# On-line Fehler-Erkennung und schnelle Wiederherstellungs-Techniken für zuverlässige eingebettete Prozessoren

Matthias Pflanz

mpflanz@de.ibm.com

Diese Arbeit präsentiert neue Verfahren zur on-line Observierung und parallelem Check von Prozessoren durch Verfeinerung und Verbesserung bekannter Techniken und Methoden sowie Einführung neuer Ideen. Motivation war die Entwicklung von Techniken zur Fehlererkennung im selben Maschinenzyklus. Des Weiteren werden Verfahren zur schnellen Fehler-Kompensation vorgestellt. Ziel war die Erhöhung der Zuverlässigkeit von Prozessoren speziell für sicherheits-kritische Systeme.

## 1 Einführung

Computer-Komponenten werden seit vielen Jahren mit einem ständig wachsenden Anteil in eingebetteten Systemen für Steuerungs- und Signalverarbeitungs-Applikationen verwendet. Der Trend von eingebetteten Computer-Systemen geht in Richtung Single-Chip-Lösungen. Sogenannte System-on-a-Chip (SoC) beinhalten unterschiedliche wiederverwendbare funktionale Blöcke (embedded cores), wie Micro-Prozessoren, DSPs, Speicher, Interface-Blöcke, etc. Die zunehmende Komplexität dieser Systeme wirft eine Reihe neuer Probleme für Logik-Designer, Test- und Verifikations-Ingenieure und Benutzer auf. Durch unzureichende Zugriffsmöglichkeit auf Teilkomponenten in SoCs sind klassische Testverfahren nicht mehr ausreichend. Damit steigt die Wahrscheinlichkeit unentdeckter Herstellungsfehler. Als spätere Ausfallursachen können außerdem transiente Fehler (Soft-Errors) oder - insbesondere in (autonomen) Systemen im Langzeiteinsatz - Alterungseffekte oder sogenannte "wachsende Defekte" in Frage kommen. Eingebettete Prozessor-basierte Systeme finden auch in sicherheits-kritischen Systemen Verwendung. Insbesondere für diese Systeme werden hohe Forderungen hinsichtlich der Zuverlässigkeit gestellt. Erschwerend sind insbesondere problematische Umgebungsbedingungen, wie starke Temperaturschwankungen, mechanische Beanspruchung durch Erschütterungen, etc. Durch verbesserte Halbleitertechnologien und damit kleinerer Transistor-Kanallängen besteht eine höhere Empfindlichkeit gegenüber transienten Effekten. Diese oft auch als Mehrfachfehler auftretenden Effekte können in einem Fertigungstest vorab nicht getestet werden. Bereits seit Jahren zeichnet sich der Trend ab, den Test (partiell) auf den Chip zu verlagern. Die Migration von Test-Komponenten auf den Chip muss dahingehend erweitert werden, dass auch effiziente on-line Check-Methoden zur Erkennung von Fehlern während des Betrie-

bes zum Einsatz kommen. Um die Verfügbarkeit von Prozessoren in eingebetteten Systemen zu gewährleisten, sind Fehler-Kompensationsmethoden notwendig. Eine Fehlerausbreitung im Prozessor kann nur durch eine schnelle Fehlerkorrektur verhindert werden. Die Art der Kompensationsmethode muss sich hierbei nach der Fehlercharakteristik richten. Transiente Fehler (verursacht z.B. durch temporäre "single event upsets") erfordern hierbei andere Maßnahmen als permanente Fehler (verursacht z.B. durch reale physikalische Defekte). Besitzen eingebettete Prozessoren die Fähigkeiten zur on-line Fehlererkennung und zur schnellen Kompensation, kann die Zuverlässigkeit und die Qualität dieser Systeme verbessert werden.

Im folgenden Abschnitt werden die Grundlagen und Ansätze zu den in dieser Arbeit beschriebenen Verfahren erläutert. Anschließend werden Techniken vorgestellt, mit denen Prozessor-Komponenten on-line überwacht werden können. Im vierten Abschnitt wird gezeigt, wie Kontrollsignale bzw. Prozessor-Zustände während des Betriebes gecheckt werden können. Eine Methode zur schnellen Fehlerkompensation in Prozessoren mit der Fähigkeit zur Unterscheidung zwischen transienten und permanenten Fehlern wird im fünften Kapitel präsentiert.

## 2 Grundlagen

Ein *eingebettetes System* (embedded system) ist eine Computer-basierte Komponente oder Komponenten-Gruppe innerhalb eines mechanischen, elektrischen oder optischen Systems. Hauptfunktionen dieses (intelligenten) Kerns ist die Steuerung der System-Funktionalität und die Verarbeitung von Daten. Ein eingebettetes System verfügt i.d.R. über Interface-Bausteine, A/D-Wandler, mixed-signal Komponenten, Speicher, Benutzer-definierte Logik und über ein oder mehrere Prozessoren (embedded processor). Durch die ständig wachsende Integrationsdichte dieser Systeme werden Verifikation und Fertigungstests für diese Systeme auf Grund des reduzierten Zugriffs auf interne Komponenten (Cores) zunehmend erschwert. Der Terminus *Fehler* lässt sich durch die englischen Begriffe *fault*, *error* und *failure* beschreiben. Eine Einwirkung von außen führt zu einer Verfälschung (fault) (z.B. auf einer Daten-Leitung). Ein Fehler (error) liegt vor, wenn diese fehlerhaften Daten in ein Register geschrieben werden. Ein Systemfehlverhalten bzw. ein Systemversagen tritt dann auf, wenn sich dieser Fehler entsprechend ausgebreitet hat (siehe auch Abb. 1).

Bei Fehlern in Prozessoren muss zwischen *permanenten* und *transienten* Effekten unterschieden werden. Permanente Fehler können auf reale physikalische Defekte, Fertigungs-Ungenauigkeiten in der Halbleiter-Struktur oder Alterungseffekte zurückgeführt werden. Sie verursachen ständige oder regelmäßig wiederkehrende Fehler. Transiente Fehler hingegen können sporadisch auftreten. Ursachen hierfür können Alpha-Teilchen, Cross-Talk zwischen Leitungen, Vdd-GND-Rauschen, u.a. sein. Hinsichtlich ihrer Auswirkung innerhalb eines Prozessors muss bei Fehlern zwischen *effektiv*, *latent* und *overwritten* differenziert werden. Ein effektiver Fehler wirkt sich sofort, der latente Fehler mit einer entsprechenden Verzögerung auf die Funktion des Prozessors aus. Ein "overwritten error" kennzeichnet einen aufgetretenen Fehler, der vor einer möglichen Ausbreitung überschrieben oder maskiert wird.

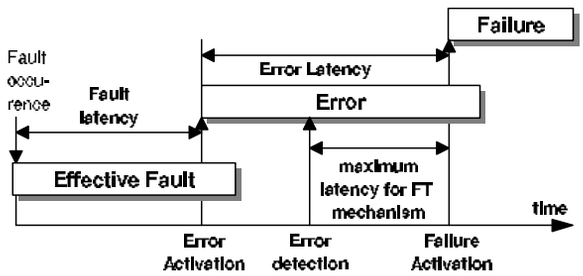


Abbildung 1: Zeitverhalten von Fault, Error und Failure

# 3 On-line Check Techniken für Prozessor Komponenten

## 3.1 State-of-the-art

Seit Jahrzehnten wurden eine Vielzahl von on-line Check-Verfahren für Prozessorkomponenten entwickelt. In der Fachliteratur wird für diese Techniken der Terminus *Concurrent checking* benutzt. Die einfachste Variante hierfür ist die Duplizierung des betreffenden Schaltkreises mit anschließendem Vergleich der Signale bzw. Daten. Das Duplikat kann entweder identisch oder divers zum Original implementiert werden. Verschiedene Fehler-Toleranz-Verfahren können ebenso zur sofortigen Fehlererkennung genutzt werden. Der Vorteil von Verdopplung oder Verdreifachung ist die Erkennung von beliebigen Fehlern (arbitrary errors). Jedoch führt z.B. eine Komponenten-Verdreifachung mit anschließendem Mehrheitsentscheid zu einem enormen Mehrbedarf an Chipfläche und Stromverbrauch. Der Faktor "zusätzliche Hardware" ist umgekehrt proportional zur Ausbeute der Chip-Produktion [CP95]. Eine weitere Variante der fehlererkennenden Schaltkreise stellt die "two-rail Logik" dar, bei der die originale und die duplizierte Logik so implementiert sind, dass sie inverse Ausgangs-Signale oder -Daten produzieren [GG93]. Eine Steigerung der Effizienz der zusätzlichen Fehlererkennungslogik kann erreicht werden, wenn der duplizierte Schaltkreis modifiziert wird. Bei der "pseudo-Duplikation" [SHB68] werden Daten durch ein und dieselbe Komponente mehrfach aber über unterschiedliche Datenpfade berechnet. Hier besteht ein Mehrbedarf nur durch die längere Berechnungszeit. Für einen on-line Check ist diese Variante nur geeignet, wenn sie zusammen mit einer Duplizierung angewandt wird [SHB68]. Durch fehlererkennende Codes können weitere Kosten eingespart werden. Ein bekanntes Beispiel ist der Paritäts-Code für Datenworte. Hamming-Codes werden sowohl bei der Überwachung und Fehlerkorrektur von Speichern als auch in Kommunikationsprotokollen (z.B. TCP/IP) verwendet. Der Berger-Code [WG] bietet die Möglichkeit der Erkennung von Einzel- und unidirektionalen Mehrfach-Fehlern in kombinatorischen Schaltkreisen. Bei der praktischen Nutzung von Codes zur Fehlererkennung muss zwischen Daten- und Hardware-Redundanz unterschieden werden. Bei Daten-Redundanz werden zusätzliche Check-Bits zum Datenwort hinzugefügt. Redundante Hardware wird genutzt, um Daten-Encoder bzw. eine Code-Prediktion zu realisieren und Ergebnisse beider zu vergleichen.

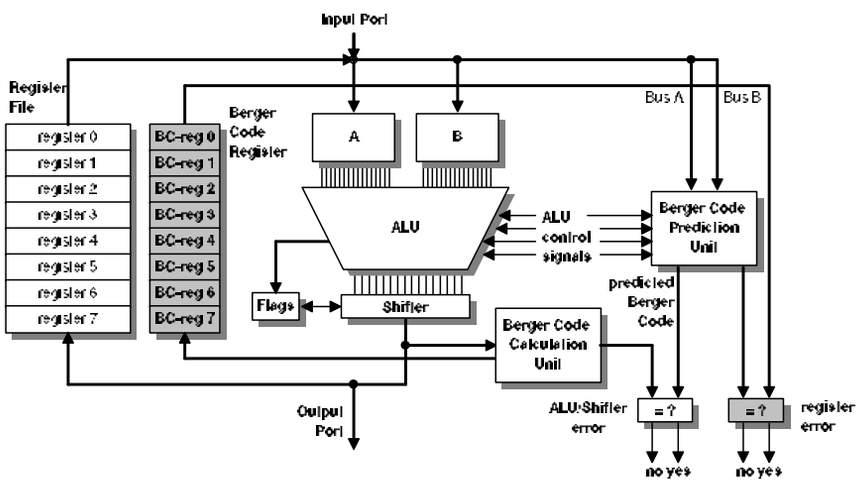


Abbildung 2: Berger-Code-Prediction für Prozessor Datenpfade

### 3.2 Berger-Code-Prediction

In diesem Abschnitt wird die Implementierung der Berger-Code-Prediction (BCP) vorgestellt. (Der Berger-Code (BC) repräsentiert die Anzahl der binären '0'-en bzw. '1'-en in einem Datenwort.) Die BCP wurde in [ZSM99] entwickelt um eine Arithmetisch-Logische Einheit (ALU) on-line überwachen zu können. Das Grundprinzip dieses Verfahrens basiert darauf, dass sich für eine ALU-Operation F = A op B der Berger-Code Fc entsprechend bestimmter Formeln vorhersagen lässt. Im Rahmen dieser Arbeit wurde dieses Verfahren auf einen kompletten Prozessor-Datenpfad angewandt. Ziel war die on-line Überwachung aller Komponenten innerhalb dieses Pfades. Weiterführend sollten auch sequentielle Komponenten in die Überwachung einbezogen werden. Dafür wurde die Ausgangsstruktur dahingehend erweitert, dass für Prozessor-Register (Register-File) zusätzliche Register implementiert wurden, um den BC des jeweiligen Registers zu speichern. Eine BCP-Unit generiert basierend auf den Eingangs-Operanden und der durch die ALU-Steuerleitungen angewählten Funktion den zu erwartenden Berger-Code Fc des ALU-Ergebnisses. Dieser wird mit dem BC des realen Ergebnisses verglichen. Wird ein Datum in das Register-File geschrieben, wird der BC ebenfalls im zusätzlichen BC-Register gespeichert. Beim Zugriff auf ein Register wird dessen erneut berechneter BC mit dem gespeicherten verglichen. Dadurch ist eine Erkennung von latenten Fehlern in Registern möglich. Dieses Verfahren ermöglicht erstmalig die gleichzeitige on-line Überwachung von kombinatorischen und sequentiellen Elemente eines Prozessors durch ein und die selbe Kodierungsmethode. Abbildung 2 zeigt die Struktur der Implementierung.

Die weiteren Untersuchungen zur BCP-Methode sollten sicherstellen, dass sich diese Verfahren auch über die bisher implementierten einfachen Strukturen mit Integer-Verarbeitung hinaus auf komplexere Prozessoren anwenden lässt. Für diesen Nachweis wurden Berger-Code-Prediktionsformeln für vierstufige Fließkomma-Additions/Subtraktions-Komponen-

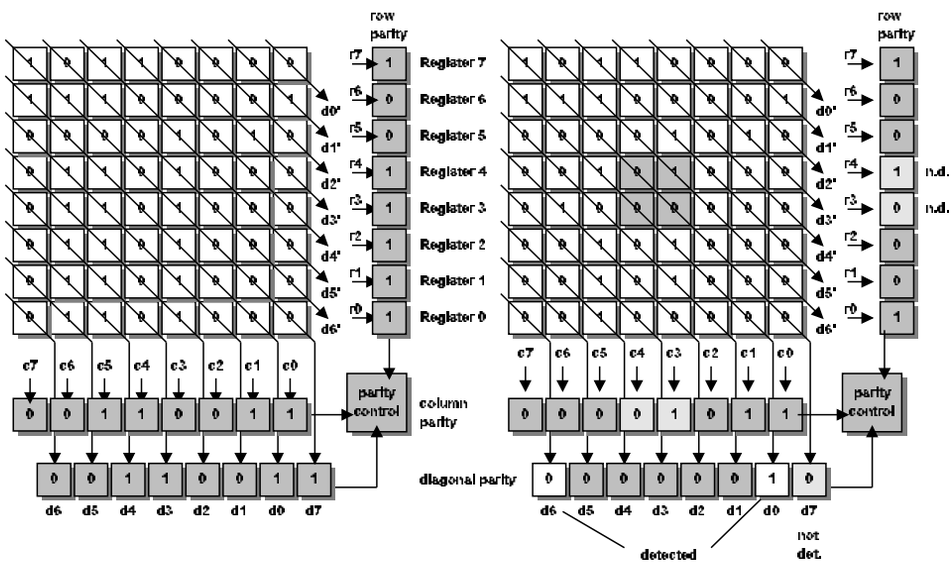


Abbildung 3: Cross-Parity Struktur und Erkennung eines 4-Bit-Fehlers

ten (FPU) mit einfacher (32-Bit) und doppelter Genauigkeit (64-Bit) entwickelt, implementiert und in umfangreichen Fehler-Injektionsexperimenten validiert. Der Overhead der implementierten BCP betrug 38% für eine FPU mit einfacher Genauigkeit (single precision) und 18% für eine mit doppelter Genauigkeit [PWV01].

### 3.3 Cross-Parity-Prediction

Bei Experimenten mit der BCP für Register zeichnete sich bei einigen Prozessor-Design bereits ab, dass der Mehrbedarf an Check-Bit-Registern für größere Register-Files unrentabel werden kann. Um hierfür effizientere on-line Check-Verfahren nutzen zu können, wurden herkömmliche Verfahren zur Überwachung von Registern untersucht. Das am häufigsten benutzte Verfahren basiert auf der Paritätskodierung. Damit diese jedoch der Zielsetzung der Mehrfachfehler-Erkennung gerecht wird, bedarf es einer massiven Modifikation. Bei der Übertragung von Datenblöcken war bereits die gemeinsame Überwachung von Zeilen- und Spalten-Parität bekannt. Ein Fehler konnte hiermit wie in einem Fadenkreuz erkannt werden. Diese Methode versagt jedoch bei Fehler mit *gerader Multiplizität*. Zum Beispiel ein vierfach-Bit-Fehler, der zusammenhängend zwei Zellen in zwei Registern verfälscht, bleibt unerkannt. Um auch diese Fehler detektieren zu können, wurde ein zusätzlicher Diagonal-Paritäts-Vektor eingeführt. Die gemeinsame Überwachung von Zeilen-, Spalten- und Diagonal-Parität wurde *Observation with Cross-Parity-Prediction* genannt [PWV01]. Abbildung 3 zeigt die Organisation der Cross-Parity-Vektoren und die Erkennung eines vierfachen Registerfehlers.

In verschiedenen Implementierungen könnte nachgewiesen werden, dass der Bedarf an zusätzlichen Check-Bit-Registern mit dieser Methode für größere Register-Files drastisch gesenkt werden konnte [P02a]. Darüber hinaus bietet die Cross-Parity-Technik die Möglichkeit einer einfach implementierbaren Fehler-Korrektur-Methode [P02b].

# 4 On-line Check Techniken für Prozessor Controll-Signale

## 4.1 State-of-the-art

Techniken zur on-line Überwachung von Komponenten sind im Kontext einer Fehlererkennung in Prozessoren nutzlos, wenn nicht auch die sie steuernden Signale überwacht werden. Eine Komponente (z.B. Datenpfad) wird als fehlerfrei deklariert, kann jedoch auf Grund falscher Kontroll-Signale fehlerhafte Ergebnisse produzieren. Kontroll-Signal- oder Controller-Fehler wirken sich oft erst mit einer gewissen Latenz aus, d.h. der Fehler-Zeitpunkt ist schwer bestimmbar. Sie führen i.d.R. zu Ausnahme-Behandlungsroutinen (Exceptions). Eine Reihe von "Kontroll-Fluss-Check"-Techniken wurden publiziert. Eine Vielzahl von Verfahren basieren auf der *Signatur-Analyse* unter Nutzung von rückgekoppelten Schieberegister-Strukturen [SS87], [Mi98]. Diese Analyseart ist für die angestrebte Fehler-Erkennung im selben Takt-Zyklus ungeeignet, da zur Prozessierung der Signaturen eine entsprechende Latenz notwendig ist. Eine einfache Methode bei der Controller-Überwachung stellt auch hier eine Duplizierung oder Verdreifachung dar. Zur Fehler-Erkennung im Prozessor-Kontroll-Fluss bietet die Zustandskodierung. Für komplexere Prozessoren ist die Realisierung der Überwachung von Zustand, Transition und Folgezustand nicht mit vernünftigen Kosten realisierbar. Durch den Check eines Folgezustandes muss auch hier eine Latenz für die Fehlererkennung in Kauf genommen werden.

## 4.2 Kontroll-Signal on-line Check mit pseudo-TMR Controller

Eine effizientere Methode als eine einfache Verdreifachung (triple-modular redundancy = TMR) wurde durch eine Form der pseudo-Redundanz entwickelt [PV98]. Bei eingebetteten Prozessor-basierten Systemen kommen oft Standard-CISC-Prozessoren zum Einsatz. Da das System jedoch i.d.R. nur einen geringen Teil des Befehlsatzes für seine Applikationen benötigt, wird auch nur ein Sub-Set aller möglichen Prozessor-Zustände erreicht. Deswegen ist es auch nur notwendig, diesen Sub-Set zu überwachen. Mit Hilfe entsprechender Software-Werkzeuge konnte der Code einer Applikation dahingehend untersucht werden, welche Kontroll-Signale benötigt werden. Für diese wurde dann eine "Applikationsspezifisch reduzierte" Kontroll-Logik generiert. Für exemplarische Prozessor-Designs wurden zum originalen Kontroller zwei reduzierte hinzugefügt. Dieser sog. "pseudo-TMR"-Controller [PV98], [PPV99] ist nun in der Lage, beliebige Fehler zu erkennen. Mit entsprechender Mehrheitsentscheidungslogik können für die in diesem System laufenden Applikationen Kontroll-Signale fehler-tolerant erzeugt werden. Der Mehraufwand für die zu-

sätzliche Hardware erreichte bei den Experimenten ca. den gleichen Umfang wie eine einfache Verdopplung des Controllers.

### 4.3 Kontroll-Signal on-line Check mit Zustandscode-Vorhersage

Bei der Abarbeitung eines Befehls (Makro-Befehle) in einem Prozessor wird je nach Architektur eine Sequenz von Mikro-Befehlen ausgeführt. Diese Mikro-Befehle/Operationen sind Kombinationen von Kontroll-Signalen. Sind zum Zeitpunkt t die Kontroll-Signale y aktiv, repräsentieren sie den momentanen *legalen* Zustand des Prozessors. Der Ansatz für die Methode des on-line Check des Prozessor-Zustandes basiert zum Einen auf der Kodierung gleichzeitig aktiver Kontroll-Signale bzw. Mikro-Operationen (Kodierung des legalen Zustandes) und der Vorhersage (Prediction) dieses Codes. Eine Zustandsprediction kann realisiert werden, wenn die entsprechenden Eingangssignale der Kontroll-Logik (Befehls-, Zeit-Variable, Flag-Signale) so kodiert werden, dass derselbe Code generiert wird, als wenn die Kontroll-Signale die Ausgangsbasis bilden würden. Diese rein kombinatorische Kodierungsmethode mit entsprechendem Vergleich gewährleistet die sofortige Erkennung illegaler Zustände. Der Mehraufwand an Hardware beträgt je nach Befehlssatz bei experimentellen Prozessor-Designs ca. 2500 Gatter-Equivalente. Die Zustandscode-Vorhersage wurde ebenfalls hinsichtlich ihrer Eignung für komplexere Designs untersucht. Sie wurde für einen 64-Bit Prozessor auf DLX-Basis implementiert. Auf Grund der Pipeline-Verabeitung wurde eine spezielle Partitionierung der Zustandsencoder entsprechend der einzelnen Pipeline-Stufen vorgenommen. Der Gesamtaufwand für die Zustandscode-Check-Komponenten betrug 5,8%.

## 5 Schnelle Fehler-Kompensation in Prozessoren

### 5.1 State-of-the-art

Bei Fehlern in Prozessoren werden unterschiedliche Methoden angewendet, um diese zu kompensieren. In einfachen Systemen wird nach einem Reset die Programm-Abarbeitung wiederholt. Möglich sind auch die Ausführung von *Interrupt-* oder *Exception-Handling-*Routinen. Durch Mehrfach-Prozessorstrukturen ist eine Kompensation von Fehlern durch Umkonfigurierung leicht möglich. Re-konfigurierbare Architekturen erlauben die Ersetzung defekter Hardware durch "cold/hot-standby" Komponenten. Eine übliche Technik insbesondere zur Kompensation transienter Fehler (soft errors) ist das Rollback [KT87]. Hierbei werden in regelmäßigen Abständen Checkpoints gesetzt. Das heißt, es werden alle Inhalte speichernder Elemente gesichert. Die Dauer zwischen einzelnen Checkpoints wird Rollback-Distanz genannt. Wird innerhalb dieser Distanz ein Fehler erkannt, wird zum letzten Checkpoint zurückgekehrt, alle Register mit den gesicherten Werten geladen und der gesamte Zyklus wiederholt. Übliche Rollback-Distanzen von mehreren 100 oder 1000 Prozessor-Takten sind insbesondere für sicherheits-kritische Systeme ungeeig-

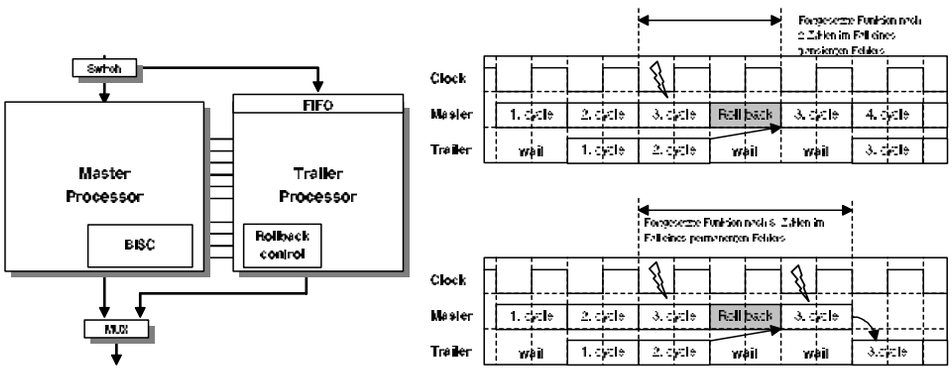


Abbildung 4: Micro-Rollback Struktur und zeitliches Verhalten

net. In [TT90] wird ein sogenanntes *Micro-Rollback* vorgestellt, bei dem Fehler in einem Register-File in kürzester Zeit kompensiert werden können. Realisiert wurde das durch ein FIFO-Register, in das die Daten parallel zum Register-File geschrieben werden. So können fehlerhafte Daten über einen bestimmten Zeitabschnitt mit wenigen Zyklen Verzögerung wiederhergestellt werden. Nachteilig ist, dass permanente Fehler nicht kompensiert werden können.

### 5.2 Micro-Rollback mit Master-Trailer-Struktur

Durch die in den vorangegangenen Abschnitten erläuterte on-line Fehler-Erkennung im selbem Prozessor-Takt ist eine schnelle Micro-Rollback-Variante möglich. Grundlage bildet eine doppelte Prozessor-Struktur - bestehend aus einem Master- und einem Trailer-Prozessor. Beide Prozessoren sind mit den entsprechenden Fehler-Erkennungs-Komponenten ausgestattet. Der Trailer verfügt darüber hinaus über einen Rollback-Controller. Master und Trailer prozessieren den selben Maschinen-Code - der Trailer allerdings mit einer Verzögerung von einem Zyklus.

Wird in einem Takt-Zyklus ein Fehler entdeckt, werden Master und Trailer gestoppt. Im folgenden Zyklus (Rollback-Zyklus) werden sämtliche Register-Inhalte des Trailers in den Master geladen. Anschließend wird der fehlerhafte Takt wiederholt. Tritt kein erneuter Fehler auf, handelte es sich bei dem Fehler um einen transienten Effekt. Wird im Wiederholungstakt ein erneuter Fehler registriert, muss von einem permanenten Effekt ausgegangen werden. In diesem Fall ist der Trailer so ausgelegt, das er die komplette Funktion des Masters übernehmen kann. So kann zum Beispiel eine Notlaufeigenschaft realisiert werden. Die maximale Verzögerung im Falle eines transienten Effekts beträgt zwei Zyklen. Der erste permanente Fehler kann nach drei Zyklen kompensiert werden (siehe Abb. 4). Die Fehler-Klassifizierung wird durch einen Vergleich von Fehlervektoren durchgeführt, die durch die on-line Check-Komponenten vor und nach dem Rollback generiert werden [PPV99]. Die Implementierung des Micro-Rollback-Verfahrens wurde mit Hilfe von Pro-

zessoren vorgenommen, die über eine sequentielle Befehlsabarbeitung verfügen. Weiterführend wurde untersucht, ob dieses Verfahren auch für Pipeline-Strukturen geeignet ist bzw. unkompliziert angepasst werden konnte. Durch Ausnutzung und durch geringfügige Modifikation der internen Struktur konnte für eine fünfstufige Prozessor-Pipeline mit Forwarding und Hazardkontrolle eine Fehlerkompensationsmethode durch Rollback realisiert werden [Ga00]. Im Unterschied zu der vorherigen Methode wurde hierbei eine Prioritäten-Steuerung entwickelt, die ausgehend von einem (on-line) erkannten Komponenten- oder Kontroll-Signal-Fehler über die Rollback-Distanz entscheidet.

## 6 Zusammenfassung

In dieser Arbeit wurden Methoden zur on-line Erkennung von transienten und permanenten Einzel- und Mehfachfehler in Prozessoren vorgestellt. Die implementierten Techniken ermöglichten eine Erkennung im selben Maschinenzyklus. Dadurch kann einer Fehler-Ausbreitung vorgebeugt werden. Ein Vorteil dieser on-line Verfahren ist die Erkennung vorab nicht-testbarer transienter Effekte während des Betriebes. In Kombination mit traditionellen Testverfahren ist so eine verbesserte Prävention gegenüber Systemausfällen gegeben. Durch eine Unterscheidung zwischen Fehler-Charakteristik und -Klasse kann eine Auswahl geeigneter Maßnahmen zur Wiederherstellung getroffen werden. Durch das vorgestellte Micro-Rollback-Verfahren können transiente Effekte bzw. der erste permanente Fehler kompensiert werden. Dadurch ist eine schnelle Wiederherstellung der Systemfunktion bzw. eine Notlaufeigenschaft gewährleistet. Ausgehend von einfachen Prozessor-Architekturen wurde im Rahmen dieser Arbeit die Erweiterbarkeit für komplexere Strukturen untersucht. Anhand von experimentellen Prozessor-Designs (z.B. mit Pipeline-Verarbeitung, Fließkomma-Einheiten) konnte nachgewiesen werden, dass die präsentierten Techniken auch für diese Strukturen hervorragend geeignet sind.

Durch die entwickelten Verfahren ist es möglich, die Verfügbarkeit von eingebetteten Prozessoren unterschiedlicher Architektur und Komplexität zu erhöhen. Dadurch wird die Zuverlässigkeit dieser Prozessoren und damit des Gesamtsystems verbessert.

## Literaturverzeichnis

[CP95] J.A. Clark and D.K. Pradhan. Fault Injection - A Method for Validating Computer-System Dependability. *IEEE Computer*, June 1995:47–56, 1995.

[Gal00] C. Galke. Rollback-Strategien für Prozessoren mit Pipeline-Struktur. Diplom-Arbeit am Lehrstuhl Technische Informatik, BTU Cottbus, Oktober 2000.

[GG93] M. Gössel and S. Graf. *Error Detection Circuits*. McGraw-Hill Book Company, London, 1993.

[KT87] R. Koo and S. Toueg. Checkpointing and Rollback-Recovery for Distributed Systems. *IEEE Transactions on Software Engineering*, SE-13, January 1987:23–31, 1987.

[MORK98] G. Miremadi, J. Ohlson, M. Riemén, and J. Karlsson. Use of Time and Address Signatures for Control Flow Checking. *Dependable Computing for Critical Applications, IEEE Computer Society*, No. 5:201–221, 1998.

[Pfl02] Matthias Pflanz. *On-line Error Detection and Fast Recover Techniques for Dependable Embedded Processors*. LNCS, Springer, Heidelberg, nr. 2270 edition, 2002.

[PPV99] M. Pflanz, F. Pompsch, and H.T. Vierhaus. An Efficient On-Line-Test and Back-up Scheme for Embedded Processors. In *Proceedings Int. Test Conference, ITC'99*, pages 964–972, 1999. Atlantic City, MA.

[PV98] M. Pflanz and H.T. Vierhaus. Generating Reliable Embedded Processors. *IEEE MICRO*, Sept./Oct.:33–41, 1998.

[PWGV02] M. Pflanz, K. Walther, C. Galke, and H.T. Vierhaus. On-line Error Detection and Correction in Storage Elements with Cross-Parity Check. In *Proceedings IEEE Int. On-line Test Workshop (IOLTW'02)*, Juli 2002. Frankreich.

[PWV01] M. Pflanz, K. Walther, and H.T. Vierhaus. On-line Error Detection Techniques for Dependable Embedded Processors with High Complexity. In *IEEE Int. On-line Test Workshop (IOLTW'01)*, Juli 2001. Italy.

[SHB68] F.F. Sellers, M.J. Hsiao, and L.W. Bernson. *Error Detecting Logic for Digital Computers*. McGraw-Hill Book Company, New York, 1968.

[SS87] M.A. Schuette and J.P. Shen. Processor Control Flow Monitoring Using Signatured Instruction Streams. *IEEE Transactions on Computers*, C-36, No. 3:264–275, March 1987.

[TT90] Y. Tamir and M. Tremblay. High-Performance Fault-Tolerant VLSI Systems Using Micro Rollback. *IEEE Trans. on Computers*, 39, No. 4:548–554, April 1990.

[WG] N.T. Wing and E. Glen. Self Checking Arithmetic Unit. US Patent 4314350, GO6F 11/14.

[ZSM99] C. Zeng, N. Saxena, and E.J. McCluskey. Finite State Machine Synthesis with concurrent Error Detection. In *Proceedings Int. Test Conference, ITC'99*, pages 672–679, 1999. Atlantic City, NJ.

**Matthias Pflanz** studierte von 1991 bis 1996 Elektrotechnik an der Brandenburgischen Technischen Universität (BTU) in Cottbus. Von 1996 bis 2001 war er wissenschaftlicher Mitarbeiter am Lehrstuhl Technische Informatik der BTU. Zu seiner Aufgaben gehörten Lehrtätigkeiten auf den Gebieten Prozessorarchitektur, Test und Entwurf integrierter Schaltungen sowie die Forschung auf den Gebieten Fehler-tolerante Schaltungen und zuverlässige Prozessor-basierte Systeme. Forschungsergebnisse wurden in mehr als 20 internationalen Publikationen veröffentlicht. Im September 2001 promovierte er an der BTU Cottbus. Seit November 2001 ist Matthias Pflanz bei der IBM Deutschland Entwicklung GmbH in Böblingen als Entwicklungsingenieur beschäftigt. Zu seinen Aufgaben zählen u.a. Logik-Design und Test-Modellierung für High-Performance- Prozessoren. Ab 2003 wird er an der Technischen Akademie Esslingen Fachvorträge z.B. zum Thema „Zuverlässige Schaltungen und Systeme“ halten.