Die Gefahren des Over-Engineerings von sicheren Systemen

Funktionale Sicherheit komplexer Software-Systeme – Teil III

Autoren: Chris Hobbs, Senior Developer, Safe Systems
Malte Mundt, Field Application Engineer

Inhalt

Ein konkretes Sicherheitsproblem mit überhastetem Aktionismus beheben zu wollen, ohne dabei sorgfältig die Verlässlichkeit des Gesamtsystems zu untersuchen, kann viel Arbeit mit wenig Nutzen bedeuten und sogar ungewollt neue, erhebliche Probleme verursachen.

Wir betrachten das hypothetische Beispiel eines Controllers für vollautomatischen Zugbetrieb (Automated Train Operations – ATO), der im Führerstand von S-Bahn-Zügen zum Einsatz kommt, welche in Höhenlagen fahren sollen, wo der höhere Neutronenfluss die Gefahr von transienten DRAM-Bitfehlern (Soft Errors) deutlich erhöht. Wir untersuchen, wie sich die Erweiterung eines 2oo2-Systems um softwarebasierte Fehlererkennung auf die Verlässlichkeit des Systems auswirkt, betrachten die Vor- und Nachteile dieses Ansatzes, und schlagen alternative Vorgehensweisen vor, die die Verlässlichkeit wirksam verbessern können.

Die Elefanten übersehen?

In diesem Beitrag ignorieren wir absichtlich zwei „dicke“ Probleme: Sicherheit und Bedienfehler. Der Einfachheit halber gehen wir davon aus, dass das hier betrachtete System absolut sicher ist und nie falsch bedient wird.

In der Praxis müssen wir diese beiden Punkte selbstverständlich berücksichtigen, denn kein System ist völlig sicher und früher oder später wird es auch Bedienfehler geben – vor allem solche, mit denen wir bei der Entwicklung nicht gerechnet haben.

Zwei tragische Korrekturen

Gut gemeinte, aber schlecht durchdachte „Lösungen“ können oftmals genau die Tragödien hervorrufen, die sie eigentlich verhindern sollten. Zwei Beispiele, eines aus der Schifffahrt und eines aus der Luftfahrt, veranschaulichen dies deutlich: Die erste Tragödie ereignete sich 1915 auf dem Chicago River. Die *SS Eastland* bekam Schlagseite und kenterte, 840 Menschen kamen ums Leben. George W. Hilton, Autor von *Eastland: Legacy of the Titanic* schreibt dazu, dass

> ... die Topplastigkeit der Eastland größtenteils auf die Anzahl und das Gewicht der vorgeschriebenen Rettungsboote zurückging. Nach dem Untergang der *Titanic* im Jahre 1912 führte eine allgemeine Panik zu der oftmals irrationalen Forderung nach mehr Rettungsbooten auf Passagierschiffen. Der Gesetzgeber, der mit den Details des Schiffbaus nicht vertraut war, war sich nicht darüber im Klaren, dass Rettungsboote selten alle und manchmal gar keine Leben retten können.[1]

Die neuen Sicherheitsvorschriften des Seaman's Act von 1915 bewirkten, dass auch auf der Eastland – einem Schiff, das sowieso schon Probleme mit Topplastigkeit, d. h. einem

zu weit oben im Schiff befindlichen Schwerpunkt hatte – Rettungsboote nachgerüstet wurden. Diese Rettungsboote machten die *Eastland* somit unsicherer statt sicherer. Ob sie überhaupt eingesetzt werden können hängt nämlich stark davon ab, auf welche Weise und wie schnell ein Schiff sinkt.

Die zweite Tragödie ereignete sich 1979 in der Antarktis. Der Air-New-Zealand-Flug 901 flog direkt in einen Berg hinein. Alle 257 Passagiere und Besatzungsmitglieder kamen ums Leben. Im Bericht der Untersuchungskommission steht, dass der Flugplan im Computersystem für diesen Flug „seit 14 Monaten fehlerhaft“ war (er wich von der genehmigten Route ab). „Der Fehler wurde erst am Tag vor dem Unglücksflug korrigiert“ und: „Der Crew war beim Briefing noch ein Ausdruck des fehlerhaften Flugplans mit den nicht genehmigten Koordinaten ausgehändigt worden, aber der Flugplan, der am Tag des Fluges in den Bordcomputer eingegeben wurde, war korrekt.“[2] Die Kommission identifizierte zwei „schuldhafte Handlungen oder Unterlassungen“: Erstens, dass die Piloten keine topografischen Karten ihrer Flugroute erhalten hatten, und zweitens, dass die Piloten nicht über die Korrekturen an der Flugroute informiert worden waren. Da sie nichts von der Korrektur der Flugroute im Computer wussten, glaubten die Piloten, sie flögen über den McMurdo-Sund. In Wirklichkeit aber flogen sie unbemerkt auf den wolkenverhüllten Mount Erebus zu und hatten keine Chance mehr, die Flughöhe zu korrigieren. Der Leiter der Untersuchungskommission, P. T. Mahon, findet klare Worte:

> Ich bin daher der Auffassung, dass der einzige und ursächliche Grund für das Unglück in dem Versagen derjenigen Mitarbeiter der Fluggesellschaft liegt, die die Flugroute über den Mt. Erebus einprogrammiert haben, ohne die Besatzung darüber zu informieren.[3]

Genau wie bei dem Untergang der *SS Eastland* erzeugte auch hier eine Maßnahme, mit der eine Gefahr (in diesem Fall eine nicht genehmigte Flugroute) abgewendet werden sollte, unbeabsichtigt eine zusätzliche Belastung für das Gesamtsystem (korrekte Navigation des Flugzeugs) und löste ungewollt eine Tragödie aus.

Wie Aussagen über die Sicherheit zu verstehen sind

Beim Entwurf eines sicheren Softwaresystems müssen ganz zu Anfang die Sicherheitsanforderungen ermittelt werden. Im Einzelnen sind dies:

- der erforderliche Verlässlichkeitsgrad bzw. die akzeptable Ausfallrate des Systems und
- die Grenzen der Aussagen zur Sicherheit, das heißt, die Bedingungen und Rahmenparameter, innerhalb derer die Aussagen zur Verlässlichkeit getroffen werden und gültig sind.

Wenn wir zeigen können, dass unser System innerhalb dieser Grenzen das erforderliche Verlässlichkeitsniveau erreicht, dürfen wir sagen, dass das System hinreichend sicher ist.

Zuverlässigkeit versus Verfügbarkeit

Verlässlichkeit ist bei einem Softwaresystem eine Kombination aus *Verfügbarkeit* (wie oft das System rechtzeitig auf Anfragen reagiert) und *Zuverlässigkeit* (wie oft die Reaktionen korrekt sind). Ein verlässliches Softwaresystem ist also ein System, das dann, wenn es benötigt wird, innerhalb der erforderlichen Zeitspanne die richtige Reaktion liefert.
Um sagen zu können, ob ein System hinreichend verlässlich ist, müssen wir sowohl die erforderliche Zuverlässigkeit als auch die erforderliche Verfügbarkeit kennen. Das relative Gewicht von Verfügbarkeit und Zuverlässigkeit variiert von System zu System – je nachdem, wofür das System vorgesehen ist.

Das Fahrrad als Anschauungsbeispiel

Am Beispiel eines Fahrrades lässt sich wunderbar das Thema Verlässlichkeit erklären, denn zum sicheren Radfahren gehört beides: Zuverlässigkeit und Verfügbarkeit. Der Radfahrer muss die richtigen Entscheidungen treffen, und zwar bezüglich Lenkung, Geschwindigkeit und Gleichgewicht – sonst landet er schnell im Graben. Und wenn er nicht zufällig ein Zirkusartist ist, der ein stehendes Fahrrad im Gleichgewicht halten kann, muss er das Rad in Bewegung halten, um nicht umzukippen.

Stellen wir uns nun ein fahrerloses Fahrrad vor, das von einem Software-Controller gesteuert wird. Der Controller muss die richtigen Entscheidungen treffen (d. h., er muss zuverlässig sein), um das Fahrrad unfallfrei an seinen Bestimmungsort zu befördern, und er muss dies fortwährend tun (er muss also verfügbar sein). Wechselt der Controller aufgrund eines Problems in seinen Design Stafe State, bedeutet dies normalerweise, dass er seinen Betrieb einstellt. Bleibt er zu lange in diesem Zustand, wäre das genauso fatal wie eine falsch getroffene Lenkentscheidung. Das führerlose Fahrrad muss fortwährend in Bewegung bleiben, um nicht umzukippen, also gibt es keinen möglichen Design Safe State für den Controller, der das Gesamtsystem (das Fahrrad) nicht gefährden würde.

Ein einfaches sicheres System

Im Folgenden betrachten wir einen sehr einfachen hypothetischen Controller für ein (ebenso hypothetisches) System für vollautomatischen Zugbetrieb (Automated Train Operations – ATO), das im Führerstand von führerlosen S-Bahn-Zügen zum Einsatz kommt (Abbildung 1). Aus Gründen der Einfachheit gehen wir davon aus, dass der Zug eine Ringstrecke befährt, und lassen das System nur vier Werte überprüfen:

- Zustand des Zugs (Fahren oder Stehen),
- Standzeit an der Haltestelle (über 90 Sekunden oder bis zu 90 Sekunden),
- Zustand der Türen (offen oder geschlossen),
- Zug in Haltestelle? (eingefahren oder nicht eingefahren).

Bei der Initialisierung steht der Zug. Er öffnet die Türen, wartet 90 Sekunden ab, schließt die Türen und fährt zur nächsten Haltestelle. Wenn der Zug in eine Haltestelle einfährt, hält er an und öffnet seine Türen, wartet usw. und arbeitet bis zum Tagesende weiter diese Endlosschleife ab. In seinem Design Safe State steht der Zug.

Der Controller

In unserer Begrifflichkeit ist der Controller ein 2oo2-System („two out of two“ – „zwei aus zweien“), denn er verfügt über zwei Verarbeitungs-Subsysteme, die darin übereinstimmen müssen, dass es sicher ist, den Controller von einem Wechsel in den Design Safe State abzuhalten. (Siehe „Das 2oo2-System“ auf Seite 6 und Abbildung 2 auf Seite 8.) Entdeckt eines der beiden Subsysteme, dass der Controller nicht sicher betrieben werden kann, wird er gestoppt. Ein 2oo2-Design verkompliziert das System, doch wir gehen hierbei davon aus, dass die Folgen eines

2oo2 oder 1oo2?

In der Begrifflichkeit der IEC 61508, Teil 6, ist unser System ein 1oo2-System („one out of two“ – „eines aus zweien“), da ein Subsystem allein das System in den Design Safe State versetzen kann.

Die unterschiedliche Bezeichnung rührt daher, dass die IEC 61508 Definitionen verwendet, die sich aus dem Hardwaredesign ableiten und die Entscheidungsarchitektur anhand der Anzahl von Stimmen beschreibt, die notwendig sind, um das System *in seinen Design Safe State zu versetzen*. Wir dagegen benutzen eine Definition, die sich im Softwaredesign durchgesetzt hat und die Entscheidungsarchitektur anhand der Anzahl von Stimmen beschreibt, die notwendig sind, um das System *von einem Wechsel in den Design Safe State abzuhalten*.

fehlerhaften Verhaltens des Controllers hinreichend unheilvoll sind, um die Zeit und den Aufwand für die Verdoppelung als gerechtfertigt erscheinen zu lassen.

Bei dieser Betrachtung sind die genauen Kriterien, anhand derer das System bestimmt, ob es den Controller in einen Design Safe State versetzen muss, weniger bedeutend als die Tatsache, dass der Controller das, was er tut, fehlerfrei tun muss. Werden die Türen bei fahrendem Zug geöffnet oder fährt der Zug bei geöffneten Türen ab, während Passagiere ein- und aussteigen, können Menschen verletzt oder getötet werden. Wir gehen daher davon aus, dass das System gemäß IEC 61508 Sicherheitsintegritätslevel 3 (SIL 3) zertifiziert sein muss, was bedeutet, dass die „Wahrscheinlichkeit eines gefährlichen Ausfalls“ niedriger als 1 zu 10^7 je fortwährende Betriebsstunde ist. Gehen wir davon aus, dass das ursprüngliche Controller-Design diese Anforderung erfüllt. Jetzt wird jedoch ein neues Risiko identifiziert, das bei Formulierung der ursprünglichen Anforderungen nicht bekannt war.

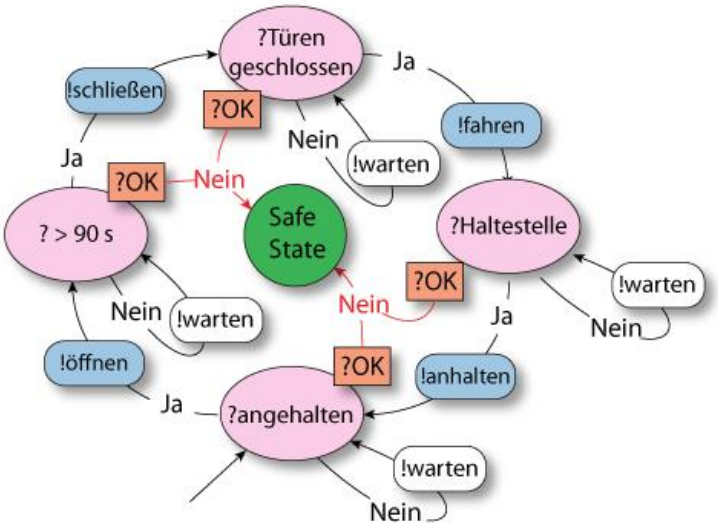

Abbildung 1. *Ein stark vereinfachtes Diagramm eines Controllers für vollautomatischen Zugbetrieb, der in einen sicheren Zustand wechselt, wenn er Gefahren oder Anomalitäten erkennt. Das System bezieht von Taktgebern und Sensoren Daten über den Standort des Zuges, seinen Bewegungszustand, den Zustand aller Zugtüren und die Zeit, seit der er an einer Haltestelle steht. Es sendet Anweisungen zum Starten und Anhalten des Zugs und zum Öffnen und Schließen der Türen. Bevor es eine Anweisung sendet, fragt es sein 2oo2-System, ob die Fortsetzung des Betriebs sicher ist.*

Eine neue Verwundbarkeit

Die Auswirkungen von Strahlung auf den Arbeitsspeicher von Computern sind zwar schon lange bekannt, aber Speicherfehler aufgrund von Höhenstrahlung wurden bei den ursprünglichen Spezifikationen nicht berücksichtigt.

Ein anderer Kontext

Unser hypothetischer Controller hat sich bereits in Rom und diversen anderen Städten bewährt. Ein neuer Kunde möchte ihn nun in einem S-Bahn-System mit vollautomatischem Zugbetrieb im Großraum La Paz-El Alto in Bolivien verwenden. La Paz-El Alto hat fast 2,5 Millionen Einwohner, die in einer Höhe von mehr als 4100 m über dem

Meeresspiegel leben (das ist höher als der Mount Erebus). Dies ist eine bedeutende Kontextänderung, weil die Gefahr, dass Höhenstrahlung transiente oder permanente Speicherfehler verursacht, mit der Höhe ansteigt. Der Kunde erbittet nun einen Nachweis, dass unser System auch dann noch die Sicherheitsanforderungen erfüllt, wenn ein erhöhtes strahlungsbedingtes Risiko transienter Speicherfehler in die Verlässlichkeitseinschätzung aufgenommen wird. Tatsächlich müssten wir aufgrund der Höhenlage auch andere Voraussetzungen und Annahmen überprüfen, etwa bezüglich der Kühlung, aber der Einfachheit halber wollen wir darauf hier verzichten.

Zunächst gratulieren wir unserem Kunden dazu, dass er erkannt hat, dass unabhängig von dem bisherigen Erfolg unseres Systems und unabhängig von der Zahl der ausfallfrei absolvierten Betriebsstunden in Rom und anderswo eine Änderung des Kontextes, in dem das System betrieben wird, einige oder gar alle der Voraussetzungen zunichtegemacht haben könnte, auf denen unsere Aussagen zur Verlässlichkeit basierten. Im Speziellen ist Höhenstrahlung – hauptsächlich die Neutronen – schon seit Langem als wesentliche Ursache für transiente Speicherfehler (Soft Errors) bekannt.[4] Je mehr Neutronen in einer gegebenen Zeitspanne eine gegebene Fläche passieren (der so genannte relative Neutronenfluss), desto höher ist das Risiko, dass diese Neutronen einen Speicherfehler verursachen. Der Neutronenfluss schwankt abhängig vom geografischen Standort und ganz besonders von der Höhe über dem Meeresspiegel. Der Neutronenflussrechner „Seutest“[5] liefert für La Paz-El Alto einen 11-mal höheren relativen Neutronenfluss als in Rom.[6]

Fehler und False Positives

Unser Problem hat zwei Seiten: Speicherfehler können einerseits Ausfälle des Controllers verursachen, und andererseits können sie so genannte „False Positives“ hervorrufen, aufgrund derer der Controller grundlos in seinen Design Safe State versetzt wird. Es ist sogar denkbar, dass die Strahlung einen transienten Speicherfehler hervorruft, der dasselbe Bit in *beiden* Subsystemen betrifft, so dass es unbemerkt zu einer Fehlfunktion des Controllers kommt, bei der aber der Zug weiterläuft, obwohl der Controller eigentlich in seinen Design Safe State versetzt werden müsste. Wesentlich wahrscheinlicher ist aber, dass ein transienter Speicherfehler in lediglich einem der Subsysteme einen False Positive des 2oo2-Systems auslöst, dadurch unnötige Abschaltungen des Controllers verursacht und er somit zu Unterbrechungen des vollautomatischen Zugbetriebs führt.

False Positives in unserem Controller kompromittieren die Sicherheit zwar nicht unbedingt *direkt*. Das 2oo2-Design, bei dem alles in einen sicheren Zustand wechselt, falls es zwischen zwei Subsystemen zu irgendeiner Art von Nichtübereinstimmung kommt, führt zu einem sehr zuverlässigen System. Aber: Verlässlichkeit (und damit Sicherheit) hängt auch von der Verfügbarkeit des Systems ab. Obwohl bei dem betrachteten System Verfügbarkeit weniger wichtig als Zuverlässigkeit erscheint (ein angehaltener Zug ist in der Regel weniger gefährlich als ein fahrender Zug), ist es doch so, dass Kompromisse bei der Verfügbarkeit des Controllers unterm Strich die Sicherheit beeinträchtigen:

- Ob wegen eines False Positive oder aus einem anderen Grund: Ein auf freier Strecke stehender Zug ist eine Störung im Betriebsablauf. Zeitliche und räumliche Distanz zu folgenden Zügen verringern sich. Damit wird die Zuverlässigkeit von Kommunikationssystemen umso wichtiger, denn entsprechend der Standorte der anderen Züge müssen die Sicherheitsabstände geregelt werden können. Die Belastung für das Gesamtsystem nimmt also zu.

- Funktioniert ein System nicht wie gefordert – ist es nicht verfügbar, obwohl es benötigt wird – finden die Menschen in der Regel Mittel und Wege, es wieder zum Laufen zu bringen. Dabei werden oftmals Sicherheitsmaßnahmen umgangen oder außer Kraft gesetzt. Meldet zum Beispiel ein Sensor immer wieder, dass eine von zwölf Zugtüren offen ist, obwohl sie geschlossen ist, finden die Bediener ggf. einen Weg, den Sensor zu überbrücken, um den Zug in Bewegung zu halten. Dabei treffen sie implizit die gefährliche Annahme, dass, wenn elf Türen geschlossen sind, wohl auch die zwölfte Tür geschlossen sein muss.

Angesichts der höheren Gefahr des Auftretens transienter Speicherfehler und ihrer möglichen Konsequenzen in dem neuen Kontext, in dem unser System zum Einsatz kommen soll, stimmen wir dem Kunden zu, dass wir uns näher mit den Auswirkungen von transienten Speicherfehlern auf die Verlässlichkeit unseres Systems beschäftigen sollten.

Softwarebasierte Fehlererkennung

Da Speicherfehler das Problem sind, erscheint es naheliegend, eine Funktion zur Erkennung von Speicherfehlern zu implementieren. Bevor wir dies tun, sollten wir uns sicher sein, dass diese Lösung:

a) wirksam ist und

b) die Sicherheit nicht beeinträchtigt.

Es folgt eine Beschreibung des Controllers, unserer Annahmen über sein 2oo2-Subsystem und dessen Umgang mit Speicherfehlern sowie unsere Berechnungen der Häufigkeiten gefährlicher Ausfälle mit und ohne softwarebasierter Fehlererkennung. Anhand der Ergebnisse dieser Berechnungen werden wir sehen, ob softwarebasierte Fehlererkennung eine akzeptable Verbesserung der Verlässlichkeit unseres Systems bewirkt, oder ob wir zu anderen Mitteln greifen sollten, um die Verlässlichkeit unseres Controllers in seinem neuen Kontext sicherzustellen.

Das 2oo2-System

Das 2oo2-Design, das unserem ATO-Controller das Verlassen seines Design Safe State ermöglicht, so dass er seine Aufgaben zum Betrieb der S-Bahn wahrnehmen kann, funktioniert wie folgt:

1. Zwei unabhängige Verarbeitungs-Subsysteme empfangen von der Umgebung dieselben Stimuli (Ereignisse).

2. Jedes der Subsysteme berechnet unabhängig von dem anderen anhand der aus der Umgebung empfangenen Ereignisse, ob der Controller in seinen Design Safe State wechseln muss.

3. Beide Subsysteme führen ihre Ergebnisse („Ja“ oder „Nein“) einem Gatter-Subsystem zu (in Abbildung 2 unten als UND-Gatter gezeigt).

4. Das Gatter-Subsystem vergleicht die Ausgaben der beiden Verarbeitungs-Subsysteme miteinander.

5. Wenn beide Subsysteme übereinstimmend melden, dass der Controller am Wechsel in den Design Safe State gehindert werden darf, darf der Controller seinen Betrieb fortsetzen.

6. Unter allen anderen Umständen zwingt das Gatter-Subsystem den Controller, in seinen Design Safe State zurückzufallen.

Annahmen über das Gatter-Subsystem

Wir wollen annehmen, dass das Gatter-System nach IEC 61508 SIL 4 zertifiziert ist. Für ein System mit geringer Nutzung („low demand mode“) bedeutet dieser Sicherheitsintegritätslevel, dass das Subsystem den Unterschied zwischen den beiden Ausgaben 9999 von 10000 Malen korrekt erkennt. Bei einem System mit häufiger Nutzung („high demand mode“) oder einem System im Dauerbetrieb („continuous mode“), das nach SIL 4 zertifiziert ist, muss die Wahrscheinlichkeit, dass ein Unterschied zwischen den Ausgaben der beiden Subsysteme nicht erkannt wird, niedriger als 10^{-8} je fortwährende Betriebsstunde sein.

Unter der Annahme, dass die Rolle des Gatter-Subsystems darin besteht, einen Wechsel des ATO-Controllers in den Design Safe State aktiv zu verhindern, müsste es eigentlich die strengeren Kriterien für Dauerbetrieb erfüllen. Für unsere konkrete Berechnung verwenden wir allerdings die konservativeren Zahlen für Systeme, die nur für geringe Nutzung ausgelegt sind. Das heißt, wir rechnen mit einem weniger verlässlichen Gatter-Subsystem.

Fehlererkennung

Das Gatter-Subsystem muss unter zwei verschiedenen Umständen eine Abweichung zwischen den Verarbeitungs-Subsystemen feststellen:

- In einem der Verarbeitungs-Subsysteme kam es zu einem nicht korrigierbaren Speicherfehler (Multibitfehler), weswegen der betreffende Prozessor den Betrieb eingestellt hat.
- Auf einem der Verarbeitungs-Subsysteme (jedoch nicht auf dem anderen) ist ein Applikationsfehler aufgetreten, der möglicherweise von einem Fehler in der Applikation, möglicherweise aber auch durch einen nicht erkannten Speicherfehler verursacht wurde.

In beiden Fällen lautet die Ausgabe mindestens eines Verarbeitungs-Subsystems nicht „OK“, so dass das Gatter-Subsystem den ATO-Controller zum Rückfall in seinen Design Safe State zwingt.

Annahmen über die Verarbeitungs-Subsysteme

Wir wollen annehmen, dass bei Abwesenheit von Speicherfehlern die Verlässlichkeit der beiden Verarbeitungs-Subsysteme in unserem 2oo2-System eine Größenordnung niedriger als SIL 1 liegt (d. h., die Wahrscheinlichkeit einer fehlerhaften Antwort liegt unterhalb von 10^{-4} je Betriebsstunde). Die nachstehenden Berechnungen gehen davon aus, dass die Zeit zwischen dem Auftreten von zwei Ausfällen einer abfallenden Exponentialverteilung mit $\lambda = 10^{-4}$ pro Stunde folgt.

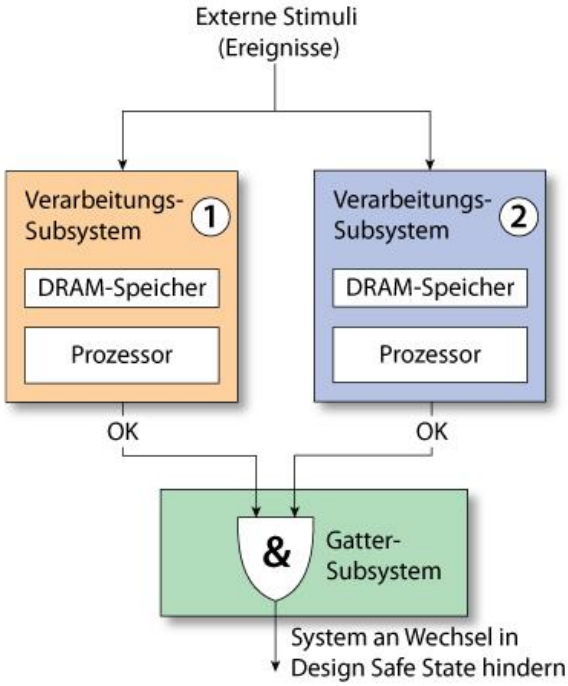

Abbildung 2. High-Level-Ansicht des 2oo2-Gatter-Systems des ATO-Controllers. Beide Verarbeitungs-Subsysteme müssen übereinstimmend melden, dass der Betrieb des Controllers sicher ist.

Die Wahrscheinlichkeit, dass beide Antworten gleichzeitig falsch sind

Es ist wichtig, die Korrelation zwischen Ausfällen der beiden Verarbeitungs-Subsysteme zu verstehen: Wie hoch ist die Wahrscheinlichkeit, dass beide Prozessoren gleichzeitig eine falsche Antwort berechnen?

Da die meisten Bugs wahrscheinlich in der Applikation (und nicht im Betriebssystem, das Millionen von Betriebsstunden im verlässlichen Feldeinsatz vorweisen kann) vorliegen und die Applikation für beide Subsysteme dieselbe ist, können wir anhand eines (hypothetischen) Fehlereinstreuungstests annehmen, dass eine 70%ige Wahrscheinlichkeit dafür besteht, dass ein Fehler in einem Verarbeitungs-Subsystem auch in dem anderen Verarbeitungs-Subsystem auftritt.

Da es sich bei den meisten während der Systemvalidierung nicht erkannten Fehlern um Heisenbugs und nicht um Bohrbugs handelt[7], ist dies vermutlich eine sehr konservative Schätzung. Wie bei allen anderen in diesem Whitepaper verwendeten Werten müsste auch die Gültigkeit dieser Annahme in Tests und Feldversuchen überprüft werden.

Annahmen über die Speicherbausteine

Wir nehmen an, dass die Speicherbausteine (DIMMs) in unserem 2oo2-System über eingebaute Einzelbit- und Mehrbit-ECC-Fehlererkennungsalgorithmen auf Basis eines Hamming-Codes mit einem Mindestabstand von 4 verfügen. Wir nehmen außerdem an, dass die Speicherbausteine *nicht* über Chipkill-Algorithmen verfügen.

Genaue Zahlen über Ausfälle von Speicherbausteinen sind leider schwer zu erhalten. Schroeder, Pinheiro und Weber veröffentlichten im Jahr 2009 unter dem Namen „DRAM Errors in the Wild: A Large-Scale Field Study“ (DRAM-Fehler in freier Wildbahn: eine groß angelegte Feldstudie)[8] eine umfassende Untersuchung, die sie an Servern von Google durchgeführt hatten. Bis dahin lagen die erwarteten Fehlerraten in der Größenordnung von 200 bis 5000 FIT[9] pro MBit, was bedeuten würde, dass bei einem 2-GB-DRAM alle 12 bis 313 Betriebsstunden mit einem Fehler zu rechnen wäre. Doch Schroeder *et. al.* fanden ein ganz anderes Bild vor: Es stellte sich heraus, dass in den DRAMs der Google-Server entweder keine oder aber eine hohe Anzahl an Fehlern auftraten. Wenn es zu Fehlern kam, so lagen die Zahlen bei 25000 bis 75000 FIT pro MBit. Ein Wert von 75000 FIT entspricht einem Fehler alle 50 Betriebsminuten bei einem 2-GB-DRAM.

Eine vertretbare Fehlerrate für unsere Berechnungen entnehmen wir den in FIT pro MBit umgerechneten Rohdaten aus Tabelle 2 (Seite 4) der Google-Studie (DRAMs für die Plattformen A, B, C, D und F). Basierend auf diesen Daten erscheint 36707 FIT pro MBit als sinnvoller Wert für korrigierbare Fehler (d. h. Einbit-Fehler, die von der ECC-Hardware des Speicherbausteins korrigiert werden).

Die Google-Studie lässt erkennen, dass bei zwischen 0,08 % und 0,3 % (im Mittel 0,22 %) der DIMMs pro Jahr ein nicht korrigierbarer Fehler auftritt, und dass es eine signifikante Korrelation zwischen korrigierbaren und nicht korrigierbaren Fehlern gibt: Wenn in einem DIMM ein korrigierbarer Fehler auftritt, ist die Wahrscheinlichkeit für das Auftreten eines nicht korrigierbaren Fehlers in demselben DIMM innerhalb desselben Monats 9- bis 47-fach höher als bei einem DIMM, in dem kein korrigierbarer Fehler beobachtet wurde.[10]

Für unser Modell nehmen wir an, dass erkannte, aber nicht korrigierbare Fehler mit einer Rate von 1 alle 1/0,22 = 4,5 Jahre je 2-GB-DIMM auftreten. Dieser Wert entspricht ungefähr 3 FIT pro MBit – einem Wert, der auch in dem Whitepaper „Soft Errors in Electronic Memory – A White Paper“ (Transiente Fehler in elektronischem Speicher – ein Whitepaper) aus dem Jahr 2004 von Tezzaron Semiconductor gestützt wird. Laut dem Whitepaper von Tezzaron treten nicht korrigierbare Fehler um eine bis zwei Zehnerpotenzen seltener auf als korrigierbare Fehler.[11]

Einen Wert können wir jedoch nicht aus den Statistiken aus der Google-Studie ableiten, und das ist die Rate der *nicht erkannten* Speicherfehler. Definitionsgemäß wurden diese Fehler nicht erkannt, so dass für sie keine Zahlen zur Verfügung stehen können.[12] Als vorsichtige Schätzung gehen wir bei der Berechnung davon aus, dass diese nicht erkannten Fehler mit einer Häufigkeit auftreten, die eine Zehnerpotenz unter der Häufigkeit nicht korrigierbarer Fehler liegt, d. h., mit 0,3 FIT pro MBit.

Zusammengefasst liegen unseren Berechnungen also die folgenden Fehlerraten zugrunde:

- Erkannte und korrigierte Fehler: 36707 FIT pro MBit
- Erkannte und nicht korrigierbare Fehler: 3 FIT pro MBit
- Nicht erkannte Fehler: 0,3 FIT pro MBit

Annahmen über den Umgang mit Speicherfehlern

Es gibt drei mögliche Arten von Speicherausfällen. Bezüglich des Umgangs mit diesen drei Fehlerarten treffen wir folgende Annahmen:

Erkannte und korrigierbare Speicherfehler

Erkannte und korrigierbare Speicherfehler werden gezählt, aber im Übrigen ignoriert, da sie von der Hardware korrigiert werden und für die Applikation nicht sichtbar sind. Angesichts der Ergebnisse der Untersuchung der Google-Server von Schroeder *et al.* wäre es in einem realen Produkt jedoch angeraten, diese Fehler zu überwachen: Korrigierbare Fehler scheinen nämlich eng mit künftigen nicht korrigierbaren Fehlern zu korrelieren.

Erkannte, aber nicht korrigierbare Speicherfehler

Fehler, die erkannt, aber nicht korrigiert werden können, führen dazu, dass der zugehörige Prozessor angehalten wird. Das wird so gut wie sicher von dem Gatter-Subsystem erkannt, so dass das System geeignete Maßnahmen zur Wiederaufnahme des Betriebs oder zum Wechsel in einen Design Safe State (siehe „Annahmen über das Gatter-Subsystem“ oben) einleiten wird. Wird der Stillstand des Prozessors jedoch nicht vom Gatter-Subsystem erkannt, kommt es zu einem gefährlichen Ausfall des Gesamtsystems.

Nicht erkannte Speicherfehler

Nicht erkannte Speicherfehler sind grundsätzlich klar die gefährlichste Fehlerart. In der Praxis betreffen zwar viele nicht erkannte Fehler nicht benutzte oder nicht initialisierte Speicherplätze und sind daher ungefährlich. Wir müssen jedoch davon ausgehen, dass einige dieser Fehler Speicherplätze betreffen werden, die in Benutzung sind. Die Folgen können relativ harmlos sein. Etwa könnte dem Benutzer in einem Meldungstext ein falscher Buchstabe angezeigt werden (harmlos ist dies natürlich nur dann, wenn der falsche Buchstabe keine Sinnänderung bewirkt und den Benutzer nicht zum Ergreifen einer falschen Maßnahme veranlasst). In einigen Fällen kann ein Fehler im aktiven Speicher aber auch dazu führen, dass die beiden Verarbeitungs-Subsysteme unterschiedliche Antworten liefern, und eine solche Diskrepanz muss von dem Gatter-Subsystem erkannt werden.

Bei den Berechnungen gehen wir davon aus, dass *alle* nicht erkannten Speicherfehler dazu führen, dass der betroffene Prozessor falsche Werte erzeugt. Dies ist eine sehr konservative Annahme, da in Wirklichkeit viele nicht erkannte Speicherfehler gar keine Auswirkung auf den betroffenen Prozessor haben werden. Wir treffen diese Annahme, da es immer klüger ist, den schlimmstmöglichen Fall anzunehmen.

Berechnung ohne softwarebasierte Fehlererkennung

Um die Häufigkeit gefährlicher Ausfälle abzuschätzen, haben wir 100 Mal eine Simulation von 10^9 Jahren (ca. 88×10^{12} Stunden) ausgeführt. Dabei haben wir genug Daten erhalten, um ein Konfidenzintervall berechnen zu können. Die Ergebnisse unserer Simulation sind in Tabelle 1 dargestellt, die wie folgt zu lesen ist: „Mit einem Konfidenzniveau von 99 % kann gesagt werden, dass unter den vorstehend aufgeführten Annahmen die Häufigkeit gefährlicher Ausfälle zwischen 7,96157 und 8,01067 FIT liegt.“

Konfidenzniveau	Wahrscheinlichkeit eines gefährlichen Ausfalls	
	Untere Grenze	Obere Grenze
95,0 %	7,96884 FIT	8,00340 FIT
97,5 %	7,96547 FIT	8,00677 FIT
99,0 %	7,96157 FIT	8,01067 FIT

Tabelle 1. Berechnungsergebnisse für die geschätzte Häufigkeit gefährlicher Ausfälle ohne softwarebasierte Fehlerkorrektur.

Kombiniert man also zwei SIL-0-Verarbeitungs-Subsysteme mit einem SIL-4-Gatter (Modus geringe Nutzung) und verzichtet dabei auf Software-Speicherfehlererkennung, so ist das System, das sich ergibt, ein SIL-4-System.

Grenzen unserer Berechnungen

Es sei an dieser Stelle angemerkt, dass wir die zweite wesentliche Berechnung, die wir unter „Annahmen über die Verarbeitungs-Subsysteme“ beschrieben haben, nicht ausgeführt haben: Wir konnten zeigen, dass das System unter den getroffenen Annahmen das geforderte Sicherheitsniveau bietet, aber wir haben nicht gezeigt, dass es die Zielvorgaben bezüglich der Verfügbarkeit erfüllen kann. Ein System, das niemals seinen Design Safe State verlässt (ein Zug, der immer steht, oder eine Ampel, die immer rot ist usw.), ist sicher, aber bestenfalls nutzlos.[13] Bei einem echten System müssten wir die Verfügbarkeit berechnen, um zu zeigen, dass unser System nicht nur zuverlässig, sondern auch verfügbar und nützlich ist (wie wir dies unter „Zuverlässigkeit versus Verfügbarkeit“ am Beispiel eines Fahrrads veranschaulicht haben).

Berechnung mit softwarebasierter Fehlererkennung

Eine softwarebasierte Fehlererkennung auf Applikationsebene arbeitet mit einer relativ geringen Geschwindigkeit (ca. 23 h zum Testen von 2 GB Speicher)[14]. Deshalb kann davon ausgegangen werden, dass ECC-Hardware sowohl korrigierbare als auch nicht korrigierbare Speicherfehler, die sie prinzipiell erkennen kann, lange vor der Software findet.

Permanente und transiente Speicherfehler

Ein permanenter Speicherfehler („Hard Error“) liegt vor, wenn ein Speicherbit keine Werte mehr annimmt. Ein transienter Speicherfehler („Soft Error“) ist nur temporär: Ein Bit verliert seinen Wert oder wird nicht korrekt gelesen, aber beim nächsten Zugriff und darüber hinaus verhält es sich wie erwartet.

Für den Software-Entwickler mag es keine große Rolle spielen, von welcher Art Speicherfehler er getroffen wird, wenn das Ergebnis sowieso ein Neustart ist, aber für das Gesamtsystem ist es überaus wichtig.

Schroeder *et al.* haben festgestellt, dass es eine Korrelation zwischen transienten Speicherfehlern und dem künftigen Auftreten permanenter Speicherfehler gibt. Weiterhin wächst mit zunehmender Verkleinerung elektronischer Komponenten auch die Geschwindigkeit, mit der diese altern und ausfallen. Transiente Fehler sind möglicherweise Vorboten eines bevorstehenden permanenten Ausfalls. Die Sicherheit hängt somit ggf. nicht nur davon ab, dass die Software (in der Regel) nach einem Speicherfehler den Betrieb wiederaufnehmen kann, sondern auch von Wartungsplänen, die den Austausch anfälliger Platinen vorsehen.

Software könnte jedoch die ECC-Hardware ergänzen und genutzt werden, um permanente Speicherfehler zu finden, die den ECC-Schaltungen entgangen sind. Eine softwarebasierte Fehlererkennung könnte also in folgender Situation nützlich sein:

1. Es tritt ein nicht erkannter Speicherfehler auf.
2. Dieser Fehler betrifft den Betrieb eines Verarbeitungs-Subsystems und bringt dieses dazu, eine falsche Antwort zu liefern.
3. Der Fehler wird nicht von dem Gatter-Subsystem abgefangen.

Für die Berechnung haben wir angenommen, dass die Prüfsoftware bei Eintreten der genannten Bedingungen eine Chance von 80 % hat, den Fehler zu finden und den betroffenen Prozessor anzuhalten. Der angehaltene Prozessor wird ggf. von dem Gatter-Subsystem abgefangen oder auch nicht.

Wir haben unsere Simulation unter Berücksichtigung dieser Annahmen erneut durchgeführt. Das Ergebnis ist interessant: Die Wahrscheinlichkeiten für gefährliche Ausfälle gegenüber den Werten aus Tabelle 1 für dasselbe System ohne softwarebasierte Fehlererkennung sind quasi unverändert geblieben. Unser System bleibt somit ein SIL4-System (Modus geringe Nutzung).

Zusammenfassung der Erkenntnisse bezüglich softwarebasierter Fehlererkennung

Das 2oo2-Modell hat sich als hervorragendes Controller-Design erwiesen, das die Sicherheit des Systems gewährleisten kann. Selbst bei sehr niedrigen Verlässlichkeitsniveaus in den Prozessorsubsystemen ist die Wahrscheinlichkeit eines gefährlichen Ausfalls *sehr* gering. Treffen unsere Annahmen zu, so ergibt sich durch Aufnahme einer softwarebasierten Erkennung von Speicherfehlern kein merklicher Unterschied für die Verlässlichkeit unseres Systems, und sowieso ist die softwarebasierte Erkennung viel zu langsam, um sich für die Behandlung von transienten Fehlern zu eignen[15]. Somit stellt sich die Erkennung von transienten Speicherfehlern per Software nicht als besonders geeignete Lösung für den Umgang mit solchen Fehlern in unserem hypothetischen ATO-Controller dar.

Ein neuer Ansatz

Unser System ist also ausreichend sicher, aber ist es auch ausreichend verfügbar oder auch nur nützlich? Wir haben festgestellt, dass False Positives bei unserem 2oo2-Design die Verfügbarkeit des Controllers signifikant beeinträchtigten können,[16] was nicht nur die Leistung des Controllers herabsetzt, sondern auch den sicheren Betrieb der S-Bahn als Ganzes gefährdet, da andere ATO-Komponenten des Steuerungssystems zusätzlich belastet werden.

Haben wir überhaupt das richtige Problem behandelt?

Wie wir gesehen haben, liegt die Wahrscheinlichkeit dafür, dass unser System fehlerhaft reagiert, innerhalb der Anforderungsgrenzen aus IEC 61508 SIL4, allerdings auf Kosten einer möglicherweise reduzierten Verfügbarkeit. Versuchen wir also, unser Problem neu zu formulieren:

1. Wir sind uns sicher, dass unser System hinreichend sicher ist.
2. Wir sind uns nicht sicher, ob unser System hinreichend verfügbar ist.

3. Wie können wir also die Verfügbarkeit unseres Systems verbessern, ohne seine Sicherheit zu beeinträchtigen?

Alternative Strategien

Im Folgenden einige Vorschläge, die zur Verbesserung der Verfügbarkeit des Controllers beitragen könnten. Wir gehen dabei davon aus, dass wir das 2oo2-Design nicht ändern, da es unseren Aussagen über die Zuverlässigkeit zugrunde liegt, und dass wir die entsprechenden Berechnungen ausführen, um die Auswirkungen jeder implementierten Änderung zu quantifizieren.

Zweite Meinungen

Falls genügend Zeit bleibt, um eine Neuberechnung anzufordern, wenn das Gatter keine Übereinstimmung festgestellt hat, so kann dies eine exzellente Lösung sein, um einen Wechsel in den Design Safe State zu vermeiden. Da transiente Speicherfehler vorübergehender Natur sind, würde im Falle einer Nichtübereinstimmung zwischen den beiden Verarbeitungs-Subsystemen eine Wiederholung der Berechnung wahrscheinlich korrekte und passende Antworten liefern und somit einen False Positive vermeiden. Erst wenn die zweite Meinung die ursprünglich festgestellte Nichtübereinstimmung bestätigt, müssen wir davon ausgehen, dass etwas nicht stimmt, und den Controller geeignete Maßnahmen einleiten lassen, wie etwa den Wechsel in einen Design Safe State.

Diese Strategie kann situationsabhängig angewendet werden. Ein Controller in einem Zug, der gerade an einer Station hält, kann sich eine Verzögerung von zwei Sekunden leisten, während derer er einen Reset durchführt und die von dem Fehler betroffenen Verarbeitungsschritte wiederholt. Ein Controller in demselben Zug, der gerade mit 50 km/h in eine Station einfährt, hat diese Option möglicherweise nicht. Die Entscheidung, eine zweite Meinung anzufordern, könnte daher abhängig von der Geschwindigkeit des Zugs und weiteren Faktoren wie seinem Standort und dem Wetter (das ggf. den Bremsweg beeinflusst) ausgelöst werden.

ECC

ECC mit Chipkill (Reparatur von bis zu 4 Bits) ist eine Möglichkeit zur Behandlung von transienten Speicherfehlern. In vielen Fällen kann schon ein SEC/DED-Algorithmus im Speicherbaustein dem System die notwendige Fehlertoleranz verleihen. Chipkill-Speicher würde eine noch größere Fehlertoleranz bewirken, da Chipkill noch mehr Speicherfehler behandeln würde, bevor sie von der Software bzw. dem Gatter-Subsystem überhaupt registriert werden. Solcher Speicher ist kostspielig, aber bei Produkten, die in geringer Stückzahl produziert werden – die Stückzahlen sind bei S-Bahn-Systemen deutlich geringer als etwa bei Autos – ist dies möglicherweise hinnehmbar und wegen des niedrigeren Speicherfehlerrisikos auch gerechtfertigt.

Austausch von Platinen

Ausgehend von den Ergebnissen der zitierten Google-Studie scheint es eine Korrelation zwischen dem Auftreten von transienten Speicherfehlern und künftigen permanenten Speicherfehlern zu geben. Möglicherweise lässt sich die Verlässlichkeit verbessern, indem wir Fehler protokollieren und Platinen austauschen, in denen besonders häufig transiente Fehler auftreten, oder indem wir Platinen generell häufiger austauschen, beispielsweise alle 20 Monate statt alle fünf Jahre.

Was haben wir übersehen?

Angesichts der Kontextänderung – ein neuer Einsatzort unseres Systems – und unseres Wissens über den Neutronenfluss in Höhenlagen war es legitim, sich auf transiente

Speicherfehler zu konzentrieren. Nachdem wir dieses Problem behandelt haben, sollten wir auch darüber nachdenken, was uns dadurch entgangen sein könnte, dass wir uns einseitig auf Lösungen für durch Höhenstrahlung verursachte transiente Speicherfehler konzentriert haben.

Ist unser System zum Beispiel anfällig für transiente Speicherfehler, die durch elektromagnetische Interferenz ausgelöst werden, welche von Handys, anderen Funksendern oder Stromquellen ausgehen kann? Gewährleisten der Installationsort des Systems und die Konstruktion des Führerstands, dass keine Strahlungsquelle dem System so nahe kommen kann, dass sie seine korrekte Funktion unabsichtlich stört? Haben die Höhenlage und die entsprechend dünnere Luft signifikante Auswirkungen auf die Kühlungsanforderungen unserer Hardware?

Da unser ATO-System bereits in einer anderen Umgebung im Einsatz ist, können wir wohl davon ausgehen, dass das System ausgehend von dem Verständnis entworfen wurde, dass einzelne Komponenten und das gesamte System in der Tat ausfallen können. Das heißt, wir haben bei der Konzipierung des Systems darauf geachtet, dass es folgende Kriterien erfüllt:

- Sicherheitskritische Komponenten sind gemäß den einschlägigen Sicherheitsstandards von anderen Komponenten und untereinander isoliert.
- Fehler werden zur Laufzeit des Systems erkannt und korrigiert.
- Ausfälle werden erkannt und je nach Möglichkeit und Anforderungen eingedämmt, oder es erfolgt ein kontrolliertes Herunterfahren und Neustarten oder ein Wechsel in einen Design Safe State.

Abschließend müssen wir uns fragen, ob die von uns vorgeschlagenen Lösungen das Ausfallrisiko senken oder steigern, und zwar sowohl in Bezug auf das konkrete von uns entworfene System (in diesem Falle den Führerstandscontroller für vollautomatischen Zugbetrieb) als auch in Bezug auf das größere Gesamtsystem, in dessen Rahmen unser System implementiert werden wird.

Wir müssen unser Anschauungsbeispiel mit dem Fahrrad wohl ein wenig anpassen, um die Fragestellungen zu erfassen, denen wir mit unserem ATO-Controller gegenüberstehen: Wir müssen nicht nur ein Fahrrad betrachten, sondern viele Fahrräder, die in einem Pulk ein Rennen fahren. Ein einzelnes Fahrrad darf auch mal langsamer werden, während sich sein Controller zurücksetzt. Ein Fahrrad in einem Rennpulk dagegen gefährdet andere Fahrräder, wenn es plötzlich bremst. Die anderen Fahrräder kommen damit möglicherweise nicht klar und schon kommt es zu einer Karambolage.

Fußnoten

1 Webseite der Eastland Memorial Society <www.eastlandmemorial.org/eastland7.shtml>

2 Office of Air Accidents Investigation, *Aircraft Accident Report No. 79-139, Air New Zealand McDonnell-Douglas DC10-30 ZK-NZP, Ross Island, Antarctica, 28. November 1979*, Wellington, 12. Juni 1980. Abs. 1.17.7. <www.nzalpa.org.nz/Portals/4/Documents/Reports/Chippindale/79-139_section1.pdf>

3 P.T. Mahon, Report of the Royal Commission to inquire into The Crash on Mount Erebus, Antarctica of a DC10 Aircraft operated by Air New Zealand Limited, Wellington, S. 157–59. <archives.govt.nz/exhibitions/currentexhibitions/chch/downloads/AntarcticReport.pdf>

4 Siehe z. B. Ray Heald, *How Cosmic Rays Cause Computer Downtime* (Wie Höhenstrahlung Computer-Ausfallzeiten verursacht) (IEEE Rel. Soc. SCV Meeting: 3/23/05) mit der Aussage: „Kosmische Ereignisse sind die vorherrschende Ursache für transiente Fehler in ICs aus Materialien mit niedriger Alphaaktivität", S. 22, und „Das Hauptproblem sind die Neutronen", S. 23. Siehe auch Tom Simonite, „Should every computer chip have a cosmic ray detector?" (Sollte jeder Computerchip einen Höhenstrahlungsdetektor haben?) *New Scientist Technology Blog*. 7. März 2008. <www.newscientist.com/blog/technology/2008/03/do-we-need-cosmic-ray-alerts-for.html>

5 Details siehe <www.seutest.com>

6 Wir haben bewusst einen Extremfall gewählt. Die Auswirkungen der Höhenstrahlung sind jedoch auch zum Beispiel in Denver (Colorado, USA), Davos (Schweiz) oder Johannesburg (Südafrika) nicht zu vernachlässigen. Noch anfälliger sind natürlich Systeme in Flugzeugen.

7 Ein Heisenbug ist ein nicht reproduzierbarer Ausfall. Ein gleichzeitiges Auftreten eines Heisenbugs in beiden Subsystemen ist unwahrscheinlich. Ein Bohrbug ist ein solider, reproduzierbarer Bug, der auf beiden Systemen auftreten würde.

8 Bianca Schroeder, Eduardo Pinheiro und Wolf-Dietrich Weber, „DRAM Errors in the Wild: A Large-Scale Field Study" (DRAM-Fehler in freier Wildbahn: eine groß angelegte Feldstudie), Seattle: SIGMETRICS/Performance 2009, 15.-19. Juni 2009.

9 Ein FIT ist ein Ausfall pro 10^9 Stunden.

10 Schroeder *et al.*, S. 6

11 Tezzaron Semiconductor, „Soft Errors in Electronic Memory—A White Paper" (Transiente Fehler in elektronischem Speicher – ein Whitepaper), 2004. <www.tezzaron.com/about/papers/soft_errors_1_1_secure.pdf>

12 Dies ist ein Nichtbeweisbarkeitsproblem. Wir können sagen, dass wir keine Anzeichen für Fehler im System gefunden haben, aber wir können nicht sagen, dass nachweislich keine Fehler im System vorliegen.

13 Eine Ampel, die immer rot ist, ist nur dann sicher, wenn man annimmt, dass die Fahrer sich auch immer an das Lichtsignal halten. In der Praxis werden die Autofahrer irgendwann die Geduld verlieren (wann genau, das hängt von der individuellen Persönlichkeit und der lokalen Fahrkultur ab ...) und nehmen das Risiko auf sich, die Kreuzung bei rotem Licht zu überqueren.

14 Unter der Annahme, dass alle 330 ms 8 KByte getestet werden.

15 Ob die Erkennung eines Fehlers, nachdem das System bereits einige Stunden falsch gearbeitet hat, überhaupt nützlich ist, muss auf Systemebene betrachtet und entschieden werden.

16 Wechselt der Controller unnötigerweise in seinen Design Safe State, so ist dies letztlich auch ein Zuverlässigkeitsproblem: Der Entscheidungsmechanismus hat die falsche Antwort geliefert.

Über QNX Software Systems

QNX Software Systems Limited, eine Tochtergesellschaft von BlackBerry, ist ein führender Anbieter von Betriebssystemen, Entwicklungswerkzeugen und Dienstleistungen für vernetzte Embedded-Systeme. Weltmarktführer wie Audi, Cisco, General Electric, Lockheed Martin und Siemens nutzen Technologie von QNX zum Beispiel in Fahrzeug-Infotainment-Einheiten, Netzwerkroutern, medizintechnischen Geräten, industriellen Automatisierungsanlagen sowie Sicherheitssystemen und anderen missions- oder betriebskritischen Anwendungen. QNX Software Systems Limited wurde 1980 gegründet. Der Hauptsitz des Unternehmens befindet sich in Ottawa, Kanada; die deutsche Niederlassung befindet sich in Hannover. QNX-Produkte werden weltweit in über 100 Ländern vertrieben. Besuchen Sie www.qnx.de und https://www.facebook.com/QNXSoftwareSystems, und folgen Sie @QNX_News auf Twitter. Weitere Informationen zu unserer Tätigkeit im Automobilbereich finden Sie unter qnxauto.blogspot.com oder folgen Sie @QNX_Auto.

www.qnx.de