch der Befehl entsprechend angepasst werden. Zunächst wird der Basisdatentyp, der durch A indiziert wird, auf Gültigkeit geprüft. Ist er gültig, wird er in die Datentypkennung DT des durch D indizierten Zieldatenspeicherelements eingetragen.

Anschließend wird der durch B indizierte Subdatentyp in das Zieldatenspeicherelement eingefügt. Im letzten Schritt stellt die Hardware sicher, dass die durch C indizierten zu setzenden Typberechtigungen eine Teilmenge der für den Basisdatentyp geltenden Rechte sind. Ist dies der Fall, so werden die Typberechtigungen TB in das Zieldatenspeicherelement übernommen. Schlägt eine der Prüfungen fehl, so wird ein Ausnahmefehler generiert.

```
ST A, B, C, D :=

WENN W([A]) ∈ Gültige_Datentypen DANN
  DT.DT([D]) := W([A]);
SONST
  Generierung_Ausnahmefehler;
ENDEWENN

DT.SDT([D]) := W([B]);
```

```
WENN TB(DT.DT([A])) UND W([C]) = W([C]) DANN
  DT.TB([D]) := W([C]);
SONST
  Generierung_Ausnahmefehler;
ENDEWENN
```

Ebenfalls von ISMA bekannt ist der Befehl Ändere Typ ÄT zur expliziten Datentypumwandlung. Dazu wendet ISMA einen umfangreichen Satz von Regeln an, die jedoch nicht im Fokus der Beschreibung dieses Befehls liegen. Hier soll das Augenmerk auf die Prüfung gerichtet werden, ob eine Typumwandlung für den abgeleiteten Datentyp überhaupt zulässig ist. Der Befehl ÄT soll den im durch A indizierten Quelldatenspeicherelement enthaltenen Datenwert mit dem Datentyp DT.DT([A]) in den Datentyp DT.DT([B]) des durch B indizierten Zieldatenspeicherelements umwandeln und in diesem speichern. Zunächst erfolgt eine Prüfung, ob die gewünschte Datentypumwandlung für das Tupel (DT.DT([A]),DT.DT([B])) überhaupt zulässig ist, wobei der bereits erwähnte, umfangreiche Regelsatz zur Anwendung kommt. Anschließend wird sichergestellt, dass die Operation Datentypänderung in den Typberechtigungen des Quelldatenspeicherelements gestattet wurde. Erst dann erfolgt die eigentliche Datentypumwandlung und das Ablegen des umgewandelten Datenwerts im durch B indizierten Zieldatenspeicherelement. Im Falle des Fehlschlagens einer der Prüfungen wird ein Ausnahmefehler generiert.

```
ÄT A, B :=

WENN Typänderung_erlaubt(DT.DT([A]), DT.DT([B])) DANN
  WENN DT.TB([A]) UND Bitmaske(ÄT) = Bitmaske(ÄT) DANN
    [B] := Datentypumwandlung([A], DT.DT([B]));
  SONST
    Generierung_Ausnahmefehler;
  ENDEWENN
SONST
  Generierung_Ausnahmefehler;
ENDEWENN
```

Analog zum Befehl Prüfe Einen Operanden PEO, der in Kapitel 4.3.3.3 vorgestellt wurde, wird ein Befehl zur Prüfung von zwei Operanden mit dem Namen Prüfe Zwei Operanden PZO bereitgestellt, der beide Operanden liest und die typischen beim Lesen von Operanden durchgeführten Prüfungen durchführt. Auch dieser stellt –

ebenso wie der Befehl PEO – eine Assertion dar, also einen reinen Prüfbefehl, der die Operanden nicht verändert und kein Ergebnis außer der Sicherstellung der angegebenen Zusammenhänge liefert. Der Befehl PZO wird bei mehreren Kennungen referenziert und es wird jeweils angegeben, welche – auf die aktuelle Kennung bezogenen – Prüfungen durch ihn durchgeführt werden. Die Durchführung weiterer Prüfungen wird durch „...“ angedeutet. Bezogen auf die Datentypkennung DT wird die Gleichheit der Daten- und Subdatentypen beider Operanden sichergestellt.

```
PZO A, B :=

...

WENN DT.DT([A]) ≠ DT.DT([B]) DANN
  Generierung_Ausnahmefehler;
ENDEWENN

WENN DT.SDT([A]) ≠ DT.SDT([B]) DANN
  Generierung_Ausnahmefehler;
ENDEWENN

...
```

4.3.4.6 Beispiele für die Nutzung hardwarebasierter abgeleiteter Datentypen

Ein gutes Beispiel für den Nutzen abgeleiteter Datentypen in Verbindung mit der WB-Eigenschaft ist die Zahlendarstellung im BCD-Format. Dabei wird jede Stelle einer Dezimalzahl in einem eigenständigen Byte im Zahlenbereich 0 bis 9 abgebildet. Es ist daher naheliegend, bei der Verwendung von BCD-Daten in einem Softwareprojekt einen von 8 Bit breiten Ganzzahlen abgeleiteten Datentyp zu definieren und durch die WB-Kennung dessen zulässigen Wertebereich auf 0 bis 9 festzulegen, wodurch es der Hardware möglich wird, den Datenwert W des Datenspeicherelements gegen diese Wertgrenzen zu prüfen und so Fehler sofort aufzudecken.

Ein Beispiel für den sinnvollen Einsatz der Einschränkung zulässiger Operationen auf einem abgeleiteten Datentyp wäre der Datentyp „Fehlercode“. Dieser würde auf einem vorzeichenlosen Ganzzahldatentyp mit einer passenden Bitbreite

basieren und die Ausführung arithmetischer Operationen würde untersagt werden – so wäre z. B. die Bedeutung des Ergebnisses der Addition der Fehlercodes FEHLER_PAPIERSTAU und FEHLER_TEMPERATUR_ZU_HOCH völlig unklar.

4.3.4.7 Spezifikation abgeleiteter Datentypen in Hochsprachen

Die meisten Hochsprachenübersetzer bieten Sprachmittel zur Definition abgeleiteter Datentypen. Für die Hochsprache C kann ein Übersetzer bei entsprechenden Typdeklarationen automatisch Subdatentypen definieren, bei denen die Hardware der Datenspezifikationsarchitektur die Einhaltung der festgelegten Regeln überwachen soll.

- Enumerationen (Aufzählungstypen) mit `enum` erlauben einerseits die Definition eines neuen abgeleiteten Datentyps und zeitgleich die Einschränkung des zulässigen Wertebereichs.
- Typdefinitionen mit `typedef` erlauben die Definition abgeleiteter Datentypen.

```
enum name {wert_1, ..., wert_n};
```

```
typedef <Basisdatentyp> <neuer_Typname>;
```

Nach dem Entwicklungsprinzip der minimalen Rechte – engl. „Principle of Least Privilege“ – sollen bei fehlender Angabe von zulässigen Operationen auf dem abgeleiteten Datentyp sämtliche Operationen untersagt werden, dem Typberechtigungsfeld TB der Datentypkennung DT daher der Wert Null zugewiesen werden. Berechtigungen können dann explizit über ein Attribut zugeteilt werden. Dafür wird

`__type_rights(<Liste zugelassener Funktionen>)`

vorgeschlagen. Als mögliche zuzulassende Operationsgruppen werden

- Datentypumwandlungen – `typeconversion` –,
- arithmetische Operationen – `arithmetic` –,
- Schiebe- und Rotationsbefehle – `shiftrotate` – und
- bitweise logische Operationen – `logical` –

vorgesehen. Der Übersetzer sorgt dann beim Anlegen von Variablen des abgeleiteten Datentyps dafür, dass durch Nutzung des Befehls Setze Typ ST neben dem Basisdatentyp und dem Identifikator des Subdatentyps auch die angegebenen Berechtigungen im Typberechtigungsfeld TB der Datentypkennung DT gesetzt werden.

Dies ergibt die in der folgenden Auflistung beispielhaft gezeigte Typdeklaration, bei der ein von den vorzeichenlosen Ganzzahlen abgeleiteter neuer Datentyp definiert wird, auf dem nur arithmetische Operationen durchgeführt werden dürfen, jedoch keine Datentypumwandlungen, Schiebe- oder Rotationsbefehle und auch keine bitweisen logischen Operationen.

```
typedef unsigned int __type_rights(arithmetic) neuer_Datentyp;
```

4.3.4.8 Eine alternative Realisierung der Definition abgeleiteter Datentypen

Eine weitere Realisierungsmöglichkeit der hardwareverständlichen Definition abgeleiteter Datentypen mit Einschränkung der zulässigen Operationen wäre deren Bekanntgabe durch die Software bei Programmstart. Dabei werden die Informationen

- Identifikator des neuen abgeleiteten Datentyps $\mathrm{DTID_{neu}}$,
- Identifikator des zugrundeliegenden Basisdatentyps $\mathrm{DTID_{Basis}}$,
- die für den neuen abgeleiteten Datentyp zu setzenden Typberechtigungen $\mathrm{TB_{neu}}$, also die Definition der erlaubten Operationen, und
- der zulässige Wertebereich des abgeleiteten Datentyps $\mathrm{WB_{neu}}$

zum Informationstupel

$$NDT := (DTID_{neu},\ DTID_{Basis},\ TB_{neu},\ WB_{neu})$$

zusammengestellt und beim Programm- oder Systemstart an die Hardware übermittelt. Bevor die Hardware die Definition des neuen abgeleiteten Datentyps in eine Datentypdefinitionstabelle DDT aufnimmt, in der auch alle Basisdatentypen spezifiziert werden, führt sie die folgenden Prüfungen durch:

- $\mathrm{DTID_{neu}} \notin$ DDT, d. h. der Identifikator des neuen Datentyps darf noch nicht benutzt worden sein, also weder für die Definition eines Basisdatentyps, noch für abgeleitete Datentypen,

- $\text{DTID}_{\text{Basis}} \in \text{DDT}$, d. h. der Identifikator des zugrundeliegenden Basisdatentyps muss existent sein,
- $\text{TB}_{\text{neu}} \subseteq \text{TB}_{\text{Basis}}$, d. h. die Typberechtigungen dürfen die zulässigen Operationen gegenüber den für den Basisdatentyp zugelassenen Operationen nur einschränken, nicht jedoch erweitern,
- $\text{WB}_{\text{neu}} \subseteq \text{WB}_{\text{Basis}}$, d. h. der Wertebereich des abgeleiteten Datentyps muss eine Teilmenge des Wertebereichs des Basisdatentyps sein.

Diese alternative Lösung zur Definition abgeleiteter Datentypen hat die folgenden Vorteile gegenüber der ersten Realisierungsmöglichkeit:

- die Datentypkennung DT besteht wie bei den bekannten Datentyparchitekturen weiterhin nur aus dem Datentypidentifikator, da die restlichen Informationen in der Datentypdefinitionstabelle DDT in der Hardware abgelegt sind,
- durch die Spezifikation des für den abgeleiteten Datentyp zulässigen Wertebereichs kann dieser bereits auf Typdefinitionsebene eingeschränkt werden, die Wertebereichskennung WB wird damit entweder überflüssig oder kann für eine detailliertere Einschränkung genutzt werden, ohne einen neuen Datentyp definieren zu müssen.

Allerdings verstößt diese alternative Realisierungsmöglichkeit gegen das in Kapitel 4.2 definierte Entwicklungsparadigma, nach dem alle die Daten beschreibenden Informationen in diesen selbst enthalten sein sollen. Bei der Übermittlung von Daten besteht die Gefahr von Inkonsistenzen, da die Informationen getrennt von den Daten gespeichert werden und damit in anderen Programmen oder Systemkomponenten genau identisch definiert werden müssen. Daher ist die erste Lösung bei der Realisierung einer Datenspezifikationsarchitektur zu bevorzugen.

4.3.4.9 Realisierung sicherer Felder

Zur Erkennung von Pufferunter- und -überläufen sollen in einer Datenspezifikationsarchitektur DSA die von Datenstrukturarchitekturen bekannten sicheren Felder durch Nutzung von Felddeskriptoren und dedizierten Feldzugriffsbefehlen realisiert werden. Dabei wird die Realisierung an jene von ISMA [125] angelehnt. Da für sie die Verwendung von speziellen Datentypkennungen unerlässlich ist, werden die sicheren Felder an dieser Stelle oberflächlich vorgestellt. Details sind [125] zu entnehmen.

ISMA definiert drei Datentypen, die zur Realisierung sicherer Datenfelder notwendig sind:

- Felddesktriptoren,
- Feldeelemente und
- Feldindizes.

In Abbildung 4.16 wird der Aufbau eines Datenfelds im Speicher gezeigt. Den Anfang eines Datenfelds bildet dabei ein Felddeskriptor, dessen Aufbau in Abbildung 4.17 dargestellt ist, der die Anzahl der im Datenfeld enthaltenen Feldelemente und deren Datentyp – genannt Felddatentyp FDT – beschreibt.

Abbildung 4.16: Aufbau eines Datenfelds im Speicher

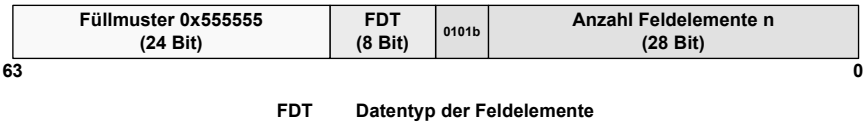


Abbildung 4.17: Aufbau eines Felddeskriptors

Der direkte Zugriff auf Feldelemente wird von der Hardware anhand der Datentypkennung Feldelement als fehlerhaft erkannt und führt zur Generierung eines Ausnahmefehlers.

```
Prüfung_Zugriff :=

WENN DT.DT([Quelle]) ∈ {Felddeskriptor,Feldelement} DANN
  Generierung_Ausnahmefehler;
ENDEWENN
```

Nur über die dedizierten Feldzugriffsbefehle Lade Aus Feld LAF und Speichere In Feld SIF kann auf ein Datenfeld zugegriffen werden. Der Verlauf des Zugriffs wird in Abbildung 4.18 veranschaulicht.

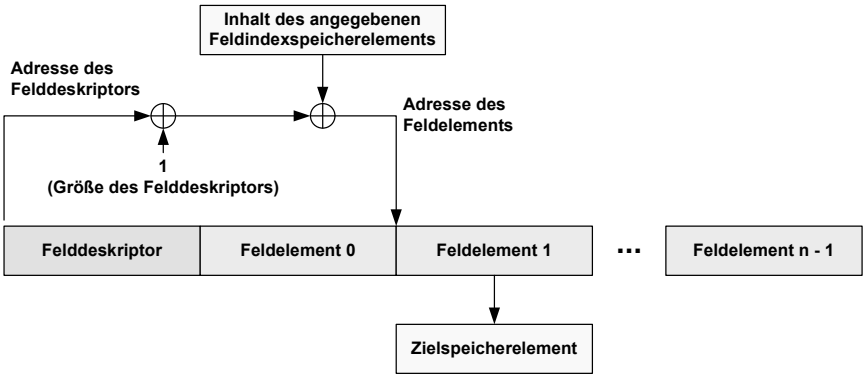


Abbildung 4.18: Zugriff auf Datenfelder über dedizierte Feldzugriffsbefehle

Vor dem eigentlichen Zugriff prüft die Hardware, ob der angegebene Feldindex auf ein Feldelement zeigt, also innerhalb der Feldgrenzen liegt. Ist dies nicht der Fall, wird ebenfalls ein Ausnahmefehler generiert. Weitere Prüfungen, wie beispielsweise, ob das Feldelement lesbare Daten enthält oder ob Schreibzugriffe gestattet sind, werden erst später in dieser Arbeit erläutert und daher hier nicht erwähnt.

```
LAF A, B, C :=

WENN DT.DT([A]) = Felddeskriptor DANN
  WENN W([B]) < Anzahl_Feldelemente([A]) DANN
    W([C]) := W([A + 1 + W([B])]);
  SONST
    Generierung_Ausnahmefehler;
  ENDEWENN
SONST
  Generierung_Ausnahmefehler;
ENDEWENN
```

4.3.4.10 Evaluation der Datentypkennung DT

Mittels der Datentypkennung lassen sich von den 20 in Kapitel 2.4 vorgestellten Fehler- und Angriffsarten die in Tabelle 4.5 gezeigten Fehler erkennen.

Tabelle 4.5: Fehlererkennung durch die Datentypkennung DT

Fehlerart	**Erkennbarkeit**
Inkompatible Datentypen	ja
Inkompatible Einheiten	nein
Wertebereichsunter- bzw. -überschreitung	nein
Genauigkeitsproblem	nein
Falsche Operandenauswahl	begrenzt
Falsche Operatorauswahl	begrenzt
Fehlerhaftes Operationsergebnis	nein
Fristüberschreitung	nein
Zyklusunterschreitung	nein
Zyklusüberschreitung	nein
Verlorengegangene Datenaktualisierung	nein
Synchronisationsfehler oder unvollständige Datenübertragung	nein
Pufferunter- oder -überläufe	ja
Fehlerhafter Datenfluss (falsche Adressaten, ...)	begrenzt
Duplizierte Daten	nein
Durch Fehler oder Störungen verfälschte Daten	begrenzt
Fehlerhafter Datenzugriff (fehlende Zugriffsrechte)	nein
Nutzung nicht initialisierter Daten	begrenzt
Angriffsart	
Gezielt verfälschte Daten	nein
Wiedereinspielungsattacke	nein

Die Verwendung von Operanden mit inkompatiblen Datentypen kann durch den Vergleich ihrer Datentypkennungen in jedem Fall erkannt werden. Das Heranziehen eines falschen Operanden kann nur dann erkannt werden, wenn der falsche Operand einen inkompatiblen Datentyp aufweist. Ebenso kann die Verwendung eines falschen Operators erkannt werden, falls dieserdurfch die Typberechtigungen TB als nicht zulässig markiert wurde. Durch die Implementierung sicherer Felder mit dedizierten Feldzugriffsbefehlen – dem Vorbild der Datenstrukturarchitekturen folgend – können Pufferunter- bzw. -überläufe sicher aufgedeckt werden. Fehlgeleitete Daten werden dann als solche erkannt, wenn deren Datentypen von den erwarteten Datentypen abweichen. Durch Störungen verfälschte Daten können nur dann erkannt werden, wenn die Störung den Inhalt der Datentypkennung betrifft. Die Nutzung nicht initialisierter Daten kann dann erkannt werden, wenn entsprechenden Datenspeicherelementen ohne lesbaren Datenwert W ein spezieller Datentyp zugewiesen wird, der den Inhalt als ungültig ausweist. Diese Vorgehensweise findet z. B. bei ISMA Anwendung, indem die betreffenden Datenspeicherelemente mit dem Datentyp „undefiniert“ versehen werden [125]. Bei einem Lesezugriff auf ein derart markiertes Datenspeicherelement wird ein Ausnahmefehler generiert. Durch die Definition abgeleiteter Datentypen lassen sich Daten noch feingranularer voneinander unterscheiden, mit dem Vorteil einer erweiterten Isolation der Daten und einer noch weiter verbesserten Fehlererkennung. Diese Verbesserung wird in der Tabelle jedoch nicht sichtbar.

4.3.5 Einheit

In [48] wird darauf hingewiesen, dass die Festlegung der Einheiten von Operanden innerhalb von Softwareprogrammen nur unzureichend erfolgt, obwohl bereits 1986 für Ada und Pascal in [80] und NewSpeak in [26] entsprechende Vorschläge unterbreitet wurden. Auf Hardwareebene wurde die Notwendigkeit der hardwareverständlichen Spezifikation der Einheiten von Datenwerten bislang völlig vernachlässigt, obwohl inkompatible Einheiten in der Vergangenheit schon zu schwerwiegenden Sachschäden wie z. B. dem Verlust des Mars Climate Orbiter MCO geführt haben (siehe Kapitel 1.1.2).

4.3.5.1 Realisierung der Einheitenkennung EI

Zur Aufdeckung der versuchten Verarbeitung von Datenwerten mit inkompatiblen Einheiten sollen alle Datenspeicherelemente, die in einer Datenspezifikationsarchi-

tektur erzeugt oder verarbeitet werden, mit einer Einheitenkennung EI versehen werden, wie in Abbildung 4.19.

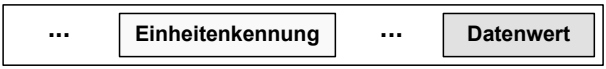


Abbildung 4.19: Datenspeicherelement mit Einheitenkennung EI und Datenwert W

Diese Einheitenkennung EI gibt die Einheit des Datenwerts in Form der vorzeichenbehafteten Potenzen der sieben SI-Basiseinheiten [94]

- Länge l in Meter m,
- Masse m in Kilogramm kg,
- Zeit t in Sekunden s,
- Stromstärke I in Ampere A,
- Temperatur T bzw. θ in Kelvin K,
- Stoffmenge n in Mol mol und
- Lichtstärke I_v bzw. J in Candela cd.

Somit lässt sich die Einheit eines Datenwerts als Tupel der vorzeichenbehafteten Potenzen dieser sieben Basiseinheiten

$$EI := (l,\ m,\ t,\ I,\ T,\ n,\ I_v)$$

darstellen und es ergibt sich der in Abbildung 4.20 gezeigte Aufbau der Einheitenkennung EI.

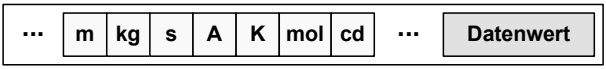


Abbildung 4.20: Aufbau der Einheitenkennung EI

In Abbildung 4.21 werden zwei Beispiele gezeigt, deren Datenwerte W identisch sind und daher in einer konventionellen Architektur – aufgrund der fehlenden hardwareverständlichen Beschreibung der Einheit der Datenwerte – nicht voneinander zu unterscheiden wären:

$30\,\frac{m}{s}$ und $30\,s^{-1} = 30\,\mathrm{Hz}$

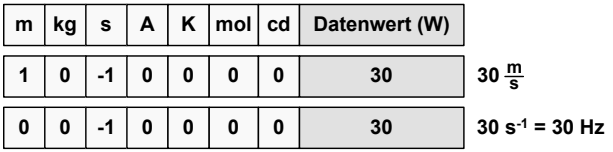


Abbildung 4.21: Beispiele für die Einheitenkennung EI

Eine konventionelle Architektur würde bei einem Vergleich der beiden Werte fälschlich deren Gleichheit feststellen, obwohl beide Werte unterschiedliche Einheiten besitzen.

Für die Einheitenkennung EI wird bei der Realisierung in Hardware eine bestimmte Anzahl an Bits für jede SI-Einheit zur Darstellung der entsprechenden Potenz dieser Einheit vorgesehen. Somit existieren positive und negative Wertebereichsgrenzen für die darstellbaren Einheiten eines Datenwerts. Bei der Definition der Einheiten von Datenwerten kann der Übersetzer zur Übersetzungszeit erkennen, dass die Einheit nicht in der zur Verfügung stehenden Anzahl von Bits darstellbar ist und damit den Übersetzungsvorgang mit einer entsprechenden Fehlermeldung beenden. Es existiert daher eine durch die realisierte Hardware vorgegebene Limitierung der darstellbaren Einheiten, die durch die Software nicht beeinflusst werden kann.

Um die Einheit eines Operanden bzw. die Einheiten mehrerer Operanden direkt bei ihrer Verwendung prüfen zu können, werden die Befehlsspeicherelemente ebenfalls mit einer Einheitenkennung EI ausgestattet, wie in Abbildung 4.22 gezeigt.

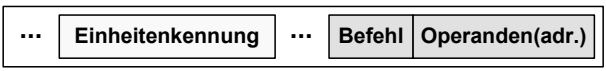


Abbildung 4.22: Einheitenkennung EI in Befehlsspeicherelementen

Diese Einheitenkennung EI innerhalb der Befehlsspeicherelemente besteht aus dem Präsenzbit P und den beiden Einheitenspezifikationen EI_A und EI_B für die Prüfung von bis zu zwei Operanden, wodurch sich der in Abbildung 4.23 dargestellte Aufbau der Kennung ergibt. Die anhand der Kennung durchgeführten Prüfungen werden im nächsten Unterkapitel detailliert vorgestellt.

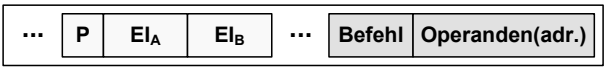


Abbildung 4.23: Aufbau der Einheitenkennung EI in Befehlsspeicherelementen

4.3.5.2 Prüfung der Einheiten von Operanden anhand der Einheitenkennung EI

Anhand der Einheitenkennung EI lässt sich bei Additionen, Subtraktionen und Vergleichsoperationen sicherstellen, dass alle Operanden die identischen Einheiten aufweisen. Auf diese Weise kann der sprichwörtliche Vergleich von Äpfeln mit Birnen sehr einfach durch die Hardware der Datenspezifikationsarchitektur als Fehler erkannt werden.

```
Prüfung_Addition = Prüfung_Subtraktion = Prüfung_Vergleich :=

WENN EI([Quelle_1]) ≠ EI([Quelle_2]) DANN
  Generierung_Ausnahmefehler;
ENDEWENN
```

Durch Nutzung der in der Einheitenkennung EI der Befehle spezifizierten erwarteten Operandeneinheiten können die Einheiten der Operanden bei lesendem Zugriff geprüft werden. Die folgenden Pseudocodeauflistungen zeigen die durchzuführenden Prüfungen für einen bzw. zwei Quelloperanden. In beiden Fällen wird zunächst geprüft, ob das Präsenzbit P innerhalb der Einheitenkennung EI der Befehle die Prüfung der Operandeneinheit bzw. Operandeneinheiten verlangt.

```
Prüfung_Lesezugriff_ein_Quelloperand :=

WENN EI.P([Befehl]) = 1 DANN
  WENN EI([Quelle_1]) ≠ EI.EI_A([Befehl]) DANN
    Generierung_Ausnahmefehler;
  ENDEWENN
ENDEWENN
```

```
Prüfung_Lesezugriff_zwei_Quelloperanden :=

WENN EI.P([Befehl]) = 1 DANN
  WENN EI([Quelle_1]) ≠ EI.EI_A([Befehl]) DANN
    Generierung_Ausnahmefehler;
  ENDEWENN
  WENN EI([Quelle_2]) ≠ EI.EI_B([Befehl]) DANN
    Generierung_Ausnahmefehler;
  ENDEWENN
ENDEWENN
```

4.3.5.3 Setzen der Einheitenkennung EI in Ergebnissen von Operationen

Bei Multiplikationen und Divisionen berechnet die Hardware die sich ergebende Einheit des Ergebnisses basierend auf den Potenzgesetzen. In Abbildung 4.24 wird die Multiplikation zweier Beispielwerte gezeigt, wobei die Prozessorhardware eine Addition der einzelnen Potenzen der EI-Kennung beider Operanden vornimmt und somit ohne Beteiligung der Software die Einheit des Ergebnisses errechnet. Bei einer Division, dargestellt in Abbildung 4.25, nimmt die Prozessorhardware eine entsprechende Subtraktion der einzelnen Potenzen vor.

Die Berechnung der Einheit des Ergebnisses wird in den zwei folgenden Pseudocodeauflistungen gezeigt. Die Hardware prüft dabei, ob bei der Berechnung der Potenzen der Einheiten Unter- bzw. Überläufe auftreten und generiert in einem solchen Fall einen Ausnahmefehler. Aus Platzgründen wird Ergebnis als Erg abgekürzt.

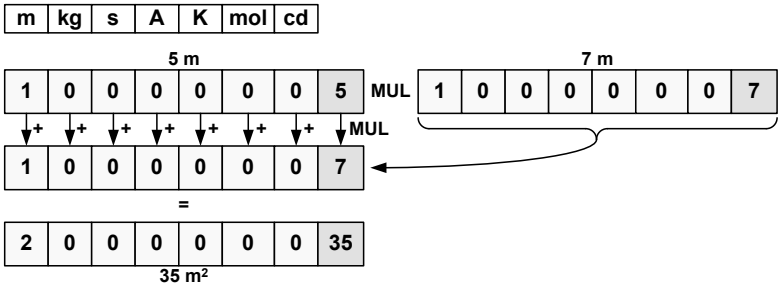


Abbildung 4.24: Multiplikation zweier Beispielwerte

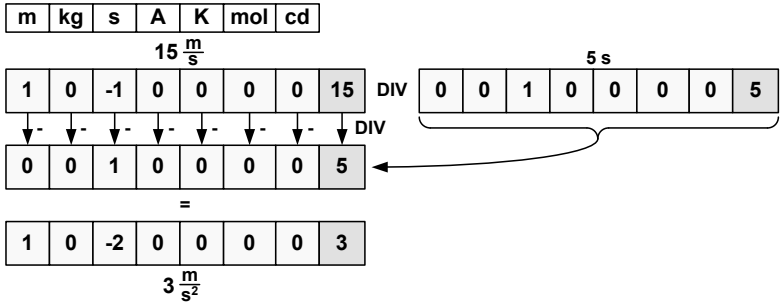


Abbildung 4.25: Division zweier Beispielwerte

```
Setzen_und_prüfen_EI_in_Ergebnis_Multiplikation :=

WIEDERHOLE ∀ i ∈ EI
  EI.i([Erg]) := EI.i([Quelle_1]) + EI.i([Quelle_2])
  WENN Unterlauf(EI.i([Erg])) ∨ Überlauf(EI.i([Erg])) DANN
    Generierung_Ausnahmefehler;
  ENDEWENN
ENDEWIEDERHOLE
```

```
Setzen_und_prüfen_EI_In_Ergebnis_Division :=

WIEDERHOLE ∀ i ∈ EI
  EI.i([Erg]) := EI.i([Quelle_1]) - EI.i([Quelle_2])
  WENN Unterlauf(EI.i([Erg])) ∨ Überlauf(EI.i([Erg])) DANN
    Generierung_Ausnahmefehler;
  ENDEWENN
ENDEWIEDERHOLE
```

4.3.5.4 Befehle zur Verwaltung der Einheitenkennung EI

Um Datenquellen das Setzen der Einheit eines Datenwerts zu ermöglichen, wird der Befehl Setze Einheit SEI definiert.

```
SEI A, B :=

WIEDERHOLE ∀ i ∈ EI
  WENN EI.i ≠ 0 DANN
    Generierung_Ausnahmefehler;
  ENDEWENN
ENDEWIEDERHOLE

EI([B]) := W([A]);
```

Dabei wird zunächst geprüft, ob alle Potenzen der SI-Basiseinheiten in der Einheitenkennung EI des durch B indizierten Zieldatenspeicherelements den Wert Null

besitzen. Diese Prüfung wird durchgeführt, um Manipulationen der Einheit eines Datenwerts zu verhindern, da es sonst möglich wäre, „unangenehme Einheiten“ zu „korrigieren“. Anschließend wird der durch A indizierte Wert in die Einheitenkennung EI des Zieldatenspeicherelements eingetragen. Schlägt die beschriebene Prüfung der Einheitenkennung fehl, so wird ein Ausnahmefehler generiert.

Um auch eine explizite Prüfung der Gleichheit der EI-Kennungen von Datenspeicherelementen zu ermöglichen, ohne dabei wie bei Vergleichsoperationen zugleich die Datenwerte zu vergleichen, wird der Befehl Vergleiche Einheit VEI verwendet. Dieser vergleicht die Einheitenkennung EI der beiden durch A und B indizierten Datenspeicherelemente und löst im Falle der Nichtübereinstimmung der Einheiten einen Ausnahmefehler aus.

```
VEI A, B :=

WENN EI([A]) ≠ EI([B]) DANN
  Generierung_Ausnahmefehler;
ENDEWENN
```

Zur Verifikation der Einheit einzelner Operanden, ohne diese mit der Einheit eines anderen Operanden zu vergleichen, wird der Befehl Prüfe Einheitenkennung Direkt PEID genutzt. Dieser prüft, ob die Einheitenkennung EI des durch A indizierten Datenspeicherelements mit dem durch B indizierten Wert übereinstimmt. Ist dies nicht der Fall, wird ein Ausnahmefehler generiert.

```
PEID A, B :=

WENN EI([A]) ≠ W([B]) DANN
  Generierung_Ausnahmefehler;
ENDEWENN
```

Bei den beiden Befehlen VEI und PEID handelt es sich um Assertionen, also reine Prüfbefehle, die keine Änderung der Operanden hervorrufen, aber im Falle der Nichterfüllung der zu prüfenden Bedingungen einen Ausnahmefehler generieren. Diese Befehle können also dazu genutzt werden, die Gleichheit der Einheiten von Operanden bzw. die Korrektheit der Einheit eines Operanden sicherzustellen, um Folgefehler aufgrund falscher Einheiten zu verhindern.

Statt der beschriebenen Befehle kann auch der in Kapitel 4.3.3.3 vorgestellte Befehl Prüfe Einen Operanden PEO zur Prüfung der Einheit des Operanden gegen die in der Einheitenkennung EI des Befehlsspeicherelements genutzt werden. Wenn das Präsenzbit der Einheitenkennung des Befehlsspeicherelements eine gültige zu prüfende Einheit markiert, so werden die Inhalte der Einheitenkennung EI des Operanden mit denen der Teilkennung EI_A der Einheitenkennung EI des Befehlsspeicherelements verglichen. Bei Nichtübereinstimmung wird ein Ausnahmefehler generiert.

```
PEO A :=

...
WENN EI.P([Befehl]) = 1 DANN
  WENN EI([A]) ≠ EI.EI_A([Befehl]) DANN
    Generierung_Ausnahmefehler;
  ENDEWENN
ENDEWENN
...
```

Sollen die Einheiten von zwei Operanden geprüft werden, so ist dies durch den Einsatz des in Kapitel 4.3.4.5 definierten Befehls Prüfe Zwei Operanden PZO möglich. Die Prüfung erfolgt dabei analog zu den beim Befehl Prüfe Einen Operanden PEO durchgeführten Prüfungen.

```
PZO A, B :=

...
WENN EI.P([Befehl]) = 1 DANN
  WENN EI([A]) ≠ EI.EI_A([Befehl]) DANN
    Generierung_Ausnahmefehler;
  ENDEWENN
  WENN EI([B]) ≠ EI.EI_B([Befehl]) DANN
    Generierung_Ausnahmefehler;
  ENDEWENN
ENDEWENN
...
```

4.3.5.5 Spezifikation der Einheiten von Operanden in Hochsprachen

Zur Spezifikation von Einheiten gibt es Ansätze für die verschiedensten Hochsprachen, darunter z. B.

- C [16],
- F# [30],
- Haskell [15, 34],
- Java [29],
- NewSpeak [26],
- Pascal und Ada [80] und
- PEARL [48].

Ein Hochsprachenübersetzer kann die Spezifikation der Einheiten der Variablen unter Nutzung des Befehls Setze Einheit SEI in deren Einheitenkennung EI übertragen. Da es dem Programmierer in einigen Sprachen möglich ist, eigene Einheiten zu definieren, die nicht auf den SI-Einheiten basieren, müssen diese zur Abbildung in der Einheitenkennung EI in Potenzen der sieben SI-Basiseinheiten überführt werden. Ist dies nicht möglich, so soll die Variablendefinition als fehlerhaft abgelehnt werden. Keinesfalls sollten entsprechende Definitionen stillschweigend akzeptiert werden, um die Variablen dann mit der Einheit 1 zur Laufzeit zu spezifizieren und damit die Sicherheitsmechanismen der Datenspezifikationsarchitektur DSA zu umgehen.

In den folgenden Unterkapiteln werden einige Beispiele für die Spezifikation der Einheiten von Variablen gezeigt, wobei auch schnell deutlich wird, dass hier verschiedenste Ansätze existieren. Für die Hochsprache C wird eine zu den Vorschlägen zur Definition weiterer Dateneigenschaften in dieser Arbeit passende Spezifikationsmöglichkeit vorgestellt.

4.3.5.5.1 Einheitenspezifikation in F#

In der Programmiersprache F# werden nach [30] Einheiten durch das Schlüsselwort `[<Measure>]` deklariert und diese dann in spitze Klammern den Variablen hinzugefügt. Dies wird in der folgenden Auflistung gezeigt, die auf Basis von [30] erstellt wurde.

```
[<Measure>] type m (* meter *)
[<Measure>] type s (* second *)
[<Measure>] type kg (* kilogram *)
[<Measure>] type N = (kg * m)/(s^2) (* Newtons *)
[<Measure>] type Pa = N/(m^2) (* Pascals *)

let v_1 = 3.1<m/s>
let v_2 = 2.7<m/s>
let x_1 = 0.2<m>
```

4.3.5.5.2 Einheitenspezifikation in Java

In [29] wird eine Erweiterung der Hochsprache Java zur Nutzung von Einheiten vorgeschlagen. Dabei können mit Hilfe des Schlüsselworts `dimension` Dimensionstypen definiert werden. Dies wird in der aus [29] stammenden Auflistung gezeigt.

```
dimension Length (meter);
dimension Mass (kilogram);
dimension Time (second);
dimension ElectricCurrent (Ampere);
dimension Temperature (Kelvin);
dimension AmountOfSubstance (mole);
dimension LuminousIntensity (Candela);

double *Time t;
double *Time t = 18.3*second;
double *Length s = 64.2*meter;
```

4.3.5.5.3 Einheitenspezifikation in Pascal

Einheiten können bzw. könnten in der Hochsprache Pascal nach [80] durch das Schlüsselwort `UNIT` deklariert werden und anschließend bei der Variablendefinition einfach hinter die Variablen geschrieben werden. Damit wäre die Spezifikation der Einheiten für den Programmierer sehr einfach. Die Veranschaulichung in der folgenden Auflistung entstammt – mit leichten Modifikationen – [80].

```
UNIT g; (* mass *)
     cm; (* length *)
     sec D' (* time *)
     cents; (* money *)

Distance := 10 km;
Density := 10.0 g|cm3
I := 11.5 eV * ChargeTarget;
```

4.3.5.5.4 Einheitenspezifikation in PEARL

Nach den in [48] gemachten Vorschlägen sollen Einheiten in PEARL-90 durch

`/*+U= <Einheit>*/`

bzw. im objektorientierten PEARL-2020 durch

`DCL <Variablenname> INV <Dimension>`

spezifiziert werden. Die folgende Auflistung als Beispiel für die Einheitendefinition in PEARL-90 wurde [48] entnommen.

```
DCL Current INV FLOAT INIT (12.0) /*+ U = mA */;
DCL Inductance INV FLOAT INIT (2.5) /*+ U = uH */;
DCL Length_cm INV FLOAT INIT (1.0) /*+ U = cm */;
DCL Windings INV FIXED INIT (10) /*+ U = 1 */;
DCL Resistance INV FLOAT INIT (5.0) /*+ U = Ohm */;
```

4.3.5.5.5 Ein neuer Vorschlag für die Hochsprache C

Die Spezifikation der Einheit einer Variablen könnte – den weiteren Vorschlägen zur Definition von Variablenattributen in dieser Arbeit folgend – durch Einführung des neuen Attributs

`__unit(<Einheit^Potenz,Einheit^Potenz,...>)`

erfolgen. Die Potenz einer in der Liste nicht aufgeführten Basiseinheit wird mit Null und die einer ohne Potenz angegebenen Basiseinheit mit Eins angenommen. In der folgenden Auflistung wird eine Variable definiert, die die Einheit $\frac{m}{s}$ besitzt.

```
int __unit(m,s^-1) variable_a;
```

4.3.5.6 Evaluation der Einheitenkennung

Aufgrund der Vorarbeiten hinsichtlich der Spezifikation der Einheiten von Variablen in Hochsprachen fällt die Realisierung der Einheitenkennung einfach aus. Der Übersetzer muss die jeweiligen Einheiten beim Übersetzungsvorgang nur noch in den Datenworten hinterlegen. Die Hardware kann anhand der Kennung bei bestimmten Operationen, wie z. B. Zuweisungen, Additionen, Subtraktionen, aber auch ggf. Zeichenketten und Bitfeldern die Kompatibilität der Operanden überprüfen und Inkompatibilitäten sofort erkennen und eine entsprechende Reaktion auslösen. Bei anderen arithmetischen Operationen wie Multiplikationen und Divisionen werden die Einheiten der Operanden entsprechend der durchzuführenden Operation verarbeitet und dem Ergebnis zugewiesen. Hat das Zieldatenspeicherelement eine vordefinierte Einheit, kann die Gleichheit der Einheit des Ergebnisses mit der des Zieldatenspeicherelements durch die Hardware geprüft werden.

Natürlich könnte während der Entwicklung eines Systems auf die Anwendung der Einheitenkennung verzichtet werden, indem die Daten einfach nicht mit entsprechenden Einheitenangaben versehen werden. Das könnte jedoch einerseits vom Übersetzer erzwungen werden, andererseits ist eine solche Umgehung des Sicherheitsmerkmals bei einer Codebegutachtung im Verlauf einer Zertifizierung sehr einfach zu identifizieren und sollte zur Ablehnung der Zertifizierung führen.

Bezogen auf die 20 in Kapitel 2.4 vorgestellten Fehler- und Angriffsarten gestattet die EI-Kennung die Aufdeckung der in Tabelle 4.6 gezeigten Fehlerarten.

Die EI-Kennung erlaubt die Erkennung von Inkompatibilitäten von Operanden, die auf nicht zueinander passenden Einheiten beruhen. Sollten sich durch die jeweilige Fehlerart inkompatible Einheitenkennungen der Operanden ergeben, können in begrenztem Umfang mittels der EI-Kennung ggf. auch weitere Fehlerarten erkannt werden:

- falsche Operandenauswahl, wenn die falschen Operanden unerwartete Einheiten besitzen,
- falsche Operatorauswahl, z. B. eine Addition statt einer Multiplikation, wobei bei nicht identischen Einheiten der Operanden der Fehler aufgedeckt würde,

Tabelle 4.6: Fehlererkennung durch die EI-Kennung

Fehlerart	**Erkennbarkeit**
Inkompatible Datentypen	nein
Inkompatible Einheiten	ja
Wertebereichsunter- bzw. -überschreitung	nein
Genauigkeitsproblem	nein
Falsche Operandenauswahl	begrenzt
Falsche Operatorauswahl	begrenzt
Fehlerhaftes Operationsergebnis	nein
Fristüberschreitung	nein
Zyklusunterschreitung	nein
Zyklusüberschreitung	nein
Verlorengegangene Datenaktualisierung	nein
Synchronisationsfehler oder unvollständige Datenübertragung	nein
Pufferunter- oder -überläufe	nein
Fehlerhafter Datenfluss (falsche Adressaten, ...)	begrenzt
Duplizierte Daten	nein
Durch Fehler oder Störungen verfälschte Daten	begrenzt
Fehlerhafter Datenzugriff (fehlende Zugriffsrechte)	nein
Nutzung nicht initialisierter Daten	(ja)
Angriffsart	
Gezielt verfälschte Daten	nein
Wiedereinspielungsattacke	nein

- fehlerhafter Datenfluss, wenn die fehlgeleiteten Daten von den Erwartungen abweichende Einheiten aufweisen und
- durch Störungen verfälschte Daten, wenn die Verfälschung die Einheitenkennung betrifft.

Die Verwendung nicht initialisierter Daten kann durch die EI-Kennung aufgedeckt werden, wenn die betroffenen Datenspeicherelemente eine unerwartete Einheitenkennung aufweisen. Zur Erhöhung der Wahrscheinlichkeit, dass ein entsprechender Zugriffsfehler aufgedeckt werden kann, ist es sinnvoll, uninitialisierten Datenspeicherelementen eine besonders ungewöhnliche Einheitenkennung zuzuweisen, z. B. die maximalen oder minimalen Potenzen in allen SI-Basiseinheiten.

4.3.6 Zugriffsrechte und Initialisierungsstatus

Durch die Dateneigenschaft Zugriffsrechte ZR kann innerhalb einer Systemkomponente spezifiziert werden, wer auf das jeweilige Datenspeicherelement Zugriff haben darf und ob dieser nur lesend oder auch schreibend erfolgen darf. Nach dem Vorbild der Befähigungsarchitekturen handelt es sich bei dieser Dateneigenschaft also um eine Befähigung, die das Geheimnisprinzip und die Kapselung von Befehlen und Daten realisiert.

Eine weitere Dateneigenschaft, die den Zugriffsrechten zuzuordnen ist, ist der Initialisierungsstatus IS, der beschreibt, ob ein Datenspeicherelement lesbare Daten enthält. Die Nutzung nicht initialisierter Variablen stellt nach [118] einen der häufigsten Datenflussfehler dar.

4.3.6.1 Realisierung der Zugriffsrechtekennung ZR

Zur Spezifikation der Zugriffsrechte ZR und des Initialisierungsstatus IS wird den Datenspeicherelementen eine Zugriffsrechtekennung ZR hinzugefügt, wie in Abbildung 4.26 dargestellt.

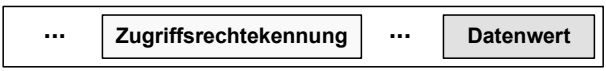


Abbildung 4.26: Datenspeicherelement mit Zugriffsrechtekennung ZR und Datenwert W

Während in Befähigungsarchitekturen derjenige Zugriff auf Daten hat, der die dazugehörige Befähigung besitzt [39, 78], verwendet ISMA eine Mikrosegmentierung, wodurch Daten und Befehle feinstgranular einzelnen Funktionen zugeordnet und vor fehlerhaftem Zugriff geschützt werden können [125]. Dazu wird der Programmspeicher, wie in Abbildung 4.27 gezeigt, in Programmmodule und diese wiederum in einzelne Funktionen unterteilt. Diese Unterteilung entspricht idealerweise der Aufteilung des Programms bzw. der Programme im Quellcode. Jedes Modul wird durch eine Modulnummer MN identifiziert, die innerhalb einer Systemkomponente eindeutig ist. Ebenso werden alle Funktionen durch eine modulweit eindeutige Funktionsnummer FN beschrieben, wodurch sich eine Funktion durch das Tupel (MN,FN) in einer Systemkomponente eindeutig identifizieren lässt. Diese Mikrosegmentierung soll für die Datenspezifikationsarchitektur DSA übernommen werden.

Neben der Zuordnung von Daten und Befehlen zu einer bestimmten Funktion, kann in der Zugriffsrechtekennung auch vermerkt werden, ob auf den in einem Datenspeicherelement enthaltenen Datenwert W nur lesend oder auch schreibend zugegriffen werden darf. Dies wird durch den Schreibrechtebeschreiber SR spezifiziert.

Die Zugriffsrechtekennung ZR beinhaltet die beschriebenen Komponenten

- Modulnummer MN,
- Funktionsnummer FN,
- Schreibrechtebeschreiber SR und
- Initialisierungsstatus IS,

wodurch sich der in Abbildung 4.28 dargestellte Aufbau ergibt.

Um eine Prüfung der Zugehörigkeit von Daten und Befehlen zur aktuell ausgeführten Funktion im aktuellen Programmmodul zu ermöglichen, bietet ISMA zwei Register an, die bei der DSA im in Abbildung 4.29 gezeigten Zugriffsrechteregister ZRR zusammengefasst werden.

Dieses Register besteht aus den zwei Teilregistern Aktuelle Modulnummer AMN und Aktuelle Funktionsnummer AFN, wie in Abbildung 4.30 dargestellt. Die Inhalte der Register werden von der Hardware bei Funktionsaufrufen automatisch gesetzt und bei jedem Zugriff auf Daten- und Befehlsspeicherelemente mit den Inhalten der Zugriffsrechtekennung ZR verglichen. Auf diese Weise kann jederzeit sichergestellt werden, dass nur die zur aktuellen Funktion gehörenden Befehle ausgeführt und die der Funktion zugeordneten Daten gelesen bzw. geschrieben werden können.

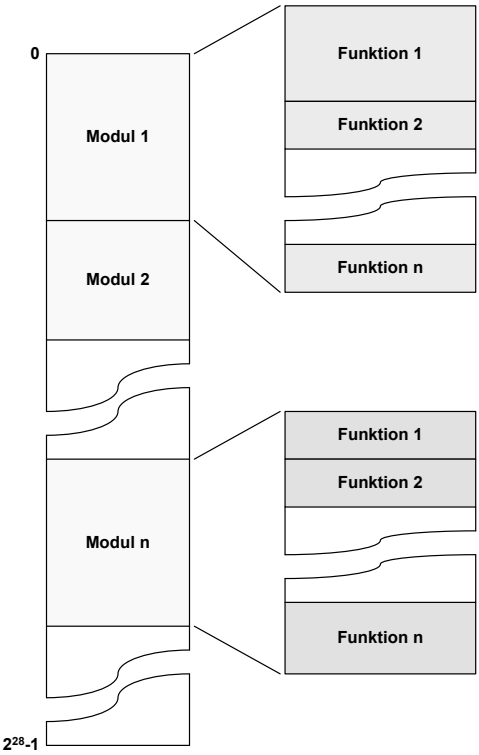


Abbildung 4.27: Mikrosegmentierung bei ISMA

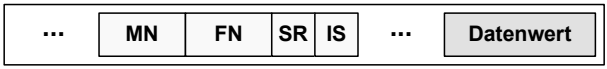


Abbildung 4.28: Aufbau der Zugriffsrechtekennung ZR

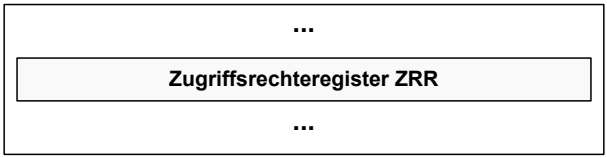


Abbildung 4.29: Zugriffsrechteregister ZRR im Registersatz der DSA

Abbildung 4.30: Aufbau des Zugriffsrechteregisters ZRR

Das genaue Vorgehen zur Prüfung der in der Zugriffsrechtekennung ZR spezifizierten Eigenschaften wird im folgenden Unterkapitel detailliert vorgestellt.

4.3.6.2 Prüfung der Zugriffsrechte anhand der Zugriffsrechtekennung ZR

Bei jedem Zugriff überprüft die Hardware einer Datenspezifikationsarchitektur DSA, ob die in den Teilregistern Aktuelle Modulnummer AMN und Aktuelle Funktionsnummer AFN des Zugriffsrechteregisters ZRR spezifizierten Modul- und Funktionsnummern der gerade ausgeführten Programmfunktion mit denen des zu lesenden oder zu schreibenden Datenspeicherelements identisch sind. Wird auf ein Datenspeicherelement zugegriffen, dass nicht zur aktuell ausgeführten Funktion gehört, so wird durch die DSA ein Ausnahmefehler generiert.

```
Prüfung_Zugriff :=

WENN ZR.MN([Quelle]) ≠ [ZRR.AMN] DANN
  Generierung_Ausnahmefehler;
ENDEWENN

WENN ZR.FN([Quelle]) ≠ [ZRR.AFN] DANN
  Generierung_Ausnahmefehler;
ENDEWENN
```

Vor jedem Schreibzugriff wird geprüft, ob das Zieldatenspeicherelement beschrieben werden darf, wozu das Zieldatenspeicherelement zuerst gelesen und der Schreibrechtebeschreiber SR innerhalb der Zugriffsrechtekennung ZR ausgewertet wird. Hat dieser den Wert Null, wodurch das Zieldatenspeicherelement des Schreibzugriffs als nur-lesbar markiert ist, so wird ein Ausnahmefehler generiert.

```
Prüfung_Schreibzugriff :=

WENN ZR.SR([Quelle]) ≠ 1 DANN
  Generierung_Ausnahmefehler;
ENDEWENN
```

Bei jedem lesenden Zugriff auf ein Datenspeicherelement prüft die Hardware der Datenspezifikationsarchitektur DSA, ob das Datenspeicherelement verarbeitbare Daten enthält, indem sie den Inhalt der Initialisierungsstatusbeschreibers IS auswertet. Ergibt diese Auswertung, dass ein Programm soeben versucht hat, Daten zu verarbeiten, denen vorab kein sinnvoller Wert zugewiesen wurde, so wird ein Ausnahmefehler generiert.

```
Prüfung_Lesezugriff :=

WENN ZR.IS([Quelle]) ≠ 1 DANN
  Generierung_Ausnahmefehler;
ENDEWENN
```

4.3.6.3 Befehle zur Verwaltung der Zugriffsrechtekennung ZR

Zum Setzen des Besitzers initialisierter lokaler Variablen bietet ISMA den Befehl Setze Elementbesitzer SEB an, der nur während der Initialisierungsphase eines Systems angewendet werden kann. Er kann sowohl auf einzelne Datenspeicherelemente, als auch auf Felddeskriptoren angewendet werden. Dabei werden Modul- und Funktionsnummer in der Zugriffsrechtekennung ZR des durch B indizierten Datenspeicherelements bzw. aller Elemente des durch B indizierten Datenfelds auf die in A spezifizierten Werte gesetzt. Die Funktion FG([Operand]) liefert dabei die Feldgröße aus dem Felddeskriptor zurück und der Index hinter dem Operandennamen gibt die zu nutzenden Bitpositionen an.

```
SEB A, B :=

WENN DT([B]) = FD DANN
  ZR.MN([B]...[B + FG([B])]) := A.31...A.16;
  ZR.FN([B]...[B + FG([B])]) := A.15...A.0;
```

```
SONST
  ZR.MN([B]) := A.31...A.16;
  ZR.FN([B]) := A.15...A.0;
ENDEWENN
```

Der Befehl Setze Nur-Lesbar SNL ist ebenfalls von ISMA bekannt und dient dem Löschen der Schreibrechte eines Datenspeicherelements. Dabei wird der Schreibrechtebeschreiber SR der Zugriffsrechtekennung ZR des durch A indizierten Datenspeicherelements auf Null gesetzt.

```
SNL A :=

ZR.SR([A]) := 0;
```

Soll ein Feldelement als nur-lesbar markiert werden, kann der Befehl Setze Feldelement Nur-Lesbar SFNL angewandt werden. Dieser setzt – analog zum Befehl SNL – den Schreibrechtebeschreiber SR der Zugriffsrechtekennung ZR des durch A und B indizierten Feldelements auf Null.

```
SFNL A, B :=

ZR.SR([B + 1 + W([A])]) := 0;
```

In ISMA wurde der Befehl Lösche Initialisierungsstatusbeschreiber LI definiert, um den Initialisierungsstatus IS eines Datenspeicherelements zurücksetzen zu können. Damit ist es möglich, Daten als nicht mehr lesbar zu markieren, was z. B. bei Puffern, denen Daten entnommen werden, der Erkennung von Lesezugriffen dient, bei denen nicht mehr gültige Daten verarbeitet werden sollen. Bei der Ausführung des Befehls wird der Initialisierungsstatusbeschreiber IS in der Zugriffsrechtekennung ZR des durch A indizierten Zieldatenspeicherelements auf den Wert Null gesetzt.

```
LI A :=

ZR.IS([A]) := 0;
```

Zum Rücksetzen des Initialisierungsstatusbeschreibers von Feldelementen dient der Befehl Lösche Initialisierungsstatusbeschreiber eines Feldelements LFI analog zum

Befehl LI. Dabei wird der Initialisierungsstatusbeschreiber IS in der Zugriffsrechtekennung ZR des durch A und B indizierten Feldelements auf Null gesetzt.

```
LFI A, B :=

ZR.IS([B + 1 + W([A])]) := 0;
```

Der in Kapitel 4.3.3.3 eingeführte Befehl Prüfe Einen Operanden PEO kann auch zur Prüfung der Zugriffsrechte eingesetzt werden. Dabei werden – neben weiteren durch „...“ angedeuteten Prüfungen – die in der folgenden Auflistung gezeigten Prüfungen eines lesenden Zugriffs durchgeführt. Dabei wird sichergestellt, dass der durch A indizierte Operand zum aktuellen Programmmodul und zur aktuellen Programmfunktion gehört. Weiterhin wird anhand des Initialisierungsstatusbeschreibers IS sichergestellt, dass das Datenspeicherelement lesbare Daten enthält.

```
PEO A :=

...

WENN ZR.MN([A]) ≠ [ZRR.AMN] DANN
  Generierung_Ausnahmefehler;
ENDEWENN

WENN ZR.FN([A]) ≠ [ZRR.AFN] DANN
  Generierung_Ausnahmefehler;
ENDEWENN

WENN ZR.IS([A]) ≠ 1 DANN
  Generierung_Ausnahmefehler;
ENDEWENN

...
```

4.3.6.4 Spezifikation von Zugriffsrechten in Hochsprachen

Die Modul- und Funktionsnummernfelder MN und FN müssen in Hochsprachen nicht manuell definiert werden. Entwicklungswerkzeuge wie Übersetzer oder Binder

können die Modulnummern auf Basis von Programmeinheiten oder Quelltextdateien vergeben. Gleiches gilt für die in Programmmodulen enthaltenen Funktionen, für die die Werkzeuge die Funktionsnummern automatisiert generieren können.

Zur Definition nur-lesbarer Daten bietet die Hochsprache C das Schlüsselwort `const` an. Soll die Schreibbarkeit während der Laufzeit geändert werden, kann eine neue intrinsische Funktion genutzt werden. Dafür wird

`__set_readonly(<Variablenname>)`

vorgeschlagen. Der Übersetzer überführt diese Funktion in Abhängigkeit von der Art der Variable in den Befehl Setze Nicht Lesbar SNL oder Setze Feldelement Nicht Lesbar SFNL.

Der Initialisierungsstatusbeschreiber wird durch die Hardware der Datenspezifikationsarchitektur automatisch gesetzt, sobald durch einen Schreibzugriff lesbare Daten in ein Datenspeicherelement geschrieben wurde. Um bei verbrauchenden Lesevorgängen, wie z. B. dem Abrufen von Pufferinhalten die betroffenen Datenspeicherelemente als nicht initialisiert bzw. ungültig zu markieren, kann ebenfalls eine neue intrinsische Funktion eingeführt werden. Diese könnte mit

`__set_not_initialized(<Variablenname>)`

bezeichnet werden. Je nach Art der Variable wird der Übersetzer dann den Befehl Lösche Initialisierungsstatus LI oder Lösche Feldelement Initialisierungsstatus LFI in das Programm einfügen. Die folgende Auflistung zeigt beispielhaft den Einsatz der beiden intrinsischen Funktionen zur Modifikation der Eigenschaften einer Variable bzw. eines Feldelements.

```
__set_readonly(x);
__set_not_initialized(feld_y[z]);
```

4.3.6.5 Evaluation der Zugriffsrechtekennung ZR

Die Erkennbarkeit der 20 in Kapitel 2.4 identifizierten Fehler- und Angriffsarten durch die Zugriffsrechtekennung ZR wird in Tabelle 4.7 dargestellt.

Das Heranziehen falscher Operanden kann nur dann erkannt werden, wenn diese anderen Programmmodulen oder -funktionen zugeordnet sind, bei Schreibzugriffen keine Schreibrechte aufweisen oder bei Lesezugriffen keine lesbaren Daten enthalten. Pufferunter- bzw. -überläufe können ebenfalls nur dann erkannt werden, wenn eine

Tabelle 4.7: Fehlererkennung durch die ZR-Kennung

Fehlerart	**Erkennbarkeit**
Inkompatible Datentypen	nein
Inkompatible Einheiten	nein
Wertebereichsunter- bzw. -überschreitung	nein
Genauigkeitsproblem	nein
Falsche Operandenauswahl	begrenzt
Falsche Operatorauswahl	nein
Fehlerhaftes Operationsergebnis	nein
Fristüberschreitung	nein
Zyklusunterschreitung	nein
Zyklusüberschreitung	nein
Verlorengegangene Datenaktualisierung	nein
Synchronisationsfehler und unvollständige Datenübertragung	nein
Pufferunter- oder -überläufe	begrenzt
Fehlerhafter Datenfluss (falsche Adressaten, ...)	nein
Duplizierte Daten	nein
Durch Störungen oder Fehler verfälschte Daten	begrenzt
Fehlerhafter Datenzugriff (fehlende Zugriffsrechte)	ja
Nutzung nicht initialisierter Daten	ja
Angriffsart	
Gezielt verfälschte Daten	nein
Wiedereinspielungsattacke	nein

der genannten Bedingungen erfüllt ist. Werden Daten durch Störungen verfälscht, dann kann die ZR-Kennung dies aufdecken, wenn die durch die Störung verursachten Bitfehler die Bits der ZR-Kennung betreffen. Fehlerhafter Datenzugriff auf nicht zur aktuellen Programmfunktion des aktuellen Programmmoduls gehörende Daten, sowie der Versuch, auf schreibgeschützte Daten schreibend zuzugreifen, können durch die ZR-Kennung zuverlässig erkannt werden. Der Versuch, nicht initialisierte Daten zu lesen, wird anhand des Initialisierungsstatusfelds der Zugriffsrechtekennung ZR aufgedeckt.

4.3.7 Quelle, Verarbeitungsweg und Ziel

Eine sehr wichtige zu prüfende Eigenschaft eines Datenflusses ist die seines Weges durch ein System. Die drei Dateneigenschaften Quelle Q, Verarbeitungsweg VW und Ziel Z beschreiben den genauen Pfad von Daten von ihrer Entstehung bis zu ihrer endgültigen Nutzung. Üblicherweise ist in Prozessautomatisierungssystemen bereits zum Zeitpunkt des Systementwurfs klar, welche Daten sich auf welchen Wegen durch ein System bewegen und dabei verarbeitet werden sollen. Werden diese Informationen in der Systemspezifikation verankert und dem Übersetzer zugänglich gemacht, so kann dieser die vorgeschriebenen Verarbeitungswege im System den Daten in Form einer Verarbeitungswegkennung hinzufügen und die Hardware die Einhaltung dieser Vorgaben während der Datenverarbeitung prüfen.

4.3.7.1 Realisierung der Verarbeitungswegkennung VW

Die drei erwähnten Dateneigenschaften Quelle Q, Verarbeitungsweg VW und Ziel Z werden in der Verarbeitungswegkennung VW zusammengefasst, wie in Abbildung 4.31 dargestellt.

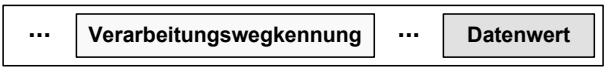


Abbildung 4.31: Datenspeicherelement mit Verarbeitungswegkennung VW und Datenwert W

Da zum Zeitpunkt der Spezifikation eines Gesamtsystems meist nur die Subsysteme feststehen, diese aber unter Umständen von verschiedenen Herstellern entwickelt

werden und somit keine Interna dieser Subsysteme bekannt sind, ist es vorstellbar, die Verarbeitungswegkennung VW in zwei Teile aufzuspalten: einen Systemteil VW_{sys} und einen Lokalteil VW_{lok}.

Der Systemteil VW_{sys}, dessen Inhalte durch die Systemspezifikation festgelegt werden müssen, beinhaltet die hardwareverständliche Beschreibung, welche Subsysteme ein bestimmter Datenfluss auf seinem Weg durchlaufen darf. Er enthält dagegen keine Beschreibung, wie dieser Verarbeitungsweg innerhalb eines Subsystems ausgestaltet sein soll.

Der Lokalteil VW_{lok}, der innerhalb eines Subsystems gesetzt wird, dient der hardwareverständlichen Beschreibung des Datenverarbeitungswegs innerhalb eines Subsystems.

Damit ergibt sich der in Abbildung 4.32 gezeigte Aufbau der Verarbeitungswegkennung VW, bestehend aus

- der Quellkennung Q, die die Quelle bzw. die Quellen eines Datenwerts angibt,
- dem Systemteil VW_{sys}, der – wie bereits beschrieben – spezifiziert, welche Systemkomponenten das betreffende Datenspeicherelement verarbeiten dürfen,
- dem Lokalteil VW_{lok}, der – wie ebenfalls bereits beschrieben – festlegt, welche Datenverarbeitungseinheiten innerhalb einer Systemkomponente auf das Datenspeicherelement zugreifen dürfen, und
- die Zielkennung Z, die beschreibt, welche Senken innerhalb des Systems die Daten nach der Verarbeitung entgegennehmen dürfen.

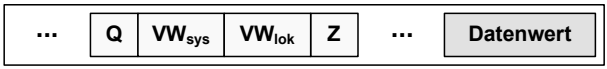


Abbildung 4.32: Aufbau der Verarbeitungswegkennung VW im Datenspeicherelement

Nimmt ein Subsystem ein Datenspeicherelement entgegen, prüft es anhand der eigenen Systemidentifikation, die mit dem Systemteil VW_{sys} der Verarbeitungswegkennung VW abgeglichen wird, ob es die Berechtigung bzw. die Fähigkeit besitzt, die Daten zu verarbeiten. Wird hierbei kein Fehler festgestellt, so wird der Lokalteil VW_{lok} der Verarbeitungswegkennung VW des Datenspeicherelements auf den

zugehörigen Wert gesetzt und somit eine Prüfung des Wegs des betreffenden Datenspeicherelements innerhalb der Systemkomponente ermöglicht.

Zur Spezifikation der erwarteten Inhalte in den Verarbeitungswegkennungen VW der Datenspeicherelemente wird den Befehlsspeicherelementen ebenfalls eine Verarbeitungswegkennung hinzugefügt, wie in Abbildung 4.33 dargestellt.

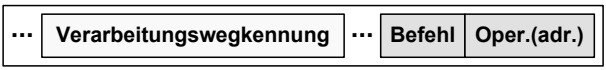


Abbildung 4.33: Befehlsspeicherelement mit Verarbeitungswegkennung VW

Dabei wird die Kennung – analog zum Aufbau der Verarbeitungswegkennung VW der Datenspeicherelemente – in die Teilkennungen Quellkennung Operand A Q_A, Quellkennung Operand B Q_B, Systemteil VW_{sys}, Lokalteil VW_{lok} und Zielkennung Z aufgeteilt, wie in Abbildung 4.34 gezeigt. Die Spezifikation der zwei Quellkennungen Q_A und Q_B erlaubt die Prüfung der Quellkennungen von bis zu zwei Operanden.

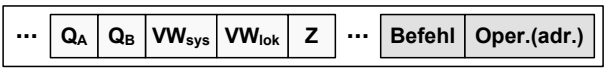


Abbildung 4.34: Aufbau der Verarbeitungswegkennung VW im Befehlsspeicherelement

Da die Inhalte der Verarbeitungswegkennung VW der Befehlsspeicherelemente bereits zur Übersetzungszeit bekannt sein müssen, diese aber nicht in jedem Fall zu diesem Zeitpunkt bekannt sind, wird der Registersatz der Datenspezifikationsarchitektur DSA um das in Abbildung 4.35 gezeigte Verarbeitungswegregister VWR erweitert. Dieses wird gleichberechtigt zur Verarbeitungswegkennung VW der Befehlsspeicherelemente zur Prüfung der Inhalte der Verarbeitungswegkennung VW der Datenspeicherelemente herangezogen und kann bei Systemstart mit den Inhalten einer Konfigurationsdatei befüllt werden.

Wie in Abbildung 4.36 gezeigt, besteht das Verarbeitungswegregister VWR aus den zwei Quellregistern Q_AR und Q_BR, dem Systemteilregister des Verarbeitungswegs

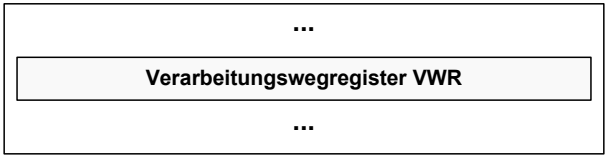


Abbildung 4.35: Verarbeitungswegregister VWR im Registersatz

$VW_{sys}R$, dem Lokalteilregister des Verarbeitungswegs $VW_{lok}R$ und dem Zielregister ZR. Die Bedeutung der Teilregister deckt sich mit der der Teilkennungen der Verarbeitungswegkennung VW der Befehlsspeicherelemente.

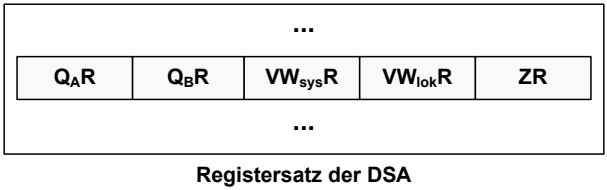


Abbildung 4.36: Aufbau des Verarbeitungswegregisters VWR

4.3.7.2 Identifikation der Systemkomponenten innerhalb der Verarbeitungswegkennung VW

Ein sehr einfacher Ansatz für die Identifikation der Systemkomponenten innerhalb der Verarbeitungswegkennung VW ist die Zuweisung einer Bitposition für jede Datenquelle des Systems in Q, jeder Datenverarbeitungseinheit in VW_{sys} und jeder Senke in Z. Ebenso werden den Datenverarbeitungsblöcken innerhalb der Datenverarbeitungseinheiten Bitpositionen innerhalb des Lokalteils VW_{lok} zugewiesen. Bei der Generierung von Daten setzt die Quelle das ihr zugewiesene Bit der Quellkennung Q, ebenso die Bits der zugriffsberechtigten Datenverarbeitungseinheiten im Systemteil VW_{sys}, – sofern bekannt – die Bits der dem Zugriff berechtigten Datenverarbeitungsblöcke innerhalb der ersten Datenverarbeitungseinheit in Lokalteil VW_{lok} und die Bits aller Senken in der Zielkennung Z, die die Daten nach deren Verarbeitung entgegennehmen dürfen.

Datenverarbeitungseinheiten, welche selbst neue Daten erzeugen, die nicht durch die Verarbeitung von Daten aus anderen Quellen entstehen, wird eine eigene Bitposition in der Quellkennung Q zugewiesen. Analog dazu erhalten Datenverarbeitungseinheiten, die bestimmte Daten als Endabnehmer in Empfang nehmen und diese nicht nach der Verarbeitung weiterleiten, zusätzlich eine Bitposition in der Zielkennung zugewiesen.

Daher kann eine Datenverarbeitungseinheit jeweils eine Bitposition in den Teilkennungen Quellkennung Q, Systemteil des Verarbeitungswegs VW_{sys} und der Zielkennung Z aufweisen.

4.3.7.3 Prüfung des Datenflusses anhand der Verarbeitungswegkennung VW

Wie bei der Realisierung der Verarbeitungswegkennung VW der Datenspeicherelemente beschrieben, kann die Prüfung des Verarbeitungswegs unter Nutzung

- der Verarbeitungswegkennung VW der Befehlsspeicherelemente oder
- des Verarbeitungswegregisters VWR im Registersatz der Datenspezifikationsarchitektur DSA

erfolgen. In den zwei folgenden Unterkapiteln wird zunächst die Durchführung der Prüfungen anhand der beiden Angaben vorgestellt, um anschließend zu erläutern, wie explizit auf die einzelnen Teilprüfungen verzichtet werden kann.

4.3.7.3.1 Durchführung der Prüfungen des Verarbeitungswegs

Bei jedem Datenzugriff in Datenverarbeitungseinheiten wird durch die Hardware geprüft, ob innerhalb des Systemteils VW_{sys} der Verarbeitungswegkennung VW das Bit an der der Verarbeitungseinheit zugeordneten Bitposition gesetzt ist, die Verarbeitung der Daten in der Einheit dadurch also gestattet ist. Dazu wird der Inhalt des Systemteils VW_{sys} mit dem Systemteilregister $VW_{sys}R$ des Verarbeitungswegregisters VWR bzw. dem Systemteil VW_{sys} der Verarbeitungswegkennung des Befehlsspeicherelements durch eine Konjunktion verknüpft. Das jeweilige Ergebnis der UND-Verknüpfung muss mit den Inhalten des Teilregisters bzw. der Teilkennung identisch sein. Ist dies nicht der Fall, dann waren nicht alle geforderten Bits in der Systemteilkennung VW_{sys} des Datenspeicherelements gesetzt und die prüfende

Datenverarbeitungseinheit darf nicht auf die Daten zugreifen, woraufhin ein Ausnahmefehler generiert wird. Aus Platzgründen wird einigen Pseudocodeauflistungen die Quelle als Q abgekürzt.

```
Prüfung_SystemteilVW :=

WENN VW.VW_sys([Q]) UND [VWR.VW_sysR] ≠ [VWR.VW_sysR] DANN
  Generierung_Ausnahmefehler;
ENDEWENN

WENN VW.VW_sys([Q]) UND VW.VW_sys([Befehl]) ≠ VW.VW_sys([Befehl]) DANN
  Generierung_Ausnahmefehler;
ENDEWENN
```

Innerhalb der Datenverarbeitungsblöcke der Datenverarbeitungseinheiten des Systems wird durch die Hardware geprüft, ob der Lokalteil $\mathrm{VW_{lok}}$ der Verarbeitungswegkennung VW der Datenspeicherelemente dem jeweiligen Datenverarbeitungsblock den Zugriff gestattet. Die Prüfung erfolgt analog zum Systemteil $\mathrm{VW_{sys}}$ der Verarbeitungswegkennung der Datenspeicherelemente.

```
Prüfung_LokalteilVW :=

WENN VW.VW_lok([Q]) UND [VWR.VW_lokR] ≠ [VWR.VW_lokR] DANN
  Generierung_Ausnahmefehler;
ENDEWENN

WENN VW.VW_lok([Q]) UND VW.VW_lok([Befehl]) ≠ VW.VW_lok([Befehl]) DANN
  Generierung_Ausnahmefehler;
ENDEWENN
```

Die Hardware von Verarbeitungseinheiten und Senken kann anhand der Quellkennung Q der Verarbeitungswegkennung VW prüfen, ob die empfangenen Daten von der erwarteten Quelle stammen, bzw. die verarbeiteten Ergebnisse der erwarteten Quellen enthalten. Dazu wird der Inhalt der Quellkennung Q des Datenspeicherelements mit dem Inhalt eines der beiden Quellregister $\mathrm{Q_AR}$ bzw. $\mathrm{Q_BR}$ und einer der beiden Quellkennungen $\mathrm{Q_A}$ bzw. $\mathrm{Q_B}$ des Befehlsspeicherelements verglichen, sofern diese ungleich Null sind. Bei Nichtübereinstimmung wird ein Ausnahmefehler generiert. Auch innerhalb der Datenquellen selbst kann die beschriebene Prüfung sinnvoll

sein, um fehlerhaft gesetzte Quellkennungen innerhalb der eigenen Datenspeicherelemente aufzudecken. Hat ein Befehl nur einen Quelloperanden, so wird dessen Quellkennung Q mit dem ersten Quellregister Q_AR und der ersten Quellkennung Q_A des Befehls verglichen, wie in der folgenden Pseudocodeauflistung dargestellt.

```
Prüfung_Quellkennung_ein_Quelloperand :=

WENN [VWR.QAR] ≠ 0 DANN
  WENN VW.Q([Quelle]) ≠ [VWR.QAR] DANN
    Generierung_Ausnahmefehler;
  ENDEWENN
ENDEWENN

WENN VW.QA([Befehl]) ≠ 0 DANN
  WENN VW.Q([Quelle]) ≠ VW.QA([Befehl]) DANN
    Generierung_Ausnahmefehler;
  ENDEWENN
ENDEWENN
```

Bei Befehlen mit zwei Quelloperanden werden die beiden Quellregister Q_AR und Q_BR, sowie die beiden Quellkennungen Q_A und Q_B des Befehls zur Prüfung der Quellkennungen der Quelloperanden herangezogen.

```
Prüfung_Quellkennung_zwei_Quelloperanden :=

WENN [VWR.QAR] ≠ 0 DANN
  WENN VW.Q([Quelle_1]) ≠ [VWR.QAR] DANN
    Generierung_Ausnahmefehler;
  ENDEWENN
ENDEWENN

WENN VW.QA([Befehl]) ≠ 0 DANN
  WENN VW.Q([Quelle_1]) ≠ VW.QA([Befehl]) DANN
    Generierung_Ausnahmefehler;
  ENDEWENN
ENDEWENN
```

```
WENN [VWR.Q_BR] ≠ 0 DANN
  WENN VW.Q([Quelle_2]) ≠ [VWR.Q_BR] DANN
    Generierung_Ausnahmefehler;
  ENDEWENN
ENDEWENN

WENN VW.Q_B([Befehl]) ≠ 0 DANN
  WENN VW.Q([Quelle_2]) ≠ VW.Q_B([Befehl]) DANN
    Generierung_Ausnahmefehler;
  ENDEWENN
ENDEWENN
```

In Senken kann die Hardware anhand der Zielkennung Z innerhalb der Verarbeitungswegkennung VW sicherstellen, dass die empfangenen Daten wirklich für die Nutzung durch die betroffene Einheit gedacht sind. Dazu werden die Inhalte mit dem Zielregister ZR bzw. der Zielkennung Z des Befehlsspeicherelements durch eine Konjunktion verknüpft. Das Ergebnis der UND-Verknüpfung muss dem Bitmuster des Registers bzw. der Kennung entsprechen. Ist dies nicht der Fall, so hat die Senke keine Rechte zur Nutzung der Daten und ein Ausnahmefehler wird generiert.

```
Prüfung_Zielkennung :=

WENN VW.Z([Quelle]) UND [VWR.ZR] ≠ [VWR.ZR] DANN
  Generierung_Ausnahmefehler;
ENDEWENN

WENN VW.Z([Quelle]) UND VW.Z([Befehl]) ≠ VW.Z([Befehl]) DANN
  Generierung_Ausnahmefehler;
ENDEWENN
```

4.3.7.3.2 Verzicht auf Teilprüfungen

Wie bei der Realisierung der Verarbeitungswegkennung VW der Datenspeicherelemente beschrieben, wird das Verarbeitungswegregister VWR als Alternative zur Verarbeitungswegkennung VW der Befehlsspeicherelemente bereitgestellt. Dies hat den Hintergrund, dass die Bitpositionen der Systemkomponenten oder Datenverarbeitungsblöcke ggf. erst in den detaillierten Spezifikationen der betroffenen Kom-

ponenten festgelegt werden können und zur Übersetzungszeit zur Einbettung in die Verarbeitungswegkennungen VW der Befehlsspeicherelemente nicht zur Verfügung stehen. Auch wenn beide Spezifikationswege – Register und Kennung innerhalb der Befehlsspeicherelemente – zeitgleich genutzt werden können, so wird nicht jede der Prüfungen immer erwünscht sein.

Auf die Prüfung der Quellkennung Q der Datenspeicherelemente kann verzichtet werden, indem die Quellregister Q_AR und Q_BR sowie die Quellkennungen Q_A und Q_B der Befehlsspeicherelemente auf Null gesetzt werden. Dies ist den Pseudocodeauflistungen im vorhergehenden Unterkapitel bereits zu entnehmen.

Auch die anderen Teilregister bzw. Teilkennungen der Befehlsspeicherelemente erlauben durch Setzen auf den Wert Null die Aussetzung der Prüfung der zugehörigen Teilkennung der Datenspeicherelemente, da die in den Auflistungen beschriebenen Bedingungen stets zutreffen, da die Gleichung

$$x \text{ UND } 0 = 0$$

für alle Bitmuster in den Teilkennungen – hier stellvertretend durch x symbolisiert – erfüllt ist.

4.3.7.4 Setzen der Verarbeitungswegkennung VW in Ergebnissen von Operationen

Die Ergebnisse der Verarbeitung von Operanden durch Operationen erhalten eine Kombination der Verarbeitungswegkennungen VW der zugrundeliegenden Operanden. Dabei werden die Quellenidentifikatoren der Operanden mit einer bitweisen Disjunktion verknüpft – also unter Anwendung einer ODER-Verknüpfung –, während die Verarbeitungswege $VW_{sys,lok}$ und die Zielidentifikatoren Z jeweils durch eine bitweise Konjunktion – also einer UND-Verknüpfung – für das Ergebnis ausgewählt werden.

Dahinter verbirgt sich die folgende Logik: Werden zwei Operanden verschiedener Quellen in einer Operation verarbeitet, z. B. die Signale zweier Temperatursensoren voneinander subtrahiert, so hat die entstehende Differenz beide Sensoren als Quellen, realisiert durch Verwendung einer ODER-Verknüpfung. Als Ziel der Differenz im Datenfluss dürfen nur diejenigen Senken als gültig markiert werden, die in beiden Operanden der Subtraktion bereits als gültige Senken markiert waren, weshalb eine UND-Verknüpfung der in den Quelldaten angegebenen Zielen erfolgt. Gleiches gilt

für die Verarbeitungswegkennung VW der Differenz, die sich dadurch ergibt, dass die Zwischenstationen, die in beiden Operanden als gültig markiert waren, auch auf die Differenz durch eine UND-Verknüpfung übertragen werden. Aus Platzgründen wird in der Pseudocodeauflistung Ergebnis als Erg abgekürzt.

```
Setzen_VW_in_Ergebnis :=

VW.Q([Erg]) := VW.Q([Quelle_1]) ODER VW.Q([Quelle_2]);
VW.VW_sys([Erg]) := VW.VW_sys([Quelle_1]) UND VW.VW_sys([Quelle_2]);
VW.VW_lok([Erg]) := VW.VW_lok([Quelle_1]) UND VW.VW_lok([Quelle_2]);
VW.Z([Erg]) := VW.Z([Quelle_1]) UND VW.Z([Quelle_2]);
```

4.3.7.5 Befehle zur Verwaltung der Verarbeitungswegkennung VW

Zur Verwaltung der Verarbeitungswegkennung mit ihren Bestandteilen Quellkennung Q, Systemteil VW_{sys} und Lokalteil VW_{lok} des Verarbeitungswegs und Zielkennung Z werden drei Befehle definiert. Bei allen Befehlen wird zunächst sichergestellt, dass die zu setzenden Teilkennungen den Wert Null aufweisen, also noch nicht gesetzt wurden. Dies dient der Verhinderung unerlaubter „Korrekturen“ der Inhalte der Verarbeitungswegkennung VW. Enthält eine der Teilkennungen des Zieldatenspeicherelements bereits gesetzte Bits, so wird ein Ausnahmefehler generiert.

Der erste Befehl Setze Gesamten Verarbeitungsweg SGVW dient dem Setzen aller vier Bestandteile der Verarbeitungswegkennung VW eines Datenspeicherelements durch eine Datenquelle. Dies ist nur dann möglich, wenn der Inhalt des Lokalteils VW_{lok} der Datenquelle bekannt ist, z. B. durch eine entsprechende Konfigurationsdatei. Die Teilkennungen der Verarbeitungswegkennung VW des durch E indizierten Zieldatenspeicherelements werden auf die durch A, B, C und D indizierten Werte gesetzt.

```
SGVW A, B, C, D, E :=

WENN VW.Q([E]) = VW.VW_sys([E]) = VW.VW_lok([E]) = VW.Z([E]) = 0 DANN
  VW.Q([E]) := W([A]);
  VW.VW_sys([E]) := W([B]);
  VW.VW_lok([E]) := W([C]);
  VW.Z([E]) := W([D]);
```

```
SONST
  Generierung_Ausnahmefehler;
ENDEWENN
```

Kennt eine Datenquelle den Lokalteil VW_{lok} nicht und kann dieser deshalb durch sie nicht gesetzt werden, so kann sie den Befehl Setze Verarbeitungsweg SVW nutzen. Dieser führt dieselben Schritte wie SGVW aus, setzt jedoch keinen neuen Wert für den Lokalteil VW_{lok} der Verarbeitungswegkennung VW, wodurch dieser – sichergestellt durch die anfängliche Überprüfung – den Wert Null beibehält. Somit können keine Datenverarbeitungsblöcke innerhalb der Datenverarbeitungseinheiten die Datenwerte verarbeiten, wenn vorab keine entsprechende Kennung gesetzt wird. Die Teilkennungen der Verarbeitungswegkennung VW des durch D indizierten Zieldatenspeicherelements werden analog zum Befehl Setze Gesamten Verarbeitungsweg SGVW mit den durch A, B und C indizierten Werten gefüllt.

```
SVW A, B, C, D :=

WENN VW.Q([D]) = VW.VW_sys([D]) = VW.VW_lok([D]) = VW.Z([D]) = 0 DANN
  VW.Q([D]) := W([A]);
  VW.VW_sys([D]) := W([B]);
  VW.Z([D]) := W([C]);
SONST
  Generierung_Ausnahmefehler;
ENDEWENN
```

Das Setzen des Lokalteils VW_{lok} der Verarbeitungswegkennung VW eines Datenspeicherelements erfolgt in Datenverarbeitungseinheiten durch Nutzung des Befehls Setze Verarbeitungsweg Lokal SVWL. Bei dessen Ausführung wird VW_{lok} mit dem durch A indizierten Bitmuster gefüllt.

```
SVWL A, B :=

WENN VW.VW_lok([B]) = 0 DANN
  VW.VW_lok([B]) := W([A]);
SONST
  Generierung_Ausnahmefehler;
ENDEWENN
```

Auch für die Verarbeitungswegkennung VW kann der in Kapitel 4.3.3.3 eingeführte Befehl Prüfe Einen Operanden PEO genutzt werden, der die in der folgenden Auflistung vorgestellten Prüfungen durchführt, ohne eine Änderung des Operanden hervorzurufen. Dabei deutet „...“ die Durchführung weiterer Prüfungen an, die in den zur jeweiligen Kennung gehörenden Kapiteln vorgestellt werden. Bezogen auf die Verarbeitungswegkennung werden die in der folgenden Auflistung gezeigten, umfangreichen Prüfungen der Kennungsinhalte durchgeführt.

```
PEO A :=

...
WENN [VWR.Q_AR] ≠ 0 DANN
  WENN VW.Q([A]) ≠ [VWR.Q_AR] DANN
    Generierung_Ausnahmefehler;
  ENDEWENN
ENDEWENN

WENN VW.Q_A([Befehl]) ≠ 0 DANN
  WENN VW.Q([A]) ≠ VW.Q_A([Befehl]) DANN
    Generierung_Ausnahmefehler;
  ENDEWENN
ENDEWENN

WENN VW.VW_sys([A]) UND [VWR.VW_sysR] ≠ [VWR.VW_sysR] DANN
  Generierung_Ausnahmefehler;
ENDEWENN

WENN VW.VW_sys([A]) UND VW.VW_sys([Befehl]) ≠ VW.VW_sys([Befehl]) DANN
  Generierung_Ausnahmefehler;
ENDEWENN

WENN VW.VW_lok([A]) UND [VWR.VW_lokR] ≠ [VWR.VW_lokR] DANN
  Generierung_Ausnahmefehler;
ENDEWENN

WENN VW.VW_lok([A]) UND VW.VW_lok([Befehl]) ≠ VW.VW_lok([Befehl]) DANN
  Generierung_Ausnahmefehler;
ENDEWENN
```

```
WENN VW.Z([A]) UND [VWR.ZR] ≠ [VWR.ZR] DANN
  Generierung_Ausnahmefehler;
ENDEWENN

WENN VW.Z([A]) UND VW.Z([Befehl]) ≠ VW.Z([Befehl]) DANN
  Generierung_Ausnahmefehler;
ENDEWENN
...
```

4.3.7.6 Veranschaulichung der Verarbeitungswegkennung

Zur Veranschaulichung der Anwendung der Verarbeitungswegkennung inklusive der Quell- und Zielkennung soll das einfache Beispiel in Abbildung 4.37 dienen.

Zunächst soll der Datenfluss oberflächlich erläutert werden. In einen technischen Prozess sind die drei Sensoren A bis C eingebracht, die in der Spezifikationsphase des Systems eindeutige Identifikatoren in Form von eindeutigen Bitmustern zugewiesen bekommen. Für Sensor A ist dies 001b, für Sensor B 010b und für Sensor C 100b. Die Sensoren erzeugen die Messergebnisse M_A bis M_C, die an die Datenverarbeitungseinheit DVE A weitergeleitet werden, die den eindeutigen Identifikator 01b trägt. Innerhalb der DVE A werden den Messergebnissen lokale Verarbeitungswegkennungen zugewiesen, wodurch die entsprechenden modifizierten Messergebnisse M_A' bis M_C' entstehen. Diese tragen daher die Information in sich, welche Datenverarbeitungsblöcke DV innerhalb von DVE A die Daten verarbeiten sollen. Zwei dieser Datenverarbeitungsblöcke sind im Beispiel dargestellt, DV A und DV B, mit den Identifikatoren 001b bzw. 010b. Durch die Verarbeitung entstehen die zwei Stellgrößen S_A und S_B, welche an die zwei Aktoren Aktor A bzw. Aktor B gesendet werden. Die Aktoren tragen dabei die Identifikatoren 001b bzw. 010b.

Der vorgesehene Datenfluss innerhalb des Systems soll nun detailliert in den Verarbeitungswegkennungen beschrieben werden, ebenso die verschiedenen Prüfungen, die die einzelnen Instanzen des Systems vornehmen.

4.3.7.6.1 Erzeugung der Prozessgrößen in den Sensoren

Die Sensoren versehen jeden der Messwerte M_A bis M_C mit einer Verarbeitungswegkennung, die auf einer hohen Ebene – ohne Detailwissen über den inneren Aufbau

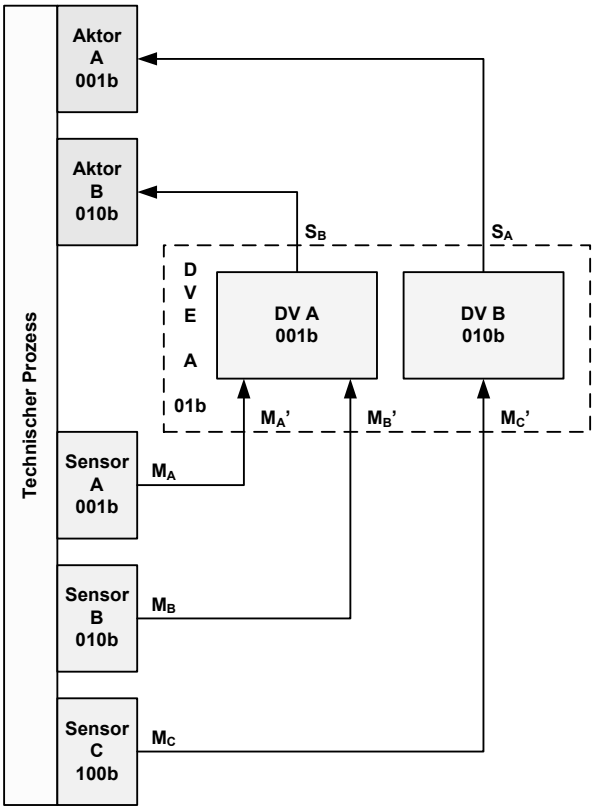


Abbildung 4.37: Einfaches Beispiel zur Veranschaulichung der Verarbeitungswegkennung

der beteiligten Datenstationen zu benötigen – den Weg der Daten durch das System bis hin zu den Aktoren festlegt. Die Darstellung dieser Kennung soll hier als Tupel der Form

$$VW := (Q,\ VW_{sys},\ VW_{lok},\ Z)$$

erfolgen, wobei Q die Quellkennung des jeweiligen Sensors enthält, VW_{sys} die die Daten verarbeitenden Einheiten auf hoher Ebene und VW_{lok} die Datenverarbeitungsblöcke innerhalb der DVE darstellen und Z das vorhergesehene Ziel durch dessen Zielkennung identifiziert. Für die Messgröße M_A, die durch Sensor A 001b gebildet wird, die durch die DVE A 01b verarbeitet werden und in die Stellgröße S_B münden und in dieser Form an Aktor B 010b gesendet werden soll, ergibt sich somit die VW-Kennung

$$M_A := (001b,01b,—,010b),$$

die vom Sensor entsprechend in M_A hinterlegt wird.

Für die beiden weiteren Prozessgrößen $M_{B,C}$ ergeben sich analog dazu

$$M_B := (010b,01b,—,010b),$$

bzw.

$$M_C := (100b,01b,—,001b).$$

4.3.7.6.2 Verarbeitung der Sensordaten in den Datenverarbeitungseinheiten

Die Prozessgrößen $M_{A,B,C}$ werden über eine Kommunikationsverbindung – z. B. einen Feldbus – an die Datenverarbeitungseinheit DVE A gesendet. Diese prüft anhand des globalen Teils der Verarbeitungswegkennung VW_{sys} der ankommenden Daten, ob diese durch sie verarbeitet werden dürfen. Für diese Prüfung kommen die den Systemteil der Verabreitungswegkennung betreffenden Inhalte des Verarbeitungswegregisters VWR und die Verarbeitungswegkennungen VW der die Daten verarbeitenden Befehle zum Einsatz. Es wird sichergestellt, dass die Gleichungen

$$VW_{sys}(M_x)\ UND\ [VWR.VW_{sys}R] = [VWR.VW_{sys}R] \text{ und}$$
$$VW_{sys}(M_x)\ UND\ VW.VW_{sys}([Befehl]) = VW.VW_{sys}([Befehl])$$

erfüllt sind, die Daten somit durch DVE A auch tatsächlich verarbeitet werden dürfen. Im vorliegenden Beispiel sollen $VWR.VW_{sys}R$ und $VW.VW_{sys}([Befehl])$ jeweils den Wert 01b aufweisen, wodurch sich für beide Gleichungen

$$VW_{sys}(M_x)\ UND\ 01b = 01b$$

ergibt und die Daten somit in der DVE A verarbeitet werden dürfen.

Die DVE fügt den Daten daraufhin den lokalen Teil VW_{lok} der Verarbeitungswegkennung hinzu, um den vorgesehenen Weg der Daten durch die DVE A über die in ihr enthaltenen Datenverarbeitungsblöcke DV A bzw. DV B zu beschreiben. Die Messwerte $M_{A,B}$ sollen dabei durch den Datenverarbeitungsblock DV A, der Messwert M_C durch DV B verarbeitet werden. So ergeben sich die um VW_{lok} erweiterten Messgrößen

M_A' := (001b,01b,001b,010b),
M_B' := (010b,01b,001b,010b) und
M_C' := (100b,01b,010b,001b).

Die Daten werden nun innerhalb der DVE A den entsprechenden Datenverarbeitungsblöcken DV A bzw. DV B zur Verarbeitung zur Verfügung gestellt. Diese prüfen nun ihrerseits, ob der lokale Teil VW_{lok} der Verarbeitungswegkennung eine Verarbeitung im jeweiligen Datenverarbeitungsblock gestattet. Wie auch bei der Prüfung des Systemteils der Daten werden hier die den Lokalteil der Verarbeitungswegkennung betreffenden Inhalte des Verarbeitungswegregisters VWR und der Verarbeitungswegkennung VW des die Daten verarbeitenden Befehls für die Prüfung der Erfüllung der Gleichungen

$VW_{lok}(M_x\text{'})\ UND\ [VWR.VW_{lok}R] = [VWR.VW_{lok}R]$ und
$VW_{lok}(M_x\text{'})\ UND\ VW.VW_{lok}([Befehl]) = VW.VW_{lok}([Befehl])$

herangezogen. Im gegebenen Beispiel soll kein Lokalteil im Verarbeitungswegregister VWR spezifiziert sein, weshalb das den Lokalteil betreffende Teilregister $VW_{lok}R$ den Wert 000b enthält. Die den Lokalteil des Verarbeitungswegs bestimmende Teilkennung VW_{lok} in der Verarbeitungswegkennung des die Daten verarbeitenden Befehls enthält für alle zum Datenverarbeitungsblock DV A gehörenden Befehle den Identifikator 001b und im Datenverarbeitungsblock DV B 010b.

Für M_A' und M_B' werden damit die Prüfungen

$VW_{lok}(M_x\text{'})\ UND\ 000b = 000b$ und
$VW_{lok}(M_x\text{'})\ UND\ 001b = 001b$,

für M_C' entsprechend

$VW_{lok}(M_x\text{'})\ UND\ 000b = 000b$ und
$VW_{lok}(M_x\text{'})\ UND\ 010b = 010b$

durchgeführt. Sind die Bedingungen erfüllt, so werden die Daten im jeweiligen Datenverarbeitungsblock verarbeitet und die Stellgrößen S_A und S_B für die Aktoren

Aktor A und Aktor B berechnet. Wird dabei nur eine Eingangsgröße verarbeitet, wie es bei der Erzeugung von S_A in DV B der Fall ist, in diesem Fall M_C', so werden Q, $VW_{sys,lok}$ und Z der Eingangsgröße einfach direkt in die VW-Kennung des Ergebnisses übertragen. Für die Stellgröße S_A ergibt sich somit

$$S_A := (Q(M_C'), VW_{sys}(M_C'), —, Z(M_C')) = (100b, 01b, —, 001b).$$

Werden mehrere Eingangsgrößen zu einem Ergebnis verarbeitet, wie im Fall von S_B, so werden die Quellkennungen Q der Eingangsgrößen ODER-verknüpft, während die Verarbeitungswegkennungen $VW_{sys,lok}$ und die Zielkennungen Z UND-verknüpft werden. Bei den Quellkennungen trägt das Ergebnis daher die Kennung aller Quellen, die an der Entstehung der zugrundeliegenden Messgrößen beteiligt waren. Die beiden anderen Kennungen, $VW_{sys,lok}$ und Z, tragen hingegen nur die Identifikatoren derjenigen Verarbeitungsinstanzen und Ziele in sich, die in allen Eingangsgrößen als gültig markiert waren. Dies resultiert im gegebenen Beispiel in der Verarbeitungswegkennung

$$\begin{aligned} S_B := &(Q(M_A') \text{ ODER } Q(M_B'), VW_{sys}(M_A') \text{ UND } VW_{sys}(M_B'), —, \\ &Z(M_A') \text{ UND } Z(M_B')) \\ = &(011b, 01b, —, 010b) \end{aligned}$$

für die Stellgröße S_B.

4.3.7.6.3 Verarbeitung der Stellgrößen in den Aktoren

Die berechneten Stellgrößen werden daraufhin über eine Kommunikationsverbindung an die jeweiligen Aktoren weitergeleitet. Diese stellen bei Erhalt der Daten anhand der Verarbeitungswegkennung sicher, dass

- die Stellgrößen auch wirklich für den jeweiligen Aktor bestimmt sind und
- die zugrundeliegenden Sensordaten von den erwarteten Quellen stammen.

Auch hier kommen die entsprechenden Inhalte des Verarbeitungswegregisters VWR und der Verarbeitungswegkennung VW der die Daten verarbeitenden Befehle zum Tragen. Nach dem Vorbild der bereits beschriebenen Prüfungen wird in den Aktoren die Einhaltung der Bedingungen

$Z(S_x)$ *UND* $[VWR.ZR] = [VWR.ZR]$ und
$Z(S_x)$ *UND* $VW.Z([Befehl]) = VW.Z([Befehl])$,

und für das konkrete Beispiel

$Z(S_B)$ *UND 010b = 010b* bzw.
$Z(S_A)$ *UND 001b = 001b*

für die Inhalte der Zielteilkennung der Daten sichergestellt. Für die erwarteten Inhalte der Quellteilkennungen ergeben sich analog dazu die durchzuführenden Prüfungen

$Q(S_x) = VWR.Q_{\{A,B\}}$, *wenn* $VWR.Q_{\{A,B\}} \neq 0$ und
$Q(S_x) = VW.Q_{\{A,B\}}$*([Befehl]), wenn* $VW.Q_{\{A,B\}}$*([Befehl])* $\neq 0$.

Für das hier beschriebene Beispiel sind die zu erfüllenden Bedingungen daher

$Q(S_A) = 011b$ und
$Q(S_B) = 100b$.

4.3.7.7 Spezifikation von Verarbeitungswegen in Hochsprachen

Zum Setzen der Verarbeitungswegkennung VW und ihrer Teilkennungen wird für die Hochsprache C die Einführung der intrinsischen Funktionen

`__set_processing_path(<Variablenname>,<Q>,<VW`$_{sys}$`>,<Z>)`,
`__set_whole_processing_path(<V.name>,<Q>,<VW`$_{sys}$`>,<VW`$_{lok}$`>,<Z>)` und
`__set_local_processing_path(<Variablenname>,<VW`$_{lok}$`>)`

vorgeschlagen, wobei die Bezeichner der Parameter der drei Funktionen den Namen der vier Teilkennungen der Verarbeitungswegkennung VW entsprechen.

Die Funktion `__set_processing_path()` wird vom Übersetzer in den Befehl Setze Verarbeitungsweg SVW überführt, um alle Teilkennungen bis auf den Lokalteil VW_{lok} der Verarbeitungswegkennung VW der Zielvariable zu setzen. Die Funktion `__set_whole_processing_path()` dient dem Setzen aller Teilkennungen und wird dementsprechend in den Befehl Setze Gesamten Verarbeitungsweg SGVW übersetzt. Die Nutzung der Funktion `__set_local_processing_path()` ermöglicht es, nur den Lokalteil VW_{lok} der Verarbeitungswegkennung VW der Zielvariable zu setzen und wird durch den Befehl Setze Verarbeitungsweg Lokal SVWL realisiert.

Der Einsatz der drei Funktionen im Quellcode wird in der folgenden Auflistung gezeigt, wobei `BM` aus Platzgründen als Bezeichnung für das jeweilige Bitmuster genutzt wird.

```
__set_processing_path(x, Q_BM, VW_SYS_BM, Z_BM);
__set_whole_processing_path(y, Q_BM, VW_SYS_BM, VW_LOK_BM, Z_BM);
__set_local_processing_path(z, VW_LOK_BM);
```

4.3.7.8 Evaluation der Verarbeitungswegkennung VW

Von den 20 in Kapitel 2.4 vorgestellten Fehler- und Angriffsarten lassen sich die in Tabelle 4.8 gezeigten Arten erkennen.

Tabelle 4.8: Fehlererkennung durch die VW-Kennung

Fehlerart	**Erkennbarkeit**
Inkompatible Datentypen	nein
Inkompatible Einheiten	nein
Wertebereichsunter- bzw. -überschreitung	nein
Genauigkeitsproblem	nein
Falsche Operandenauswahl	(ja)
Falsche Operatorauswahl	nein
Fehlerhaftes Operationsergebnis	nein
Fristüberschreitung	nein
Zyklusunterschreitung	nein
Zyklusüberschreitung	nein
Verlorengegangene Datenaktualisierung	nein
Synchronisationsfehler oder unvollständige Datenübertragung	nein
Pufferunter- oder -überläufe	begrenzt
Fehlerhafter Datenfluss (falsche Adressaten, ...)	ja
Duplizierte Daten	nein
Durch Fehler oder Störungen verfälschte Daten	begrenzt
Fehlerhafter Datenzugriff (fehlende Zugriffsrechte)	nein
Nutzung nicht initialisierter Daten	(ja)
Angriffsart	
Gezielt verfälschte Daten	nein
Wiedereinspielungsattacke	nein

Die Verwendung falscher Operanden in einer Operation kann durch die Verarbeitungswegkennung dann aufgedeckt werden, wenn in deren Verarbeitungswegken-

nung VW die aktuelle Datenverarbeitungseinheit oder ein Programmmodul innerhalb einer solchen Einheit nicht als gültiger Teilnehmer der Verarbeitung der Daten aufgeführt ist. Auf die gleiche Weise können in begrenztem Maß Pufferunter- und -überläufe erkannt werden, wenn unter bzw. über die Grenzen eines Puffers hinaus gelesen wird. Beim entsprechenden fehlerhaften Schreibzugriffen kann der Fehler frühestens beim nächsten lesenden Zugriff aufgedeckt werden, wenn die genannten Bedingungen erfüllt sind. Fehlerhafter Datenfluss durch Adressierungsfehler, also z. B. das Versenden von Daten an falsche Adressaten oder das Weiterreichen von Daten an die falschen Programmmodule kann durch die Verarbeitungswegkennung sicher erkannt werden. Die Nutzung nicht initialisierter Daten kann nur dann erkannt werden, wenn Datenspeicherelementen mit entsprechenden ungültigen Inhalten z. B. eine leere Verarbeitungswegkennung zugewiesen wird, wodurch keine System- oder Programmeinheit lesend auf die Daten zugreifen kann, ohne einen Ausnahmefehler zu verursachen.

4.3.8 Zeitschritt

Werden zur Erfassung von Messgrößen eines technischen Prozesses intelligente Sensoren eingesetzt, so werden diese die zu messende physikalische Größe in bestimmten Zeitabständen erfassen, umwandeln und zur Übertragung an eine Datenverarbeitungseinheit bereitstellen bzw. selbst übertragen. Bei der Nutzung konventioneller Sensoren, die eine Messgröße analog an eine Datenverarbeitungseinheit übermitteln, wird diese Größe ebenfalls zeitdiskret in ein digitales Signal umgewandelt. Es liegt daher in beiden Fällen nahe, den Daten ihren diskreten Entstehungszeitpunkt als Dateneigenschaft hinzuzufügen.

Doch nicht nur Messwerte können mit der Eigenschaft ihres diskreten Entstehungszeitpunkts versehen werden. Ebenso sind Variablen in Schleifen Änderungen zu diskreten Zeitpunkten unterworfen. Bei der Übertragung von Datenfeldern kann anhand des Zeitschritts aller Feldelemente sichergestellt werden, dass alle Elemente aktualisiert wurden. Werden Daten nebenläufig verwendet, so besteht die Gefahr von Inkonsistenzen bei konkurrierendem Zugriff. Auch hier kann eine Zeitschrittangabe helfen, zu erkennen, ob die betroffenen Daten inkonsistente Stände aufweisen. Es gilt also neben der reinen Angabe von Zeitschritten auch die temporalen Zusammenhänge zwischen den Zeitschritten von Operanden zu beschreiben.

4.3.8.1 Formale Notation der Zeitschrittkennung

Um die diskreten Entstehungszeitschritte von Operanden und die temporalen Zusammenhänge zwischen den Zeitschritten von Operanden verständlich darstellen zu können, ist eine formale Notation beider Merkmale notwendig. Diese wird hier vorgestellt, wobei die Zeitschrittangabe von Operanden und die Spezifikation der temporalen Zusammenhänge der Operanden von Operationen getrennt betrachtet werden.

4.3.8.1.1 Angabe des Zeitschritts von Operanden

Bei Operanden wird der erwartete Zeitschritt des Operanden unterhalb von diesem angegeben. Doch nicht alle Operanden besitzen einen diskreten Entstehungszeitpunkt, so z. B. Konstanten. Soll ein Operand keinen Zeitschritt aufweisen, dann wird dies durch

$$\underset{t:*}{K}$$

angegeben, wodurch klar wird, dass hier explizit auf die Spezifikation eines Zeitschritts verzichtet wurde. Würde die Angabe einfach weggelassen, könnte auch ein Spezifikationsfehler vorliegen, die Angabe also schlicht vergessen worden sein.

Besitzt ein Operand eine Zeitschrittangabe, wird diese durch

$$\underset{t:n}{x(n)}$$

beschrieben. Es handelt sich dabei um eine relative Angabe, hier n, die natürlich erst durch die Relation zu den Zeitschritten weiterer Operanden mit Zeitschrittkennung an Bedeutung gewinnt. Dies wird anhand eines späteren Beispiels klarer.

4.3.8.1.2 Angabe der zu prüfenden temporalen Beziehung bei Operatoren

Bei Operatoren wird die zu prüfende temporale Beziehung der beteiligten Operanden oberhalb der Operanden angegeben. Um auch bei den Operatoren durch eine explizite Angabe auf die Prüfung der temporalen Beziehung verzichten zu können, wird

$$\underset{Op}{\overset{\Delta t:*}{}}$$

definiert, um dies zu verdeutlichen.

Soll die temporale Beziehung der Operanden zueinander jedoch geprüft werden, wird die Differenz der erwarteten Zeitschritte oberhalb des Operators angegeben. Dabei wird in dieser Arbeit festgelegt, dass dabei stets der Zeitschritt des rechts des Operators stehenden Operanden von dem des links stehenden abzuziehen ist. Die Angabe der Prüfungsbedingung wird also durch

$$\overset{\Delta t:1}{Op}$$

beschrieben. Im angegebenen Beispiel soll die Differenz zwischen den Zeitschritten des linken und rechten Operanden also 1 sein.

Es kann notwendig sein, den Zeitschritt des Ergebnisses Operation manuell zu erhöhen, z. B. falls der zweite Operand eine Konstante ohne eigenen Zeitschritt ist. Dies ist z. B. bei Zählervariablen der Fall, auf die eine Konstante ohne Zeitschrittangabe addiert wird. Um hier das Ergebnis mit einem aktualisierten Zeitschritt zu versehen, wird bei einer solchen Operation zunächst – wie gehabt – der jüngere Zeitschritt der beiden Operanden ausgewählt, dieser dann jedoch um eins erhöht und dann dem Ergebnis der Operation zugewiesen. In der formalen Notation wird diese Erhöhung mit einem + gekennzeichnet. Ein Befehl, der keine Prüfung der temporalen Beziehung der Operanden durchführen, jedoch den Zeitschritt des Ergebnisses um Eins erhöhen soll, wird damit mit

$$\overset{\Delta t:*,+1}{Op}$$

und ein Befehl, der eine Differenz zwischen den Zeitschritten beider Operanden prüfen soll durch

$$\overset{\Delta t:1,+1}{Op}$$

beschrieben.

4.3.8.1.3 Beispiele für die formale Notation der Zeitschrittkennung

Zur Verdeutlichung der vollständigen Notation werden nun zwei Beispiele gezeigt. Die Addition zweier Datenwerte A und B, die temporal einen Zeitschritt auseinanderliegen sollen, wird nach der vorgestellten Notation durch

$$\underset{t:n}{A} \overset{\Delta t:1}{+} \underset{t:n-1}{B}$$

beschrieben. Der Wert von B ist also einen diskreten Zeitschritt älter als der Wert von A und die temporale Beziehung beider Operanden zueinander wird über dem Operator + angegeben.

Wird das Beispiel um eine Zuweisung des Ergebnisses zu C ergänzt, welches nach der vorangegangenen und vor der zu erfolgenden Zuweisung im fehlerfreien Fall den Zeitschritt $n-1$ haben muss, vervollständigt sich der Term zu

$$\underset{t:n-1}{C} \overset{\Delta t:-1}{=} \underset{t:n}{A} \overset{\Delta t:1}{+} \underset{t:n-1}{B}$$

um die Angabe des erwarteten Zeitschritts von C und der Angabe der zu prüfenden temporalen Beziehung zwischen C und dem Ergebnis der Addition.

Ein weiteres Beispiel soll den Einsatz der Erhöhung des Zeitschritts des jüngeren Operanden zeigen, wie es z. B. bei der Überwachung des Datenflusses von Zählervariablen notwendig ist.

$$\underset{t:n-1}{A} \overset{\Delta t:-1}{=} \underset{t:n-1}{A} \overset{\Delta t:*,+1}{+} \underset{t:*}{K}$$

Auf die Zählervariable A soll die Konstante K addiert werden, die als Konstante keinen Zeitschritt aufweist und daher die Zeitschrittangabe $t:*$ trägt. Entsprechend muss bei der Addition keine temporale Beziehung zwischen A und K geprüft werden, was durch $\Delta t:*$ spezifiziert wird. Allerdings soll der Zeitschritt des Additionsergebnisses um eins erhöht werden, daher trägt die Addition die Angabe $\Delta t:*,+1$. Die Variable A hat auf beiden Seiten der Gleichung den Zeitschritt $n-1$. Nach der Addition von A und K wird dem Ergebnis der Zeitschritt $n-1+1$, also n zugewiesen. Bei der anschließenden Zuweisung des Ergebnisses zu A wird die Differenz zwischen dem bisherigen Zeitschritt von A und dem Ergebnis der Addition gebildet und verifiziert, dass diese -1 ist.

4.3.8.2 Realisierung der Zeitschrittkennung ZS

Eine Datenquelle, z. B. ein intelligenter Sensor, versieht die von ihr generierten zeitdiskreten Datenwerte mit einem aufsteigenden Zeitschritt, der im Datenspeicherelement in der Zeitschrittkennung ZS abgelegt wird, wie in Abbildung 4.38 dargestellt.

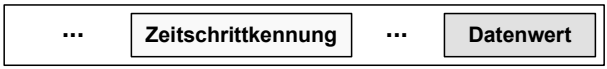


Abbildung 4.38: Datenspeicherelement mit Zeitschrittkennung ZS und Datenwert W

Da für die Angabe des Zeitschritts in der Praxis ein Kennungsfeld mit einer festgelegten Bitbreite verwendet werden wird, ist die Anzahl der darstellbaren Zeitschrittwerte endlich. Darum werden die Zeitschritte als Restklassenring $\mathbb{Z}/2^n\mathbb{Z}$ dargestellt, wobei n die Anzahl der für die Zeitschrittkennung vorgesehenen Bits angibt. Nicht immer werden Operanden einen Zeitschritt besitzen. So werden z. B. Konstanten keinen Zeitschritt besitzen, aber es kann ebenso vorkommen, dass eine Quelle einem Datenspeicherelement keinen Zeitschritt zuweist oder zuweisen kann. Daher wird ein weiteres Bit innerhalb der Zeitschrittkennung benötigt, das als Präsenzbit P genutzt wird. Dieses zeigt an, ob ein auswertbarer Zeitschritt in der Zeitschrittkennung hinterlegt ist, wobei der Wert Null angibt, dass kein Zeitschritt vorhanden ist, während der Wert Eins das Vorhandensein eines auswertbaren Zeitschritts spezifiziert.

Damit ergibt sich der in Abbildung 4.39 gezeigt Aufbau der Zeitschrittkennung eines Datenspeicherelements.

Abbildung 4.39: Aufbau der Zeitschrittkennung ZS bei Datenspeicherelementen

Der Befehlssatz der Datenspezifikationsarchitektur wird so angepasst, dass jeder Befehl ebenfalls eine Zeitschrittkennung erhält. Diese kann eine Vorgabe enthal-

ten, in welcher temporalen Beziehung die vom Befehl zu verarbeitenden Operanden zueinander stehen sollen, wie in Abbildung 4.40 gezeigt.

Abbildung 4.40: Befehlsspeicherelement mit Zeitschrittkennung ZS

Werden in einer Datenspezifikationsarchitektur Dreiadressbefehle eingesetzt, wie dies z. B. bei ISMA [125] der Fall ist, so können innerhalb der Zeitschrittkennung ZS des Befehls auch zwei temporale Beziehungen zu spezifizieren sein: die zwischen den Operanden der Operation und die zwischen dem Ergebnis der Operation und dem Zieldatenspeicherelement, in das das Ergebnis der Operation im Zuge der Ausführung der Wertzuweisung gespeichert werden soll. Da es möglich sein soll, explizit auf die Prüfung der temporalen Beziehungen der Operanden zu verzichten, werden auch in der Zeitschrittkennung ZS der Befehle ein bzw. zwei Präsenzbits P genutzt, das angibt bzw. die angeben, ob die Zeitschrittkennung zu prüfende temporale Beziehungen enthält. Der Wert Null gibt bei den Präsenzbits P an, dass keine Zeitschrittdifferenz zu prüfen ist, während der Wert Eins eine Prüfung anfordert. Weiterhin enthält die Zeitschrittkennung ZS in den Befehlen die Teilkennung +1, die angibt, ob der Zeitschritt des Ergebnisses der Operation um Eins erhöht werden soll. Daher ergibt sich der in Abbildung 4.41 dargestellte Aufbau der Zeitschrittkennung ZS bei Befehlen.

Abbildung 4.41: Aufbau der Zeitschrittkennung ZS bei Befehlsspeicherelementen

4.3.8.3 Prüfung der temporalen Beziehungen von Operanden anhand der Zeitschrittkennung ZS

Ein Befehl mit zwei Operanden enthält in seiner Zeitschrittkennung ZS eine zu verifizierende temporale Beziehung Δt der Zeitschritte der beiden Operanden, sowie

ein Präsenzbit P, welches angibt, ob eine solche Prüfung durchgeführt werden soll. Die folgende Auflistung zeigt, welche Prüfungen die Hardware bei einem Befehl mit zwei Operanden anhand der Zeitschrittkennungen ZS durchführt. Wenn das Präsenzbit P des Befehls eine zu prüfende Zeitschrittdifferenz Δt festlegt, dann wird die Differenz der Zeitschritte der Operanden gebildet, sofern beide Operanden einen Zeitschritt besitzen. Andernfalls gilt die Bedingung als erfüllt. Aus Platzgründen werden in den folgenden Pseudocodeauflistungen Quelle_1 als Q_1, Quelle_2 als Q_2 und Ziel als Z abgekürzt.

```
Prüfung_Zeitschrittdifferenz_2_Operanden :=

WENN ZS.P([Befehl]) = 1 DANN
  WENN ZS.P([Q_1]) = ZS.P([Q_2]) = 1 DANN
    WENN ZS.ZS([Q_1]) - ZS.ZS([Q_2]) ≠ ZS.Δt([Befehl]) DANN
      Generierung_Ausnahmefehler;
    ENDEWENN
  ENDEWENN
ENDEWENN
```

Bei Befehlen mit drei Operanden werden in der Zeitschrittkennung ZS des Befehls zwei zu prüfende temporale Beziehungen Δt und zwei Präsenzbits P angegeben. Die durchgeführten Prüfungen werden in der folgenden Auflistung gezeigt. Dabei bezeichnet „Ergebnis“ das innerhalb der arithmetisch-logischen Einheit entstandene Ergebnis der durchgeführten Operation. Die Ermittlung des Zeitschritts des Ergebnisses wird im folgenden Unterkapitel erläutert.

```
Prüfung_Zeitschrittdifferenz_3_Operanden :=

WENN ZS.P_Op([Befehl]) = 1 DANN
  WENN ZS.P([Q_1]) = ZS.P([Q_2]) = 1 DANN
    WENN ZS.ZS([Q_1]) - ZS.ZS([Q_2]) ≠ ZS.Δt_Op([Befehl]) DANN
      Generierung_Ausnahmefehler;
    ENDEWENN
  ENDEWENN
ENDEWENN
```

```
WENN ZS.P_zuw([Befehl]) = 1 DANN
  WENN ZS.P([Z]) = ZS.P([Ergebnis]) = 1 DANN
    WENN ZS.ZS([Z]) - ZS.ZS([Ergebnis]) ≠ ZS.Δt_zuw([Befehl]) DANN
      Generierung_Ausnahmefehler;
    ENDEWENN
  ENDEWENN
ENDEWENN
```

4.3.8.4 Setzen der Zeitschrittkennung ZS in Ergebnissen von Operationen

Die Hardware bildet bei der Ausführung von Operationen die vorzeichenbehaftete Differenz der beiden vorzeichenlosen Zeitschrittangaben der Operanden, wenn beide Operanden eine Zeitschrittkennung besitzen. Dabei wird der Inhalt der Zeitschrittkennung ZS des zweiten Operanden von der des ersten Operanden abgezogen und bewertet. Ist die Differenz positiv, dann geht die Datenverarbeitungseinheit davon aus, dass der erste Operand die höhere und damit jüngere Zeitschrittkennung aufweist und überträgt diese in die Zeitschrittkennung ZS des Ergebnisses. Ist die Differenz negativ, wird entsprechend die Zeitschrittkennung ZS des zweiten Operanden für das Ergebnis ausgewählt.

Diese Vorgehensweise birgt ein Risiko: Überschreitet die Differenz der beiden Zeitschrittkennungen ZS der beiden Operanden den Wertebereich der vorzeichenbehafteten Darstellung der Zeitschrittkennungen, so wählt die Hardware fälschlicherweise die Zeitschrittkennung des älteren Operanden aus und überträgt diese in das Ergebnis. Die Auswirkungen eines solchen Fehlers sind – bezogen auf die Sicherheit des Gesamtsystems – gering, da die fehlerhafte Auswahl des Zeitschritts bei der nächsten Prüfung der temporalen Beziehungen aufgedeckt wird, frühestens bei einer nachfolgenden Zuweisung des Ergebnisses. Allerdings hat dieser Fehler natürlich entsprechend starke Auswirkungen auf die Verfügbarkeit des Systems. Es gilt also, die Zeitschrittkennung in der Hardware mit genügender Bitbreite zu realisieren, damit die beschriebenen Überläufe und die damit verbundenen Probleme nicht auftreten.

Ist an der Operation ein Operand ohne Zeitschrittkennung beteiligt, so wird dem Ergebnis der Operation automatisch der Zeitschritt des Operanden mit Zeitschrittkennung zugewiesen. Besitzt keiner der beiden Operanden eine Zeitschrittangabe, so erhält auch das Ergebnis keine Zeitschrittangabe. Ist in der Zeitschrittkennung

die Vorgabe gesetzt, den Zeitschritt des Ergebnisses um Eins zu erhöhen, dargestellt durch die gesetzte Teilkennung +1, dann wird der dem Ergebnis zugewiesene Zeitschritt um Eins erhöht, sofern das Ergebnis einen Zeitschritt erhält.

```
Setzen_ZS_in_Ergebnis :=

WENN ZS.P([Quelle_1]) = ZS.P([Quelle_2]) = 1 DANN
  WENN ZS.ZS([Quelle_1]) - ZS.ZS([Quelle_2]) ≥ 0 DANN
    ZS.P([Ergebnis]) := 1;
    ZS.ZS([Ergebnis]) := ZS.ZS([Quelle_1]);
  SONST
    ZS.P([Ergebnis]) := 1;
    ZS.ZS([Ergebnis]) := ZS.ZS([Quelle_2]);
  ENDEWENN
SONST WENN ZS.P([Quelle_1]) = 1 DANN
  ZS.P([Ergebnis]) := 1;
  ZS.ZS([Ergebnis]) := ZS.ZS([Quelle_1]);
SONST WENN ZS.P([Quelle_2]) = 1 DANN
  ZS.P([Ergebnis]) := 1;
  ZS.ZS([Ergebnis]) := ZS.ZS([Quelle_2]);
SONST
  ZS.P([Ergebnis]) := 0;
ENDEWENN

WENN ZS.P([Ergebnis]) = 1 ∧ ZS.+1([Befehl]) = 1 DANN
  ZS.ZS([Ergebnis]) := ZS.ZS([Ergebnis]) + 1;
ENDEWENN
```

4.3.8.5 Befehle zur Verwaltung der Zeitschrittkennung ZS

Zum Setzen der Inhalte der Zeitschrittkennung ZS eines Datenspeicherelements wird für die Datenquellen der Befehl Setze Zeitschritt SZS eingeführt, mit dessen Hilfe der Zeitschritt des durch C indizierten Speicherelements auf den durch A indizierten Wert gesetzt werden kann, sofern dies gewünscht ist. Mit dem durch B indizierten Operanden kann bestimmt werden, ob ein Zeitschritt zu setzen ist oder ob das Präsenzbit P der Zeitschrittkennung ZS auf Null gesetzt werden soll, um das Fehlen eines Zeitschritts zu markieren.

```
SZS A, B, C :=

WENN W([B]) ≠ 0 DANN
  ZS.P([C]) := 1;
  ZS.ZS([C]) := W([A]);
SONST
  ZS.P([C]) := 0;
ENDEWENN
```

Um eine ähnliche Funktionalität wie ADI bzw. SSM des Oracle SPARC M7 Prozessors bieten zu können (siehe Kapitel 3.8), wird der Befehl Prüfe Zeitschritt Absolut PZSA definiert, mit dessen Hilfe der Zeitschritt innerhalb der Zeitschrittkennung des durch A indizierten Operanden mit dem durch B indizierten absoluten Zeitschrittwert verglichen wird. Dabei ist zu beachten, dass der Befehl keinen herkömmlichen Vergleichsbefehl darstellt, dessen Ergebnis durch eine bedingte Sprunganweisung ausgewertet werden kann. Stattdessen wird bei Nichtübereinstimmung ein Ausnahmefehler generiert, der zum Programmabbruch führt.

```
PZSA A, B :=

WENN ZS.P([A]) ≠ 0 DANN
  WENN ZS.ZS([A]) ≠ W([B]) DANN
    Generierung_Ausnahmefehler;
  ENDEWENN
SONST
  Generierung_Ausnahmefehler;
ENDEWENN
```

Zur Prüfung einer relativen temporalen Beziehung zweier Operanden kann der in Kapitel 4.3.4.5 eingeführte Befehl Prüfe Zwei Operanden PZO genutzt werden. Neben weiteren Prüfungen – in der folgenden Auflistung durch „...“ angedeutet – prüft der Befehl die Differenz der Zeitschritte der beiden durch A und B indizierten Operanden anhand der Inhalte der Zeitschrittkennung ZS des Befehls, ohne dabei einen der Operanden zu verändern.

```
PZO A, B :=

...

WENN ZS.P([Befehl]) = 1 DANN
  WENN ZS.P([A]) = ZS.P([B]) = 1 DANN
    WENN ZS.ZS([A]) - ZS.ZS([B]) ≠ ZS.Δt([Befehl]) DANN
      Generierung_Ausnahmefehler;
    ENDEWENN
  ENDEWENN
ENDEWENN

...
```

4.3.8.6 Beispiel 1: Digitale Filter

In Abbildung 4.42 wird ein FIR-Filter 1. Ordnung in kanonischer Form dargestellt. Der aktuelle Abtastwert $x(n)$ wird links oben eingespeist, mit dem Filterkoeffizienten A_0 multipliziert. Darauf wird der mit dem Filterkoeffizienten A_1 multiplizierte vorherige Abtastwert $x(n-1)$ addiert, der durch Verwendung des Verzögerungsglieds z^{-1} um einen Takt verzögert wurde. In der Praxis wird ein Verzögerungsglied einfach durch Zwischenspeicherung eines Abtastwerts in einer Variablen für die Verwendung im darauffolgenden Verarbeitungsschritt des Nachfolgeabtastwerts realisiert.

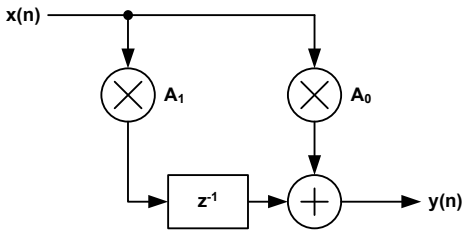


Abbildung 4.42: FIR-Filter 1. Ordnung in kanonischer Form

Die durchgeführte Berechnung lässt sich auch in Form der in Gleichung 4.1 gezeigten Differentialgleichung darstellen.

$$y(n) = A_0 \cdot x(n) + A_1 \cdot x(n-1) \tag{4.1}$$

Ein solches Filter stellt einen typischen von vielen möglichen Anwendungsfällen in der Verarbeitung von Abtastwerten dar. Da bei der zyklischen Berechnung Abtast- und Ergebniswerte mit unterschiedlichen Entstehungszeitpunkten vorliegen, lässt sich das Verfahren der Zeitschrittkennungen zur Datenflussüberwachung hier anschaulich einsetzen, wozu Gleichung 4.1 entsprechend mit Zeitschrittkennungen versehen wird und somit Gleichung 4.2 entsteht.

$$\underset{t:n-1}{y(n)} \overset{\Delta t:-1}{=} \underset{t:*}{A_0} \overset{\Delta t:*}{\cdot} \underset{t:n}{x(n)} \overset{\Delta t:1}{+} \underset{t:*}{A_1} \overset{\Delta t:*}{\cdot} \underset{t:n-1}{x(n-1)} \tag{4.2}$$

Der Abtastwert $x(n)$ soll spezifikationsgemäß immer den Zeitschritt n aufweisen, während sein Vorgänger, erzeugt durch die Verzögerung durch das Verzögerungsglied z^{-1}, den Zeitschritt $(n-1)$ haben soll. Dieser zeitliche Zusammenhang soll durch die Hardware überwacht werden. Die Filterkoeffizienten A_0 und A_1 sind Konstanten, die keinen Zeitschritt besitzen und darum in der Gleichung mit $t:*$ versehen sind. Bei den Multiplikationen des aktuellen Abtastwerts und seines Vorgängers mit den Filterkoeffizienten können deshalb keine temporalen Beziehungen zwischen den Operanden der Multiplikationen geprüft werden, was durch die Angabe $\Delta t:*$ dargestellt wird. Die skalierten Abtastwerte werden dann addiert, wobei die beiden Teilergebnisse den Zeitschritt des Abtastwerts besitzen und somit einen Zeitschritt auseinanderliegen sollen. Für die Addition gilt somit $\Delta t:1$. Die abschließende Zuweisung weist das Ergebnis, das den jüngeren Zeitschritt der beiden skalierten Abtastwerte erhalten hat, also $t:n$, der Zielvariablen $y(n)$ zu, die zu diesem Zeitpunkt noch den Zeitschritt des letzten Berechnungsschritts $t:n-1$ aufweist. Bei der Zuweisung wird also eine Zeitschrittdifferenz von $\Delta t:-1$ erwartet.

4.3.8.7 Beispiel 2: Gleitende Summe

Als zweites Beispiel soll die Berechnung einer gleitenden Summe vorgestellt werden, bei der die Summe von m aufeinanderfolgenden Abtastwerten berechnet werden soll, wie in Abbildung 4.43 dargestellt.

Der naive Ansatz zur Berechnung ist die komplette Neuberechnung der Summe $y(n)$ über die entsprechende Anzahl gespeicherter Abtastwerte, gezeigt in Gleichung 4.3,

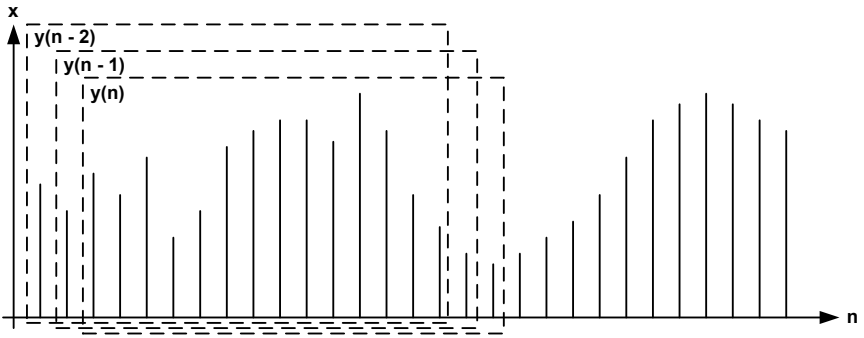


Abbildung 4.43: Gleitende Summe über diskrete Abtastwerte

wobei $x(n)$ den aktuellen Abtastwert und d die Anzahl der in der Summe zu erfassenden Abtastwerte darstellen. Die Berechnung lässt sich erheblich effizienter gestalten, indem von der aktuellen Summe der jeweils letzte Abtastwert $x(n-d)$ abgezogen und der neueste, also $x(n)$, aufaddiert wird, wodurch sich die Summe nach Gleichung 4.4 berechnet.

$$y(n) = \sum_{i=n-d+1}^{n} x(i) \tag{4.3}$$

$$y(n) = y(n-1) - x(n-d) + x(n) \tag{4.4}$$

Unter Verwendung der Zeitschrittkennungen lässt sich die Berechnung der Summe nach Gleichung 4.4 sehr effizient durch die Hardware überwachen. Dazu werden die notwendigen temporalen Bedingungen in den arithmetischen Operationen explizit angegeben. Je nachdem, in welcher Reihenfolge der Übersetzer die Berechnungen durchführt, sind die temporalen Bedingungen unterschiedlich, wie in den Gleichungen 4.5 und 4.6 zu sehen.

$$\underset{t:n-1}{y(n)} \overset{\Delta t:-1}{=} [\underset{t:n-1}{y(n-1)} \overset{\Delta t:d-1}{-} \underset{t:n-d}{x(n-d)}] \overset{\Delta t:-1}{+} \underset{t:n}{x(n)} \tag{4.5}$$

$$\underset{t:n-1}{y(n)} \overset{\Delta t:-1}{=} \underset{t:n-1}{y(n-1)} \overset{\Delta t:-1}{+} [-\underset{t:n-d}{x(n-d)} \overset{\Delta t:-d}{+} \underset{t:n}{x(n)}] \tag{4.6}$$

Im ersten Fall – siehe Gleichung 4.5 – wird zunächst der älteste Abtastwert $x(n-d)$ von der Summe abgezogen, wodurch sich eine temporale Distanz von $d-1$ zwischen

den beiden Operanden ergibt. Anschließend wird der aktuelle Abtastwert $x(n)$ aufaddiert, wobei dann die beiden Summanden die temporale Distanz von -1 besitzen. Die abschließende Zuweisung der neuen Summe auf die vorhergehende hat erwartungsgemäß einen temporalen Abstand von -1.

Etwas anders sehen die zu prüfenden temporalen Beziehungen im zweiten Fall – siehe Gleichung 4.6 – aus: zunächst wird die Differenz des ältesten Abtastwerts $x(n-d)$ und des aktuellen Abtastwerts $x(n)$ gebildet, wobei die beiden Operanden einen zeitlichen Abstand von $-d$ besitzen müssen. Hier ist Vorsicht geboten: die Differenz kann auch negative Werte annehmen, weshalb Δt vorzeichenbehaftet ist. Die Differenz beider Abtastwerte wird anschließend auf die bisherige Summe addiert, wobei diese ein Abtastintervall älter als die Differenz sein soll. Es muss also Δt gleich -1 gelten. Gleiches gilt für die anschließende Zuweisung des Ergebnisses.

4.3.8.8 Beispiel 3: Regelkreis

Im dritten Beispiel wird der in Abbildung 4.44 gezeigte Regelkreis mit Hilfe von Zeitschrittkennungen mit der Möglichkeit zur Erkennung von entsprechenden Datenflussfehlern ausgestattet.

Die Variable x bildet nach [124] die zu regelnde Größe ab, z. B. die Temperatur im Kessel eines chemischen Prozesses. Die Führungsgröße w gibt vor, auf welchen Wert x geregelt werden soll. Dabei wird die Größe x durch einen Sensor erfasst und der Regeleinrichtung zur Verfügung gestellt. Durch Subtraktion wird aus x und w die Regeldifferenz e bestimmt, die durch den Regler verarbeitet und in die Reglerausgangsgröße y_R überführt wird. Ein Stellglied – ein Aktor, also im Beispiel eine Heizung bzw. Kühlung – gibt die Stellgröße y aus und beeinflusst damit die Regelstrecke. Auf die Regelstrecke wirken eine oder mehrere Störgrößen, deren Einfluss durch die Regeleinrichtung ausgeregelt werden muss.

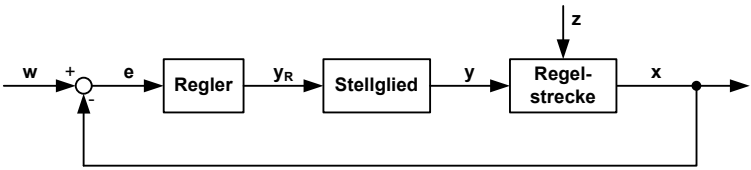


Abbildung 4.44: Ein typischer Regelkreis (nach [124])

Die Regeldifferenz $e(n)$ wird bestimmt, indem der aktuelle Sensorwert $x(n)$ von der aktuellen Führungsgröße $w(n)$ abgezogen wird, wie in Gleichung 4.7 gezeigt. Durch das Einfügen der Zeitschrittkennungen ergibt sich daraus Gleichung 4.8.

$$e(n) = w(n) - x(n) \tag{4.7}$$

$$\underset{t:n-1}{e(n)} \overset{\Delta t:-1}{=} \underset{t:n}{w(n)} \overset{\Delta t:0}{-} \underset{t:n}{x(n)} \tag{4.8}$$

Der Regler wandelt die Regeldifferenz $e(n)$ über die Übertragungsfunktion $R(e(n))$ in die Reglerausgangsgröße $y_r(n)$ um, die in Gleichung 4.9 mit Zeitschrittkennungen dargestellt wird.

$$\underset{t:n-1}{y_R(n)} \overset{\Delta t:-1}{=} \underset{t:n}{R(e(n))} \tag{4.9}$$

Innerhalb der Übertragungsfunktion $R(e(n))$ des Reglers können weitere Zeitschrittprüfungen stattfinden, um den Datenfluss innerhalb der Funktion zu überwachen.

4.3.8.9 Spezifikation von Zeitschritten und temporalen Beziehungen in Hochsprachen

Für gewisse Schleifenkonstrukte kann der Übersetzer einer Hochsprache selbstständig die Zeitschritte von Zählervariablen festlegen und temporale Zusammenhänge innerhalb der Schleifen überwachen. Für komplexere Terme ist jedoch die manuelle Spezifikation von Zeitschritten notwendig.

Für die Hochsprache C wird vorgeschlagen, die Syntax

`<Variablenname>@<Zeitschrittangabe>`

für die Spezifikation der erwarteten Zeitschritte von Operanden zu verwenden. Dies ermöglicht dem Übersetzer, die zu prüfenden temporalen Beziehungen der Operanden eines Terms zu berechnen und in den Zeitschrittkennungen zs der Befehle zu hinterlegen.

Für die Differentialgleichung

$$y(n) = A_0 \cdot x(n) + A_1 \cdot x(n-1)$$

des in Kapitel 4.3.8.6 gezeigten FIR-Filters ergibt sich damit die in der folgenden Auflistung gezeigte Notation in C.

```
float FIRFilter(float x)
{
  #define A0 = 0.5;
  #define A1 = 0.2;

  static float x_old = 0.0;
  float y;

  y@* = A0@* * x@n + A1@* * x_old@(n-1);

  return(y);
}
```

Die Funktion `FIRFilter()` berechnet y(n) für das als Parameter übergebene x(n). In der Funktion selbst wird dabei der Wert von x(n-1) in der Variable `x_old` gespeichert, die zwei Filterkoeffizienten `A0` und `A1` werden in der Funktion definiert. Der Rückgabewert y wird als lokale Variable definiert, wodurch es nicht möglich ist, einen Zeitschritt für diese anzugeben, da sie bei jedem Aufruf der Funktion neu im Speicher angelegt wird. Daher wird die Variable `y` im Term mit der Zeitschrittangabe `@*` versehen, um dem Übersetzer anzuzeigen, dass die Variable keinen Zeitschritt besitzt. Die beiden Filterkoeffizienten besitzen – da es sich bei ihnen um Konstanten handelt – ebenfalls keinen Zeitschritt und erhalten eine entsprechende Angabe im Term. Die beiden Variablen `x` und `x_old` werden mit dem zugehörigen Zeitschritt `@n` bzw. `@(n-1)` gekennzeichnet.

Bei der Übersetzung kann der Übersetzer anhand der angegebenen Zeitschritte feststellen, dass die beiden Multiplikationen der Filterkoeffizienten `A0` und `A1` mit `x` und `x_old` keine zu überwachende temporale Beziehung aufweisen, wodurch die Ergebnisse jeweils den Zeitschritt von `x` bzw. `x_old` erhalten. Bei der Addition dieser Ergebnisse kann der Übersetzer die temporale Beziehung der beiden Summanden herleiten und in der Zeitschrittkennung ZS der Addition spezifizieren. Bei der Zuweisung des Ergebnisses zur Variablen `y` erhält diese dadurch den jüngeren der beiden Zeitschritte {n,n-1}, also n. Bei der Zuweisung ist – wie bereits erwähnt – die Prüfung einer temporalen Beziehung nicht möglich, da `y` eine lokale Variable ohne Vergangenheit ist.

4.3.8.10 Evaluation der Zeitschrittkennung

Die Erkennbarkeit der 20 in Kapitel 2.4 vorgestellten Fehler- und Angriffsarten durch die Zeitschrittkennung ZS wird in Tabelle 4.9 gezeigt.

Tabelle 4.9: Fehlererkennung durch die ZS-Kennung

Fehlerart	**Erkennbarkeit**
Inkompatible Datentypen	nein
Inkompatible Einheiten	nein
Wertebereichsunter- bzw. -überschreitung	nein
Genauigkeitsproblem	nein
Falsche Operandenauswahl	(ja)
Falsche Operatorauswahl	nein
Fehlerhaftes Operationsergebnis	nein
Fristüberschreitung	nein
Zyklusunterschreitung	nein
Zyklusüberschreitung	nein
Verlorengegangene Datenaktualisierung	ja
Synchronisationsfehler oder unvollständige Datenübertragung	ja
Pufferunter- oder -überläufe	(ja)
Fehlerhafter Datenfluss (falsche Adressaten, ...)	(ja)
Duplizierte Daten	ja
Durch Fehler oder Störungen verfälschte Daten	begrenzt
Fehlerhafter Datenzugriff (fehlende Zugriffsrechte)	nein
Nutzung nicht initialisierter Daten	(ja)
Angriffsart	
Gezielt verfälschte Daten	nein
Wiedereinspielungsattacke	begrenzt (mit S)

S: Signaturkennung

Werden für eine Operation die falschen Operanden ausgewählt, was z. B. durch Adressierungsfehler verursacht werden kann, so kann dies mittels der Zeitschrittkennung ZS dann aufgedeckt werden, wenn die Zeitschrittkennung des falschen Operanden nicht der vorgegebenen temporalen Beziehung zwischen den beiden Operanden entspricht. Verlorengegangene Datenaktualisierungen, Synchronisationsfehler und

unvollständige Datenübertragungen können durch die Prüfung der temporalen Beziehung der an der Operation beteiligten Operanden aufgedeckt werden. Gleiches gilt für Fehler, durch die Daten dupliziert, also als neue, aktuelle Daten verwendet werden sollen. Pufferunter- bzw. -überläufe können nur indirekt erkannt werden, wenn die durch den Fehler überschriebenen Daten verwendet werden und der Zeitschritt durch das Überschreiben die in der entsprechenden Operation spezifizierte temporale Beziehung nicht mehr erfüllt. Fehlgeleitete Datenspeicherelemente können bei ihrer Verwendung durch die Zeitschrittkennung dann aufgedeckt werden, wenn der Zeitschritt der betroffenen Datenspeicherelemente nicht die vorgegebene temporale Beziehung erfüllt. Duplizierte Daten können zuverlässig aufgedeckt werden, da hier die Prüfung der temporalen Beziehung den Fehler aufdeckt. Durch Störungen verfälschte Daten können nur dann unter Zuhilfenahme der Zeitschrittkennung erkannt werden, wenn die Störung direkt die Zeitschrittkennung verfälscht. Wiedereinspielungsattacken können zusammen mit einer Signaturkennung S zumeist erkannt werden, da die Zeitschrittkennung dann durch einen Angreifer nicht manipuliert werden kann und dem erwarteten Zeitschritt entsprechen muss. Es ist jedoch vorstellbar, dass der Angreifer die Wiedereinspielungsattacke zum Systemstart beginnt und auf diese Weise veraltete Daten, die jedoch den erwarteten Zeitschritt aufweisen, dem System als aktuelle Daten präsentiert. Deshalb kann die Erkennbarkeit von Wiedereinspielungsattacken unter zusätzlicher Verwendung einer Signaturkennung S nur als begrenzt gewertet werden.

4.3.9 Frist

Eine Datenquelle, z. B. ein intelligenter Sensor, liefert in Echtzeitsystemen häufig zyklisch diskrete Werte, die innerhalb einer bestimmten Zeitspanne verarbeitet und an eine Senke, z. B. einen Aktor, weitergeleitet werden müssen. Werden diese Daten nicht innerhalb einer vorgesehenen Zeit verarbeitet, werden sie nutzlos und können im schlimmsten Fall sogar zu gefährlichen Ausgaben führen. Entsprechend bietet es sich an, die betroffenen Daten mit einer Fristkennung FR zu versehen, die den absoluten Zeitpunkt angibt, ab dem die Daten nicht mehr verwendet werden dürfen.

4.3.9.1 Realisierung der Fristkennung FR

Zur Erkennung von Fehlern, die dazu führen, dass veraltete Daten verwendet werden sollen, versieht eine Quelle alle Datenwerte, die sie erzeugt, mit einer Fristkennung FR, wie in Abbildung 4.45 dargestellt.

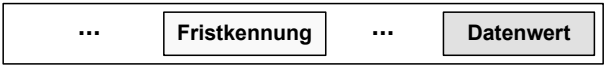


Abbildung 4.45: Datenspeicherelement mit Fristkennung FR und Datenwert W

Diese Fristkennung FR spezifiziert den spätesten Zeitpunkt, zu dem die jeweiligen Inhalte eines Datenspeicherelements noch verarbeitet werden dürfen.

Um die Frist des Ergebnisses einer Operation neu setzen zu können, werden auch die Befehle mit einer Fristkennung versehen, wie in Abbildung 4.46 gezeigt. Diese enthält entweder den Wert Null, wenn die kürzeste der Fristen der Operanden als Frist des Ergebnisses gesetzt werden soll, oder eine relative Fristangabe t_{FR}, die für das Ergebnis gesetzt werden soll.

Abbildung 4.46: Befehlsspeicherelement mit Fristkennung FR, Befehl und Operanden

4.3.9.2 Prüfung der Frist von Daten anhand der Fristkennung FR

Alle verarbeitenden Instanzen prüfen die Datenwerte bei jedem Lesezugriff auf die Einhaltung der Frist. Wird lesend auf einen Datenwert zugegriffen, dessen Frist bereits abgelaufen ist, so wird ein Ausnahmefehler generiert.

```
Prüfung_Lesezugriff :=

WENN FR([Quelle]) < Aktuelle_Zeit DANN
  Generierung_Ausnahmefehler;
ENDEWENN
```

Durch die Einführung eines neuen Befehls kann ermittelt werden, ob bestimmte Daten noch eine zur Verarbeitung ausreichende Restgültigkeitsdauer aufweisen. Ist dies nicht der Fall, so kann die Verarbeitungseinheit sehr frühzeitig reagieren, indem sie

- sofort – und damit weit vor Ablauf der Frist – einen Ausnahmefehler wegen der erkannten Überlastsituation generiert oder

- im Sinne der allmählichen Leistungsabsenkung [45] eine alternative Bearbeitungsroutine auf die Daten anwendet, die ggf. ungenauer ist, dafür aber eine kürzere Laufzeit aufweist.

Ein entsprechender Befehl wird im folgenden Unterkapitel vorgestellt.

4.3.9.3 Setzen der Fristkennung FR in Ergebnissen von Operationen

Um auch die zeitlichen Bedingungen verarbeiteter Daten überwachen zu können, werden den Ergebnissen von Operationen Fristen in deren Fristkennung FR zugewiesen, die wie folgt gesetzt werden:

- enthält der Befehl die Angabe einer relativen Zeitspanne t_{FR}, so wird diese auf die aktuelle Uhrzeit addiert und als neue Frist des Ergebnisses gesetzt,

- ansonsten wird dem Ergebnis die Frist der Operanden zugewiesen, die einen kürzeren verbleibenden Gültigkeitszeitraum definiert.

Jedem Befehl, der Operanden zu einem Ergebnis verarbeitet, wird daher eine relative Zeitangabe hinzugefügt, die die zu setzende Frist des Ergebnisses beeinflusst. Diese Frist wird auf Null gesetzt, wenn die kürzeste Frist der Operanden als Frist des Ergebnisses übernommen werden soll.

```
Setzen_FR_in_Ergebnis :=

WENN FR.Δt_FR([Befehl]) ≠ 0 DANN
  FR([Ergebnis]) := Aktuelle_Uhrzeit + FR.Δt_FR([Befehl]);
SONST WENN FR([Quelle_1]) ≤ FR([Quelle_1]) DANN
  FR([Ergebnis]) := FR([Quelle_1]);
SONST
  FR([Ergebnis]) := FR([Quelle_2]);
ENDEWENN
```

4.3.9.4 Befehle zur Verwaltung der Fristkennung FR

Zum Setzen einer Frist in Datenquellen dient der Befehl Setze Frist SFR, der die Gültigkeitsfrist in der Fristkennung FR des durch A indizierten Datenspeicherelements auf den Zeitpunkt setzt, der sich durch die Addition der in der Fristkennung FR des Befehls angegebenen relativen Frist und der aktuellen Uhrzeit ergibt.

```
SFR A :=

FR([A]) := Aktuelle_Uhrzeit + FR.Δt_FR([Befehl]);
```

Zur Ermittlung des verbleibenden Gültigkeitszeitraums wird der Befehl Ermittle Verbleibenden Gültigkeitszeitraum EVG eingeführt. Dieser stellt zunächst sicher, dass die Frist des durch A indizierten Datenspeicherelements noch nicht verstrichen ist. Ist sie bereits verstrichen, wird sofort ein Ausnahmefehler generiert. Sind die Daten noch gültig, also die Frist noch nicht abgelaufen, dann wird dem durch B indizierten Zieldatenspeicherelement die Differenz zwischen Frist und aktueller Zeit zugewiesen. Dieser Befehl kann – wie bereits im vorhergehenden Unterkapitel beschrieben – dazu genutzt werden, um Überlastsituationen, die die Einhaltung von Gültigkeitsfristen der Daten gefährden würden, frühzeitig zu erkennen und entsprechend zu reagieren.

```
EVG A, B :=

WENN Aktuelle_Zeit ≤ FR([A]) DANN
  W([B]) := Aktuelle_Zeit - FR([A]);
SONST
  Generierung_Ausnahmefehler;
ENDEWENN
```

Für die Prüfung der in der Fristkennung FR angegebenen Frist kann der in Kapitel 4.3.3.3 eingeführte Befehl Prüfe Einen Operanden PEO genutzt werden, der die in der folgenden Auflistung vorgestellten Prüfungen durchführt, ohne eine Änderung des Operanden hervorzurufen. Dabei deutet „...“ die Durchführung weiterer Prüfungen an, die in den zur jeweiligen Kennung gehörenden Kapiteln vorgestellt werden. Bezogen auf die Fristkennung FR stellt PEO sicher, dass die Frist des Operanden noch nicht verstrichen ist und die im Datenspeicherelement enthaltenen Daten somit noch gültig sind.

```
PEO A :=

...

WENN FR([A]) < Aktuelle_Zeit DANN
  Generierung_Ausnahmefehler;
ENDEWENN

...
```

4.3.9.5 Spezifikation der Frist von Operanden in Hochsprachen

Zur Spezifikation der Frist einer Variablen könnte in Übersetzern für die Hochsprache C das Attribut

`__relative_deadline(<relative Zeitangabe>)`

definiert werden. Die Definition einer Variable, der bei jeder Zuweisung eine Frist von 100 ms zugewiesen werden soll, wird exemplarisch in der folgenden Auflistung gezeigt.

```
unsigned int __relative_deadline(100ms) variable_t;
```

Der Übersetzer würde in der Fristkennung FR von Operationen, die der Variablen neue Werte zuweisen, die relative Frist spezifizieren, die das Ergebnis erhalten soll. Alternativ kann der Übersetzer explizit den Befehl Setze Frist SFR nutzen, um die Frist der Variablen zu setzen.

4.3.9.6 Evaluation der Fristkennung FR

Die Erkennbarkeit der einzelnen in Kapitel 2.4 vorgestellten Fehler- und Angriffsarten durch die Fristkennung FR ist in Tabelle 4.10 dargestellt.

Fehler bei der Operandenauswahl, z. B. durch Adressierungsfehler, können durch die Fristkennung FR nur dann aufgedeckt werden, wenn die Frist des falschen Operanden bereits abgelaufen ist. Ist die von der Datenquelle gesetzte Gültigkeitsfrist von Daten abgelaufen, kann diese Fehlerart in jedem Fall durch die Hardware erkannt

Tabelle 4.10: Fehlererkennung durch die FR-Kennung

Fehlerart	**Erkennbarkeit**
Inkompatible Datentypen	nein
Inkompatible Einheiten	nein
Wertebereichsunter- bzw. -überschreitung	nein
Genauigkeitsproblem	nein
Falsche Operandenauswahl	begrenzt
Falsche Operatorauswahl	nein
Fehlerhaftes Operationsergebnis	nein
Fristüberschreitung	ja
Zyklusunterschreitung	nein
Zyklusüberschreitung	nein
Verlorengegangene Datenaktualisierung	(ja)
Synchronisationsfehler oder unvollständige Datenübertragung	(ja)
Pufferunter- oder -überläufe	nein
Fehlerhafter Datenfluss (falsche Adressaten, ...)	nein
Duplizierte Daten	begrenzt
Durch Fehler oder Störungen verfälschte Daten	begrenzt
Fehlerhafter Datenzugriff (fehlende Zugriffsrechte)	nein
Nutzung nicht initialisierter Daten	(ja)
Angriffsart	
Gezielt verfälschte Daten	nein
Wiedereinspielungsattacke	ja (mit S)

S: Signaturkennung

werden. Ebenso können verlorengegangene Datenaktualisierungen, Synchronisationsfehler und unvollständige Datenübertragungen erkannt werden, allerdings nur, wenn die Frist der nicht aktualisierten Daten bereits verstrichen ist. Die Verfälschung von Daten durch Störungen kann nur dann durch die Fristkennung aufgedeckt werden, wenn die Störung die Fristkennung direkt betrifft und die Frist so verändert, dass sie als verstrichen interpretiert wird. Die Nutzung nicht initialisierter Daten kann erkannt werden, wenn den Datenspeicherelementen bei Systemstart eine bereits abgelaufene Frist zugewiesen wird. Wiedereinspielungsattacken können dann zuverlässig erkannt werden, wenn eine Signaturkennung S verwendet wird, die verhindert, dass die Fristkennung durch den Angreifer verfälscht wird.

4.3.10 Zykluszeit

Die Spezifikation der Zykluszeit erlaubt es einer DSA-Einheit, das Ausbleiben eines zyklisch erwarteten Datenwerts frühzeitig zu erkennen, also das Überschreiten einer maximal zulässigen Zykluszeit. Ebenso kann der zu frühe erneute Empfang eines zyklischen Datenwerts erkannt werden, was auf Fehler innerhalb des Senders oder der an der Datenübermittlung beteiligten Geräte hindeutet.

4.3.10.1 Realisierung der Zykluszeitkennung ZY

Um einer Datenspezifikationsarchitektur die Überwachung minimaler und maximaler Zykluszeiten einzelner Datenspeicherelemente zu ermöglichen, werden diese mit einer Zykluszeitkennung ZY versehen, wie in Abbildung 4.47 dargestellt.

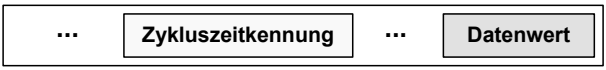


Abbildung 4.47: Datenspeicherelement mit Zykluszeitkennung ZY und Datenwert W

Diese besteht aus drei Komponenten,

- einem Identifikator, der entweder allein oder zusammen mit der Quellkennung Q in der Verarbeitungswegkennung VW der Daten – siehe Kapitel 4.3.7 – eine eineindeutige Identifikation des jeweiligen Datenspeicherelements erlaubt,

- der Spezifikation des frühesten zulässigen Zeitpunkts ZY_{min}, zu dem Daten mit dem identischen Identifikator erneut empfangen werden dürfen und
- die Angabe des spätesten zulässigen Zeitpunkts ZY_{max}, zu dem Daten mit dem identischen Identifikator empfangen werden dürfen,

und hat damit den in Abbildung 4.48 gezeigten Aufbau.

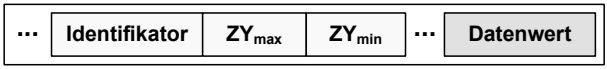


Abbildung 4.48: Aufbau der Zykluszeitkennung ZY in Datenspeicherelementen

Zum automatischen Setzen der Zykluszeitkennung des Ergebnisses einer Operation werden auch die Befehlsspeicherelemente mit einer Zykluszeitkennung ZY versehen, wie in Abbildung 4.49 gezeigt.

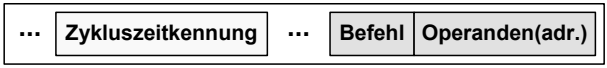


Abbildung 4.49: Befehlsspeicherelement mit Zykluszeitkennung ZY, Befehl und Operanden

Neben dem für das Ergebnis zu setzenden Identifikator enthält die Zykluszeitkennung ZY eines Befehlsspeicherelements zwei relative Zeitangaben ΔZY_{min} und ΔZY_{max}, die das zulässige relative Zeitfenster beschreiben, innerhalb dessen eine Aktualisierung des Datenwerts mit dem angegebenen Identifikator erfolgen darf. Der Aufbau der Zykluszeitkennung ZY innerhalb der Befehlsspeicherelemente ist in Abbildung 4.50 dargestellt. Soll das Ergebnis einer Operation keine Zykluszeitkennung erhalten, so wird dem Identifikator und den beiden relativen Zeitangaben jeweils der Wert Null zugewiesen.

Wird ein Identifikator ungleich Null spezifiziert, so können die beiden relativen Zeitangaben ΔZY_{min} und ΔZY_{max} auch dazu genutzt werden, die Überwachung der Zykluszeit zu beenden, indem einer Angabe oder beiden Angaben der Wert Null zugewiesen wird.

Abbildung 4.50: Aufbau der Zykluszeitkennung ZY in Befehlsspeicherelementen

4.3.10.2 Prüfung des Aktualisierungszyklus von Daten anhand der Zykluszeitkennung ZY

Um eine Überwachung der beiden Zeitgrenzen zu erlauben, wird dem Prozessor einer Datenspezifikationsarchitektur eine Zyklusüberwachungseinheit ZÜE zur Seite gestellt, die sich von dem in [46] vorgestellten Ereignisprozessor für eine GPS-basierte Zeitsteuereinheit und der Ereignisverwaltungseinheit EVE von ISMA [125] ableitet. Sie übernimmt die Funktion marktüblicher Zeitüberwachungsbausteine. Diese Zeitüberwachungsbausteine verfügen in der Regel über eine eigene Takterzeugung und überwachen die Einhaltung einer maximalen, oder bei Bausteinen mit Zeitfenster die Einhaltung einer minimalen und maximalen Zeitgrenze, innerhalb derer der Baustein ein Lebenszeichen in Form eines Impulses erwartet. Bleibt ein solches Signal aus, kann z. B. ein Rücksetzsignal oder eine Unterbrechung durch den Baustein ausgelöst werden. Entsprechende Zeitüberwachungen kommen auch bei sicherheitsgerichteten Feldbussen wie z. B. dem in Kapitel 3.11.2.1 vorgestellten PROFIsafe zum Einsatz. Diese bestehenden Lösungen haben den Nachteil, dass jeweils nur eine für alle Daten geltende Frist oder ein entsprechendes Fristintervall überwacht werden kann. Die Zyklusüberwachungseinheit einer DSA hingegen kann für jedes Datenspeicherelement dank des eindeutigen Identifikators eine minimale und eine maximale Zeitgrenze individuell für eine Vielzahl von Datenspeicherelementen überwachen. Da verschiedene Datenarten in sehr unterschiedlichen diskreten Zeitintervallen erzeugt und versendet werden können, ist eine entsprechend differenzierte Betrachtung sinnvoll. Als Beispiele für entsprechend unterschiedliche Aktualisierungsintervalle sind in Tabelle 4.11 verschiedene Messgrößen zusammen mit einem möglichen Intervall angegeben. So würde z. B. die aktuelle Position des Werkstücks bzw. des Bearbeitungswerkzeugs einer CNC-Maschine alle 10 µs aktualisiert, während die Abfrage des Status einer an der Maschine angebrachten Bedientaste nur alle 100 ms erfolgen müsste. Als dritte Größe könnte die Temperatur der Motoren der CNC-Maschine überwacht werden, wobei diese aufgrund der thermischen Trägheit der Motoren z. B. nur alle 10 s zu erfassen wäre.

Die Zykluszeitkennung ZY aller Datenspeicherelemente, die von einer DSA-Einheit empfangen werden, wird vom Prozessor der DSA an die Zyklusüberwachungsein-

Tabelle 4.11: Messgrößen und deren Aktualisierungsintervalle

Messgröße	Aktualisierungsintervall
Position und Geschwindigkeit in CNC-Maschine	10 µs
Zustand einer Bedientaste	100 ms
Motortemperatur	10 s

heit ZÜE weitergeleitet. Diese ist der Ereignisüberwachungseinheit von ISMA [125] nachempfunden und wird in Abbildung 4.51 gezeigt.

Die Zyklusüberwachungseinheit ZÜE besteht aus der Zyklussteuereinheit ZSE, die über eine Datenverbindung mit dem Hauptprozessor der DSA verbunden ist, und zwei Listen, die von der ZSE verwaltet werden. In diesen Listen werden alle mindestens einmalig empfangenen Datenwerte, bei denen die frühesten bzw. spätesten zulässigen Aktualisierungszeitpunkte $\mathrm{ZY}_{\mathrm{min}}$ bzw. $\mathrm{ZY}_{\mathrm{max}}$ auf Werte ungleich Null gesetzt sind, abgelegt.

Die erste Liste enthält alle Datenidentifikatoren zusammen mit dem frühesten zulässigen Aktualisierungszeitpunkt in Form des Tupels

(Datenidentifikator, ZY_{min}),

zu der ein Datenspeicherelement mit dem jeweiligen Datenidentifikator wieder an die ZÜE übermittelt werden, also von der DSA empfangen werden darf. Sie wird deshalb als Zyklusüberwachungsliste Minimum $\mathrm{ZÜL}_{\mathrm{min}}$ bezeichnet.

Analog zur ersten Liste enthält die zweite die Datenidentifikatoren zusammen mit der zugehörigen maximalen Zykluszeit $\mathrm{ZY}_{\mathrm{max}}$ in Form des Tupels

(Datenidentifikator, ZY_{max})

und wird darum als Zyklusüberwachungsliste Maximum $\mathrm{ZÜL}_{\mathrm{max}}$ bezeichnet. Die Einträge in dieser Liste werden nach chronologisch aufsteigenden spätesten zulässigen Aktualisierungszeitpunkten sortiert, wodurch der erste Eintrag der Liste immer der Eintrag ist, dessen Zeitbedingung als nächstes verletzt werden wird.

Bei einem aktiven Rücksetzsignal löscht die ZSE beide Listen.

```
Rücksetzen_ZÜE :=

ZÜL_min := ∅;
ZÜL_max := ∅;
```

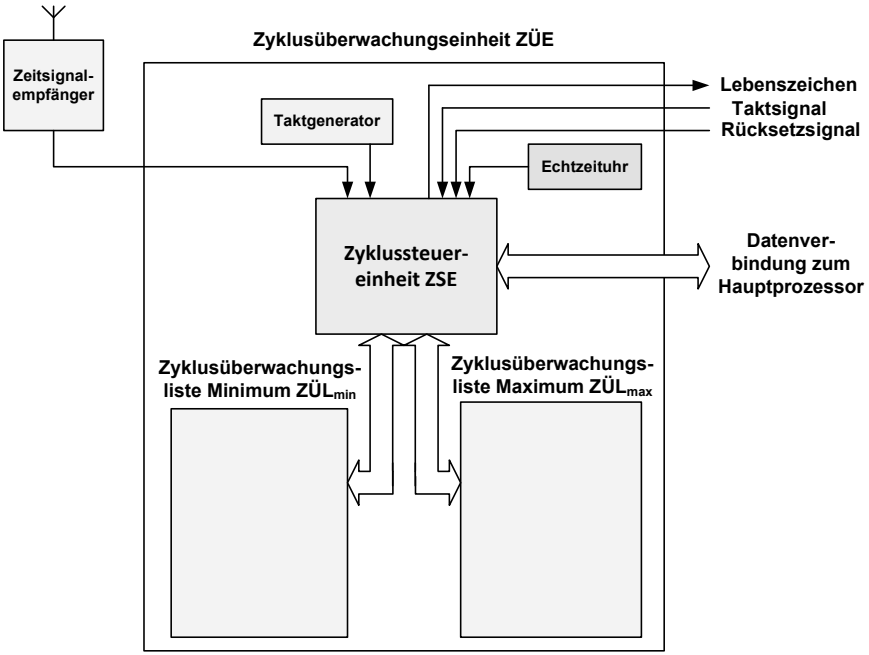


Abbildung 4.51: Aufbau der Zyklusüberwachungseinheit ZÜE

Wird ein Datenwert durch die DSA-Einheit verarbeitet, meldet diese den Identifikator der Daten zusammen mit ihren frühesten und spätesten zulässigen Aktualisierungszeitpunkten $ZY_{min,neu}$ bzw. $ZY_{max,neu}$ an die ZÜE, sofern der Identifikator nicht Null ist. Die ZSE prüft zunächst, ob der Identifikator gültig ist – dieser darf nicht Null sein – und sucht daraufhin anhand des Identifikators einen zugehörigen Eintrag der betreffenden Daten in der $ZÜL_{min}$. Falls ein entsprechender Eintrag gefunden wird, wird geprüft, ob die Bedingung

$$ZY_{min,alt} \leq Aktuelle\_Uhrzeit$$

erfüllt ist. Ist dies nicht der Fall, so wird ein Ausnahmefehler generiert, weil die Daten vor dem frühesten zulässigen Aktualisierungszeitpunkt empfangen wurden. Ist die Bedingung jedoch erfüllt, so wird der Eintrag aus der $ZÜL_{min}$ entfernt.

Im nächsten Schritt sucht die ZSE einen zum Identifikator gehörenden Eintrag in der $ZÜL_{max}$. Ist ein entsprechender Eintrag vorhanden, so wird – der Vollständigkeit halber – verifiziert, dass der bislang gültige späteste zulässige Aktualisierungszeitpunkt noch nicht verstrichen ist, also

$$ZY_{max} > Aktuelle\_Uhrzeit$$

erfüllt ist. Auch hier wird bei Nichterfüllung der Bedingung ein Ausnahmefehler generiert, da die Aktualisierung erst nach dem spätesten zulässigen Aktualisierungszeitpunkt erfolgte. Wurde auch diese zweite Zeitbedingung eingehalten, so wird der zum Identifikator gehörende Eintrag aus der $ZÜL_{max}$ entfernt.

Abschließend legt die ZSE in den beiden Zyklusüberwachungslisten $ZÜL_{min}$ und $ZÜL_{max}$ neue Einträge an, sofern die zu überwachenden Zeitpunkte $ZY_{min,neu}$ bzw. $ZY_{max,neu}$ nicht auf Null gesetzt wurden, um die jeweilige Überwachung des frühesten bzw. spätesten zulässigen Aktualisierungszeitpunkts zu beenden.

```
Prüfung_bei_Eintreffen Identifikator, ZY_min,neu, ZY_max,neu :=

WENN Identifikator ≠ 0 DANN
  WENN Identifikator ∈ ZÜL_min DANN
    WENN ZY_min,alt(Identifikator) > Aktuelle_Uhrzeit DANN
      Generierung_Ausnahmefehler;
    ENDEWENN
    ZÜL_min := ZÜL_min \ (Identifikator,ZY_min,alt);
  ENDEWENN
```

```
SONST
  Generierung_Ausnahmefehler;
ENDEWENN

WENN Identifikator ∈ ZÜL_max DANN
  WENN ZY_max,alt(Identifikator) < Aktuelle_Uhrzeit DANN
    Generierung_Ausnahmefehler;
  ENDEWENN
  ZÜL_max := ZÜL_max \ (Identifikator,ZY_max,alt);
ENDEWENN

WENN ZY_min,neu > 0 DANN
  ZÜL_min := ZÜL_min ∪ (Identifikator,ZY_min,neu);
ENDEWENN

WENN ZY_max,neu > 0 DANN
  ZÜL_max := ZÜL_max ∪ (Identifikator,ZY_max,neu);
ENDEWENN
```

In hinreichend kurzen Zeitabständen prüft die ZSE, ob die spätesten zulässigen Aktualisierungszeitpunkte der in der ZÜL_{max} gespeicherten Einträge noch nicht überschritten wurden. Durch die Sortierung ist es ausreichend, die Zeitbedingung des ersten Eintrags der Liste mit der aktuellen Uhrzeit zu vergleichen. Ist der angegebene Zeitpunkt bereits verstrichen, so liegt eine Verletzung der Echtzeitbedingungen vor und ein Ausnahmefehler wird generiert.

```
Zyklische_Prüfung :=

WENN ZÜL_max ≠ ∅ DANN
  WENN Aktuelle_Uhrzeit > ZÜL_max[0] DANN
    Generierung_Ausnahmefehler;
  ENDEWENN
ENDEWENN
```

Bei einer optimierten Lösung würde sich die ZSE der ZÜE mittels eines Zeitgebers auf die Notwendigkeit der Überprüfung des ersten Eintrags der ZÜL_{max} hinweisen lassen. Dadurch wird eine zyklische Überprüfung unnötig. Bei jeder Aktualisierung

der $\text{ZÜL}_{\max}$, bei der sich der erste Eintrag ändert, würde der Zeitpunkt der nächsten Signalisierung durch den Zeitgeber ebenfalls aktualisiert werden.

4.3.10.3 Setzen der Zykluszeitkennung ZY von Ergebnissen von Operationen

Mittels der Inhalte der Zykluszeitkennung ZY der Befehlsspeicherelemente wird dem Ergebnis von Operationen automatisch eine Zykluszeitkennung ZY hinzugefügt. Die Zykluszeitkennung ZY der Befehle besteht aus einem zu setzenden Identifikator und den zwei relativen Zeitangaben $\Delta\text{ZY}_{\min}$ und $\Delta\text{ZY}_{\max}$, die das Zeitfenster für eine Aktualisierung des Datenwerts festlegen. Das Ergebnis einer Operation erhält keine Zykluszeitkennung, wenn der Identifikator auf Null gesetzt wird, wobei in diesem Fall auch beide Zeitangaben Null sein müssen, da sonst wird ein Ausnahmefehler generiert wird. Ist der Identifikator ungleich Null, dann wertet der Befehl die beiden Zeitangaben $\Delta\text{ZY}_{\min}$ und $\Delta\text{ZY}_{\max}$ aus. Ist eine der Zeitangaben Null, wodurch angezeigt wird, dass die jeweilige Zeitgrenze nicht überwacht oder die Überwachung eingestellt werden soll, so wird der Wert Null in die zugehörige Teilkennung der Zykluszeitkennung ZY des Ergebnisses der Operation eingetragen. Zeitangaben ungleich Null werden auf die aktuelle Uhrzeit aufaddiert, wodurch sich ein absoluter Zeitpunkt ergibt, der in die entsprechende Teilkennung eingetragen wird.

```
Setzen_und_Prüfen_ZY_in_Ergebnis :=

WENN ZY.Identifikator([Befehl]) ≠ 0 DANN
  ZY.Identifikator([Ergebnis]) := ZY.Identifikator([Befehl]);
  WENN ZY.ΔZY_min([Befehl]) = 0 DANN
    ZY.ZY_min([Ergebnis]) := 0;
  SONST
    ZY.ZY_min([Ergebnis]) := Aktuelle_Uhrzeit + ZY.ΔZY_min([Befehl]);
  ENDEWENN
  WENN ZY.ΔZY_max([Befehl]) = 0 DANN
    ZY.ZY_max([Ergebnis]) := 0;
  SONST
    ZY.ZY_max([Ergebnis]) := Aktuelle_Uhrzeit + ZY.ΔZY_max([Befehl]);
  ENDEWENN
SONST WENN ZY.ΔZY_min([Befehl]) = ZY.ΔZY_max([Befehl]) = 0 DANN
  ZY.Identifikator([Ergebnis]) := 0;
  ZY.ZY_min([Ergebnis]) := 0;
```

```
  ZY.ZY_max([Ergebnis]) := 0;
SONST
  Generierung_Ausnahmefehler;
ENDEWENN
```

4.3.10.4 Befehle zur Verwaltung der Zykluszeitkennung ZY

Zum Setzen der Komponenten der Zykluszeitkennung ZY kann neben der Zykluszeitkennung ZY in den Befehlsspeicherelementen auch ein dedizierter Befehl verwendet werden, der mit Setze Zykluszeitkennung SZY bezeichnet wird. Ist der durch A indizierte zu setzende Identifikator gleich Null, so soll die Zykluszeitkennung des durch D indizierten Zieldatenspeicherelements durch Setzen aller Werte auf Null gelöscht werden, was jedoch nur durchgeführt wird, wenn die beiden durch B und C indizierten Zeitangaben ebenfalls Null sind. Bei einem Identifikator ungleich Null wird, falls beide Zeitangaben ungleich Null sind, sichergestellt, dass ein korrektes Zeitintervall definiert wird, der zweite Wert also nicht kleiner als der erste ist. Bei Verletzung dieser Bedingung wird ein Ausnahmefehler generiert. Ansonsten wird der Identifikator innerhalb der Zykluszeitkennung ZY des durch D indizierten Zieldatenspeicherelements auf den angegebenen Wert gesetzt. Die beiden Zeitangaben werden – wenn sie einen Wert ungleich Null besitzen – zur aktuellen Uhrzeit addiert und in ZY_{min} bzw. ZY_{max} eingetragen. Ansonsten wird die jeweilige Teilkennung auf Null gesetzt, um die Überwachung der jeweiligen Zeitgrenze zu beenden.

```
SZY A, B, C, D :=

WENN W([A]) ≠ 0 DANN
  WENN W([B]) ≠ 0 ∧ W([C]) ≠ 0 DANN
    WENN W([B]) > W([C]) DANN
      Generierung_Ausnahmefehler;
    ENDEWENN
  ENDEWENN
  ZY.Identifikator([D]) := W([A]);
  WENN W([B]) = 0 DANN
    ZY.ZY_min([D]) := 0;
  SONST
    ZY.ZY_min([D]) := Aktuelle_Uhrzeit + W([B]);
  ENDEWENN
```

```
  WENN W([C]) = 0 DANN
    ZY.ZY_max([D]) := 0;
  SONST
    ZY.ZY_max([D]) := Aktuelle_Uhrzeit + W([C]);
  ENDEWENN
SONST WENN W([B]) = W([C]) = 0 DANN
  ZY.Identifikator([D]) := 0;
  ZY.ZY_min([D]) := 0;
  ZY.ZY_max([D]) := 0;
SONST
  Generierung_Ausnahmefehler;
ENDEWENN
```

Der in Kapitel 4.3.3.3 eingeführte Befehl Prüfe Einen Operanden PEO kann auch zur Prüfung der Zykluszeitbedingungen eines Operanden eingesetzt werden. Dabei werden – neben weiteren durch „...“ angedeuteten Prüfungen – die in der Zykluszeitkennung ZY des Operanden spezifizierten Daten in Form einer Nachricht an die Zyklusüberwachungseinheit ZÜE übermittelt. Diese verifiziert anhand ihrer internen Listen die Einhaltung der Zykluszeitbedingungen des Operanden, aktualisiert diese und generiert bei Verletzung der gesetzten Bedingungen einen Ausnahmefehler. Aus Platzgründen wird in der Pseudocodeauflistung Identifikator durch Id abgekürzt.

```
PEO A :=

...
WENN ZY.Id([A]) ≠ 0 DANN
  ZÜE_Nachricht := (ZY.Id([A]), ZY.ZY_min([A]), ZY.ZY_max([A]));
ENDEWENN
...
```

4.3.10.5 Eine alternative Realisierung der Zykluszeitüberwachung

Eine weitere Realisierungsmöglichkeit der Überwachung der zyklischen Aktualisierung besteht darin, die Zykluszeitkennung ZY allein durch den bereits vorgestellten eineindeutigen Datenidentifikator zu realisieren, wie in Abbildung 4.52 dargestellt wird.

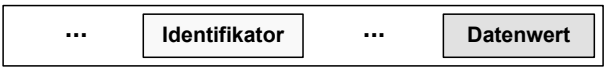


Abbildung 4.52: Alternative Realisierung der Zykluszeitkennung ZY

Der Aufbau und die Funktion der Zyklusüberwachungseinheit bleiben bei dieser Lösung unverändert. Statt jedoch die Daten für die minimale und maximale Zykluszeit in den Datenspeicherelementen in der Zykluszeitkennung ZY zu spezifizieren, würden die Daten bei der alternativen Lösung einer Konfigurationsdatei – die zur Beschleunigung des Zugriffs in einer Tabelle im Speicher zwischengespeichert sein sollte – entnommen, in der jedem Datenidentifikator ein entsprechendes Tupel ($\text{Zyklus}_{\text{min}}$,$\text{Zyklus}_{\text{max}}$) zugeordnet ist.

Während die alternative Lösung den Vorteil einer deutlich kleineren Zykluszeitkennung ZY hat, da weder die minimal noch die maximal zulässige Zykluszeit im Datenspeicherelement selbst hinterlegt werden müssen, so widerspricht die Lösung jedoch eindeutig dem in Kapitel 4.2 formulierten Entwicklungsparadigma, nach dem alle Eigenschaften von Daten auch in diesen selbst enthalten sein sollen. Daher ist die erste Lösung bei der Realisierung einer Datenspezifikationsarchitektur DSA zu bevorzugen.

4.3.10.6 Spezifikation der Frist von Operanden in Hochsprachen

Zur Definition der relativen Zykluszeiten einer Variablen könnte ein Übersetzer – hier exemplarisch für die Hochsprache C – das Attribut

```
__relativecyclicwindow(<frühester relativer Zeitpunkt>,
                       <spätester relativer Zeitpunkt>)
                      <Variablenname>
```

definieren, mit dessen Hilfe die zwei relativen Zeitpunkte für die früheste bzw. späteste zulässige Aktualisierung definiert werden können.

Bei jeder Aktualisierung einer entsprechend deklarierten Variable setzt der Übersetzer ein neues zulässiges Aktualisierungsfenster in der Zykluszeitkennung ZY der Variable durch Spezifikation der zu setzenden Daten in der Zykluszeitkennung ZY des jeweiligen Befehls oder unter Nutzung des dedizierten Befehls Setze Zykluszeit SZY.

In der folgenden Auflistung wird die Definition einer Variable gezeigt, die von einem Empfänger frühestens 100 ms nach der Zuweisung und spätestens 150 ms danach empfangen werden darf.

```
unsigned char __relativecyclicwindow(100ms,150ms) variable_j;
```

4.3.10.7 Evaluation der Zykluszeitkennung

Die Zykluszeitkennung ZY erlaubt die Überwachung eines zyklisch zu aktualisierenden Datenwerts auf die Einhaltung eines in den Daten spezifizierten Zeitfensters. Die Zyklusüberwachungseinheit ZÜE entspricht einer Sammlung von Überwachungszeitgebern mit Zeitfensterüberwachung, bei der jedem empfangenen Datenwert ein Überwachungszeitgeber zugeordnet ist. Dadurch können Werte mit verschiedensten zulässigen Empfangszeitfenstern überwacht werden, ohne die Einschränkung typischer Überwachungszeitgeber, dabei nur ein Zeitfenster für alle Werte verwenden zu können.

Die Implementierung der Zykluszeitkennung ZY zusammen mit der Zyklusüberwachungseinheit ZÜE erlaubt damit die Erkennung der in Tabelle 4.12 gezeigten Arten der 20 in Kapitel 2.4 identifizierten Fehler- und Angriffsarten.

Fristüberschreitungen lassen sich dann durch die Kombination aus ZY und ZÜE erkennen, wenn bei der Übertragung der Daten durch Verzögerungen bereits der späteste zulässige Zeitpunkt für den Empfang der nächsten Aktualisierung des Datenwerts verstrichen ist. Die ZÜE erkennt dies beim Einfügen der Daten in die ZÜL_{max}, da die aktuelle Zeit den zulässigen spätesten Empfangszeitpunkt bereits überschreitet. Allerdings ist die Erkennung dieser Fehlerart über die Fristkennung FR wahrscheinlicher und zuverlässiger. Zyklusunter- und -überschreitungen können durch die beschriebenen Merkmale zuverlässig aufgedeckt werden. Verlorengegangene Datenaktualisierungen können – analog zur Fristüberschreitung – dann erkannt werden, wenn der späteste zulässige Zeitpunkt bei Empfang der Daten bereits verstrichen ist. Duplizierte Daten können nur dann erkannt werden, wenn beim Empfang eines Duplikats der früheste zulässige Empfangszeitpunkt einer Aktualisierung noch nicht verstrichen ist. Wiedereinspielungsattacken können dadurch aufgedeckt werden, dass die Zykluszeitkennungen ZY der aufgezeichneten und erneut gesendeten Datenspeicherelemente bereits verstrichene späteste zulässige Zeitpunkte aufweisen.

Tabelle 4.12: Fehlererkennung durch die ZY-Kennung

Fehlerart	**Erkennbarkeit**
Inkompatible Datentypen	nein
Inkompatible Einheiten	nein
Wertebereichsunter- bzw. -überschreitung	nein
Genauigkeitsproblem	nein
Falsche Operandenauswahl	nein
Falsche Operatorauswahl	nein
Fehlerhaftes Operationsergebnis	nein
Fristüberschreitung	begrenzt
Zyklusunterschreitung	ja
Zyklusüberschreitung	ja
Verlorengegangene Datenaktualisierung	(ja)
Synchronisationsfehler oder unvollständige Datenübertragung	nein
Pufferunter- oder -überläufe	nein
Fehlerhafter Datenfluss (falsche Adressaten, ...)	nein
Duplizierte Daten	begrenzt
Durch Störungen oder Fehler verfälschte Daten	begrenzt
Fehlerhafter Datenzugriff (fehlende Zugriffsrechte)	nein
Nutzung nicht initialisierter Daten	nein
Angriffsart	
Gezielt verfälschte Daten	nein
Wiedereinspielungsattacke	ja (mit S)

S: Signaturkennung

4.3.11 Integritätsprüfung und Adresse

Zur Integritätsprüfung wird jedes Datenspeicherelement mit einer Fehlererkennungskodierung mit hinreichendem minimalem Hamming-Abstand versehen, anhand derer die Hardware bei jedem Zugriff auf ein Datenspeicherelement prüfen kann, ob dessen Inhalte korrekt sind und keine Datenverfälschungen durch Störungen oder Fehler aufgetreten sind. Der Einsatz entsprechender Integritätsprüfungsverfahren wird in der IEC 61508 gefordert [51–53].

4.3.11.1 Realisierung der Integritätsprüfung IP

Um der Hardware eine Integritätsprüfung von Datenspeicherelementen zu ermöglichen, wird jedes Datenspeicherelement mit einer Integritätsprüfungskennung IP versehen, wie in Abbildung 4.53 dargestellt.

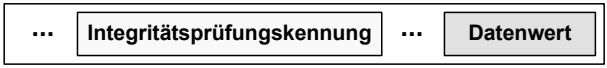


Abbildung 4.53: Datenspeicherelement mit Integritätsprüfungskennung IP und Datenwert W

In der IEC 61508 wird der Einsatz polynomialer Codes oder eines Hamming-Codes vorgeschlagen [52, 53]. Bei ISMA kommt nach [125] ein Erweiterter-(128,120)-Hamming-Code mit einem Hamming-Abstand von 4 zum Einsatz, mit dessen Hilfe Dreifachbitfehler erkannt werden können. Eine Korrektur von erkannten Fehlern sollte nach [51, 53] nicht durchgeführt werden, um keine gültigen, jedoch falschen Daten zu erzeugen. Da die Datenspeicherelemente bei einer Datenspezifikationsarchitektur DSA eine wesentlich höhere Bitbreite aufweisen, muss ein Hamming-Code mit passender Breite zum Einsatz kommen.

4.3.11.2 Prüfung der Datenspeicherelementintegrität anhand der Integritätsprüfungskennung IP

Bei jedem Zugriff – egal, ob lesend oder schreibend – wird das jeweilige Datenspeicherelement gelesen und dessen Integrität anhand der Integritätsprüfungskennung IP verifiziert. Zu diesem Zweck kommt die Prüfungsfunktion $\mathcal{I}_{\text{prüf}}$ zum Einsatz.

```
Prüfung_Integrität_ohne_Adresse :=

WENN I_prüf([Quelle]) ≠ gültig DANN
  Generierung_Ausnahmefehler;
ENDEWENN
```

4.3.11.3 Einbeziehung der Adresse AD in die Integritätsprüfung

Durch die Integration der Adresse AD eines Datenspeicherelements in die Integritätsprüfung können Fehler erkannt werden, die dazu führen, dass ein anderes Datenspeicherelement zur Verarbeitung herangezogen wird, als eigentlich vorgesehen war. Eine automatische Prüfung durch die Hardware kann z. B. dadurch realisiert werden, dass die Adresse eines Datenspeicherelements in die Integritätsprüfung IP einbezogen wird. Ein entsprechendes Vorgehen wird in [51, 53] vorgeschlagen. Bei der Integrationsprüfung eines aus dem Speicher gelesenen Datenspeicherelements wird dann die Adresse des angeforderten Datenspeicherelements mit den Inhalten des Datenspeicherelements verknüpft – z. B. durch eine Exklusiv-ODER-Verknüpfung – und danach die Integrität des Datenspeicherelements mit der implementierten Integritätsprüfung geprüft. Sollte ein falsches Datenspeicherelement aus dem Speicher geladen worden sein, so wird der Fehler in diesem Moment erkannt.

```
Prüfung_Integrität_mit_Adresse :=

WENN I_prüf([Quelle] EXODER Adresse(Quelle)) ≠ gültig DANN
  Generierung_Ausnahmefehler;
ENDEWENN
```

4.3.11.4 Setzen der Integritätsprüfungskennung in Zieldatenspeicherelementen

Beim schreibenden Zugriff auf ein Datenspeicherelement werden die Inhalte der Integritätsprüfungskennung IP durch Anwendung der Generierungsfunktion $\mathcal{I}_{\text{gen}}$ neu gebildet und dem Datenspeicherelement hinzugefügt. Je nachdem, ob die Adresse des Datenspeicherelements in die Integrationsprüfung einbezogen werden soll oder

nicht, wird die Generierungsfunktion nur auf die Inhalte des Datenspeicherelements oder deren Verknüpfung mit der Adresse des Datenspeicherelements angewendet.

```
Setzen_Integritätskennung_ohne_Adresse :=

IP([Ziel]) := I_gen([Ziel]);
```

```
Setzen_Integritätskennung_mit_Adresse :=

IP([Ziel]) := I_gen([Ziel] EXODER Adresse(Ziel));
```

4.3.11.5 Befehle zur Verwaltung der Integritätsprüfung IP

Die Verwaltung der Integritätsprüfungskennung IP erfolgt automatisch durch die Hardware der Datenspezifikationsarchitektur DSA. Der in Kapitel 4.3.3.3 eingeführte Befehl Prüfe Einen Operanden PEO kann dazu genutzt werden, die Integrität eines Operanden sicherzustellen, ohne diesen zu ändern. Dabei deutet „...“ die Durchführung weiterer Prüfungen an, die in den zur jeweiligen Kennung gehörenden Kapiteln vorgestellt werden. Bezogen auf die Integritätsprüfung wird geprüft, ob die in der Integritätsprüfungskennung IP spezifizierte Prüfsumme den Inhalten des Datenspeicherelements unter Einbeziehung der Adresse des Datenspeicherelements entspricht.

```
PEO A :=

...

WENN I_prüf([A] EXODER Adresse(A)) ≠ gültig DANN
  Generierung_Ausnahmefehler;
ENDEWENN

...
```

4.3.11.6 Datenportale zur Behandlung abgehender und ankommender Daten

Die in Kapitel 4.3.11.3 beschriebene Einbeziehung der Adresse in die Integritätsprüfung bringt ein Problem bei der Übertragung von Datenspeicherelementen mit einer entsprechenden Integritätsprüfungskennung IP zwischen Systemkomponenten mit sich, da der Empfänger die Adresse des Datenspeicherelements im Speicher des Senders nicht kennen kann. Zur Lösung des Problems werden sogenannte Datenportale DP eingeführt, von denen zwei Varianten existieren: Je nachdem, ob Daten gesendet oder empfangen werden sollen, werden diese als Datenausgangsportale DAP oder Dateneingangsportale DEP bezeichnet. Die Funktion der Datenportale DP wird in Abbildung 4.54 dargestellt.

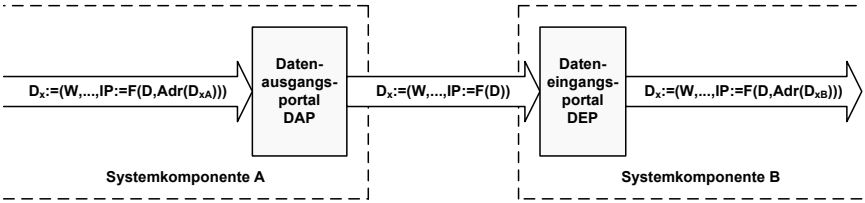


Abbildung 4.54: Funktionsweise der Datenportale bei der Integritätsprüfungskennung IP

Die Integritätsprüfungskennung IP des Datenspeicherelements D_x, welches von der Systemkomponente A an die Systemkomponente B übertragen werden soll, wird nach

$$IP_{vor\_DAP} := \mathcal{I}_{gen}\left([D_x], \text{Adresse}(D_{xA})\right)$$

durch die Generierungsfunktion $\mathcal{I}_{gen}$ aus den Inhalten des Datenspeicherelements D_x und dessen Adresse D_{xA} gebildet. Das Datenausgangsportal DAP auf Senderseite prüft die Integrität des zu sendenden Datenspeicherelements D_x nochmals und entfernt dann die Adresse des Datenspeicherelements aus der Integritätsprüfungskennung IP, wodurch diese nach

$$IP_{nach\_DAP} := \mathcal{I}_{gen}\left([D_x]\right)$$

nur noch aus den Inhalten des Datenspeicherelements D_x gebildet wird. Anschließend kann das Datenspeicherelement an den Empfänger übertragen werden.

```
DAP :=

WENN I_prüf([Quelle] EXODER Adresse(Quelle)) ≠ gültig DANN
  Generierung_Ausnahmefehler;
ENDEWENN

[Ziel] := [Quelle];
IP([Ziel]) := I_gen([Ziel]);
```

Im Empfänger wird die Integrität des empfangenen Datenspeicherelements D_x im Dateneingangsportal DEP zunächst nochmals geprüft, bevor es mit einer neuen Integritätsprüfungskennung IP nach

$$\mathrm{IP}_{\mathrm{nach\_DEP}} := \mathcal{I}_{\mathrm{gen}}\left([D_x], \mathrm{Adresse}(D_{xB})\right)$$

mit der neuen Adresse D_{xB} versehen und im Speicher des Empfängers an der entsprechenden Adresse abgelegt wird. Bei allen folgenden Lesezugriffen innerhalb des Empfängers B kann die Integrität des betreffenden Datenspeicherelements unter Einbeziehung seiner Adresse erfolgen.

Stellt ein Dateneingangsportal DEP eine Verfälschung eines empfangenen Datenspeicherelements fest, so kann es einen erneuten Sendungsversuch beim Datenausgangsportal des Senders der Daten anfordern. Ein entsprechendes Vorgehen zur Behandlung von Übertragungsfehlern wurde in Kapitel 4.1.3.2 statt der sofortigen Generierung eines Ausnahmefehlers vorgeschlagen, da Übertragungsfehler auf Kommunikationsleitungen recht häufig vorkommen können.

```
DEP :=

WENN I_prüf([Quelle]) ≠ gültig DANN
  Erneutes_Senden_anfordern;
ENDEWENN

[Ziel] := [Quelle];
IP([Ziel]) := I_gen([Ziel] EXODER Adresse(Ziel));
```

4.3.11.7 Evaluation der Integritätsprüfung IP

Die Nutzung einer Integritätsprüfung ist Stand der Technik, sie wurde z. B. bereits im in Kapitel 3.3.1.1 vorgestellten Telefunken TR 4 in Form der sog. „Dreierprobe“ eingesetzt, um Verfälschungen von Datenworten und Instruktionen zu erkennen. Auch die Einbeziehung der Adresse eines Datenspeicherelements in die Integritätsprüfung ist nicht neu, sie wird durch die IEC 61508 in [51, 53] vorgeschlagen.

Durch die Integritätsprüfung können von den 20 in Kapitel 2.4 vorgestellten Fehler- und Angriffsarten die in Tabelle 4.13 gezeigten Fehlertypen erkannt werden.

Wird die Adresse AD eines Datenspeicherelements in die Prüfung der Integrität eines Datenspeicherelements mit einbezogen, so lassen sich falsche Operanden, die statt der gewünschten Operanden aus dem Speicher gelesen wurden, zuverlässig erkennen. Innerhalb der durch den minimalen Hammingabstand des eingesetzten Integritätsprüfungsverfahrens vorgegebenen Grenzen der Fehlererkennbarkeit können Verfälschungen der Daten durch Störungen oder Fehler zuverlässig erkannt werden. Die Nutzung nicht initialisierter Daten könnte dann aufgedeckt werden, wenn Datenspeicherelemente ohne gültige Daten zeitgleich keine gültige IP-Kennung aufweisen, mit dem Nachteil, die restlichen Kennungen des betroffenen Datenspeicherelements nicht auswerten zu können. Durch Manipulationen eines Angreifers verfälschte Daten können nur dann erkannt werden, wenn es der Angreifer versäumt, die IP-Kennung nach der Manipulation entsprechend zu korrigieren.

4.3.12 Signatur und Adresse

Die Signatur S ist die am aufwendigsten zu realisierende Dateneigenschaft. Mit ihr kann, unter Anwendung asymmetrischer Verschlüsselungsverfahren, ein Datenspeicherelement durch eine Quelle kryptographisch signiert und damit vor unbefugter Manipulation geschützt werden. Datensenken können, ohne dass es weiterer Softwareanweisungen bedarf, bei jedem Zugriff die Authentizität eines entsprechend gesicherten Datenspeicherelements verifizieren.

4.3.12.1 Realisierung der Signaturkennung S

Zur Realisierung der Signaturkennung S wird jede Datenquelle mit einem Schlüsselpaar bestehend aus einem geheimen und einem öffentlichen Schlüssel versehen. Mit Hilfe des geheimen Schlüssels erstellt die Hardware beim schreibenden Zugriff auf ein Datenspeicherelement eine Signatur von dessen Inhalten und fügt diese dem

Tabelle 4.13: Fehlererkennung durch die IP-Kennung

Fehlerart	**Erkennbarkeit**
Inkompatible Datentypen	nein
Inkompatible Einheiten	nein
Wertebereichsunter- bzw. -überschreitung	nein
Genauigkeitsproblem	nein
Falsche Operandenauswahl	ja (mit AD)
Falsche Operatorauswahl	nein
Fehlerhaftes Operationsergebnis	nein
Fristüberschreitung	nein
Zyklusunterschreitung	nein
Zyklusüberschreitung	nein
Verlorengegangene Datenaktualisierung	nein
Synchronisationsfehler oder unvollständige Datenübertragung	nein
Pufferunter- oder -überläufe	nein
Fehlerhafter Datenfluss (falsche Adressaten, ...)	nein
Duplizierte Daten	nein
Durch Störungen oder Fehler verfälschte Daten	ja
Fehlerhafter Datenzugriff (fehlende Zugriffsrechte)	nein
Nutzung nicht initialisierter Daten	begrenzt
Angriffsart	
Gezielt verfälschte Daten	begrenzt
Wiedereinspielungsattacke	nein

AD: Einbeziehung der Datenspeicherelementadresse

Datenspeicherelement als Signaturkennung S hinzu, wie in Abbildung 4.55 dargestellt.

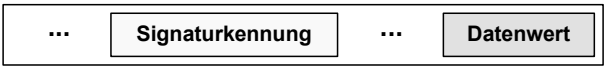


Abbildung 4.55: Datenspeicherelement mit Signaturkennung S und Datenwert W

Der Signaturprozess folgt dem allgemeinen Schema digitaler Signaturen [83]:

- Mittels einer kryptographischen Hashfunktion wie z. B. SHA-1 wird die Signatur der Inhalte des Datenspeicherelements ohne die Signaturkennung S gebildet.
- Die Signatur wird mit dem jeweiligen geheimen Schlüssel z. B. via RSA chiffriert.
- Die chiffrierte Signatur wird in der Signaturkennung S gespeichert.

Verfahren, bei dem die gesamten oder Teile der zu signierenden Daten gemeinsam mit der Signatur chiffriert und bei der Prüfung der Signatur wiederhergestellt werden, wie es in der ISO/IEC 9796-2 [62] normiert ist, könnten bei einer Datenspezifikationsarchitektur DSA zwar genutzt werden, haben allerdings einen entscheidenden Nachteil: Da die Inhalte eines derart signierten Speicherelements ganz oder teilweise chiffriert sind, muss zunächst die Dechiffrierung durchgeführt werden, bevor die Hardware die Inhalte der einzelnen Kennungen prüfen kann. Es ist daher das oben beschriebene Verfahren zu bevorzugen.

Während die Integritätsprüfung IP dem primären Ziel dient, Datenverfälschungen durch verschiedene Fehler- und Störungseinflüsse zu erkennen, kann mittels der Signaturkennung S effektiv verhindert werden, dass ein Angreifer absichtlich gefälschte Daten in ein System einbringt und diese Daten dann z. B. zu gefährlichen Systemreaktionen führen. Ein entsprechendes Angriffsszenario wäre z. B. der Versuch, einen chemischen Prozess in einen kritischen Zustand zu überführen, indem der ermittelte Temperaturwert eines Temperatursensors gefälscht wird.

Zusammen mit den Merkmalen Zeitschritt ZS und Frist FR kann eine weitere Angriffsart zuverlässig erkannt werden: Wiedereinspielungsattacken. Bei einem solchen Angriff versucht ein Angreifer, vorab aufgezeichnete, also ältere, gültige Daten dem

System als aktuelle Daten „anzubieten“, mit dem Ziel, den Prozess seinen Wünschen entsprechend zu manipulieren. Da die Signatur sicherstellt, dass ein Angreifer nicht in der Lage ist, die Zeitschritt- ZS oder Fristkennung FR zu verändern, kann eine Datensenke die veralteten Daten als solche identifizieren und entsprechend reagieren.

Da bei Nutzung einer Signaturkennung auch Verfälschungen der Inhalte von Datenspeicherelementen erkannt werden können, die nicht durch Manipulation, sondern durch Störungen oder Fehler aufgetreten sind, kann in diesem Fall auf die Implementierung einer Integritätsprüfungskennung IP verzichtet werden.

4.3.12.2 Prüfung der Authentizität anhand der Signaturkennung S

Die zu den geheimen Schlüsseln der Quellen gehörenden öffentlichen Schlüssel werden in der Quelle selbst und in den Datensenken abgelegt und zur Prüfung der Authentizität von Datenspeicherelementen herangezogen. Diese Authentizitätsprüfung führt die Hardware bei jedem Zugriff auf ein entsprechend gesichertes Datenspeicherelement durch. Kann die Authentizität eines Datenspeicherelements nicht verifiziert werden, weil das Datenspeicherelement

- durch Störungen oder Fehler verändert oder
- durch einen Angreifer gezielt manipuliert wurde,

so wird ein Ausnahmefehler generiert.

Der Prüfprozess beim Zugriff auf ein Datenspeicherelement folgt dem allgemeinen Schema zur Prüfung digitaler Signaturen [83]:

- Unter Nutzung des passenden öffentlichen Schlüssels wird die chiffrierte Signatur dechiffriert.
- Unter Anwendung der zur Erstellung der Signatur genutzten kryptographischen Hashfunktion wird eine Vergleichssignatur der vorliegenden Inhalte des zu prüfenden Datenspeicherelements gebildet.
- Die Vergleichssignatur wird mit der dechiffrierten Signatur verglichen. Im Falle der Übereinstimmung gilt das Speicherelement als authentifiziert und unverändert.

Die folgende Pseudocodeauflistung zeigt die Prüfung des Signaturkennung S ohne Einbeziehung der Adresse des Speicherelements, wobei die Prüfungsfunktion $\mathcal{S}_{\text{prüf}}$ die Inhalte des Datenspeicherelements unter Nutzung des öffentlichen Schlüssels öS verifiziert.

```
Prüfung_Signatur_ohne_Adresse :=

WENN S_prüf([Quelle], öS) ≠ gültig DANN
  Generierung_Ausnahmefehler;
ENDEWENN
```

4.3.12.3 Einbeziehung der Adresse AD in die Signaturkennung

Da sich anhand der Signaturkennung S Integrität und Authentizität eines Datenspeicherelements verifizieren lassen, wird eine zusätzliche Integritätsprüfung anhand Integritätsprüfungskennung IP überflüssig. Zur Aufdeckung von Adressierungsfehlern sollte daher analog zur Integritätsprüfungskennung IP die Adresse des Datenspeicherelements in die Bildung der Signatur einbezogen werden.

```
Prüfung_Signatur_mit_Adresse :=

WENN S_prüf([Quelle] EXODER Adresse(Quelle), öS) ≠ gültig DANN
  Generierung_Ausnahmefehler;
ENDEWENN
```

4.3.12.4 Setzen der Signaturkennung in Zieldatenspeicherelementen

Wie auch bei Anwendung der Integritätsprüfungskennung IP werden beim schreibenden Zugriff auf ein Datenspeicherelement die Inhalte der Signaturkennung S durch Anwendung der Generierungsfunktion $\mathcal{S}_{\text{gen}}$ neu gebildet und dem Datenspeicherelement hinzugefügt. Je nachdem, ob die Adresse des Datenspeicherelements in die Integrationsprüfung einbezogen werden soll oder nicht, wird die Generierungsfunktion nur auf die Inhalte des Datenspeicherelements oder deren Verknüpfung mit der Adresse des Datenspeicherelements angewendet.

```
Setzen_Signaturkennung_ohne_Adresse :=

S([Ziel]) := S_gen([Ziel]);
```

```
Setzen_Signaturkennung_mit_Adresse :=

S([Ziel]) := S_gen([Ziel] EXODER Adresse(Ziel));
```

4.3.12.5 Befehle zur Verwaltung der Signatur S

Wie auch bei der Integritätsprüfung IP erfolgt die Verwaltung der Signaturkennung S automatisch durch die Hardware der Datenspezifikationsarchitektur DSA. Der in Kapitel 4.3.3.3 eingeführte Befehl Prüfe Einen Operanden PEO kann jedoch dazu genutzt werden, die Integrität eines Operanden anhand seiner Signatur sicherzustellen, ohne diesen zu ändern. Dabei deutet „...“ die Durchführung weiterer Prüfungen an, die in den zur jeweiligen Kennung gehörenden Kapiteln vorgestellt wurden.

Bezogen auf die Signaturkennung S wird geprüft, ob die in der Kennung spezifizierte Signatur den Inhalten des Datenspeicherelements unter Einbeziehung der Adresse des Datenspeicherelements entspricht.

```
PEO A :=

...

WENN S_prüf([A] EXODER Adresse(A), öS) ≠ gültig DANN
  Generierung_Ausnahmefehler;
ENDEWENN

...
```

4.3.12.6 Datenportale zur Behandlung abgehender und ankommender Daten

Die in Kapitel 4.3.12.3 beschriebene Einbeziehung der Adresse in die Signaturkennung S führt beim Datenaustausch zwischen Systemkomponenten zum identischen Problem, welches bereits bei der Beschreibung der Integritätsprüfungskennung IP in Kapitel 4.3.11.6 erwähnt wurde: Der Empfänger von Daten hat kein Wissen über die Adresse eines Datenspeicherelements im Speicher des Senders und kann diese deshalb nicht in die Prüfung der Signaturkennung S einbeziehen. Zudem müssen die Datenspeicherelemente vom Empfänger umsigniert werden, damit die Datenspeicherelemente mit dem geheimen Schlüssel des Empfängers statt dem des Senders signiert sind. Zur Lösung des Problems können die ebenfalls in Kapitel 4.3.11.6 beschriebenen Datenportale DP in Form der Datenausgangsportale DAP auf Sender- bzw. der Dateneingangsportale DEP auf Empfängerseite genutzt werden. Die Funktion der Datenportale DP in Bezug auf die Signaturkennung S wird in Abbildung 4.56 dargestellt.

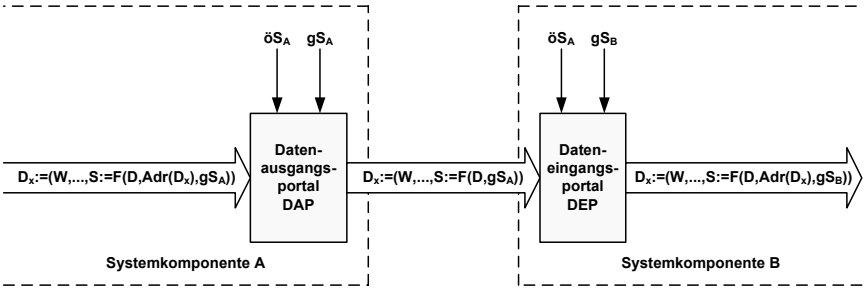


Abbildung 4.56: Funktionsweise der Datenportale bei der Signaturkennung S

Die Signaturkennung S des Datenspeicherelements D_x, welches von der Systemkomponente A an die Systemkomponente B übertragen werden soll, wird nach

$$S_{vor\_DAP} := \mathcal{S}_{gen}\left([D_x], \text{Adresse}(D_{xA}, gS_A)\right)$$

durch die Generierungsfunktion $\mathcal{S}_{gen}$ aus den Inhalten des Datenspeicherelements D_x, dessen Adresse D_{xA} und dem geheimen Schlüssel gS_A gebildet. Das Datenausgangsportal DAP auf Senderseite prüft zunächst die Integrität und Authentizität

des zu sendenden Datenspeicherelements D_x nochmals und entfernt dann die Adresse des Datenspeicherelements aus der Signaturkennung, wodurch diese nach

$$S_{nach\_DAP} := \mathcal{S}_{gen}\left([D_x], gS_A\right)$$

nur noch aus den Inhalten des Datenspeicherelements D_x und dem geheimen Schlüssel gS_A der Systemkomponente A gebildet wird. Anschließend kann das Datenspeicherelement an den Empfänger übertragen werden.

```
DAP :=

WENN S_prüf([Quelle] EXODER Adresse(Quelle), öS_eigen) ≠ gültig DANN
  Generierung_Ausnahmefehler;
ENDEWENN

[Ziel] := [Quelle];
S([Ziel]) := S_gen([Ziel], gS_eigen);
```

Im Empfänger werden Integrität und Authentizität des empfangenen Datenspeicherelements D_x im Dateneingangsportal DEP anhand der Signaturkennung S zuerst unter Zuhilfenahme des öffentlichen Schlüssels $öS_A$ der sendenden Systemkomponente geprüft, bevor es mit einer neuen Signaturkennung S nach

$$S_{nach\_DEP} := \mathcal{S}_{gen}\left([D_x], \text{Adresse}(D_{xB}, gS_B)\right)$$

mit der neuen Adresse D_{xB} unter Nutzung des geheimen Schlüssels gS_B des Empfängers B versehen und im Speicher des Empfängers an der entsprechenden Adresse abgelegt wird. Bei allen weiteren Zugriffen können Integrität und Authentizität des betreffenden Datenspeicherelements dann unter Einbeziehung des öffentlichen Schlüssels $öS_B$ und der Adresse des Datenspeicherelements verifiziert werden.

Wird ein Datenspeicherelement als verfälscht erkannt, so wird das korrespondierende Datenausgangsportal DAP auf Senderseite zu einem erneuten Sendungsversuch aufgefordert. Diese Art der Behandlung von Übertragungsfehlern wurde in Kapitel 4.1.3.2 statt der sofortigen Generierung eines Ausnahmefehlers vorgeschlagen, da Übertragungsfehler auf Kommunikationsleitungen recht häufig vorkommen können.

```
DEP :=

WENN S_prüf([Quelle], öS_fremd) ≠ gültig DANN
  Erneutes_Senden_anfordern;
ENDEWENN

[Ziel] := [Quelle];
S([Ziel]) := S_gen([Ziel] EXODER Adresse(Ziel), gS_eigen);
```

4.3.12.7 Evaluation der Signaturkennung S

Die Sinnhaftigkeit einer Signatur pro einzelnem Datenspeicherelement ist kritisch zu hinterfragen – schließlich ist der damit verbundene Aufwand immens, da die Generierung und Prüfung der Signaturen großen Rechenaufwand mit sich bringt. In den meisten Fällen wird es ausreichen, die Kommunikationsverbindungen zwischen Sensoren und Datenverarbeitungseinheiten und zwischen Datenverarbeitungseinheiten und Aktoren mit Signaturen zu versehen und damit deren Authentizität nur auf den Übertragungswegen sicherzustellen.

Die Nutzung von derart feingranularen und aufwendigen Authentifizierungsmaßnahmen wird sich nur in Anwendungsfällen rechtfertigen lassen, bei denen mit verschiedensten Arten von Angriffen auch innerhalb der Datenverarbeitungseinheiten selbst zu rechnen ist, wie es z. B. bei Chipkarten der Fall ist. Entsprechende Angriffsarten werden in [73] ausführlich vorgestellt.

Die Ziele solcher Angriffe sind nach [73]

- das Ausspähen von in den Karten gespeicherten, geheim gehaltenen Daten, wie z. B. kryptographische Schlüssel oder
- die Veränderung von auf der Karte gespeicherten Daten, um z. B. Guthaben zu erhöhen oder erweiterte Zugangsberechtigungen zu erhalten.

Dabei werden in [73] die folgenden Angriffskategorien identifiziert:

- manipulative Angriffe, bei denen Manipulationen am Chip selbst vorgenommen werden,
- observative Angriffe, bei denen der Chip selbst bzw. sein Verhalten untersucht werden und

- semi-invasive Angriffe, bei denen versucht wird, über verschiedenste Angriffsmethoden das Verhalten des Chips im Sinne des Angreifers zu beeinflussen.

Besonders zur Erkennung der letztgenannten Angriffsart, den semi-invasiven Angriffen, bei denen das Verhalten des Chips oder Signale innerhalb des Chips manipuliert werden sollen, könnte eine Signaturkennung genutzt werden. Es dürfte für einen Angreifer sehr schwierig sein, neben den gewünschten Manipulationen zeitgleich auch die Signaturkennungen so zu beeinflussen, dass diese der sich ergebenden Signatur der manipulierten Daten entspricht. An dieser Stelle sollte – auch wenn es nicht im Fokus dieser Arbeit liegt – für derartige Anwendungen darüber nachgedacht werden, nicht nur Daten, sondern auch Befehle mit einer Signaturkennung zu versehen, um auch hier Manipulationsversuche in größerem Umfang aufdecken zu können.

Die Signaturkennung S ermöglicht die Erkennung der in Tabelle 4.14 gezeigten Fehler- und Angriffsarten.

Wird die Adresse AD eines Datenspeicherelements analog zur Integritätsprüfungskennung IP in die Prüfung der Integrität eines Datenspeicherelements mit einbezogen, so lassen sich falsche Operanden, die statt der gewünschten Operanden aus dem Speicher gelesen wurden, zuverlässig erkennen.

Fehlgeleitete Daten können dann erkannt werden, wenn der Empfänger bei Anwendung des öffentlichen Schlüssels der Quelle, von der er Daten erwartet, die Signatur der fehlgeleiteten Daten nicht verifizieren kann. Durch Störungen oder Manipulation verfälschte Daten können durch die Anwendung der Signaturkennung aufgedeckt werden. Die Nutzung nicht initialisierter Daten kann dann erkannt werden, wenn diesen keine gültige Signatur zugewiesen ist. Wiedereinspielungsattacken können zusammen mit den Zeitschritt- ZS oder Fristkennungen FR sicher erkannt werden.

4.3.13 Redundante diversitäre arithmetisch-logische Einheit

Eine Fehlerkategorie kann mit den bisher eingeführten Kennungen nur unzureichend abgedeckt werden: „Fehlerhafte Operationen“. Dazu gehören

- das Heranziehen falscher Operanden,
- die Anwendung eines falschen Operators und
- die Erzeugung fehlerhafter Ergebnisse

Tabelle 4.14: Fehlererkennung durch die S-Kennung

Fehlerart	**Erkennbarkeit**
Inkompatible Datentypen	nein
Inkompatible Einheiten	nein
Wertebereichsunter- bzw. -überschreitung	nein
Genauigkeitsproblem	nein
Falsche Operandenauswahl	ja (mit AD)
Falsche Operatorauswahl	nein
Fehlerhaftes Operationsergebnis	nein
Fristüberschreitung	nein
Zyklusunterschreitung	nein
Zyklusüberschreitung	nein
Verlorengegangene Datenaktualisierung	nein
Synchronisationsfehler oder unvollständige Datenübertragung	nein
Pufferunter- oder -überläufe	nein
Fehlerhafter Datenfluss (falsche Adressaten, ...)	(ja)
Duplizierte Daten	nein
Durch Fehler oder Störungen verfälschte Daten	ja
Fehlerhafter Datenzugriff (fehlende Zugriffsrechte)	nein
Nutzung nicht initialisierter Daten	(ja)
Angriffsart	
Gezielt verfälschte Daten	ja
Wiedereinspielungsattacke	ja (mit ZS, FR, ZY)

AD: Einbeziehung der Datenspeicherelementadresse; ZS: Zeitschrittkennung;
FR: Fristkennung; ZY: Zykluszeitkennung

bei der Ausführung von Operationen. Während die Verwendung falscher Operanden schon großteils durch die Einbeziehung der Datenspeicherelementadresse oder eines entsprechenden Identifikators in die Integritätsprüfung IP bzw. die Signatur S aufgedeckt werden kann, können die beiden anderen Fehlerarten mit den bisherigen Mitteln nur in sehr begrenztem Umfang erkannt werden. Um auch diese Fehlerarten sicher erkennen zu können, wird die arithmetisch-logische Einheit ALE, engl. „arithmetic logic unit ALU“, in einer Datenspezifikationsarchitektur DSA redundant und diversitär aufgebaut.

4.3.13.1 Realisierung der redundanten diversitären arithmetisch-logischen Einheit

Dem Vorbild der Prozessoren für sicherheitsgerichtete Anwendungen in Kapitel 3.2.1 folgend, soll die Realisierung der redundanten ALE wie folgt geschehen:

- die redundante ALE wird räumlich möglichst weit entfernt und gedreht auf dem Die des DSA-Prozessors untergebracht, um räumliche Diversität zu erreichen,
- die redundante ALE wird zeitlich um zwei Takte versetzt betrieben, um zeitliche Diversität zu gewährleisten und
- die Ergebnisse beider ALE werden miteinander verglichen und bei Nichtübereinstimmung wird ein Ausnahmefehler generiert.

Zusätzlich wird gefordert, die beiden ALE diversitär aufzubauen, um etwaige Entwurfs- und Implementierungsfehler innerhalb der ALE aufdecken zu können und zu verhindern, dass im Falle eines solchen Fehlers beide ALE die identischen falschen Ergebnisse ausgeben, die dann durch einen Vergleich nicht mehr als fehlerhaft zu erkennen wären.

4.3.13.2 Evaluation der redundanten diversitären arithmetisch-logischen Einheit

Die redundante diversitäre Ausführung der ALE erlaubt es, Fehlfunktionen der ALE aufzudecken, was eine durch die bisher vorgestellten Fehlererkennungsmerkmale nur unzureichend abgedeckte Fehlerkategorie ist. Die Erkennbarkeit der 20 in Kapitel 2.4 vorgestellten Fehler- und Angriffsarten wird in Tabelle 4.15 dargestellt.

Die Verarbeitung falscher Operanden kann durch eine redundant diversitäre ALE nur dann aufgedeckt werden, wenn der Fehler, der zur falschen Auswahl der Operanden führte, innerhalb einer der beiden ALE aufgetreten ist, da sich dann – sofern die Operanden verschiedene Datenwerte beinhalten – unterschiedliche Ausgaben ergeben, die durch den Vergleicher aufgedeckt werden. Ist ein Adressierungsfehler auf dem gemeinsamen Datenbus die Ursache der Falschauswahl, so werden beide ALE die geforderte Operation auf den identischen falschen Operatoren ausführen und entsprechend identische falsche Ergebnisse ausgeben, weshalb der Vergleicher den Fehler in diesem Fall nicht entdecken kann.

Tabelle 4.15: Fehlererkennung durch die redundante diversitäre ALE

Fehlerart	**Erkennbarkeit**
Inkompatible Datentypen	nein
Inkompatible Einheiten	nein
Wertebereichsunter- bzw. -überschreitung	nein
Genauigkeitsproblem	nein
Falsche Operandenauswahl	(ja)
Falsche Operatorauswahl	ja
Fehlerhaftes Operationsergebnis	ja
Fristüberschreitung	nein
Zyklusunterschreitung	nein
Zyklusüberschreitung	nein
Verlorengegangene Datenaktualisierung	nein
Synchronisationsfehler oder unvollständige Datenübertragung	nein
Pufferunter- oder -überläufe	nein
Fehlerhafter Datenfluss (falsche Adressaten, ...)	nein
Duplizierte Daten	nein
Durch Fehler oder Störungen verfälschte Daten	nein
Fehlerhafter Datenzugriff (fehlende Zugriffsrechte)	nein
Nutzung nicht initialisierter Daten	nein
Angriffsart	
Gezielt verfälschte Daten	nein
Wiedereinspielungsattacke	nein

Die Durchführung einer falschen Operation, also zum Beispiel die Anwendung einer Addition statt einer Subtraktion auf die Operanden kann eine redundante diver-

sitäre ALE zuverlässig aufdecken. Gleiches gilt für Fehler innerhalb der ALE, die dazu führen, dass das Ergebnis einer Operation fehlerhaft ist. Beide Fehlerarten können durch den Vergleich der Ergebnisse beider ALE erkannt werden.

4.4 Übersicht der Kennungen in Daten- und Befehlsspeicherelementen

Werden alle vorgestellten Arten von Kennungen zur Erkennung von Fehlern bei der Realisierung einer Datenspezifikationsarchitektur genutzt, so ergeben sich die in den Abbildungen 4.57 und 4.58 gezeigten Aufbauten von Daten- und Befehlsspeicherelementen, die systemweit einheitlich sind. Die angedeuteten Größen der einzelnen Kennungen sind dabei nicht exakt maßstäblich, sondern nur grob der Größe der jeweiligen Kennung nachempfunden.

Die Datenspeicherelemente, gezeigt in Abbildung 4.57, bestehen aus

- dem Datenwert W, der bei Messwertdatentypen aus den beiden Intervallgrenzen W_{min} und W_{max} und bei allen anderen Datentypen nur aus dem Datenwert W im unteren Teil des Wertefelds besteht,
- der Wertebereichskennung WB, bestehend aus den Teilkennungen W_{unten} und W_{oben},
- der Datentypkennung DT, bestehend aus den Teilkennungen Basisdatentyp DT, Subdatentyp SDT und der Typberechtigungen TB,
- der Einheitenkennung EI,
- der Zugriffsrechtekennung ZR, bestehend aus den Teilkennungen Modul- und Funktionsnummer MN und FN, Schreibrechte SR und Initialisierungstatus IS,
- der Verarbeitungswegkennung VW, bestehend aus den Teilkennungen Quell-Q, Verarbeitungsweg- VW und Zielkennung Z,
- der Zeitschrittkennung ZS, bestehend aus den Teilkennungen Präsenzbit P und Zeitschrittangabe ZS,
- der Fristkennung FR,
- der Zykluszeitkennung ZY, bestehend aus Identifikator, frühestem und spätestem zulässigen Aktualisierungszeitpunkt ZY_{min} und ZY_{max}, und
- der Integritätsprüfungskennung IP bzw. der Signaturkennung S.

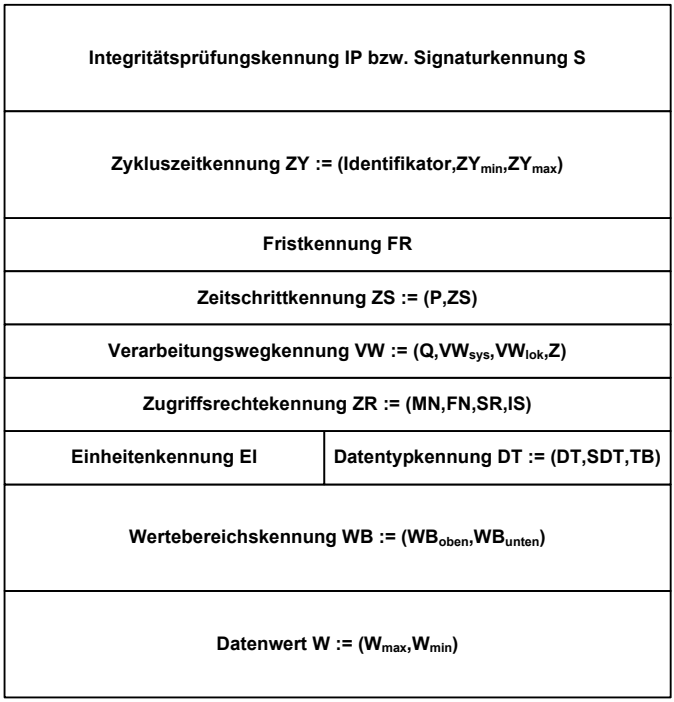


Abbildung 4.57: Aufbau der Datenspeicherelemente einer DSA

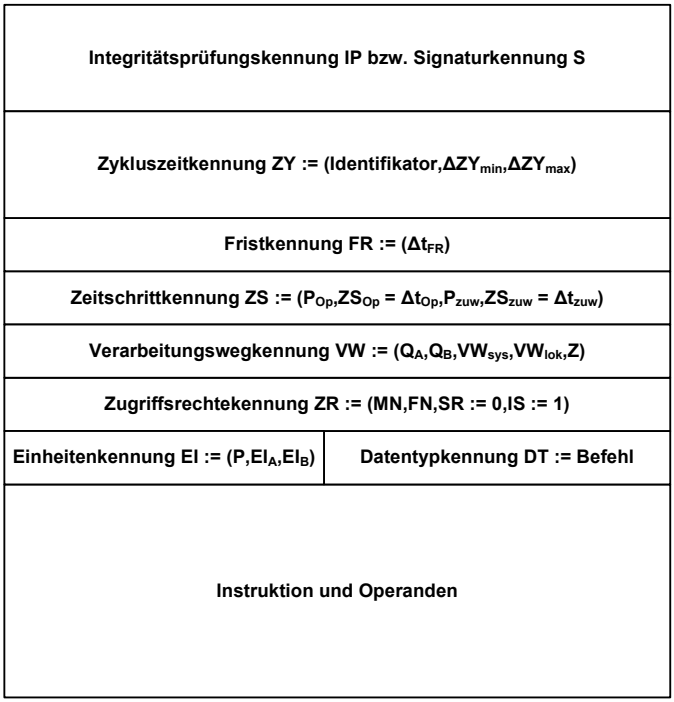


Abbildung 4.58: Aufbau der Befehlsspeicherelemente einer DSA

Die Befehlsspeicherelemente, deren Aufbau in Abbildung 4.58 dargestellt ist, beinhalten die Instruktion und die Angabe von Operanden bzw. Operandenadressen, sowie

- die Datentypkennung DT, die das Speicherelement explizit als Befehl ausweist,
- die Einheitenkennung EI, bestehend aus dem Präsenzbit P und den zu verifizierenden Einheitenkennungen EI_A und EI_B für bis zu zwei Quelloperanden der Instruktion,
- die Zugriffsrechte ZR mit den Teilkennungen Modul- und Funktionsnummer MN und FN, Schreibrechte SR (muss Null sein, d. h. nur lesbar), Initialisierungsstatus IS (muss stets Eins sein, um anzudeuten, dass die Inhalte verwend- und damit ausführbar sind),
- die Verarbeitungswegkennung VW, bestehend aus den Teilkennungen Quellkennungen Q_A und Q_B für bis zu zwei Quelloperanden, System- und Lokalteil der Verarbeitungswegkennung VW_{sys} bzw. VW_{lok} und der Zielkennung Z, die die zu prüfenden Bitmuster spezifizieren,
- die Zeitschrittkennung ZS, bestehend aus den Teilkennungen Präsenzbit der Zeitschrittdifferenz der Operanden P_{Op}, Zeitschrittdifferenz der Operanden Δt_{Op}, Präsenzbit der Zeitschrittdifferenz der Zuweisung P_{zuw} und Zeitschrittdifferenz der Zuweisung Δt_{zuw},
- die Fristkennung FR, welche die neu zu setzende relative Frist Δt_{FR} oder den Wert Null enthält,
- die Zykluszeitkennung, die den im Ergebnis der Operation zu setzenden Identifikator, sowie die zwei relativen Zeitangaben enthält, die das Zeitfenster definieren, innerhalb dessen eine Aktualisierung des Datenwerts erfolgen soll, und
- die Integritätsprüfungskennung IP bzw. die Signaturkennung S.

Die Signatur von Befehlsspeicherelementen wurde im die Signaturkennung S beschreibenden Kapitel 4.3.12 nicht erwähnt, da der Fokus dieser Arbeit auf der Überwachung von Daten, ihren Eigenschaften und ihrem Weg innerhalb eines Systems liegt. Die Authentifizierung jedes einzelnen Befehls könnte jedoch in entsprechend anspruchsvollen Anwendungen zum Einsatz kommen. Sie ist – analog zur Authentizitätsprüfung jedes einzelnen Datenspeicherelements bei jedem Zugriff – mit erheblichen Laufzeitaufwänden zur Prüfung der jeweiligen Signaturen verbunden.

Weitere Bereiche innerhalb der Befehlsspeicherelemente können für befehlsspezifische Kennungen wie die in ISMA genutzten Sprungzielmarkierungen verwendet werden.

4.5 Übersicht der speziellen Register

Zusätzlich zu den im vorangegangenen Unterkapitel zusammengefassten Kennungen in Daten und Befehlsspeicherelementen wird der Registersatz einer Datenspezifikationsarchitektur DSA um die zwei in Abbildung 4.59 dargestellten speziellen Register

- das Zugriffsrechteregister ZRR, bestehend aus den Teilregistern Aktuelle Modulnummer AMN und Aktuelle Funktionsnummer AFN, und
- das Verarbeitungswegregister VWR, welches aus den Teilregistern Quellenregister für bis zu zwei Operanden Q_AR und Q_BR, den System- und Lokalteilregistern des Verarbeitungswegs $VW_{sys}R$ und $VW_{lok}R$ und dem Zielregister ZR.

zur Prüfung von Kennungsinhalten der zu verarbeitenden Daten erweitert.

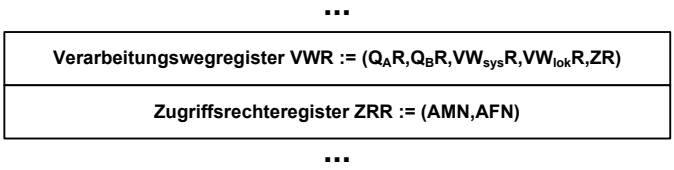


Abbildung 4.59: Spezielle Register im Registersatz der DSA

4.6 Pseudocode einer Instruktion

Zur Veranschaulichung des Funktionsprinzips einer Datenspezifikationsarchitektur und des Umfangs der durch die Hardware parallel zur eigentlichen Ausführung einer Operation durchgeführten Prüfungen, wird in der folgenden Auflistung der Pseudocode der Addition

$$C = A + B$$

gezeigt. Auch wenn die einzelnen Prüfungen im Pseudocode sequentiell dargestellt sind, finden sie – soweit möglich – zeitgleich, also parallelisiert statt. Diese Parallelisierung stellt auch im Fehlerfall kein Problem dar: Selbst wenn eine Operation z. B. mit fehlerhaften Operanden durchgeführt wird und dieser Umstand erst einige Takte versetzt durch eine parallel durchgeführte Prüfung festgestellt wird, wird dennoch ein Ausnahmefehler generiert, der zum Programmabbruch führt, wodurch das fehlerhafte Ergebnis nicht mehr zu gefährlichen Ausgaben der Datenverarbeitungseinheit führen kann.

Der Additionsbefehl ADD besitzt drei Operanden in Form der Speicheradressen A und B der beiden Summanden, sowie der Speicheradresse C des Zieldatenspeicherelements, in welches das Ergebnis gespeichert werden soll.

Die Darstellung des Pseudocodes der Addition und der mit ihr verbundenen Prüfungen erfolgt in dem in Kapitel 4.3.1.3 erläuterten Format. Obwohl die Integritätsprüfungskennung IP überflüssig ist, wenn eine Signaturkennung S zum Einsatz kommt, werden im Pseudocode der Vollständigkeit halber beide Kennungen eingesetzt.

Aus Platzgründen wird in der Pseudocodeauflistung Identifikator als Id abgekürzt.

```
;=== Pseudocode der Additionsinstruktion ADD ===
ADD A, B, C :=

;=== Prüfung der Wertebereiche ===
;=== (Kapitel 4.3.3) ===
WENN DT.DT([A]) ∈ Messwertdatentypen DANN
  WENN [W_min([A]);W_max([A])] ⊈ WB([A]) DANN
    Generierung_Ausnahmefehler;
  ENDEWENN
SONST
  WENN W([A]) ∉ WB([A]) DANN
    Generierung_Ausnahmefehler;
  ENDEWENN
ENDEWENN
```

```
WENN DT.DT([B]) ∈ Messwertdatentypen DANN
  WENN [W_min([B]);W_max([B])] ⊈ WB([B]) DANN
    Generierung_Ausnahmefehler;
  ENDEWENN
SONST
  WENN W([B]) ∉ WB([B]) DANN
    Generierung_Ausnahmefehler;
  ENDEWENN
ENDEWENN

WENN WB([A]) ⊈ DT.DT([A]) DANN
  Generierung_Ausnahmefehler;
ENDEWENN

WENN WB([B]) ⊈ DT.DT([B]) DANN
  Generierung_Ausnahmefehler;
ENDEWENN

WENN WB([C]) ⊈ DT.DT([C]) DANN
  Generierung_Ausnahmefehler;
ENDEWENN

;=== Prüfung der Kompatibilität der Datentypen ===
;=== (Kapitel 4.3.4) ===
WENN DT.DT([A]) ≠ DT.DT([B]) DANN
  Generierung_Ausnahmefehler;
ENDEWENN

WENN DT.SDT([A]) ≠ DT.SDT([B]) DANN
  Generierung_Ausnahmefehler;
ENDEWENN

WENN DT.TB([A]) ≠ DT.TB([B]) DANN
  Generierung_Ausnahmefehler;
ENDEWENN

WENN DT.DT([A]) ≠ DT.DT([C]) DANN
  Generierung_Ausnahmefehler;
ENDEWENN
```

```
WENN DT.SDT([A]) ≠ DT.SDT([C]) DANN
  Generierung_Ausnahmefehler;
ENDEWENN

WENN DT.TB([A]) ≠ DT.TB([C]) DANN
  Generierung_Ausnahmefehler;
ENDEWENN

;=== Prüfung der Durchführbarkeit der Operation ADD ===
;=== (Kapitel 4.3.4) ===
WENN DT.TB([A]) UND Bitmaske(ADD) ≠ Bitmaske(ADD) DANN
  Generierung_Ausnahmefehler;
ENDEWENN

;=== Prüfung der Kompatibilität der Einheiten ===
;=== (Kapitel 4.3.5) ===
WENN EI([A]) ≠ EI([B]) DANN
  Generierung_Ausnahmefehler;
ENDEWENN

WENN EI([A]) ≠ EI([C]) DANN
  Generierung_Ausnahmefehler;
ENDEWENN

;=== Verifikation der Einheiten ===
;=== (Kapitel 4.3.5) ===
WENN EI.P([Befehl]) = 1 DANN
  WENN EI([A]) ≠ EI.EI_A([Befehl]) DANN
    Generierung_Ausnahmefehler;
  ENDEWENN
  WENN EI([B]) ≠ EI.EI_B([Befehl]) DANN
    Generierung_Ausnahmefehler;
  ENDEWENN
ENDEWENN
```

```
;=== Prüfung der Zugriffsrechte ===
;=== (Kapitel 4.3.6) ===
WENN ZR.MN([A]) ≠ [ZRR.AMN] DANN
  Generierung_Ausnahmefehler;
ENDEWENN

WENN ZR.FN([A]) ≠ [ZRR.AFN] DANN
  Generierung_Ausnahmefehler;
ENDEWENN

WENN ZR.MN([B]) ≠ [ZRR.AMN] DANN
  Generierung_Ausnahmefehler;
ENDEWENN

WENN ZR.FN([B]) ≠ [ZRR.AFN] DANN
  Generierung_Ausnahmefehler;
ENDEWENN

WENN ZR.MN([C]) ≠ [ZRR.AMN] DANN
  Generierung_Ausnahmefehler;
ENDEWENN

WENN ZR.FN([C]) ≠ [ZRR.AFN] DANN
  Generierung_Ausnahmefehler;
ENDEWENN

WENN ZR.IS([A]) ≠ 1 DANN
  Generierung_Ausnahmefehler;
ENDEWENN

WENN ZR.IS([B]) ≠ 1 DANN
  Generierung_Ausnahmefehler;
ENDEWENN

WENN ZR.SR([C]) ≠ 1 DANN
  Generierung_Ausnahmefehler;
ENDEWENN
```

```
;=== Prüfung und Setzen der Verarbeitungswege VW ===
;=== (Kapitel 4.3.7) ===
WENN [VWR.QAR] ≠ 0 DANN
  WENN VW.Q([A]) ≠ [VWR.QAR] DANN
    Generierung_Ausnahmefehler;
  ENDEWENN
ENDEWENN

WENN VW.QA([Befehl]) ≠ 0 DANN
  WENN VW.Q([A]) ≠ VW.QA([Befehl]) DANN
    Generierung_Ausnahmefehler;
  ENDEWENN
ENDEWENN

WENN VW.VWsys([A]) UND [VWR.VWsysR] ≠ [VWR.VWsysR] DANN
  Generierung_Ausnahmefehler;
ENDEWENN

WENN VW.VWsys([A]) UND VW.VWsys([Befehl]) ≠ VW.VWsys([Befehl]) DANN
  Generierung_Ausnahmefehler;
ENDEWENN

WENN VW.VWlok([A]) UND [VWR.VWlokR] ≠ [VWR.VWlokR] DANN
  Generierung_Ausnahmefehler;
ENDEWENN

WENN VW.VWlok([A]) UND VW.VWlok([Befehl]) ≠ VW.VWlok([Befehl]) DANN
  Generierung_Ausnahmefehler;
ENDEWENN

WENN [VWR.QBR] ≠ 0 DANN
  WENN VW.Q([B]) ≠ [VWR.QBR] DANN
    Generierung_Ausnahmefehler;
  ENDEWENN
ENDEWENN
```

```
WENN VW.Q_B([Befehl]) ≠ 0 DANN
  WENN VW.Q([B]) ≠ VW.Q_B([Befehl]) DANN
    Generierung_Ausnahmefehler;
  ENDEWENN
ENDEWENN

WENN VW.VW_sys([B]) UND [VWR.VW_sysR] ≠ [VWR.VW_sysR] DANN
  Generierung_Ausnahmefehler;
ENDEWENN

WENN VW.VW_sys([B]) UND VW.VW_sys([Befehl]) ≠ VW.VW_sys([Befehl]) DANN
  Generierung_Ausnahmefehler;
ENDEWENN

WENN VW.VW_lok([B]) UND [VWR.VW_lokR] ≠ [VWR.VW_lokR] DANN
  Generierung_Ausnahmefehler;
ENDEWENN

WENN VW.VW_lok([B]) UND VW.VW_lok([Befehl]) ≠ VW.VW_lok([Befehl]) DANN
  Generierung_Ausnahmefehler;
ENDEWENN

VW.Q([C]) := VW.Q([A]) ODER VW.Q([B]);
VW.VW_sys([C]) := VW.VW_sys([A]) UND VW.VW_sys([B]);
VW.VW_lok([C]) := VW.VW_lok([A]) UND VW.VW_lok([B]);
VW.Z([C]) := VW.Z([A]) UND VW.Z([B]);

;=== Prüfung und Setzen der Zeitschrittkennungen ZS ===
;=== (Kapitel 4.3.8) ===
WENN ZS.P_0p([Befehl]) = 1 DANN
  WENN ZS.P([A]) = ZS.P([B]) = 1 DANN
    WENN ZS.ZS([A]) - ZS.ZS([B]) ≠ ZS.Δt_0p([Befehl]) DANN
      Generierung_Ausnahmefehler;
    ENDEWENN
  ENDEWENN
ENDEWENN
```

```
WENN ZS.P([A]) = ZS.P([B]) = 1 DANN
  WENN ZS.ZS([A]) - ZS.ZS([B]) ≥ 0 DANN
    ZS.P([Ergebnis]) := 1;
    ZS.ZS([Ergebnis]) := ZS.ZS([A]);
  SONST
    ZS.P([Ergebnis]) := 1;
    ZS.ZS([Ergebnis]) := ZS.ZS([B]);
  ENDEWENN
SONST WENN ZS.P([A]) = 1 DANN
  ZS.P([Ergebnis]) := 1;
  ZS.ZS([Ergebnis]) := ZS.ZS([A]);
SONST WENN ZS.P([B]) = 1 DANN
  ZS.P([Ergebnis]) := 1;
  ZS.ZS([Ergebnis]) := ZS.ZS([B]);
SONST
  ZS.P([Ergebnis]) := 0;
ENDEWENN

WENN ZS.P([Ergebnis]) = 1 ∧ ZS.+1([Befehl]) = 1 DANN
  ZS.ZS([Ergebnis]) := ZS.ZS([Ergebnis]) + 1;
ENDEWENN

WENN ZS.P_zuw([Befehl]) = 1 DANN
  WENN ZS.P([C]) = ZS.P([Ergebnis]) = 1 DANN
    WENN ZS.ZS([C]) - ZS.ZS([Ergebnis]) ≠ ZS.Δt_zuw([Befehl]) DANN
      Generierung_Ausnahmefehler;
    ENDEWENN
  ENDEWENN
ENDEWENN

;=== Prüfung und Setzen der Fristen ===
;=== (Kapitel 4.3.9) ===
WENN Aktuelle_Zeit > FR([A]) DANN
  Generierung_Ausnahmefehler;
ENDEWENN
```

```
WENN Aktuelle_Zeit > FR([B]) DANN
  Generierung_Ausnahmefehler;
ENDEWENN

WENN FR.ΔtFR([Befehl]) = 0 DANN
  WENN FR([A]) < FR([B]) DANN
    FR([C]) := FR([A]);
  SONST
    FR([C]) := FR([B]);
  ENDEWENN
SONST
  FR([C]) := Aktuelle_Uhrzeit + FR.ΔtFR([Befehl]);
ENDEWENN

;=== Senden der Zykluszeitkennungsdaten an die ZÜE ===
;=== (Kapitel 4.3.10) ===
WENN ZY.Id([A]) ≠ 0 DANN
  ZÜE_Nachricht := (ZY.Id([A]), ZY.ZYmin([A]), ZY.ZYmax([A]));
ENDEWENN

WENN ZY.Id([B]) ≠ 0 DANN
  ZÜE_Nachricht := (ZY.Id([B]), ZY.ZYmin([B]), ZY.ZYmax([B]));
ENDEWENN

;=== Setzen der Zykluszeitkennung ZY des Ergebnisses ===
;=== (Kapitel 4.3.10) ===
WENN ZY.Id([Befehl]) ≠ 0 DANN
  ZY.Id([C]) := ZY.Id([Befehl]);
  WENN ZY.ΔZYmin([Befehl]) = 0 DANN
    ZY.ZYmin([C]) := 0;
  SONST
    ZY.ZYmin([C]) := Aktuelle_Uhrzeit + ZY.ΔZYmin([Befehl]);
  ENDEWENN
  WENN ZY.ΔZYmax([Befehl]) = 0 DANN
    ZY.ZYmax([C]) := 0;
  SONST
    ZY.ZYmax([C]) := Aktuelle_Uhrzeit + ZY.ΔZYmax([Befehl]);
  ENDEWENN
```

```
SONST WENN ZY.ΔZY_min([Befehl]) = ZY.ΔZY_max([Befehl]) = 0 DANN
  ZY.Id([C]) := 0;
  ZY.ZY_min([C]) := 0;
  ZY.ZY_max([C]) := 0;
SONST
  Generierung_Ausnahmefehler;
ENDEWENN

;=== Prüfung der Datenintegrität anhand IP ===
;=== (Kapitel 4.3.11) ===
WENN I_prüf([A] EXODER Adresse(A)) ≠ gültig DANN
  Generierung_Ausnahmefehler;
ENDEWENN

WENN I_prüf([B] EXODER Adresse(B)) ≠ gültig DANN
  Generierung_Ausnahmefehler;
ENDEWENN

WENN I_prüf([C] EXODER Adresse(C)) ≠ gültig DANN
  Generierung_Ausnahmefehler;
ENDEWENN

;=== Prüfung der Datenintegrität anhand Signatur S ===
;=== (Kapitel 4.3.12) ===
WENN S_prüf([A] EXODER Adresse(A), öS) ≠ gültig DANN
  Generierung_Ausnahmefehler;
ENDEWENN

WENN S_prüf([B] EXODER Adresse(B), öS) ≠ gültig DANN
  Generierung_Ausnahmefehler;
ENDEWENN

WENN S_prüf([C] EXODER Adresse(C), öS) ≠ gültig DANN
  Generierung_Ausnahmefehler;
ENDEWENN
```

```
;=== Ausführung der Addition ===
;=== (Intervallarithmetik Kapitel 4.3.2) ===
WENN DT.DT([A]) ∈ Messwertdatentypen ∧
     DT.DT([B]) ∈ Messwertdatentypen DANN
  W_min([C]) := W_min([A]) + W_min([B]);
  W_max([C]) := W_max([A]) + W_max([B]);
SONST WENN DT.DT([A]) ∈ Messwertdatentypen DANN
  W_min([C]) := W_min([A]) + W([B]);
  W_max([C]) := W_max([A]) + W([B]);
SONST WENN DT.DT([B]) ∈ Messwertdatentypen DANN
  W_min([C]) := W([A]) + W_min([B]);
  W_max([C]) := W([A]) + W_max([B]);
SONST
  W([C]) := W([A]) + W([B]);
ENDEWENN

;=== Setzen des Initialisierungsstatus ===
;=== (Kapitel 4.3.6) ===
ZR.IS([C]) := 1;

;=== Prüfung, ob Ergebnis innerhalb Zielwertebereich liegt ===
;=== (Kapitel 4.3.3) ===
WENN DT.DT([C]) ∈ Messwertdatentypen DANN
  WENN [W_min([C]);W_max([C])] ⊈ WB([C]) DANN
    Generierung_Ausnahmefehler;
  ENDEWENN
SONST
  WENN W([C]) ∉ WB([C]) DANN
    Generierung_Ausnahmefehler;
  ENDEWENN
ENDEWENN

;=== Übertragung des Zeitschritts in das Zielspeicherelement ===
;=== (Kap. 4.3.8) ===
ZS.P([C]) := ZS.P([Ergebnis]);
ZS.ZS([C]) := ZS.ZS([Ergebnis]);
```

```
;=== Setzen der Integritätsprüfung des Ergebnisses ===
;=== (Kapitel 4.3.11) ===
IP([C]) := I_gen([C] EXODER Adresse(C));

;=== Abschließende Signatur der Ergebnisses ===
;=== (Kapitel 4.3.12) ===
S([C]) := S_gen([C] EXODER Adresse(C), gS);
```

4.7 Anforderungen an die Systemkomponenten

In den folgenden Unterkapiteln wird beschrieben, welche speziellen Anforderungen an nach dem Vorbild einer Datenspezifikationsarchitektur entwickelte Systemkomponenten zu stellen sind. Dabei werden

- die Schnittstellen zu konventionellen Systemkomponenten und
- die Nutzung hochpräziser synchronisierter Uhren

vorgestellt und näher erläutert.

4.7.1 Schnittstellen zu konventionellen Systemkomponenten

Es ist anzunehmen, dass in Systemen, die Systemkomponenten mit den vorgestellten Merkmalen einer Datenspezifikationsarchitektur DSA nutzen, auch konventionelle Komponenten eingebracht werden, auch wenn diese tunlichst auf ein Minimum zu reduzieren sind und keine oder nur wenig sicherheitskritische Funktionen übernehmen sollten. Diese konventionellen Gerätschaften werden Daten generieren und kommunizieren, ohne deren Eigenschaften dabei in den geforderten expliziten Kennungen bekanntzugeben. Es ist daher zu fordern, dass die Daten von allen DSA-Systemkomponenten vor der Verwendung mit allen bekannten Eigenschaften zu markieren sind, um bei der nachfolgenden Verwendung so viele Fehler- und Angriffsarten wie möglich aufdecken zu können. Die zu setzenden Dateneigenschaften könnten den Systemkomponenten z. B. im Zuge einer Konfiguration bekanntgeben werden.

4.7.2 Hochpräzise synchronisierte Uhren

Zur Bewertung der in den Datenspeicherelementen der Datenspezifikationsarchitektur angegebenen Fristen, also

- der Fristkennung FR und
- der Zykluszeitkennung ZY,

ist es notwendig, dass alle Systemkomponenten hochpräzise und synchronisierte Uhren aufweisen. Als Zeitbasis können z. B. satellitenbasierte Systeme wie GPS, GLONASS oder Galileo dienen, wobei zu beachten ist, dass

- die entsprechenden Signale nicht immer mit hinreichender Signalstärke zu empfangen sind, z. B. innerhalb von Gebäuden, und dass
- zumindest die Betreiber des GPS immer wieder damit drohen, im Konfliktfall die Systeme – zumindest teilweise – abzuschalten.

Es ist davon auszugehen, dass es nicht wirtschaftlich ist, sämtliche Quellen, Datenverarbeitungseinheiten und Senken jeweils mit einem entsprechenden Empfänger zu versehen. Daher müssen Zeitsynchronisationsmechanismen zum Einsatz kommen, wie z. B. die in [35] vorgeschlagene Zeitsynchronisation in einem Doppelringbussystem.

4.8 Konfiguration der Systemkomponenten

Aufgrund der detaillierten Spezifikation der in Kapitel 4.2 identifizierten Dateneigenschaften ist es notwendig, das Wissen über diese Eigenschaften an alle Systemkomponenten zu verteilen. Damit nun nicht für jede Anwendung neue Softwarepakete für die eingesetzten Sensoren, Datenverarbeitungseinheiten und Aktoren übersetzt und getestet werden müssen, wird für ein auf einer Datenspezifikationsarchitektur basierendes System ein Konfigurationsmechanismus ähnlich der Konfiguration der sicherheitsgerichteten Feldbusprotokolle PROFIsafe [56] und CIP Safety [55] vorgeschlagen. Dabei werden die notwendigen Dateneigenschaften den Systemkomponenten in Form von Konfigurationsdateien zur Verfügung gestellt. Für die einzelnen Systemkomponenten werden dabei unterschiedliche Informationen benötigt, die in den folgenden Unterkapiteln identifiziert werden.

4.8.1 Konfiguration der Datenquellen

Für die Konfiguration der Datenquellen – also z. B. von Sensoren – werden die folgenden Dateneigenschaften benötigt, die dem Gerät durch eine entsprechende Konfigurationsdatei zur Verfügung gestellt werden müssen:

- die Quellkennung Q der Quelle, die in die Verarbeitungswegkennung VW aller erzeugten Daten der Quelle eingeht,
- für alle zu generierenden Daten:
  - die vorbestimmten Verarbeitungswege durch das System, die in die Verarbeitungswegkennung VW in Form der Komponenten VW_{sys}, – sofern möglich –VW_{lok} und Zielkennung Z eingehen,
  - die zu setzende relative Frist, mit der die Daten bei ihrer Erzeugung versehen werden, die auf die aktuelle Zeit addiert wird, um in den generierten Daten die absolute Zeitangabe in Form der Fristkennung FR zu setzen,
  - falls notwendig, die Spezifikation des einzusetzenden Datentyps DT zur Darstellung des Datenwerts W, des Subdatentyps SDT, der Typberechtigungen TB und des Wertebereichs WB,
  - den in den Daten zu vermerkenden gültigen Wertebereich, der in der Wertebereichskennung WB hinterlegt wird und eine Plausibilitätsprüfung auf Empfängerseite gestattet,
- im Falle des Einsatzes der Signaturkennung S ein Schlüsselpaar bestehend aus
  - einem geheimen Schlüssel zur Generierung der kryptographischen Signaturen der erzeugten Daten und
  - einem öffentlichen Schlüssel zur Prüfung der Signaturen eigener Daten.

Sollten Datenquellen auch Daten von anderen Systemkomponenten erwarten, so müssen ggf. weitere Vorgaben durch die Konfigurationsdatei erfolgen, die der Beschreibung der Konfiguration von Datensenken zu entnehmen sind.

4.8.2 Konfiguration der Datenverarbeitungseinheiten

Die Datenverarbeitungseinheiten benötigen ebenfalls einen umfangreichen Satz an Informationen, um die entsprechenden Prüfungen zu gestatten:

- die Verarbeitungswegkennung VW_{sys}, die die Datenverarbeitungseinheit identifiziert und die in der Verarbeitungswegkennung VW der Daten erwartet wird,
- für alle zu empfangenden Daten:
  - die Quellkennung Q der Quelle, die in der Verarbeitungswegkennung VW erwartet wird,
  - die minimale und maximale Zykluszeit, falls es sich um zyklisch übermittelte Daten handelt; diese Zeiten werden von der Zyklusüberwachungseinheit ZÜE genutzt,
  - falls notwendig, die Datentypen DT, Subdatentypen SDT und Typberechtigungen TB,
- im Falle des Einsatzes einer Signaturkennung S
  - die öffentlichen Schlüssel aller Quellen, deren Daten verarbeitet werden sollen,
  - einen eigenen geheimen Schlüssel, mit dem die erzeugten Verarbeitungsergebnisse signiert werden sollen und
  - einen eigenen öffentlichen Schlüssel zur Prüfung der erzeugten Daten.
- das gewünschte Verhalten im Falle der Generierung eines Ausnahmefehlers, also beispielsweise wie ein sicherer Zustand eingenommen und gehalten werden kann.

Die lokalen Anteile der Verarbeitungswegkennung VW, VW_{lok}, werden innerhalb des Softwareprojekts der Datenverarbeitungseinheit definiert und werden nicht durch die Konfigurationsdatei vorgegeben. Sollten diese jedoch bereits durch die Quellen in der VW-Kennung der Daten vorgegeben werden, so ist es notwendig, die entsprechenden Kennungen zu exportieren und in die Konfigurationsdateien der Quellen einzubetten.

Sollte eine Datenverarbeitungseinheit zusätzlich zur reinen Verarbeitung von Daten auch eigene Daten, wie z. B. Diagnosedaten, erzeugen, so ist die Konfigurationsdatei um die benötigten Vorgaben, die bei der Konfiguration von Datenquellen spezifiziert wurden, zu ergänzen.

In Kapitel 4.7.1 wird beschrieben, dass die von konventionellen Sensoren oder Komponenten übermittelten Daten möglicherweise nicht alle von einer Datenspezifikationsarchitektur vorgeschlagenen Dateneigenschaften in Kennungen abbilden oder sogar analoge Signale übermitteln. Ist dies der Fall, so muss die Konfigurationsdatei

das notwendige Wissen bereitstellen, um die Daten innerhalb der Datenverarbeitungseinheiten mit allen notwendigen Kennungen ausstatten zu können.

4.8.3 Konfiguration der Datensenken

Schlussendlich werden auch alle Datensenken innerhalb eines Systems mit einer Konfigurationsdatei parametriert, die die folgenden Parameter spezifizieren muss:

- die Zielkennung Z der Senke, die in der Verarbeitungswegkennung der empfangenen Daten erwartet wird,
- für alle zu empfangenden Daten:
  - die Quellkennung Q der Quelle, die in der Verarbeitungswegkennung VW erwartet wird,
  - ggf. die erwarteten Komponenten $VW_{sys,lok}$ in der Verarbeitungswegkennung der Daten,
  - die minimale und maximale Zykluszeit, falls es sich um zyklisch übermittelte Daten handelt; diese Zeiten werden von der Zyklusüberwachungseinheit ZÜE genutzt,
  - falls notwendig, die Datentypen DT, Subdatentypen SDT und Typberechtigungen TB,
- im Falle des Einsatzes einer Signaturkennung S die öffentlichen Schlüssel aller Quellen, deren Daten verarbeitet werden sollen.
  - die öffentlichen Schlüssel aller Datenverarbeitungseinheiten, deren Steuersignale verarbeitet werden sollen,
  - einen eigenen geheimen Schlüssel, mit dem interne Daten signiert werden und
  - einen eigenen öffentlichen Schlüssel zur Prüfung der erzeugten internen Daten,
- das gewünschte Verhalten im Falle der Generierung eines Ausnahmefehlers, also beispielsweise wie ein sicherer Zustand eingenommen und gehalten werden kann.

Falls die Datensenken auch Daten generieren können, z. B. wenn Stellgrößen durch die Datenverarbeitungseinheiten zurückgelesen oder Diagnoseinformationen generiert werden, so sind der Konfigurationsdatei entsprechende weitere Vorgaben hinzuzufügen, die der Beschreibung der Konfiguration von Datenquellen zu entnehmen sind.

4.8.4 Konfiguration der Systemüberwachungseinheit

Auch die in Kapitel 4.1.2 beschriebene Systemüberwachungseinheit SÜE wird über eine entsprechende Konfigurationsdatei parametriert. Diese spezifiziert:

- im Falle des Einsatzes einer Signaturkennung S die öffentlichen Schlüssel aller Datenverarbeitungseinheiten,
- das gewünschte Verhalten im Falle eines Ausfalls von Sensoren, Datenverarbeitungseinheiten oder Aktoren, ebenso wie bei Nichtübereinstimmung von durch die Datenverarbeitungseinheiten generierten Steuersignalen, beispielsweise, wie ein sicherer Zustand eingenommen und gehalten werden kann.

Neben diesen minimal notwendigen Informationen können weitere Festlegungen wie beispielsweise

- die erwarteten Quellkennungen und
- die erwarteten Zielkennungen

der Steuersignale vorgenommen werden. Dies ermöglicht der Systemüberwachungseinheit SÜE eine erweiterte Prüfung der von den Datenverarbeitungseinheiten gelieferten Steuersignale.

4.8.5 Erkennung konfigurationsbezogener Inkonsistenzen

Um sicherzustellen, dass alle Systemkomponenten zueinander passende Konfigurationsdateien verwenden, kann ein im sicherheitsgerichteten Feldbusprotokoll PROFIsafe [56] eingesetztes Verfahren angewandt werden, bei dem verschiedene Konfigurationsinformationen in die Prüfsummenberechnung eingehen. Es ist also vorstellbar, die Konfigurationsdaten mit einer Versionsnummer zu versehen, die von den Systemkomponenten in die Berechnung und Prüfung der Integritätsprüfung IP einbezogen werden. Dadurch können etwaige Abweichungen der Stände der Konfigurationsdateien aufgedeckt werden.

4.9 Anforderungen an Begutachtungen und Audits

In den verschiedenen Entwicklungsphasen innerhalb von Projekten werden Begutachtungen und Audits dazu genutzt, die Einhaltung von Vorgaben, die u. a. der Normengebung entstammen, sicherzustellen. Weitere Vorgaben, wie z. B. Programmierrichtlinien oder Einschränkungen der zulässigen Sprachmittel können dabei ebenfalls geprüft werden.

Bezogen auf die Fehlererkennungsmerkmale der Datenspezifikationsarchitektur sollten die folgenden Fragen in die entsprechenden Prüfungen eingehen:

- Werden alle Daten und deren Eigenschaften in der jeweiligen Phase des Entwicklungszyklus korrekt und in hinreichender Tiefe beschrieben?
- Werden alle für das Projekt relevanten Kennungsarten der Datenspezifikationsarchitektur genutzt?
- Werden die Kennungen in der vorgesehenen Art und in vollem Umfang genutzt?
  - Werden alle Messwerte durch Messwertdatentypen abgebildet und wird deren Genauigkeit korrekt abgebildet?
  - Werden Wertebereichseinschränkungen vorgenommen, die fehlerhafte Parameter und Rückgabewerte aufdecken können?
  - Werden alle Datentypen, die auf dem selben Basisdatentyp beruhen, aber eine andere semantische Bedeutung haben, durch Subdatentypen voneinander isoliert? Werden die zugelassenen Operationen auf eine sinnvolle Untermenge eingeschränkt?
  - Werden alle Werte mit korrekten Einheiten in der Einheitenkennung EI versehen? Wurde sichergestellt, dass die für die Potenz der jeweiligen SI-Basiseinheit zur Verfügung stehende Bitbreite in den Einheitenteilkennungen für die Anwendung ausreicht?
  - Werden die Zugriffsrechte in vollem Umfang inklusive der Schreibrechte SR und des Intialisierungsstatus IS genutzt?
  - Werden alle Daten mit einer gültigen Verarbeitungswegkennung VW ausgestattet, ohne pauschal alle Datenverarbeitungseinheiten und Senken als gültige Stationen zu markieren?

  - Werden diskrete Messwerte mit einer Zeitschrittangabe versehen und werden die temporalen Zusammenhänge von Operanden bei der Verarbeitung geprüft? Wurde sichergestellt, dass die maximale zu prüfende Zeitschrittdifferenz durch die Zeitschrittkennung ZS dargestellt werden kann?
  - Werden alle Daten mit begrenzter Gültigkeit mit sinnvollen Fristen ausgestattet, die Gültigkeitszeiträume also nicht länger als notwendig gestaltet?
  - Wird die Einhaltung der zeitlichen Grenzen aller zyklisch erfassten und übermittelten Daten mit Hilfe der Zykluszeitkennung ZY geprüft? Werden dabei sinnvolle Zeitfenster definiert?
  - Wurde die korrekte Entscheidung bzgl. der Nutzung der Integritätsprüfungskennung IP oder der Signaturkennung S basierend auf den individuellen Schutzanforderungen der Anwendung getroffen?
- Werden Redundanz- und Diversitätsarten richtig und in hinreichendem Umfang eingesetzt?
- Wird – sofern vorhanden – der sichere Zustand des Systems bzw. der einzelnen Systemkomponenten korrekt in den Konfigurationsdateien beschrieben?
- Sind die Konfigurationsdateien für die verschiedenen Systemkomponenten zueinander konsistent?

Die Nutzung der Kennungen auf die vorgesehene Art und in hinreichendem Umfang wird deshalb so betont, da es teilweise möglich ist, die Sicherheitsmechanismen der Datenspezifikationsarchitektur trotz Anwendung der Kennungen zu umgehen. Ein Beispiel für eine solche Umgehungsmaßnahme wäre, allen Datenwerten die Einheit 1 zuzuweisen, d. h. alle Potenzen der sieben SI-Basiseinheiten auf Null zu setzen. Eine weitere Art der Umgehung wichtiger Fehlererkennungsmaßnahmen könnte das Setzen der Gültigkeit der Daten auf extrem hohe Werte sein, um die Notwendigkeit der Berechnung sinnvoller Zeitgrenzen zu vermeiden.

4.10 Realisierung der Datenspezifikationsarchitektur als Datenflussarchitektur

Während der Betrachtung des Stands von Wissenschaft und Technik wurde in Kapitel 3.6 das Funktionsprinzip statischer und dynamischer Datenflussarchitekturen vorgestellt. Dabei wurde kritisiert, dass diese keine Merkmale zur Erkennung der

20 in Kapitel 2.4 vorgestellten Fehler- und Angriffsarten aufweisen. Da Datenflussarchitekturen einen am Datenfluss orientierten Aufbau besitzen und einen hohen Grad an Parallelisierung ermöglichen, soll in diesem Kapitel eine Erweiterung der Funktionsblöcke der Datenflussarchitekturen vorgestellt werden, die die Erkennung der genannten Fehler- und Angriffsarten weitmöglichst erlauben soll.

4.10.1 Erweiterung der Funktionsblöcke um Lebenszeichen und Diagnose

Zunächst werden die zur Verarbeitung der Daten genutzten Funktionsblöcke bzw. Verarbeitungseinheiten um einen Lebenszeichen- und einen Diagnoseausgang LZ bzw. D erweitert, wodurch sich der in Abbildung 4.60 dargestellt Aufbau ergibt.

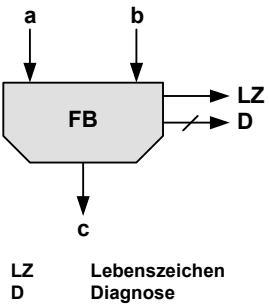


Abbildung 4.60: Erweiterung der Funktionsblöcke um Lebenszeichen und Diagnose

Im fehlerfreien Betrieb generiert jeder Funktionsblock ein alternierendes Lebenszeichensignal LZ. Diese Lebenszeichensignale LZ werden zu einem globalen Lebenszeichensignal zusammengefasst, wie in Abbildung 4.61 gezeigt. Nur, wenn keiner der Funktionsblöcke einen Fehler meldet, gibt auch die gesamte Datenverarbeitungseinheit ein entsprechendes Lebenszeichen aus, das durch eine Systemüberwachungseinheit ausgewertet werden kann.

Nun soll betrachtet werden, inwieweit die Inhalte der Kennungen der Datenspeicherelemente in den gemäß Abbildung 4.60 gestalteten Funktionsblöcken geprüft werden können. Dabei können die folgenden Prüfungen der Operanden ohne zusätzliche Erweiterungen der Funktionsblöcke durchgeführt werden:

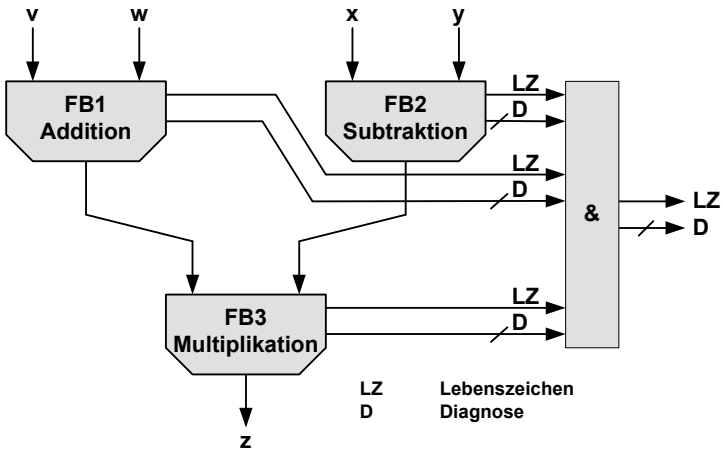


Abbildung 4.61: Zusammenfassung von Lebenszeichen LZ und Diagnose D

- die in den Datenwertfeldern W abgelegten Datenwerte der Operanden – bei Messbereichsdatentypen auch deren Werteintervalle – können anhand der Wertebereichskennung WB auf ihre Plausibilität hin geprüft werden,
- die Datentypkompatibilität der Operanden zueinander kann anhand Datentypkennung überprüft werden, wobei
  - die Datentypen DT,
  - die Subdatentypen SDT und
  - die Typberechtigungen TB der Operanden identisch und die im Funktionsblock durchzuführende Operation durch die Typberechtigungen TB gestattet sein müssen,
- bei Additionen, Subtraktionen und Vergleichen kann anhand der Einheitenkennung EI die Gleichheit der Einheiten aller Operanden verifiziert werden,
- die Initialisierungsstatusbeschreiber IS innerhalb der Zugriffsrechtekennungen ZR müssen bei allen Operanden den Wert Eins aufweisen, da sie les- und damit verarbeitbare Daten enthalten müssen,
- die Integrität der Operanden kann anhand der Integritätsprüfung IP verifiziert werden.

Die Kennungen des Ergebnisses der im Funktionsblock ausgeführten Operation werden wie folgt gesetzt:

- der Datenwert W – bei Messwertdatentypen das Datenwertintervall – ergibt sich als Ergebnis der durchgeführten Operation,
- die Datentypkennung DT wird auf die Inhalte der Operanden gesetzt,
- die Einheitenkennung EI wird bei Additionen und Subtraktionen auf die Einheit der Operanden, bei Multiplikationen und Divisionen auf die sich nach Anwendung der Potenzgesetze auf die Einheiten der Quelloperanden ergebende Einheit gesetzt,
- der Initialisierungsstatusbeschreiber IS innerhalb der Zugriffsrechtekennung ZR wird auf den Wert Eins gesetzt, da das Ergebnis einer Operation ja immer verwendbare Daten enthält,
- die Verarbeitungswegkennung VW des Ergebnisses ergibt sich aus den in Kapitel 4.3.7 vorgestellten Verknüpfungen der Teilkennungen der Verarbeitungswegkennungen VW der Operanden,
- die Zeitschrittkennung ZS wird auf den jüngsten Zeitschritt der Operanden gesetzt,
- der Fristkennung FR des Ergebnisses wird die kürzeste verbleibende Frist der Operanden zugewiesen und
- die Integritätsprüfung IP wird entsprechend der Inhalte des Ergebnisses gesetzt.

Eine Übersicht über die Kennungen, deren Inhalte bei Quelloperanden durch die nach Abbildung 4.60 gestalteten Funktionsblöcke geprüft bzw. bei den Ergebnissen gesetzt werden können, gibt Tabelle 4.16. Wie der Tabelle zu entnehmen ist, bestehen bei den folgenden Kennungen Einschränkungen:

- Bei der Wertebereichkennung WB können nur die Wertebereiche der Quelloperanden gegen den vorgegebenen Wertebereich geprüft werden, beim Ergebnis kann jedoch keine derartige Prüfung erfolgen, da kein Zieldatenspeicherelement im Speicher existiert, welches einen entsprechenden Wertebereich definieren kann.
- Der Zeitschritt in der Zeitschrittkennung ZS des Ergebnisses kann nur auf den jüngsten der Zeitschritte der Operanden gesetzt werden, ein automatisches

Inkrementieren des Zeitschritts – wie von der +1-Teilkennung bekannt – kann nicht erfolgen.

- Die Frist des Ergebnisses in dessen Fristkennung FR kann nur auf die kürzere der Fristen der beiden Quelloperanden gesetzt werden. Eine neue Frist kann mangels entsprechender Information nicht gesetzt werden.

Zur Beseitigung dieser Einschränkungen und der Prüfung bzw. dem Setzen der bislang nicht prüf- bzw. setzbaren Kennungen sollen die Funktionsblöcke im folgenden Unterkapitel nochmals erweitert werden.

Tabelle 4.16: Prüfung der Kennungsinhalte durch erweiterte Funktionsblöcke

Kennung	Teilkennung	Prüfen	Setzen
Wertebereich WB	(alle)	(X)	-
Datentyp DT	(alle)	X	X
Einheit EI	(alle)	X	X
Zugriffsrechte ZR	Modulnummer MN	-	-
	Funktionsnummer FN	-	-
	Schreibrechte SR	-	-
	Initialisierungsstatus IS	X	X
Verarbeitungsweg VW	(alle)	-	X
Zeitschritt ZS	(alle)	-	(X)
Frist FR	(alle)	-	(X)
Zykluszeit ZY	(alle)	-	-
Integrität IP	(alle)	X	X
Signatur S	(alle)	-	-

X: prüf- bzw. setzbar, (X): mit Einschränkungen prüf- bzw. setzbar,
-: nicht prüf- bzw. setzbar

4.10.2 Verbesserung der Fehlererkennung durch zusätzliche Erweiterungen

Sollen innerhalb der Funktionsblöcke weitere Prüfungen durchgeführt werden, so werden zusätzliche Erweiterungen der Funktionsblöcke notwendig:

- durch das Hinzufügen der Verfügbarkeit einer Zeitquelle kann die Gültigkeit der Operanden anhand deren Fristkennung FR verifiziert werden, ebenso wird dadurch die lokale Prüfung der Zykluszeitkennung ZY ermöglicht,
- die Festlegung des zulässigen Wertebereichs des Verarbeitungsergebnisses erlaubt dessen entsprechende Prüfung,
- das Bereitstellen von Verarbeitungsweginformationen erlaubt die Prüfung der Inhalte der Verarbeitungswegkennung VW,
- die Spezifikation der zu prüfenden temporalen Beziehung der Operanden erlaubt deren Prüfung anhand der Zeitschrittkennungen ZS und
- die Bereitstellung des öffentlichen Schlüssels erlaubt die Prüfung der Authentizität und Integrität der Operanden anhand der Signaturkennungen S.

Auch das Setzen der Kennungen des Ergebnisses von Funktionsblöcken kann durch zusätzliche Erweiterungen unterstützt werden:

- die Bereitstellung des geheimen Schlüssels erlaubt die Signierung der Inhalte des Ergebnisses in der Signaturkennung S,
- die oben bereits erwähnte Festlegung des zulässigen Wertebereichs des Verarbeitungsergebnisses kann in dessen Wertebereichskennung WB eingetragen werden,
- die Spezifikation einer relativen Frist Δt_{FR} erlaubt das Setzen einer von den Fristen der Quelloperanden unabhängigen neuen Frist FR für das Ergebnis und
- die Angabe eines Identifikators und jeweils eines frühesten und spätesten relativen Zeitpunkts zur Aktualisierung des betreffenden Datums erlaubt das Setzen der Zykluszeitkennung ZY des Ergebnisses.

Damit ergibt sich der in Abbildung 4.62 dargestellte Aufbau der Funktionsblöcke unter Einbeziehung der beschriebenen Erweiterungen.

Auch für die in Abbildung 4.62 gezeigten nochmals erweiterten Funktionsblöcke wird in Tabelle 4.17 übersichtlich zusammengefasst, welche der Kennungen der Datenspeicherelemente der Quelloperanden geprüft bzw. beim Ergebnis gesetzt werden können.

Bis auf drei Teilkennungen der Zugriffsrechtekennung ZR sind mit den genannten Erweiterungen der Funktionsblöcke nun alle Kennungen in vollem Umfang prüf- bzw. setzbar.

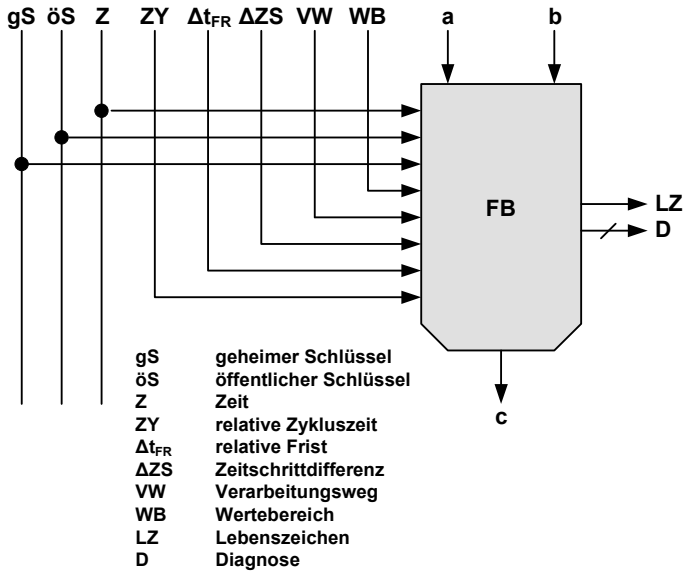


Abbildung 4.62: Funktionsblock mit Zusatzeingängen

Tabelle 4.17: Prüfung der Kennungsinhalte durch erneut erweiterte Funktionsblöcke

Kennung	Teilkennung	Prüfen	Setzen
Wertebereich WB	(alle)	X	X
Datentyp DT	(alle)	X	X
Einheit EI	(alle)	X	X
Zugriffsrechte ZR	Modulnummer MN	-	-
	Funktionsnummer FN	-	-
	Schreibrechte SR	-	-
	Initialisierungsstatus IS	X	X
Verarbeitungsweg VW	(alle)	X	X
Zeitschritt ZS	(alle)	X	X
Frist FR	(alle)	X	X
Zykluszeit ZY	(alle)	X	X
Integrität IP	(alle)	X	X
Signatur S	(alle)	X	X

X: prüf- bzw. setzbar, -: nicht prüf- bzw. setzbar

4.10.3 Weiterhin bestehende Einschränkungen

Trotz der vorgestellten Erweiterungen der Funktionsblöcke für die Realisierung der Datenspezifikationsarchitekturmerkmale auf Basis einer Datenflussarchitektur bleiben Teile der Zugriffsrechtekennung ZR unprüfbar. Die Zieldatenspeicherelemente existieren nicht als Variablen in einem Speicher, sondern nur als Nachrichten zwischen Funktionsblöcken und können daher auch nicht formatiert sein. Dies betrifft vor allem die Zugriffsrechte, da keine Aufteilung des Programms in Module existiert, wodurch weder Modul- noch Funktionsnummern MN bzw. FN in der Zugriffsrechtekennung ZR überprüft werden können. Weiterhin ist es nicht möglich, den Schreibrechtebeschreiber SR in der Zugriffsrechtekennung zu prüfen. Da die beschriebenen Teilkennungen der Zugriffsrechtekennung ZR – bis auf den Initialisierungsstatusbeschreiber IS – bei der Realisierung einer Datenspezifikationsarchitektur auf Basis einer Datenflussarchitektur weder setz- noch prüfbar sind, können diese ignoriert werden. Ein Weglassen der betroffenen Teilkennungen Modul- und Funktionsnummer MN bzw. FN und des Schreibrechtebeschreibers SR aus Platzspargründen sollte unterbleiben, da sich ansonsten Inkompatibilitäten mit anderen Systemkomponenten ergeben könnten.

5 Evaluation der Datenspezifikationsarchitektur

Die in dieser Arbeit vorgestellte Datenspezifikationsarchitektur DSA soll nun anhand verschiedener Gesichtspunkte evaluiert werden. Zuerst wird die Art der Datenabbildung in einer DSA mit der des Stands der Technik verglichen und die Architektur anschließend im Zusammenhang mit Datentyp-, Datenstruktur- und Befähigungsarchitekturen eingeordnet. Es folgen die Evaluation der Erkennbarkeit der 20 in Kapitel 2.4 vorgestellten datenflussbezogenen Fehler- und Angriffsarten und eine Betrachtung, wie die in Kapitel 1.1 vorgestellten Fehlerfälle aus der Praxis durch eine DSA frühzeitig erkannt und wie ihre Auswirkungen reduziert oder sogar vermieden hätten werden können. Den Abschluss der Evaluation der DSA bildet die Analyse der Speicherausnutzung der Architektur, auch wenn der Speicherverbrauch bei den heutzutage verfügbaren Speichergrößen kein an eine moderne Architektur anzulegendes Bewertungskriterium sein darf, der Vollständigkeit halber aber trotzdem betrachtet werden soll.

5.1 Evaluation der Datenabbildung der DSA

Die verschiedenen Abbildungen von Daten inklusive aller Eigenschaften, die von der jeweiligen Architektur oder dem jeweiligen Verfahren explizit in einem Datenspeicherelement D spezifiziert werden, werden hier nochmals anhand der in Kapitel 4.2 vorgestellten Dateneigenschaften übersichtlich dargestellt, inklusive der in dieser Arbeit vorgestellten Datenspezifikationsarchitektur DSA.

In **konventionellen Architekturen** bilden die Datenspeicherelemente D einzig einen Datenwert W in einem nur implizit bekannten und spezifizierten Datenformat in der Form

$$D := W$$

ab. Alle weiteren Eigenschaften der Daten werden durch die Art des Zugriffs und die anschließende Verwendung der Daten implizit festgelegt. Entsprechend wenige Möglichkeiten gibt es für die Hardware, fehlerhaften Umgang mit den Datenwerten oder auch fehlerhafte Daten zu erkennen. Dabei gab es in der Vergangenheit sehr leistungsfähige und vor allem sehr einfache Ansätze, mehr Eigenschaften von Datenspeicherelementen in einer für die Hardware verständlichen Form darzustellen.

In **Datentyparchitekturen** werden die Datenspeicherelemente um Datentypkennungen DT ergänzt, die in Form zusätzlicher Bits den Datentyp der im Datenspeicherelement enthaltenen Datenwerte beschreiben und für die Hardware auswertbar machen [1, 36, 39].

$$D := (W, DT)$$

Auf diese Weise kann die Hardware die Kompatibilität der Datentypen der Datenwerte vor deren Nutzung prüfen, um z. B. sicherzustellen, dass nur Datenwerte mit kompatiblen Datentypen bei arithmetischen Operationen verwendet werden.

Zur hardwareverständlichen Beschreibung komplexerer, über atomare Datentypen hinausgehender Datenstrukturen werden in **Datenstruktur-** bzw. **Deskriptorarchitekturen** entsprechende Datentypidentifikatoren vorgesehen, z. B. in Form von Deskriptoren [39, 78].

Des Weiteren kann man Zugriffrechte ZR in den Datenspeicherelementen verankern, wie es in **Befähigungsarchitekturen** vorgesehen ist [39, 78]. Diese Beschreibung der Zugriffrechte gestattet es der Hardware, diese Rechte bei Zugriffen zu prüfen. Dadurch erweitert sich das Tupel, welches ein Datenspeicherelement beschreibt, zu:

$$D := (W, DT, ZR)$$

Zugriffsrechte können dabei Lese-, Schreib- und für Code auch Ausführungsrechte, ggf. auch die Zuordnung von Code und Daten zu bestimmten Programmeinheiten sein.

In einigen **Datentyp-, Datenstruktur- und Befähigungsarchitekturen** werden die Datenspeicherelemente zusätzlich mit einem Merkmal zur Integritätsprüfung IP, also zur Erkennung von Verfälschungen der Datenworte versehen [1, 39], wodurch sich das Datenabbildungstupel für Datentyp- und Datenstrukturarchitekturen zu

$$D := (W, DT, IP)$$

und für Befähigungsarchitekturen

$$D := (W,\ DT,\ ZR,\ IP)$$

ergänzt.

Die inhärent sichere Mikroprozessorarchitektur **ISMA** [125] fügt den Datenspeicherelementen neben den bereits erwähnten Kennungen einen Initialisierungsstatusbeschreiber hinzu, die es erlaubt, die Verwendung nicht initialisierter Datenspeicherelemente als Fehler zu erkennen, also den Versuch, lesend auf ein Datenspeicherelement zuzugreifen, dem vorher kein gültiger Datenwert zugewiesen wurde. Diese Kennung wird als Initialisierungsstatus IS ebenfalls im Datenspeicherelement abgelegt. Damit werden Datenspeicherelemente in ISMA wie folgt dargestellt:

$$D := (W,\ DT,\ ZR,\ IP,\ IS)$$

ISMA verwendet Datenspeicherelemente mit einer einheitlichen Breite von 128 Bit, wobei nur 64 Bit für die Speicherung der eigentlichen Datenwerte W und die restlichen 64 Bit für Sicherungs- und Verwaltungsdaten, also zur Definition von DT, ZR, IP und IS verwendet werden.

Bei der **ANBD-Kodierung** kommen keine Kennungen zum Einsatz. Die Datenwerte werden stattdessen kodiert, indem die Fehlererkennungsmerkmale durch arithmetische Operationen mit dem eigentlichen Datenwert verbunden werden, wodurch sich die Datenabbildung

$$D := (W \cdot IP + ZS + AD)$$

ergibt.

Die in dieser Arbeit vorgestellte **Datenspezifikationsarchitektur DSA** verwendet – bei Nutzung aller vorgestellten Kennungen – das Tupel

$$D := (W,\ WB,\ DT,\ EI,\ ZR,\ VW,\ ZS,\ FR,\ ZY,\ IP,\ S)$$

mit

$$VW := (Q,\ VW_{sys},\ VW_{lok},\ Z),$$

$$ZR := (MN,\ FN,\ SR,\ IS)$$

und der Einbeziehung der Adresse AD in die Integritätsprüfung IP bzw. die Signatur S zur Abbildung das Datenwerts und seiner Eigenschaften in einem Datenspeicherelement. Zur Abbildung der Genauigkeit von Messwerten und der Fehlerfortpflanzung werden die Datenwerte W bei speziellen Messwertdatentypen in der Form

$$W := (W_{min},\ W_{max})$$

dargestellt. Damit übertrifft sie alle bisherigen Datenabbildungen in Hinsicht auf den Umfang der Beschreibung der Dateneigenschaften deutlich.

5.2 Einordnung der entstandenen Architektur

In Abbildung 5.1 wird veranschaulicht, wie die entstandene Datenspezifikationsarchitektur bezogen auf die bislang bekannten Architekturarten einzuordnen ist.

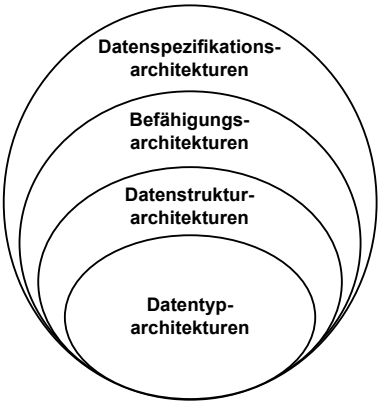


Abbildung 5.1: Einordnung der Datenspezifikationsarchitekturen

Den Anfang bilden die Datentyparchitekturen mit der expliziten hardwareverständlichen Angabe des Datentyps der in einem Datenspeicherelement enthaltenen Daten. Datenstrukturarchitekturen bieten zusätzliche Merkmale zur hardwareverständlichen Definition komplexer Datenstrukturen und bilden somit eine Übermenge der Datentyparchitekturen. Durch die Zurverfügungstellung von Merkmalen zur Spezifikation von Zugriffsrechten bilden Befähigungsarchitekturen wiederum eine Übermenge der Datenstrukturarchitekturen.

Die in dieser Arbeit vorgestellte Datenspezifikationsarchitektur DSA erweitert die Datenabbildung gegenüber den Befähigungsarchitekturen nochmals deutlich, indem den Datenspeicherelementen weitere Dateneigenschaften in einer hardwareverständlichen Form hinzugefügt werden. Durch diese zusätzlichen Merkmale bildet die

DSA eine Übermenge der Befähigungsarchitekturen. Damit kann man die entstandene DSA auch als neue Architekturgattung oberhalb der bekannten Gattungen Datentyp-, -struktur- und Befähigungsarchitekturen betrachten.

5.3 Evaluation anhand der Fehlererkennungsmöglichkeiten

Die in dieser Arbeit vorgestellte Datenspezifikationsarchitektur soll nun anhand der Erkennbarkeit der 20 in Kapitel 2.4 vorgestellten Fehler- und Angriffsarten bewertet werden. Dazu wird die Erkennbarkeit der einzelnen Fehler- bzw. Angriffsarten in Tabelle 5.1 zusammen mit der Kennung oder dem Verfahren gezeigt, welches die Erkennung der jeweiligen Fehlerart erlaubt. Dabei werden – aus Gründen der Übersichtlichkeit – diejenigen Kennungen oder Merkmale der Architektur nicht erwähnt, die nur eine begrenzte oder bedingte Erkennbarkeit der einzelnen Fehler- bzw. Angriffsarten erlauben. Diese eingeschränkten Fehlererkennungsmöglichkeiten können in der Einzelevaluation des jeweiligen Merkmals nachgeschlagen werden.

Die Datentypkennung DT ermöglicht die Erkennung der Verwendung von Operanden mit inkompatiblen Datentypen. Über die EI-Kennung kann die Inkompatibilität der Einheiten der Operanden aufgedeckt werden, also z. B. der sprichwörtliche Vergleich von Äpfeln mit Birnen.

Mit Hilfe der Wertebereichskennung können Datenwerte eines Datenspeicherelements beim Lesen auf ihre Plausibilität geprüft werden. Beim Schreiben in ein Datenspeicherelement mit vordefiniertem Wertebereich kann die Hardware prüfen, ob der zu schreibende Wert innerhalb des definierten Wertebereichs liegt. Die Verwendung von Messwertdatentypen mit Angabe eines Werteintervalls ermöglicht das Aufdecken genauigkeitsbezogener Fehler.

Die Auswahl falscher Operanden kann die DSA durch Einbeziehung der Adresse oder des Identifikators eines adressierten Operanden in dessen Integritätsprüfung IP bzw. Signaturkennung S erkennen. Eine redundante diversitäre arithmetisch-logische Einheit ALE erlaubt es, eine durch Fehler verursachte Falschauswahl von Operatoren und Berechnungsfehler innerhalb der ALE zu erkennen.

Eine Fristüberschreitung in Form der Nutzung von Daten außerhalb ihres Gültigkeitszeitraums kann die Hardware anhand der FR-Kennung erkennen. Verletzungen der vorgesehenen frühesten und spätesten zulässigen Zeitpunkte zur Aktualisierung eines Datenspeicherelements, also Zyklusunter- und -überschreitungen,

Tabelle 5.1: Fehlererkennung durch die Datenspezifikationsarchitektur DSA

Fehlerart	**Erkennbarkeit**	**Merkmal**	**Kapitel**
Inkompatible Datentypen	ja	DT-Kennung	4.3.4
Inkompatible Einheiten	ja	EI-Kennung	4.3.5
Wertebereichsunter- bzw. -überschreitung	ja	WB-Kennung	4.3.3
Genauigkeitsproblem	ja	Messwertdaten-typen mit Werteintervall	4.3.2
Falsche Operandenauswahl	ja	AD in IP- bzw. S-Kennung	4.3.11, 4.3.12
Falsche Operatorauswahl	ja	Diversitäre ALE	4.3.13
Fehlerhaftes Operationsergebnis	ja	Diversitäre ALE	4.3.13
Fristüberschreitung	ja	FR-Kennung	4.3.9
Zyklusunterschreitung	ja	ZY-Kennung mit ZÜE	4.3.10
Zyklusüberschreitung	ja	ZY-Kennung mit ZÜE	4.3.10
Verlorengegangene Datenaktualisierung	ja	ZS-Kennung	4.3.8
Synchronisationsfehler oder unvollständige Datenübertragung	ja	ZS-Kennung	4.3.8
Pufferunter- oder -überläufe	ja	Sichere Felder, DT-Kennung	4.3.4
Fehlerhafter Datenfluss (falsche Adressaten, ...)	ja	VW-Kennung	4.3.7
Duplizierte Daten	ja	ZS-Kennung	4.3.8
Durch Fehler oder Störungen verfälschte Daten	ja	IP-Kennung, S-Kennung	4.3.11, 4.3.12
Fehlerhafter Datenzugriff (fehlende Zugriffsrechte)	ja	ZR-Kennung	4.3.6
Nutzung nicht initialisierter Daten	ja	ZR-Kennung	4.3.6
Angriffsart			
Gezielt verfälschte Daten	ja	S-Kennung	4.3.12
Wiedereinspielungsattacke	ja	S-Kennung mit ZS-, FR- und ZY-Kennung	4.3.12, 4.3.8, 4.3.9, 4.3.10

können durch die Kombination der Zykluszeitkennung ZY und der Zyklusüberwachungseinheit ZÜE aufgedeckt werden, die dazu für alle zu überwachenden Daten entsprechende Listeneinträge verwaltet.

Verlorengegangene Datenaktualisierungen, Synchronisationsfehler und unvollständige Datenübertragungen sowie duplizierte Daten können durch die Nutzung der Zeitschrittkennung erkannt werden. Datenzugriffe außerhalb von Feldern oder Puffern können durch die von Datenstrukturarchitekturen bekannten sicheren Feldzugriffsmechanismen unter Nutzung von Felddatentypen und dedizierten Feldzugriffsbefehlen als Fehler erkannt werden. Fehlerhafter Datenfluss kann dadurch erkannt werden, dass die Daten nicht dem in der VW-Kennung festgelegten Weg durch das System folgen.

Werden Daten durch Störungen oder Fehler verfälscht, kann dies – im Rahmen des minimalen Hammingabstands des eingesetzten Verfahrens zur Integritätsprüfung – im Zuge der Integritätsprüfung unter Nutzung der IP-Kennung bzw. – bei Nutzung einer kryptographischen Signatur pro Datenspeicherelement – der S-Kennung erkannt werden.

Fehlerhafte Datenzugriffe durch falsche Programmteile oder auf fehlerhafte Weise, sowie die Nutzung nicht initialisierter Daten werden durch die Zugriffsrechtekennung ZR aufgedeckt.

Durch einen Angreifer gezielt verfälschte Daten können durch die Signaturkennung S aufgedeckt werden, da der Angreifer zwar die Daten manipulieren, aber ohne den geheimen Schlüssel der Quelle die Daten anschließend nicht mit einer gültigen Signatur versehen kann. Auch den Versuch, aufgezeichnete – und damit an sich gültige und unverfälschte Daten – zu einem für den Angreifer günstigen Zeitpunkt durch eine Wiedereinspielungsattacke dem System als aktuelle Daten zu präsentieren, kann eine DSA erkennen, da die FR-, ZS- und ZY-Kennungen die veralteten Daten als solche identifizieren. Die Verwendung der Signaturkennung S sorgt dafür, dass der Angreifer die beiden Kennungen nicht in seinem Sinne ändern kann.

In Tabelle 5.2 wird die entstandene Datenspezifikationsarchitektur DSA nochmals dem Stand von Wissenschaft und Technik gegenübergestellt, wobei deren Leistungsfähigkeit in Bezug auf die Erkennbarkeit der verschiedenen Fehler- und Angriffsarten besonders deutlich wird. Details zur Bewertung des Stands von Wissenschaft und Technik können der Zusammenfassung in Kapitel 3.12 bzw. den detaillierten Betrachtungen der einzelnen Verfahren und Architekturen in den Kapiteln 3.1 bis 3.11 entnommen werden.

Tabelle 5.2: Vergleich der DSA mit dem Stand von Wissenschaft und Technik bzgl. der Fehlererkennbarkeit

Fehler- bzw. Angriffsart	x86, ARM	SGP	DT, DS, BA	DFA	ISMA	ADI SSM	ANBD	DDFV	TCP / IP	PS, CS	DSA
Inkompatible Datentypen	-	-	+	-	+	-	-	-	-	-	+
Inkompatible Einheiten	-	-	-	-	-	-	-	-	-	-	+
Wertebereichsverletzung	◯ x86	-	-	-	-	-	-	-	-	-	+
Genauigkeitsproblem	-	-	-	-	-	-	-	-	-	-	+
Falsche Operanden	-	+	-	-	-	-	(+) BD	◯	-	-	+
Falsche Operatoren	-	+	-	-	-	-	(+) BD	+	-	-	+
Fehlerhafte Operation	-	+	-	-	-	-	(+) BD	-	-	-	+
Fristüberschreitung	-	◯	-	-	◯	-	-	-	-	(+)	+
Zyklusunterschreitung	-	-	-	-	-	-	-	-	-	-	+
Zyklusüberschreitung	-	-	-	-	-	-	-	-	-	(+)	+
Verlorengeg. Aktualisier.	-	-	-	-	-	(+)	(+) D	-	(+)	(+)	+
Synchronisationsfehler und unvollst. Übertragung	-	-	-	-	-	(+)	(+) D	-	(+)	(+)	+
Pufferunter- oder -überlauf	(+) x86	-	+ DS	-	+	(+)	(+) D	-	-	-	+
Fehlerh. Datenfluss	-	◯	◯ BA	-	◯	-	-	◯	(+)	(+)	+
Duplizierte Daten	-	-	-	-	-	(+)	-	-	(+)	(+)	+
Durch Störungen oder Fehler verfälschte Daten	◯	(+)	+	-	+	-	(+) AN	-	+	+	+
Fehlerh. Datenzugriff (fehlende Zugriffsrechte)	◯	◯	+ BA	-	+	-	-	-	-	-	+
Nicht initialisierte Daten	-	-	◯	-	+	(+)	(+) BD	-	-	-	+
Gezielte Verfälschung	-	-	-	-	-	-	-	-	-	◯	+
Wiedereinspielungsattacke	-	-	-	-	-	-	-	-	-	◯	+

Fehlererkennung: - nicht mögl., ◯ begrenzt mögl., (+) mit Einschränkungen mögl., + mögl.; SGP: Proz. für sicherheitsger. Anwendungen, DT, DS, BA: Datentyp-, -struktur-, Befähigungsarch., DFA: Datenflussarch., PS: PROFIsafe, CS: CIP Safety, DSA: Datenspezifikationsarch.

5.4 Evaluation anhand der Fehlerbeispiele

Die Leistungsfähigkeit der vorgestellten Datenspezifikationsarchitektur DSA soll nun anhand der in Kapitel 1.1 dieser Arbeit vorgestellten Beispiele evaluiert werden, wobei die Leistungsfähigkeit der Fehlererkennungsmerkmale der DSA deutlich zu Tage tritt.

5.4.1 Selbstzerstörung der Ariane 5

Die Rakete Ariane 5 zerstörte sich nach [79], da es bei Berechnungen zu Überläufen kam. Diese wurden zwar erkannt, allerdings wurden die generierten Diagnosedaten, die dies kenntlich machen sollten, zusammen mit einem Datenblock mit gültigen Flugdaten an den Bordrechner übermittelt, der die Diagnosedaten fehlerhafterweise als Flugdaten interpretierte. In der Folge neigte sich die Rakete zu stark, worauf die Selbstzerstörung ausgelöst wurde.

Die Hardware einer Datenspezifikationsarchitektur wäre zwar nicht in der Lage gewesen, die Berechnungsüberläufe zu vermeiden, sie hätte aber die fehlerhafte Interpretation der Diagnosedaten als Flugdaten aufdecken können. Zum einen ist es unwahrscheinlich, dass alle Diagnosedaten enthaltenden Datenspeicherelemente die zu den Flugdaten enthaltenden Datenspeicherelementen identischen Datentypen aufgewiesen haben, d. h. die aus Datentyparchitekturen bekannte Datentypkennung DT hätte an dieser Stelle die Fehler aufdecken können. Gleiches dürfte für die Einheiten gelten: auch hier wäre mit Abweichungen der Diagnosedaten gegenüber realen Flugdaten zu rechnen gewesen, hätte sich also durch die Einheitenkennung EI aufdecken lassen. Weiterhin dürften Verarbeitungsweg und Zieldatensenke der Diagnosedaten deutlich von jenen der Flugdaten abweichen – was in einer in dieser Arbeit vorgestellten Datenspezifikationsarchitektur in den Diagnosedaten explizit in den Kennungen Verarbeitungsweg VW und Ziel Z angegeben worden wäre. Die Hardware hätte also bei der ersten Verwendung der Diagnosedaten in einem Flugdatenauswertungsmodul erkannt, dass hier Daten fehlgeleitet wurden.

Der Absturz hätte jedoch auch bei Anwendung der in dieser Arbeit vorgestellten Fehlererkennungsmethoden nicht vermieden werden können, da ein weiterer, schwerwiegender Entwicklungsfehler begangen wurde [79]: die beiden Navigationssysteme waren nicht diversitär entwickelt worden und wiesen beide den identischen Fehler auf und fielen somit zeitgleich aus. Die Rakete wäre daher also auch bei erfolgreicher und frühzeitiger Fehlererkennung nicht zu retten gewesen.

5.4.2 Verlust der NASA-Sonde Mars Climate Orbiter

Der Mars Climate Orbiter MCO ging verloren, da verschiedene Entwicklungsgruppen mit verschiedenen Maßeinheiten – die eine in SI-, die andere in englischen – entwickelt hatten [81]. Bei einer auf Basis dieser Arbeit entwickelten Datenspezifikationsarchitektur werden die Einheiten jedes Operanden in der EI-Kennung explizit und hardwareverständlich festgelegt. Dabei können nur SI-Einheiten zum Einsatz kommen, wodurch ein solcher Fehler von vornherein vermieden wird. Sollte auf die Angabe der Einheiten absichtlich oder unabsichtlich verzichtet werden, ist dies im Rahmen einer Entwurfs- oder Codebegutachtung einfach aufzudecken und sollte zur Ablehnung einer Zertifizierung der betroffenen Software führen. Spätestens in Testläufen würde die Hardware einer Datenspezifikationsarchitektur die Inkompatibilität der Einheiten feststellen und verbliebene derartige Fehler aufdecken, selbst wenn die verursachten Abweichungen innerhalb von Testtoleranzen liegen würden.

5.4.3 Bestrahlungsgerät Therac 25

Neben weiteren Fehlern führten Inkonsistenzen in den Behandlungsparametern, verursacht durch fehlerhafte Synchronisierungsmechanismen, beim Therac 25 in mindestens sechs Fällen zu massiven Strahlungsüberdosen, die teilweise zum Tod des Patienten führten.

Durch den Einsatz der in dieser Arbeit vorgestellten Zeitschrittkennungen ZS wäre mit hoher Wahrscheinlichkeit die Vermischung aktualisierter und veralteter Parameterwerte bei deren Auswertung anhand der Ungleichheit der Zeitschritte der einzelnen Operanden aufzudecken gewesen.

Auch die Nutzung der Fristkennung FR hätte bei der Verhinderung der Unfälle helfen können. Ein Setzen der Fristkennung der Behandlungsparameter auf das Ende einer ersten Behandlung hätte alle veralteten Parameter bei der folgenden Behandlung aufgedeckt, da die Frist dieser Parameter abgelaufen wäre.

5.4.4 Sicherheitslücke Heartbleed

Die Auswirkung des Heartbleed-Fehlers [22] – die fehlerhafte Herausgabe von Daten, die nicht zur aktuellen Kommunikationsverbindung zwischen Angreifer und Opfer gehörten – hätte bei Verwendung der in dieser Arbeit beschriebenen Merkmale

einer Datenspezifikationsarchitektur auf die folgenden Weisen aufgedeckt werden können:

- durch die Nutzung der aus Befähigungsarchitekturen bekannten Zugriffsrechtekennungen ZR,
- durch Anwendung der in ISMA [125] vorgestellten Initialisierungsstatusbeschreibers IS oder
- bei Verwendung der in dieser Arbeit vorgestellten Zeitschritt- ZS und Fristkennungen FR.

Die Zugriffsrechtekennungen ZR hätten den Fehler dann aufdecken können, wenn den Speicherinhalten des Opfers verschiedene Befähigungen zugewiesen worden wären. Der fehlerhafte Zugriff einer Verbindungsinstanz auf die Daten einer anderen wäre damit zu erkennen gewesen.

Es ist anzunehmen, dass nicht alle Dateninhalte, die wegen des Heartbleed-Fehlers aus dem Speicher des Opfers gelesen werden konnten, aktuell genutzte und somit gültige Daten enthalten haben. Hier hätte die Markierung des freien Speichers, der keine für die Verwendung gültigen Daten mehr enthält, als nicht-initialisiert mit Hilfe der Initialisierungsstatusbeschreibers IS dafür gesorgt, dass die Hardware den Versuch, diese Daten zu lesen, als Fehler erkannt hätte.

Eine weitere Möglichkeit, den genannten Fehler aufzudecken, hätte die Zeitschrittkennung ZS einer Datenspezifikationsarchitektur geboten. Die übertragenen Daten innerhalb einer Verbindung hätten von der DSA mit Zeitschrittkennungen versehen werden können. Beim Erzeugen der Antwort auf die Nachricht des Angreifers hätte die Hardware der DSA sicherstellen können, dass alle in der Antwort enthaltenen Datenworte dieselbe Zeitschrittkennung aufweisen. Auch die Fristkennung FR hätte zur Erkennung des Fehlers genutzt werden können. Veraltete Daten außerhalb des der Kommunikationsverbindung zugewiesenen Puffers, deren in der Fristkennung angegebene Frist abgelaufen war, hätten beim Lesen zur Generierung eines Ausnahmefehlers geführt.

Eine Datenverbindung, bei der die genannten Prüfungen einen Fehler aufgedeckt hätten, hätte durch entsprechende Fehlerbehandlungsmaßnahmen z. B. einfach beendet werden können, ohne dass der Angreifer Daten hätte abgreifen können.

5.5 Evaluation der Speicherausnutzung

Ein Nachteil der Datenspezifikationsarchitektur DSA ist der immense Speicherbedarf für die verschiedenen Kennungen bezogen auf die Größe des eigentlichen Datenwerts. Während die inhärent sichere Mikroprozessorarchitektur ISMA maximal 50 % des Speichers für Datenwerte nutzen kann [125], ist dies bei einer DSA noch weitaus weniger. Die Speicherausnutzung wird nun für Daten- und Befehlsspeicherelemente getrennt anhand vorgeschlagener Bitbreiten für die einzelnen Kennungen betrachtet.

5.5.1 Speicherausnutzung der Datenspeicherelemente

Die Breite der Bitfelder, die für die einzelnen Kennungen einer Datenspezifikationsarchitektur DSA benötigt werden, wird in dieser Arbeit nicht festgelegt, da sie unter anderem vom Einsatzgebiet abhängt. Zur Evaluation der Speicherplatznutzung sollen nun für die einzelnen Bereiche eines DSA-Datenspeicherelements sinnvolle Bitbreiten exemplarisch in Tabelle 5.3 angenommen werden, um die effektive Speicherplatznutzung für Datentypen verschiedener Bitbreiten berechnen zu können.

5.5.1.1 Datenspeicherelementbreite bei Nutzung der Integritätsprüfung IP

Bei Verzicht auf die Signaturkennung S und der Nutzung der Integritätsprüfungskennung IP berechnet sich die Breite $\omega(D_{\mathrm{IP}})$ des Datenspeicherelements D nach Gleichung 5.1, wobei die Funktion ω die Bitbreite der einzelnen Datenspeicherelementbestandteile bzw. des gesamten Datenspeicherelements zurückliefert.

$$\omega(D_{\mathrm{IP}}) = \sum \omega(i) \quad \forall i \in \{\mathrm{W}, \mathrm{WB}, \mathrm{DT}, \mathrm{EI}, \mathrm{ZR}, \mathrm{VW}, \mathrm{ZS}, \mathrm{FR}, \mathrm{ZY}, \mathrm{IP}\} \tag{5.1}$$

Damit ergibt sich für die in Tabelle 5.3 angegebenen Bitbreiten der einzelnen Kennungen eine Datenwortbreite $\omega(D_{\mathrm{IP}})$ von 640 Bit nach Gleichung 5.2.

$$\omega(D_{\mathrm{IP}}) = (128 + 128 + 24 + 35 + 18 + 32 + 9 + 64 + 140 + 11)\,\mathrm{Bit} = 640\,\mathrm{Bit} \tag{5.2}$$

Sollen Datenspeicherelemente in einem Lese- bzw. Schreibzyklus komplett übertragen werden können, so könnte dies durch den parallelen Anschluss von 10 Speicherbausteinen mit einer Datenbusbreite von 64 Bit realisiert werden.

Tabelle 5.3: Vorgeschlagene Bitbreiten der Kennungen in Datenspeicherelementen

Kennung	**Vorgeschl. Bitbreite**	**Anmerkung**
Datenwert W	128	Verschiedene Datentypen mit einer Breite von 1 bis 64 Bit, Messwertdatentypen als Intervall mit zwei Intervallgrenzen je 64 Bit
Wertebereich WB	128	Unter- und Obergrenze mit je 64 Bit
Datentyp DT	24	Basisdatentyp DT 8 Bit, Subdatentyp SDT 8 Bit, Typberechtigungen TB 8 Bit
Einheit EI	56	Potenzen der sieben SI-Basiseinheiten mit je 8 Bit
Zugriffsrechte ZR	22	Modul- MN und Funktionsnummer FN je 10 Bit, Schreibrechte SR 1 Bit, Initialisierungsstatus 1 Bit
Verarbeitungsweg VW	38	Quelle Q 10 Bit, Verarbeitungswege $\mathrm{VW}_{\{\mathrm{sys,lok}\}}$ je 9 Bit, Ziel Z 10 Bit
Zeitschritt ZS	17	Präsenzbit 1 Bit, Zeitschritt 16 Bit
Frist FR	64	absoluter Zeitpunkt 64 Bit
Zykluszeit ZY	152	Identifikator 24 Bit, $\mathrm{ZY}_{\{\mathrm{min,max}\}}$ je 64 Bit
Integritätsprüfung IP	11	(640,629)-Erweiterter-Hamming-Code
Signatur S	1024	kryptographische Signatur für RSA-Schlüssel mit einer Länge von 1024 Bit

5.5.1.2 Datenspeicherelementbreite bei Nutzung der Signaturkennung S

Soll aus Sicherheitsgründen eine Signaturkennung S zum Einsatz kommen, so wird die Integritätsprüfungskennung IP überflüssig und die Breite $\omega(D_\mathrm{S})$ berechnet sich nach Gleichung 5.3.

$$\omega(D_\mathrm{S}) = \sum \omega(i) \quad \forall i \in \{\mathrm{W}, \mathrm{WB}, \mathrm{DT}, \mathrm{EI}, \mathrm{ZR}, \mathrm{VW}, \mathrm{ZS}, \mathrm{FR}, \mathrm{ZY}, \mathrm{S}\} \tag{5.3}$$

Dadurch ergibt sich für die in Tabelle 5.3 vorgeschlagenen Bitbreiten der einzelnen Kennungen eine Datenwortbreite $\omega(D_\mathrm{S})$ von 1653 Bit nach Gleichung 5.4.

$$\omega(D_{\mathrm{S}}) = (128 + 128 + 24 + 35 + 18 + 32 + 9 + 64 + 140 + 1024)\,\mathrm{Bit} = 1653\,\mathrm{Bit} \quad (5.4)$$

Bei Einsatz der Signaturkennung S müssten daher 25 Speicherbausteine mit einer Datenbusbreite von 64 Bit zum Einsatz kommen, um ein Datenspeicherelement in einem Lese- bzw. Schreibzyklus transferieren zu können. Dabei bleiben 11 Bits ungenutzt, die auf die verschiedenen Kennungen aufgeteilt werden können. Um eine realistische Speicherausnutzung zu berechnen, wird daher die auf die nächste 64-Bit-Grenze aufgerundete Breite $\omega(D_{\mathrm{S}})^*$ der Datenspeicherelemente nach Gleichung 5.5 verwendet.

$$\omega(D_{\mathrm{S}})^* = \omega(D_{\mathrm{S}}) + 11\,\mathrm{Bit} = 1664\,\mathrm{Bit} \quad (5.5)$$

5.5.1.3 Berechnung der Speicherausnutzung der Datenspeicherelemente

Die Speicherausnutzung η wird berechnet, indem die Breite der Datenwerte verschiedener Datentypen ins Verhältnis zur Gesamtbreite eines Datenspeicherelements gestellt wird. Bei Einsatz der Integritätsprüfungskennung IP berechnet sich die Speicherausnutzung $\eta_{\mathrm{D,IP}}$ nach Gleichung 5.6.

$$\eta_{\mathrm{D,IP}} = \frac{\omega(\mathrm{W}_{\mathrm{Datentyp}})}{\omega(D_{\mathrm{IP}})} \quad (5.6)$$

Wird hingegen die Signaturkennung S verwendet, so berechnet sich $\eta_{\mathrm{D,S}}$ nach Gleichung 5.7.

$$\eta_{\mathrm{D,S}} = \frac{\omega(\mathrm{W})}{\omega(D_{\mathrm{S}})^*} \quad (5.7)$$

Auf Basis der in Tabelle 5.3 vorgeschlagenen Bitbreiten der einzelnen Kennungen ergeben sich für verschiedene Datentypbitbreiten die in Tabelle 5.4 dargestellten Speicherausnutzungswerte.

Tabelle 5.4: Speicherausnutzung für verschiedene Datenwertbitbreiten

Bitbreite Datenwert W	Speicherausnutzung $\eta_{\mathbf{D,IP}}$	Speicherausnutzung $\eta_{\mathbf{D,S}}$
128	20,00 %	7,69 %
64	10,00 %	3,85 %
32	5,00 %	1,92 %
16	2,50 %	0,96 %
8	1,25 %	0,48 %
1	0,16 %	0,06 %

5.5.2 Speicherausnutzung der Befehlsspeicherelemente

Die Breite der Kennungen innerhalb der Befehlsspeicherelemente B hängt von der Breite der Kennungen innerhalb der Datenspeicherelemente D ab. Basierend auf den in Tabelle 5.3 vorgeschlagenen Bitbreiten der Kennungen in den Datenspeicherelementen ergeben sich die in Tabelle 5.5 angegebenen Bitbreiten der Kennungen innerhalb der Befehlsspeicherelemente. Wie bei den Daten- gilt es auch bei den Befehlsspeicherelementen zu unterschieden, ob eine Integritätsprüfungskennung IP oder eine Signaturkennung S zum Einsatz kommen soll, basierend auf den jeweiligen Sicherheitsanforderungen der jeweiligen Applikation.

5.5.2.1 Befehlsspeicherelementbreite bei Nutzung der Integritätsprüfung IP

Bei Einsatz der Integritätsprüfungskennung IP berechnet sich die Breite des Befehlsspeicherelements $\omega(B_{\mathrm{IP}})$ nach Gleichung 5.8. Dabei wird die Abkürzung „InstOp“ für den für die Instruktion und die Operanden reservierten Bereich im Befehlsspeicherelement verwendet.

$$\omega(B_{\mathrm{IP}}) = \sum \omega(i) \quad \forall i \in \{\mathrm{InstOp}, \mathrm{DT}, \mathrm{EI}, \mathrm{ZR}, \mathrm{VW}, \mathrm{ZS}, \mathrm{FR}, \mathrm{ZY}, \mathrm{IP}\} \tag{5.8}$$

Für die in Tabelle 5.5 genannten Breiten der verschiedenen Kennungen ergibt sich somit nach Gleichung 5.9 für $\omega(B_{\mathrm{IP}})$ ein Wert von 596 Bit.

$$\omega(B_{\mathrm{IP}}) = (128 + 24 + 113 + 22 + 48 + 34 + 64 + 152 + 11)\,\mathrm{Bit} = 596\,\mathrm{Bit} \tag{5.9}$$

Tabelle 5.5: Bitbreiten der Kennungen in Befehlsspeicherelementen

Kennung	Bitbreite	Anmerkung
Instruktion und Operanden	128	
Datentyp DT	24	Basisdatentyp DT 8 Bit := Befehl, 16 Bit ungenutzt
Einheit EI	113	Präsenzbit und Potenzen der sieben SI-Basiseinheiten mit je 8 Bit für zwei Operanden
Zugriffsrechte ZR	22	Modul- MN und Funktionsnummer FN je 10 Bit, Schreibrechte SR 1 Bit := 0, Initialisierungsstatus 1 Bit := 1
Verarbeitungsweg VW	48	Quellen $Q_{\{A,B\}}$ je 10 Bit, Verarbeitungswege $VW_{\{sys,lok\}}$ je 9 Bit, Ziel Z 10 Bit
Zeitschritt ZS	34	Zwei Präsenzbits je 1 Bit, zwei Zeitschritte je 16 Bit
Frist FR	64	absoluter Zeitpunkt 64 Bit
Zykluszeit ZY	152	Identifikator 24 Bit, $ZY_{\{min,max\}}$ je 64 Bit
Integritätsprüfung IP	11	(640,629)-Erweiterter-Hamming-Code
Signatur S	1024	kryptographische Signatur für RSA-Schlüssel mit einer Länge von 1024 Bit

Damit lässt sich eine einheitliche Bitbreite für Daten- und Befehlsspeicherelemente von 640 Bit festlegen.

Die nicht genutzten 44 Bit können für weitere kontrollflussbezogene Kennungen genutzt werden. Entsprechende Vorschläge für derartige Kennungen werden in den Weiterführungsmöglichkeiten in Kapitel 6 genannt.

Die ungenutzen Bits werden im Zuge der Aufrundung der Bitbreite auf die nächste 64-Bit-Grenze in $\omega(B_{\mathrm{IP}})^*$ nach Gleichung 5.10 einbezogen.

$$\omega(B_{\mathrm{IP}})^* = (128+24+113+22+48+34+64+152+11+44)\,\mathrm{Bit} = 640\,\mathrm{Bit} \quad (5.10)$$

5.5.2.2 Befehlsspeicherelementbreite bei Nutzung der Signaturkennung S

Wird statt der Integritätsprüfungskennung IP die Signaturkennung S zur Prüfung von Integrität und Authentizität der Befehlsspeicherelemente eingesetzt, so berechnet sich die Breite der Befehlsspeicherelemente $\omega(B_\mathrm{S})$ nach Gleichung 5.11.

$$\omega(B_\mathrm{S}) = \sum \omega(i) \quad \forall i \in \{\mathrm{InstOp}, \mathrm{DT}, \mathrm{EI}, \mathrm{ZR}, \mathrm{VW}, \mathrm{ZS}, \mathrm{FR}, \mathrm{ZY}, \mathrm{S}\} \tag{5.11}$$

Setzt man auch wieder die in Tabelle 5.5 angegebenen Breiten der einzelnen Kennungen ein, so ergibt sich nach Gleichung 5.12 für $\omega(B_\mathrm{S})$ ein Wert von 1609 Bit.

$$\omega(B_\mathrm{S}) = (128 + 24 + 113 + 22 + 48 + 34 + 64 + 152 + 1024)\,\mathrm{Bit} = 1609\,\mathrm{Bit} \tag{5.12}$$

Damit lässt sich auch bei Nutzung der Signaturkennung S eine mit der Breite der Datenspeicherelemente identische Breite der Befehlsspeicherelemente von 1664 Bit definieren, wobei 55 Bit ungenutzt bleiben. Zur korrekten Berechnung der effektiven Speicherausnutzung werden diese ungenutzen Bits nach Gleichung 5.13 in $\omega(B_\mathrm{S})^*$ einbezogen.

$$\omega(B_\mathrm{S})^* = (128+24+113+22+48+34+64+152+1024+55)\,\mathrm{Bit} = 1664\,\mathrm{Bit} \tag{5.13}$$

5.5.2.3 Berechnung der Speicherausnutzung der Befehlsspeicherelemente

Wie bei den Datenspeicherelementen soll nun auch für die Befehlsspeicherelemente die Speicherausnutzung η berechnet werden. Dazu wird die Bitbreite des für die Instruktionen und Operanden reservierten Bereichs der Befehlsspeicherelemente ins Verhältnis zu dessen gesamter Breite gesetzt. Bei Einsatz der Integritätsprüfungskennung IP berechnet sich die Speicherausnutzung $\eta_\mathrm{B,IP}$ nach Gleichung 5.14.

$$\eta_\mathrm{B,IP} = \frac{\omega(\mathrm{InstOp})}{\omega(B_\mathrm{IP})^*} \tag{5.14}$$

Bei Einsatz der Signaturkennung S berechnet sich $\eta_\mathrm{B,S}$ nach Gleichung 5.15.

$$\eta_\mathrm{B,S} = \frac{\omega(\mathrm{InstOp})}{\omega(B_\mathrm{S})^*} \tag{5.15}$$

Die sich so ergebenden Speicherausnutzungen $\eta_{\mathrm{B,IP}}$ und $\eta_{\mathrm{B,S}}$ werden in Tabelle 5.6 gezeigt.

Tabelle 5.6: Speicherausnutzung der Befehlsspeicherelemente

Speicherausnutzung $\eta_{\mathrm{B,IP}}$	**Speicherausnutzung $\eta_{\mathrm{B,S}}$**
20,00 %	7,69 %

5.5.3 Evaluation der Speicherausnutzung

Die geringe Speicherausnutzung von 0,16 bis 20,00 % bei Einsatz der Integritätsprüfungskennung IP bzw. 0,06 bis 7,69 % bei Einsatz der Signaturkennung S mag verschwenderisch wirken und gegen den Einsatz der Merkmale einer Datenspezifikationsarchitektur DSA sprechen. Aber die heute verfügbaren Speichergrößen und deren niedrige Preise rechtfertigen keine Ablehnung der Vorzüge in Form der umfassenden Fehlererkennung einer DSA aufgrund der geringen Speicherausnutzung.

Die angesprochenen Datenbusbreiten sind in der Praxis durchaus gebräuchlich. So nutzt z. B. die AMD-Grafikkarte W8100 einen 512 Bit breiten Datenbus zwischen der Zentraleinheit der Grafikkarte und dem auf ihr untergebrachten Speicher [5]. Neuere Modelle bieten sogar bis zu 4096 Bit breite Datenbusse als Schnittstelle zum Grafikspeicher [6].

Die große Anzahl an Bits, die zur Darstellung eines einzelnen Datenwerts innerhalb einer DSA benötigt werden, hat neben der geringen Speicherausnutzung weitere Nachteile:

- Werden die Daten aus einem angebundenen Speicher gelesen, dessen Datenbusbreite unterhalb der Datenspeicherelementbreite der Datenspezifikationsarchitektur liegt, so wird die Übertragung des gesamten Datenspeicherelements signifikant länger dauern, als es für den alleinigen Datenwert W in einer konventionellen Architektur notwendig ist. Damit hat ein Datenspeicherelement in einer DSA bei der Übertragung gegenüber einer konventionellen Architektur eine größere zeitliche Ausdehnung.
- Wird ein Speicher mit einer Datenbusbreite verwendet, die der Bitbreite eines Datenspeicherelements entspricht, so hat dieser Datenbus ggf. eine größere räumliche Ausdehnung gegenüber jenen von konventionellen Architekturen.

In beiden Fällen kann dies dazu führen, dass mehr Bitfehler bei der Übertragung der Datenspeicherelemente bei Störungen durch Umgebungseinflüsse auftreten können,

- bei serieller Übertragung von Teilen der Datenspeicherelemente durch die längere Übertragungsdauer und
- bei einer entsprechend hohen Datenbusbreite durch die größere räumliche Angriffsfläche.

Unter Verwendung der heute möglichen Fertigungsprozesse von integrierten Schaltungen können die beschriebenen Nachteile bzgl. erhöhter Bitfehlerraten relativiert werden, da man heutzutage verhältnismäßig große Mengen an Speicher direkt auf dem Die des Prozessors oder zumindest im gleichen Gehäuse unterbringen kann, so z. B. 128 MiB in manchen Intel x86-Prozessoren der Haswell-Generation [71]. Damit lassen dich die Leitungslängen zwischen Speicher und Prozessor minimieren und die Störsicherheit deutlich erhöhen.

6 Zusammenfassung und Weiterführungsmöglichkeiten

Zum Abschluss dieser Arbeit werden deren Beiträge zum Stand von Wissenschaft und Technik nochmals aufgelistet und anschließend Möglichkeiten der Weiterführung aufgezeigt.

6.1 Zusammenfassung der Ergebnisse der Arbeit

Die vorliegende Arbeit leistet die folgenden Beiträge zum Stand von Wissenschaft und Technik:

- die Identifikation von 20 datenflussbezogenen Fehler- und Angriffsarten,
- die Zusammenstellung einer umfassenden Sammlung der Eigenschaften von Daten in sicherheitsgerichteten Echtzeitsystemen und
- die Vorstellung der Datenspezifikationsarchitektur DSA, die unter Verwendung von umfangreichen Kennungen die identifizierten Eigenschaften von Daten hardwareverständlich beschreibt und es der Hardware dadurch ermöglicht, anhand dieser Eigenschaften den Datenfluss zu überwachen und die identifizierten Fehler- und Angriffsarten zu erkennen.

Die gegenüber dem Stand von Wissenschaft und Technik neuen Merkmale der Datenspezifikationsarchitektur DSA sind:

- die Definition von Messwertdatentypen in Form eines Werteintervalls zur Darstellung fehlerbehaftete Werte, um die Fortpflanzung dieser Fehler bei der Werteverarbeitung durch Intervallarithmetik verfolgen und eventuelle Genauigkeitsprobleme zu erkennen, zusammen mit speziellen Befehlen zur Prüfung der Genauigkeit,

- eine Wertebereichskennung [131], die es auf Hardwareebene erlaubt, einerseits die Plausibilität der in Datenworten enthaltenen Datenwerte zu prüfen, andererseits beim Ablegen eines Datenwerts in ein Datenwort eventuell auftretende Wertebereichsunter- bzw. -überschreitungen sofort aufzudecken,
- die Erweiterung der von Datentyparchitekturen bekannten Datentypkennungen [127] um abgeleitete Datentypen, bei denen die zulässigen Operationen eingeschränkt werden können, was eine verbesserte Isolation und damit eine erweiterte Prüfung der Kompatibilität der Datentypen von Operanden erlaubt,
- eine Einheitenkennung [126], die die Einheit des in einem Datenspeicherelement gespeicherten Datenwertes in Form von Potenzen der sieben SI-Basiseinheiten in hardwareverständlicher Weise beschreibt und es der Hardware somit gestattet, die Kompatibilität der Einheiten von Operanden bei Operationen sicherzustellen,
- eine Verarbeitungswegkennung [130], die den Weg der Daten von den sie erzeugenden Quellen über die Datenverarbeitungseinheiten bis hin zu den Datensenken beschreibt und der Hardware die Prüfung ermöglicht, ob alle Daten dem für sie vorgesehenen Weg durch das System folgen,
- eine Zeitschrittkennung [132], die den diskreten Entstehungszeitpunkt der Daten beschreibt, sowie eine Erweiterung der Befehle um Angaben der relativen temporalen Beziehungen der Operanden zueinander, mit deren Hilfe sich Synchronisations-, Aktualisierungs- und Zugriffsfehler aufdecken lassen,
- eine Fristkennung [128], die die hardwaregestützte Überwachung des Gültigkeitszeitraums von Daten erlaubt, die Echtzeitbedingungen unterworfen sind,
- eine Zykluszeitkennung [133], die es der Hardware ermöglicht, Unter- und Überschreitungen des Zyklus von zyklisch erwarteten Daten schnellstmöglich als Fehler zu erkennen,
- eine Signaturkennung [129], mit deren Hilfe sich alle Daten innerhalb eines Systems authentifizieren lassen, um bestimmte Angriffsszenarien auf die Datensicherheit zu erkennen, z. B. Wiedereinspielungsattacken,
- Datenportale in Form von Dateneingangs- und -ausgangsportalen, die es ermöglichen, Daten mit Einbeziehung der Adresse in die Integritätsprüfung bzw. Signatur zwischen Systemkomponenten zu übertragen; bei Nutzung einer kryptographischen Signatur der Daten übernehmen die Dateneingangsportale zusätzlich die Aufgabe der Prüfung der Signatur des Absenders und der Umsignierung mit dem eigenen geheimen Schlüssel und

- die Vorstellung einer Realisierungsmöglichkeit der Merkmale einer Datenspezifikationsarchitektur DSA in Datenflussarchitekturen durch Erweiterung der Verarbeitungseinheiten.

Dank dieser neuen und einigen weiteren dem Stand von Wissenschaft und Technik zuzurechnenden Merkmalen ist die vorgestellte Datenspezifikationsarchitektur in der Lage, alle 20 vorgestellten Fehler- und Angriffsarten zu erkennen.

6.2 Weiterführungsmöglichkeiten

Die in dieser Arbeit vorgestellte Datenspezifikationsarchitektur DSA konzentriert sich auf die detaillierte Spezifikation von Dateneigenschaften zur Fehlervermeidung und Erkennung von trotzdem verbleibenden oder auftretenden Datenflussfehlern.

Zusätzlich sollte die Erkennung von Kontrollflussfehlern durch Verfahren, wie sie durch Gollub in [41] vorgestellt wurden, bei der Realisierung einer DSA berücksichtigt werden. Ebenso sollten die ergänzenden Merkmale, die in der inhärent sicheren Mikroprozessorarchitektur ISMA in [125] beschrieben wurden, Eingang in die Realisierung einer DSA finden. Als Beispiele für entsprechende Merkmale sind dabei Ansprungbefehle, der nicht für die Software zugängliche Stapelspeicher und der Verzicht auf Unterbrechungen zu nennen. Auf arithmetische Register sollte bei der Realisierung ebenfalls verzichtet werden, wie es in [115] vorgeschlagen und bei ISMA entsprechend umgesetzt wurde.

Diese Arbeit könnte daher wie folgt fortgeführt werden:

- Entwurf und Realisierung einer DSA auf einem FPGA, idealerweise unter Einbeziehung der folgenden Erweiterungen:
  - der oben erwähnten kontrollflussbezogenen Fehlervermeidungs- und -erkennungsmerkmale von Gollub [41], wobei die Merkmale in Form weiterer Kennungen innerhalb der Befehlsspeicherelemente realisiert werden könnten,
  - der ebenfalls oben erwähnten zusätzlichen Sicherheitsmerkmale von ISMA [125],
  - der Erweiterung der Einheitenkennung auf Basis der Vorschläge in [110], z. B. durch Einführung einer weiteren Einheit für m_r und Skalierungsfaktoren,

  - der Einführung einer Färbungskennung – engl. „taint“ –, die es erlaubt, ungeprüfte Benutzereingaben oder Kommunikationsnachrichten also solche zu kennzeichnen und die Nutzung dieser Daten durch die Hardware zu überwachen, wie z. B. in HDFI [114] oder PUMP [31, 33] vorgestellt,

- Erweiterung bestehender bzw. Entwicklung neuer Entwicklungswerkzeuge wie Übersetzer und Binder, um Software für eine DSA erstellen zu können, unter Berücksichtigung und Erweiterung der Vorschläge zur Spezifikation der Dateneigenschaften in dieser Arbeit,
- Test der Fehlererkennungsraten einer DSA unter Nutzung von Fehlerinjektionsverfahren,
- Evaluation des zusätzlichen Laufzeitbedarfs durch Prüfungen, die parallel zu den eigentlichen Operationen ausgeführt werden, jedoch einen höheren Laufzeitbedarf als diese aufweisen, sowie Prüfungen, die nicht parallel zu den eigentlichen Operationen ausgeführt werden können, und schlussendlich ggf.
- Entwicklung eines auf der FPGA-Realisierung der DSA aufsetzenden ASICs.

Datentyp- und -strukturarchitekturen und die in ihnen genutzten einfachen und leistungsfähigen Fehlererkennungsmerkmale sind weitgehend in Vergessenheit geraten. Literatur über diese Architekturarten ist meist nur noch in Antiquariaten aufzufinden. Entsprechend wäre die Fortführung und Weiterentwicklung der Ergebnisse dieser Arbeit sehr zu begrüßen, um diesen dasselbe Schicksal zu ersparen.

Literaturverzeichnis

[1] AEG Datenverarbeitung: TR 4 Bedienungshandbuch

[2] AIRBUS: Fly-by-wire; http://www.airbus.com/innovation/proven-concepts/in-design/fly-by-wire/

[3] J. Åkerberg, M. Björkman: Exploring Network Security in PROFIsafe; Computer Safety, Reliability, and Security; Vol. 5775 of the series Lecture Notes in Computer Science; S. 67–80; 2009

[4] AMD: AMD64 Architecture Programmer's Manual Vol. 2: System Programming; http://developer.amd.com/wordpress/media/2012/10/24593_APM_v2.pdf

[5] AMD: AMD FirePro W8100 Professional Graphics; http://www.amd.com/en-us/products/graphics/workstation/firepro-3d/8100

[6] AMD: Radeon Pro Duo; http://www.amd.com/en-us/products/graphics/desktop/radeon-pro-duo

[7] ARM Limited: Migrating from IA-32 to ARM; Application Note 274; ARM DAI 0274; 2011

[8] R. C. Baumann, E. B. Smith: Neutron-Induced Boron Fission as a Major Source of Soft Errors in Deep Submicron SRAM Devices; Reliability Physics Symposium, 2000. Proceedings, 38th Annual 2000 IEEE International; S. 152–157; 2000

[9] T. Beierlein, O. Hagenbruch: Taschenbuch Mikroprozessortechnik; 4. Auflage, 2011; Carl Hanser Verlag; ISBN 978-3-446-42331-2

[10] D. Bovet, M. Cesati: Understanding the Linux Kernel; 1. Auflage, 2000; O'Reilly Verlag; ISBN 0-596-00002-2

[11] A. Bradbury, G. Ferris, R. Mullins: Tagged memory and minion cores in the lowRISC SoC; 2014; http://www.lowrisc.org/docs/

[12] U. Brinkschulte, T. Ungerer: Mikrocontroller und Mikroprozessoren; 3. Auflage, 2010; Springer Verlag; ISBN 978-3-642-05397

[13] D. T. Brown: Error Detecting and Correcting Binary Codes for Arithmetic Operations; IRE Transactions on Electronic Computers; Vol. EC-9, Issue 3; 1960

[14] M. Broy: Challenges in Automotive Software Engineering; ICSE '06 Proceedings of the 28th international conference on Software engineering, S. 33–42; 2006

[15] B. Buckwalter: The dimensional package; Haskell; `https://hackage.haskell.org/package/dimensional`

[16] S. Chandra, T. Reps: Physical Type Checking for C; PASTE '99 Proceedings of the 1999 ACM SIGPLAN-SIGSOFT workshop on Program analysis for software tools and engineering; S. 66–75; 1999

[17] R. N. Charette: This Car Runs on Code; `http://spectrum.ieee.org/transportation/systems/this-car-runs-on-code`; 2009

[18] B. Chelf: Measuring software quality - A Study of Open Source Software; Coverity; `https://www.coverity.com/library/pdf/open_source_quality_report.pdf`

[19] H.-C. Chi: ARM Processor Architecture; CSIE34600 Introduction to Embedded System Design; `http://soc.csie.ndhu.edu.tw/source/introemb-13/unit2.ppt`

[20] H.-C. Chi: ARM Instructions; CSIE34600 Introduction to Embedded System Design; `http://soc.csie.ndhu.edu.tw/source/introemb-13/unit1.ppt`

[21] S. Chiricescu, A. DeHon, D. Demange, S. Iyer, A. Kliger, G. Morrisett, B. C. Pierce, H. Reubenstein, J. M. Smith, G. T. Sullivan, A. Thomas, J. Tov, C. M. White, D. Wittenberg: SAFE: A Clean-Slate Architecture for Secure Systems; `http://www.crash-safe.org/docs/HST2013-SAFE.html`; 2013

[22] CODENOMICON: The Heartbleed Bug; `http://heartbleed.com/`; 2014

[23] R. P. Colwell, E. F. Gehringer, E. D. Jensen: Performance effects of architectural complexity in the Intel 432; ACM Transactions on Computer Systems (TOCS), Vol. 6, Issue 3; S. 296–339; 1988

[24] Coverity Scan: 2013 Open Source Report

[25] I. F. Currie: NewSpeak: a reliable programming language; High-Integrity Software, Part of the series Software Science and Engineering; S. 122–158; 1989

[26] I. F. Currie: NewSpeak: an unexceptional language; Software Engineering Journal; Vol. 1, Issue 4; S. 170–176; 1986

[27] B. Dasarathy: Timing Constraints of Real-Time Systems: Constructs for Expressing Them, Methods of Validating Them; IEEE Transactions on Software Engineering, Vol.11, Issue 1; 80–86; 1985

[28] A. DeHon, T. F. Knight, Jr., B. Krikeles, B. Loyall, G. Morrisett, B. C. Pierce, J. M. Smith, H. Reubenstein, J. Rosenberg, O. Shivers, G. Sullivan, C. White: SAFE: A Semantically Aware Foundation Environment for CRASH; http://www.crash-safe.org/assets/BAE-SAFE-CRASH-pub.pdf; 2010

[29] A. v. Delft: A Java Extension With Support for Dimensions; Software-Practice & Experience; Vol. 29, Issue 7; S. 605–616; 1999

[30] D. Delimarsky: Units of Measure (F#); Microsoft MSDN; https://msdn.microsoft.com/visualfsharpdocs/conceptual/units-of-measure-[fsharp]

[31] U. Dhawan, C. Hriţcu, R. Rubin, N. Vasilakis, S. Chiricescu, J. M. Smith, T. F. Knight, B. C. Pierce, A. DeHon: Architectural Support for Software-Defined Metadata Processing; http://www.crash-safe.org/docs/PUMP-ASPLOS2015.html; 2015

[32] U. Dhawan, A. Kwon, E. Kadric, C. Hriţcu, B. C. Pierce, J. M. Smith, G. Malecha, G. Morrisett, T. F. Knight, Jr., A. Sutherland, T. Hawkins, A. Zyxnfryx, D. Wittenberg, P. Trei, S. Ray, G. Sullivan, A. DeHon: Hardware Support for Safety Interlocks and Introspection; http://www.crash-safe.org/docs/HWInterlocks-SASO-AHANS2012.html; 2012

[33] U. Dhawan, N. Vasilakis, R. Rubin, S. Chiricescu, J. M. Smith, T. F. Knight, Jr., B. C. Pierce, A. DeHon: PUMP: A Programmable Unit for Metadata Processing; http://www.crash-safe.org/docs/PUMP-HASP-2014.html; 2014

[34] R. Eisenberg: The Units package; Haskell; https://hackage.haskell.org/package/units

[35] T. Erdner, W. A. Halang, J. K.-Y. Ng, S. K. Chun Chan: Synchronisation der lokalen Uhren an ringförmigen Übertragungsmedien angeschlossener Einheiten; Patent DE10253534B4; 2002

[36] E. Feustel: On the Advantages of Tagged Architectures; IEEE Transactions on Computers, Volume C-22, Number 7, S. 644–656; 1973

[37] E. Feustel: The Rice research computer: a tagged architecture; AFIPS '72 (Spring) Proceedings of the May 16-18, 1972, spring joint computer conference, S. 369–377; 1972

[38] P. Forin: Vital Coded Microprocessor Principles and Application for Various Transit Systems; 1989; IFAC Control, Computers, Communications; S. 79–84; Paris

[39] W. Giloi: Rechnerarchitektur; 2. Auflage, 1993; Springer-Verlag; ISBN 3-540-56355-5

[40] GNU Pascal: 6.2.11.1 Subrange Types; `http://www.gnu-pascal.de/gpc/Subrange-Types.html`

[41] L. Gollub: Verfahren zur Kontrollflussüberwachung in sicherheitsgerichteten Rechensystemen; 2014, VDI Verlag; ISBN 978-3-18-383210-1

[42] Google: Say Hello to Waymo; `https://storage.googleapis.com/sdc-prod/v1/press/Waymo_One-Pager_Introduction.pdf`

[43] D. Gove, R. Prakash: Detecting memory access errors; 2015; `https://blogs.oracle.com/raj/resource/Silicon-Secured-Memory-Application.pdf`

[44] W. A. Halang, R. M. Konakovsky: Sicherheitsgerichtete Echtzeitsysteme; 1. Auflage, 1999; Springer-Verlag; ISBN 3-486-24036-6

[45] W. A. Halang, R. J. Lauber: Echtzeitsysteme I, Kurs 21311; Version 1.0.1; Stand Februar 2007

[46] W. A. Halang, Z. Li: Echtzeitsysteme II, Kurs 21312; Version 1.0.1; Stand Dezember 2009

[47] D. Hansen: Intel Memory Protection Extensions (Intel MPX) for Linux; 2016; `https://01.org/blogs/2016/intel-mpx-linux`

[48] C. Houben: Integration of Physical Units into the Real-time Programming Language PEARL; IFAC-PapersOnLine, Volume 48, Ausgabe 4; S. 123–128; 2015

[49] IBM: IBM System/360 System Summary; IBM Systems Reference Library; File Number S360-00; Order No. GA22-6810-12

[50] IC Insights: Qualcomm and Samsung Pass AMD in MPU Ranking; http://www.icinsights.com/news/bulletins/Qualcomm-And-Samsung-Pass-AMD-In-MPU-Ranking/; abgerufen am 03.03.2014

[51] IEC 61508-2:2010: Functional safety of electrical / electronic / programmable electronic safety-related systems - Requirements for electrical / electronic / programmable electronic safety-related systems (Edition 2.0, 2010-04)

[52] IEC 61508-3:2010: Functional safety of electrical / electronic / programmable electronic safety-related systems - Part 3: Software requirements (Edition 2.0, 2010-04)

[53] IEC 61508-7:2010: Functional safety of electrical / electronic / programmable electronic safety-related systems - Overview of techniques and measures (Edition 2.0, 2010-04)

[54] IEC 61784-3:2016: Industrial communication networks - Profiles - Part 3: Functional safety fieldbuses – General rules and profile definitions (Edition 3.0, 2016-05)

[55] IEC 61784-3-2:2010: Industrial communication networks - Profiles - Part 3-2: Functional safety fieldbuses - Additional specifications for CPF 2 (Edition 2.0, 2010-06)

[56] IEC 61784-3-3:2010: Industrial communication networks - Profiles - Part 3-3: Functional safety fieldbuses - Additional specifications for CPF 3 (Edition 2.0, 2010-06)

[57] IEEE Computer Society: IEEE Standard for Interval Arithmetic; IEEE Std 1788-2015; 2015

[58] Infineon: Infineon AURIX powered by TriCore, Highly integrated and performance optimized, 32-bit microcontrollers for automotive and industrial applications; 2016; http://www.infineon.com/dgdl/Infineon-New+Tricore+Family+Brochure-BC-v01_00-EN.pdf?fileId=db3a30431f848401011fc664882a7648

[59] Intel: 80386 System Software Writer's Guide; 1991; ISBN 1-55512-023-7

[60] Intel: Control-flow Enforcement Technology Preview; Revision 1.0; 2016; https://software.intel.com/sites/default/files/managed/4d/2a/control-flow-enforcement-technology-preview.pdf

[61] ISO: IEC 80000-13:2008, Quantities and units Part 13: Information science and technology

[62] ISO/IEC 9796-2:2002, Information technology - Security techniques - Digital signature schemes giving message recovery - Part 2: Integer factorization based mechanisms

[63] ITU-T: X.200, Information Technology - Open Systems Interconnection - Basic Reference Model: The Basic Model; (07/1994); 1994

[64] F. Kaderali: Kommunikationsnetze und -protokolle; `http://www.kaderali.de/fileadmin/vorlesungsskripte/Buch%20KP%20(A4).pdf`; 2005

[65] R. Kirchner, U. W. Kulisch: Hardware Support for Interval Arithmetic; Reliable Computing, June 2006, Volume 12, Issue 3, S. 225–237; 2006

[66] M. Kompf: Die 12 häufigsten Programmierfehler; `http://cplus.kompf.de/artikel/errc.html`; abgerufen am 18.01.2014

[67] R. Konakovsky: Definition und Berechnung der Sicherheit von Autmatisierungssystemen; 1. Auflage, 1977; Vieweg Verlag; ISBN 3-528-03327-4

[68] I. Koren, C. Krishna: Fault-Tolerant Systems; 1. Auflage, 2007; Morgan Kaufmann Verlag; ISBN 978-0-12-088525-1

[69] KOSMOS: Computer-Praxis CP1; 1983; Franckh'sche Verlagshandlung

[70] H. Krebs, U. Haspel: Ein Verfahren zur Software-Verifikation; Regelungstechnische Praxis rtp, 26; S. 73–78; 1984

[71] N. Kurd, M. Chowdhury, E. Burton, T. P. Thomas, C. Mozak, B. Boswell, P. Mosalikanti, M. Neidengard, A. Deval, A. Khanna, N. Chowdhury, R. Rajwar, T. M. Wilson, R. Kumar: Haswell: A Family of IA 22 nm Processors; IEEE Journal of Solid-State Circuits, Vol. 50, Issue 1; S. 49–58; 2014

[72] D. Kushner: The Real Story of Stuxnet; `http://spectrum.ieee.org/telecom/security/the-real-story-of-stuxnet`

[73] P. Laackmann, M. Janke: 25 Jahre Chipkarten-Angriffe; `https://events.ccc.de/congress/2013/Fahrplan/system/attachments/2227/original/25_Jahre_Chipkartenangriffe-Marcus_Janke_Peter_Laackmann.pdf`; 2013

[74] M. Larabel: GCC Soars Past 14.5 Million Lines Of Code & I'm Real Excited For GCC 5; 2015

https://www.phoronix.com/scan.php?page=news_item&px=MTg3OTQ

[75] R. Lauber, P. Göhner: Prozessautomatisierung 1; 3. Auflage, 1999; Springer-Verlag; ISBN 3-540-65318-X

[76] B. Łent: A Contribution To The Design Of A Disjunctive Computer Architecture For Real Time Control Systems; Dissertation; FernUniversität in Hagen; 1995

[77] N. G. Leveson, C. S. Turner: An Investigation of the Therac-25 Accidents; Computer, Vol. 26, Issue 7; S. 18–41; 1993

[78] H. Levy: Capability-Based Computer Systems; 1984; Digital Equipment Corporation; ISBN 9-9323376-22-3

[79] J.-L. Lions et al.: Ariane 501 Inquiry Board report; 1996; http://esamultimedia.esa.int/docs/esa-x-1819eng.pdf

[80] R. Männer: Strong Typing and Physical Units; ACM SIGPLAN Notices, Vol. 21, Issue 3; S. 11–20; 1986

[81] Mars Climate Orbiter Mishap Investigation Board Phase I Report; November 10, 1999; ftp://ftp.hq.nasa.gov/pub/pao/reports/1999/MCO_report.pdf

[82] A. Mayr: The Architecture of the Burroughs B5000 - 20 Years Later and Still Ahead of the Times?; 1982; http://www.smecc.org/The%20Architecture%20%20of%20the%20Burroughs%20B-5000.htm

[83] A. J. Menezes, P. C. van Oorschot, S. A. Vanstone: Handbook of Applied Cryptography; 1996; CRC Press; ISBN 0-8493-8523-7

[84] H.-P. Messmer: PC-Hardwarebuch; 6. Auflage, 2000; Addison-Weslay Verlag; ISBN 3-8273-1461-5

[85] A. Meixner, D. J. Sorin: Error Detection Using Dynamic Dataflow Verification; 16th International Conference on Parallel Architecture and Compilation Techniques (PACT 2007); S. 104–118; 2007

[86] MITRE Corporation: 2011 CWE/SANS Top 25 Most Dangerous Software Errors; https://cwe.mitre.org/top25/archive/2011/2011_cwe_sans_top25.pdf; abgerufen am 18.01.2014

[87] G. Myers: Advances in Computer Architecture; 2. Auflage, 1978; John Wiley & Sons; ISBN 0-471-07878-6

[88] National Highway Traffic Safety Administration (NHTSA): RECALL Subject: Software may Disable Steering in Cold Temperatures; NHTSA Campaign Number: 13V588000; 2013

[89] NISSAN: Nissan Pivo Concept Press Kit: Overview; `http://nissannews.com/en-US/nissan/usa/releases/435dd488-658e-433a-a57a-cd0184e4b51c`

[90] E. Normand: Single Event Upset at Ground Level; IEEE Transactions on Nuclear Science, Vol. 43, Issue 6; S. 2742–2750; 1996

[91] NXP: Safety Manual for MPC5744P; Dokumentennummer MPC5744PSM; Rev. 3; 2014

[92] A. Opler: Das IBM-System/360 und seine Programmiertechniken; 1968; R. Oldenbourg Verlag

[93] Oracle: Introduction to SPARC M7 and Application Data Integrity (ADI); `https://swisdev.oracle.com/_files/What-Is-ADI.html`

[94] Organisation Intergouvernementale de la Convention du Mètre: The International System of Units (SI); 8e edition; 2006

[95] S. Phillips: M7: Next Generation SPARC; Hotchips 26; 2014; `http://www.hotchips.org/wp-content/uploads/hc_archives/hc26/HC26-12-day2-epub/HC26.12-8-Big-Iron-Servers-epub/HC26.12.820-Next_Gen_SPARC_Phillips-Oracle-FinalPub.pdf`

[96] S. Ramesh: Software's Significant Impact on the Automotive Industry; Frost & Sullivan Market Insight; 2008

[97] J. A. Rawlinson: Report on the Therac-25, OCTRF/OCI Physicists Meeting, Kingston, Ontario, Canada; 1987

[98] RFC 791: Internet Protocol, DARPA Internet Program, Protocol Specification; September 1981; `http://www.rfc-base.org/rfc-791.html`

[99] RFC 792: Internet Control Message Protocol, DARPA Internet Program, Protocol Specification; September 1981; `http://www.rfc-base.org/rfc-792.html`

[100] RFC 793: Transmission Control Protocol, DARPA Internet Program, Protocol Specification; September 1981; `http://www.rfc-base.org/rfc-793.html`

[101] RFC 1700: Assigned Numbers; Oktober 1994; http://www.rfc-base.org/rfc-1700.html

[102] RFC 2460: Internet Protocol, Version 6 (IPv6), Specification; Dezember 1998; http://www.rfc-base.org/rfc-2460.html

[103] SAFECode, S. Simpson et al.: Fundamental Practices for Secure Software Development; 2. Auflage, 2011; http://www.safecode.org/publications/SAFECode_Dev_Practices0211.pdf

[104] G. Sapper: Rechenanlage TELEFUNKEN TR4; http://www.qslnet.de/member/dj4kw/tr4.htm

[105] J. Sauerer: Smart Sensors; AMA Fachverband für Sensorik e.V., Wunstorf; Forschungsverbund Erneuerbare Energien -FVEE-, Berlin: Sensorik für erneuerbare Energien und Energieeffizienz: Beiträge zum Workshop vom AMA Fachverband für Sensorik e.V. und vom ForschungsVerbund Erneuerbare Energien am 12. und 13. März 2013 in Berlin-Adlershof; S. 18–24; 2013

[106] SAFE: A secure computing platform; http://www.crash-safe.org/

[107] SEMI: Why Moore Matters; http://semi.org/en/node/55026; 2015

[108] U. Schiffel: Hardware Error Detection Using AN-Codes; Dissertation; Technische Universität Dresden; 2011

[109] L. Schleupner: Perfekt sichere Kommunikation in der Automatisierungstechnik; Dissertation; FernUniversität in Hagen; 2012

[110] R. Schlick, W. Herzner, T. Le Sergent: Checking SCADE Models for Correct Usage of Physical Units; Computer Safety, Reliability, and Security; Volume 4166 of the series Lecture Notes in Computer Science; S. 358–371; 2006

[111] B. Schneier: Heartbleed; http://www.schneier.com/blog/archives/2014/04/heartbleed.html

[112] D. P. Siewiorek, R. S. Swarz: Reliable computer systems: design and evaluation; 3. Auflage, 1998; A K Peters, Ltd.; ISBN 1-56881-092-X

[113] N. Shimizu: Nissan Puts Steer-by-Wire on the Road: An In-Depth Look at the Technology; Nikkei BP Japan Technology Report / A1403-058-005

[114] C. Song, H. Moon, M. Alam, I. Yun, B. Lee, T. Kim, W. Lee, Y. Paek: HDFI: Hardware-Assisted Data-Flow Isolation; IEEE Symposium on Security and Privacy; 2016

[115] H. Stieger, W. A. Halang: Eine hochsprachenorientierte Rechnerarchitektur ohne arithmetische Register; 1. Auflage, 2003; IFB Verlag Paderborn; ISBN 3-931263-39-8

[116] P. Taidi: SAS and Oracle SPARC M7 Silicon Secured Memory; 2016; `https://blogs.oracle.com/partnertech/entry/sas_and_oracle_sparc_m7`

[117] J. R. Taylor: Fehleranalyse; 1. Auflage, 1988; VCH Verlagsgesellschaft mbH; ISBN 3-527-26878-2

[118] J. Teller: Problematik der Datenflussfehler; Angewandte Informatik, Ausgabe 29, Nr. 6; S. 240–247; 1987

[119] Tesla: All Tesla Cars Being Produced Now Have Full Self-Driving Hardware; `https://www.tesla.com/blog/all-tesla-cars-being-produced-now-have-full-self-driving-hardware`

[120] Texas Instruments: Safety Manual for RM48x Hercules ARM-Based Safety Critical Microcontrollers, User's Guide; Dokumentennummer SPNU577D; 2015

[121] S. Tucker Taft, R. A. Duff: Ada 95 reference manual: language and standard libraries; international standard ISO/IEC 8652:1995(E); 1997; Springer-Verlag; ISBN 3-540-63144-5

[122] P. M. Ulbrich: Ganzheitliche Fehlertoleranz in eingebetteten Softwaresystemen; Dissertation; Friedrich-Alexander-Universität Erlangen-Nürnberg; 2014

[123] University of Cambridge: Capability Hardware Enhanced RISC Instructions (CHERI); `https://www.cl.cam.ac.uk/research/security/ctsrd/cheri.html`

[124] L. Wendt: Taschenbuch der Regelungstechnik mit MATLAB und Simulink; 10. Auflage, 2014; Europa-Lehrmittel; ISBN 978-3-80-855679-5

[125] S. Widmann: Eine inhärent sichere Mikroprozessorarchitektur; 2015; VDI Verlag; ISBN 978-3-18-384310-7

[126] S. Widmann, W. A. Halang: Vorrichtung und Verfahren zur gerätetechnischen Erkennung inkompatibler Operandeneinheiten in Datenverarbeitungseinheiten; Patentanmeldung beim Deutschen Patent- und Markenamt

[127] S. Widmann, W. A. Halang: Vorrichtung und Verfahren zur gerätetechnischen Einschränkung der zulässigen Operationen auf Daten in Datenverarbeitungs-

einheiten; Patentanmeldung beim Deutschen Patent- und Markenamt

[128] S. Widmann, W. A. Halang: Vorrichtung und Verfahren zur gerätetechnischen Erkennung der Datennutzung außerhalb ihres Gültigkeitszeitraums in Datenverarbeitungseinheiten; Patentanmeldung beim Deutschen Patent- und Markenamt

[129] S. Widmann, W. A. Halang: Vorrichtung und Verfahren zur gerätetechnischen Erkennung von absichtlichen oder durch Störungen und / oder Fehler verursachten Datenverfälschungen in Datenverarbeitungseinheiten; Patentanmeldung beim Deutschen Patent- und Markenamt

[130] S. Widmann, W. A. Halang: Vorrichtung und Verfahren zur gerätetechnischen Erkennung von Datenflussfehlern in Datenverarbeitungseinheiten und -systemen; Patentanmeldung beim Deutschen Patent- und Markenamt

[131] S. Widmann, W. A. Halang: Vorrichtung und Verfahren zur gerätetechnischen Erkennung von Wertebereichsverletzungen von Datenwerten in Datenverarbeitungseinheiten; Patentanmeldung beim Deutschen Patent- und Markenamt

[132] S. Widmann, W. A. Halang: Vorrichtung und Verfahren zur gerätetechnischen Erkennung von Synchronisations- und Datenaktualisierungsfehlern in Datenverarbeitungseinheiten; Patentanmeldung beim Deutschen Patent- und Markenamt

[133] S. Widmann, W. A. Halang: Vorrichtung und Verfahren zur gerätetechnischen Erkennung von Verletzungen von zyklischen Echtzeitbedingungen in Datenverarbeitungseinheiten und -systemen; Patentanmeldung beim Deutschen Patent- und Markenamt

[134] xkcd: Heartbleed Explanation; http://xkcd.com/1354/

[135] N. Zeldovich, H. Kannan, M. Dalton, C. Kozyrakis: Hardware Enforcement of Application Security Policies Using Tagged Memory; Stanford University; 2008; http://www.scs.stanford.edu/~nickolai/papers/zeldovich-loki.pdf

Online-Buchshop für Ingenieure

VDI nachrichten **BUCHSHOP**

Online-Shops

Fachliteratur und mehr - jetzt bequem online recherchieren & bestellen unter: www.vdi-nachrichten.com/Der-Shop-im-Ueberblick

Täglich aktualisiert: Neuerscheinungen VDI-Schriftenreihen

Im Buchshop von vdi-nachrichten.com finden Ingenieure und Techniker ein speziell auf sie zugeschnittenes, umfassendes Literaturangebot.

Mit der komfortablen Schnellsuche werden Sie in den VDI-Schriftenreihen und im Verzeichnis lieferbarer Bücher unter 1.000.000 Titeln garantiert fündig.

Im Buchshop stehen für Sie bereit:

VDI-Berichte und die Reihe **Kunststofftechnik:**

Berichte nationaler und internationaler technischer Fachtagungen der VDI-Fachgliederungen

Fortschritt-Berichte VDI:

Dissertationen, Habilitationen und Forschungsberichte aus sämtlichen ingenieurwissenschaftlichen Fachrichtungen

Newsletter „Neuerscheinungen":

Kostenfreie Infos zu aktuellen Titeln der VDI-Schriftenreihen bequem per E-Mail

Autoren-Service:

Umfassende Betreuung bei der Veröffentlichung Ihrer Arbeit in der Reihe Fortschritt-Berichte VDI

Buch- und Medien-Service:

Beschaffung aller am Markt verfügbaren Zeitschriften, Zeitungen, Fortsetzungsreihen, Handbücher, Technische Regelwerke, elektronische Medien und vieles mehr – einzeln oder im Abo und mit weltweitem Lieferservice

Die Reihen der Fortschritt-Berichte VDI:

1 Konstruktionstechnik/Maschinenelemente
2 Fertigungstechnik
3 Verfahrenstechnik
4 Bauingenieurwesen
5 Grund- und Werkstoffe/Kunststoffe
6 Energietechnik
7 Strömungstechnik
8 Mess-, Steuerungs- und Regelungstechnik
9 Elektronik/Mikro- und Nanotechnik
10 Informatik/Kommunikation
11 Schwingungstechnik
12 Verkehrstechnik/Fahrzeugtechnik
13 Fördertechnik/Logistik
14 Landtechnik/Lebensmitteltechnik
15 Umwelttechnik
16 Technik und Wirtschaft
17 Biotechnik/Medizintechnik
18 Mechanik/Bruchmechanik
19 Wärmetechnik/Kältetechnik
20 Rechnerunterstützte Verfahren (CAD, CAM, CAE CAQ, CIM ...)
21 Elektrotechnik
22 Mensch-Maschine-Systeme
23 Technische Gebäudeausrüstung

ISBN 978-3-18-385610-7