Funktionale Sicherheit: Zertifizierte Mikrokontroller Selbsttest-Software

Was Sie für eine IEC 61508 Zertifizierung beachten sollten

Dr. Jörg Koch, Renesas Electronics Europe

Eine Selbsttest-Software ist ein grundlegender Baustein bei der Entwicklung von Anwendungen im Bereich Funktionaler Sicherheit. In komplexen Komponenten wie der CPU eines Mikrokontrollers ist es schwierig, die geforderte Diagnoseabdeckung nachweislich zu erfüllen. Hierzu sind detaillierte Kenntnisse der zugrundeliegenden Hardware erforderlich. Die Validierung der geforderten Testabdeckung kann mithilfe gezielter Fehlersimulation erreicht werden. Dieser Beitrag beschäftigt sich mit der Entwicklung einer Selbsttest-Software für die CPU der Renesas RX631/63N Mikrokontroller-Familie. Die Entwicklung der hier vorgestellten CPU Selbsttest-Software wurde nach IEC 61508 durchgeführt und durch einen TÜV Rheinland Zertifizierungsprozess begleitet.

An eine CPU Selbsttest-Software wird eine Reihe von Anforderungen gestellt. Sie soll mit einem ausreichenden Diagnosedeckungsgrad zufällige Hardware-Fehler der CPU während der Laufzeit entdecken. In einem sicherheitsrelevanten System liegt die notwendige Detektions-Zeit unterhalb der „Process Safety Time“. Diese ist anwendungsabhängig und kann in zeitkritischen Systemen bis hinab zu wenigen Millisekunden betragen. Außerdem sollte die Selbsttest-Software nur einen begrenzten Teil des zur Verfügung stehenden Programmspeichers belegen und eine möglichst kurze Interrupt-Maskierungszeit aufweisen. Beim Entwurf der Selbsttest-Software muss eine Balance zwischen diesen teils gegenläufigen Anforderungen erreicht werden. Der Entwicklungsprozess hat darüber hinaus den Anforderungen der IEC 61508 zu genügen, soll die Selbsttest-Software in sicherheitsgerichteten Anwendungen mit entsprechendem SIL/SC eingesetzt werden. Die Zertifizierung der Selbsttest-Software durch ein unabhängiges Institut wie dem TÜV Rheinland hilft, die entsprechenden hohen Qualitätsanforderungen zu erfüllen.

Grundlagen der CPU Selbsttest-Software

Die Selbsttest-Software ist modular aufgebaut. Der gesamte Test besteht aus 40 Testmodulen. Jedes Testmodul kann einzeln oder auch in Gruppen aufgerufen werden, je nach dem maximal von der Anwendung zur Verfügung gestellten Zeitfenster für einen Teiltest. Der gesamte Test muss innerhalb der Process Safety Time ausführbar sein.

Ziel der Selbsttest-Software ist die Stimulierung und Detektion von zufälligen Hardware-Fehlern der CPU. Eine Optimierung des Diagnosedeckungsgrades muss sowohl beim Stimulus als auch bei der Detektion erfolgen. Um einen gut kontrollierbaren Stimulus zu erreichen, erfolgt die Programmierung der Testmodule in Assembler. Das User-Interface jedes Testmoduls ist allerdings in C realisiert. Die Resultate der verschiedenen sog. „elementaren“ Tests eines Testmoduls werden in einem Signaturwert verknüpft. Jedes Testmodul liefert nach Ausführung einen Ist-Signaturwert, anhand dessen bei Abweichung vom Soll-Signaturwert der Test als

fehlerhaft erkannt werden kann. Die Art der Signaturbildung beeinflusst die Effizienz des Detektionsprozesses. Bei der beschriebenen CPU Selbsttest-Software wird eine 32-bit Signatur benutzt, die die elementaren Testergebnisse mit Hilfe einer Kombination von Exklusiv-Oder- und Bit-Rotations-Funktion verknüpft. Jedes Testmodul liefert das Ergebnis „Pass“ oder „Fail“ und speichert zusätzlich die ermittelte Signatur in einem vordefinierten Speicherbereich. Diese Signaturen können dann z.B. von einem externen „Safety-Prozessor“ ausgewertet werden ohne sich auf die Auswertung durch die getestete CPU selber verlassen zu müssen.

Es ist wichtig anzumerken, dass eine CPU Selbsttest-Software immer durch die Anwendung eines Watchdog-Timers flankiert werden muss, da es Fehler in der CPU mit fatalen Folgen für die Ablaufsteuerung gibt. Solche Fehler werden in der Regel durch einen Watchdog-Timer erkannt.

Fehlersimulation
Grundlage zur Fehlersimulation ist die sogenannte „Netzliste“ des Mikrokontrollers, die die gesamte Logik in ASCII-Format darstellt. Mit Hilfe dieser Netzliste kann mit entsprechenden Software-Tools das Verhalten des Mikrokontrollers bei Einstreuen von Fehlern simuliert werden.

Stuck-At-0 / Stuck-At-1 Fehlereinstreuung

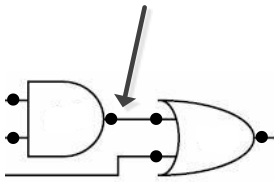


Abb. 1: Stuck-At Fehlersimulation

Es wurde das sog. Stuck-At Fehlermodell benutzt, das weitverbreitet und etabliert ist und gut die verschiedenen physikalischen Fehler in einer Logikschaltung modelliert. Dabei wird der Wert eines Ein- oder Ausgangs eines logischen Gatters der Netzliste auf „1“ oder „0“ geklemmt und damit die Auswirkungen eines Kurzschlusses zur Versorgungsspannung oder GND untersucht (Abb. 1). Die Ausführung der Selbsttest-Software mit eingestreutem Fehler wird dann simuliert und Stimulation und Detektion des Fehlers kontrolliert. Dabei wird auch der Einsatz eines Watchdog-Timers berücksichtigt. Insgesamt mussten etwa 200000 mögliche Stuck-At Fehler in der CPU simuliert werden, was nur durch massive Parallelprozessierung in vernünftiger Zeit durchgeführt werden kann.

Bei der Fehlersimulation konnten Untermodule der CPU separat untersucht werden, die verschiedene Funktionen wie Register, Rechenlogik oder logische Operationen realisieren. Dies erleichterte die gezielte Verbesserung der Selbsttest-Software in den entsprechenden Bereichen.

Optimierung der Selbsttest-Software
Die Selbsttest-Software wurde in mehreren Schritten entwickelt. Den Anfang bildete eine durch ein erfahrenes Software-Team erstellte Basisversion, deren Entwicklung sich an den entsprechenden Anwenderhandbüchern für die Hardware und den Befehlssatz des Mikrokontrollers orientierte.

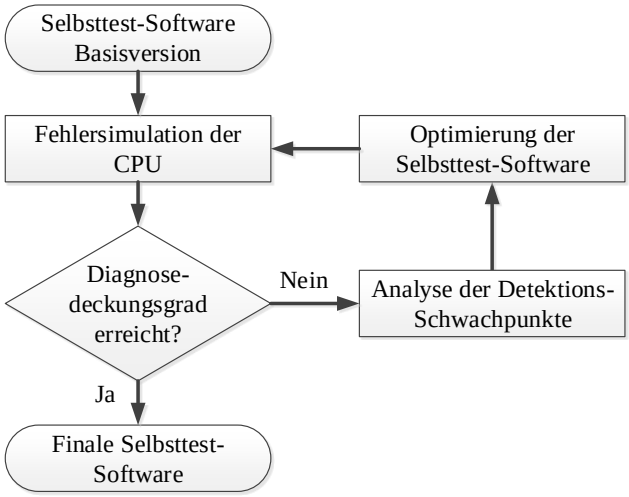


Abb. 2: Optimierungsschritte zur Entwicklung der Selbsttest-Software

Der Diagnosedeckungsgrad der Basisversion wurde mit Hilfe der Fehlersimulation bestimmt. Auf Grundlage der Ergebnisse für die verschiedenen CPU Untermodule wurde die Selbsttest-Software entsprechend optimiert (Abb. 2). Dabei mussten auch die oben bereits erwähnten weiteren Anforderungen an Laufzeit, Speicherplatzbedarf und Interrupt-Maskierungszeit berücksichtigt werden.

Abb. 3 zeigt den Verlauf des Anteils der entdeckten Stuck-At Fehler an der Gesamtfehlerzahl der CPU (blauer Verlauf). Um den Diagnosedeckungsgrad der CPU zu ermitteln, müssen noch zwei weitere Korrekturen vorgenommen werden (roter Verlauf).

Zum einen werden alle Fehler, die prinzipiell keine gefährlichen Auswirkungen haben können und entsprechend nicht in der Fehlersimulation beobachtet werden können, von der Gesamtzahl der Fehler subtrahiert. Diese sog. No-Effect Fehler werden nicht bei der Berechnung des Diagnosedeckungsgrades berücksichtigt. Typische Vertreter sind Fehler in Bereichen der Testlogik der CPU wie z.B. der Scan-In Eingang eines Flip-Flops. Diese Fehler können nur bei Aktivierung der entsprechenden Testlogik stimuliert und entdeckt werden. Bei der hier verwendeten CPU beträgt die Anzahl dieser No-Effect Fehler etwa 12%.

Darüber hinaus verlangt die IEC 61508 im Gegensatz z.B. zur ISO 26262 die Betrachtung des sog. DC Fehlermodells für einen Diagnosedeckungsgrad von 90%. Das DC Fehlermodell betrachtet neben Stuck-At auch weitere Fehlertypen wie Stuck-Open oder Bridge. Die erzielten Detektionswerte für Stuck-At Fehler müssen durch einen Korrekturfaktor, der z.B. aus theoretischen Betrachtungen gewonnen werden kann, entsprechend vermindert werden.

Man sieht in Abb. 3 gut die Auswirkungen der Verbesserung der Diagnosedeckung nach entsprechenden Optimierungsschritten der Selbsttest-Software. Dabei hat die Basisversion bereits einen recht hohen Detektionswert von über 70%. Dies entspricht nach oben erwähnter Korrektur einem Diagnosedeckungsgrad von etwa 79% für gefährliche zufällige Hardware-Fehler. Obwohl bei der Entwicklung der Basisversion bereits ein hoher Aufwand durch ein erfahrenes Software-Team betrieben wurde, liegt der Diagnosedeckungsgrad noch deutlich unter dem Zielwert von 90%.

Die finale Version der Selbsttest-Software konnte nach entsprechender Optimierung und nach Anwendung der oben beschriebenen Korrekturen einen Diagnosedeckungsgrad von 91% erreichen. Durch die Optimierung konnte die Anzahl der nicht entdeckten gefährlichen Fehler in der CPU mehr als halbiert werden.

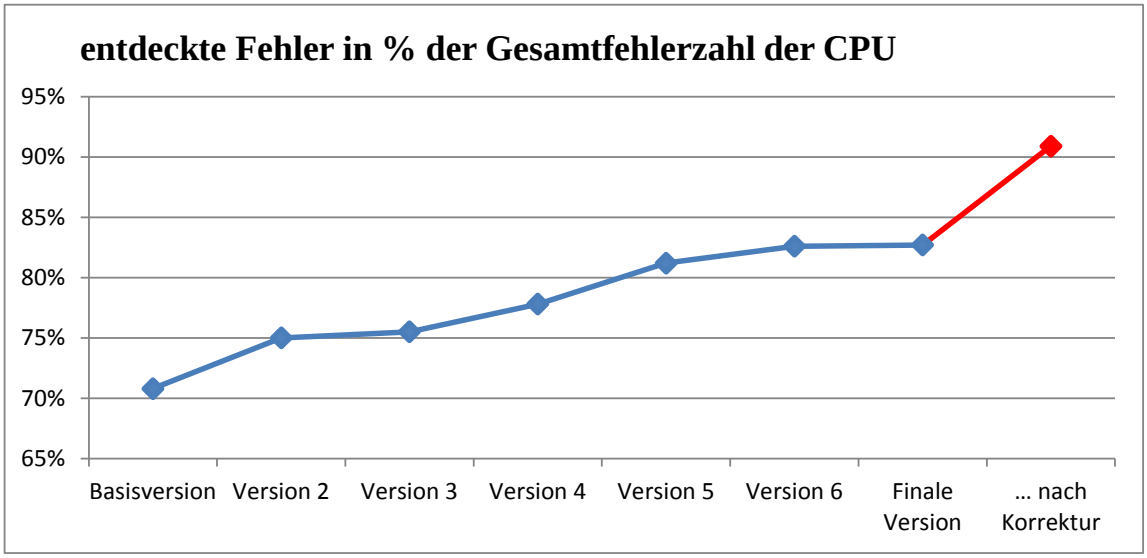


Abb. 3: Optimierungsverlauf der Selbsttest-Software

Ein näherer Blick auf die Schwachstellen der Basisversion der Selbsttest-Software zeigte vor allen Dingen bei den Rechenoperationen und im Bereich der Floating-Point-Unit Verbesserungsbedarf. Bei den Rechenoperationen musste der Wertebereich der Operanden deutlich erweitert werden, um einen ausreichenden Detektionswert des jeweiligen CPU-Untermoduls zu erreichen. Auch die Tests der Arbeitsregister konnten stark verbessert werden. Dies wurde z.B. durch Erweitern von registerbasierten Adressierungen und registerbasierten arithmetischen Operationen erreicht. Auch bei

speziellen Funktionen der untersuchten CPU (String-Funktionen, Befehls-Pipeline, etc.) konnten durch die Fehlersimulation und der Kenntnis der Struktur der CPU erhebliche Verbesserungen erreicht werden.

Anforderungen der IEC 61508

Die Selbsttest-Software wurde entsprechend den in IEC 61508 definierten Anforderungen zur Einhaltung des SIL3/SC3 Sicherheitsniveaus entwickelt. Eine zentrale Rolle spielt hierbei die lückenlose Dokumentation bei den verschiedenen Schritten der Entwicklung.

Die grundlegende Planung zu Beginn der Maßnahmen zur Erreichung sicherheitsgerichteter Ziele wird im Safety Plan dokumentiert. Er definiert z.B. alle an der Aktivität beteiligten Gruppen mit entsprechenden Schnittstellen, die verwendeten Tools mit entsprechender Qualifizierung, das Konfigurations-Management, Software Guidelines und Software Lebenszyklus-Phasen. Ein Tailoring der Anforderungen von Teil 3 der IEC 61508 an die Software-Entwicklung ist ebenfalls Teil des Safety Plans. Detaillierte Planungsdokumente bzgl. Verifikation, Validierung, etc. ergänzen den Safety Plan.

Eine wichtige Rolle bei der Entwicklung nach IEC 61508 spielt auch die klare Definition und Dokumentation von Requirements, z.B. in Bezug auf Funktion, Entwicklung und Verifikation der Software. Ein Requirements-Check muss ebenfalls Teil der Verifikation sein.

Eine nach IEC 61508 zertifizierte Entwicklungsumgebung inklusive Build-Chain erlaubt eine flexible Entwicklung von Software zur Verwendung in sicherheitsgerichteten Anwendungen. Ist eine solche vorzertifizierte Entwicklungsumgebung nicht verfügbar, muss der Nachweis, dass die Entwicklungsumgebung für die Entwicklung von Software in sicherheitsgerichteten Anwendungen geeignet ist, in Eigenarbeit erbracht werden. Alternativ muss der Output für jede Anwendung separat verifiziert werden.

Wie oben bereits erwähnt fordert die IEC 61508 die Betrachtung des DC Fehlermodells für einen Diagnosedeckungsgrad von 90%. Die notwendige Korrektur bedingt, dass bei der Fehlersimulation auf Grundlage des Stuck-At Fehlermodells ein Detektionswert von über 90% erreicht werden muss.

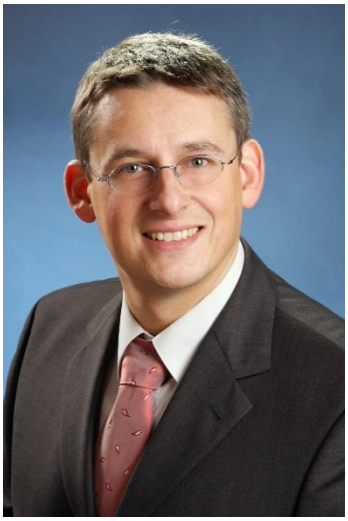
Zusammenfassung

Bei der Erstellung einer CPU Selbsttest-Software kann durch die Fehlersimulation der CPU eine erhebliche Optimierung des Diagnosedeckungsgrades des Tests erreicht werden. Aufgrund der gemachten Erfahrungen kommen wir zu der Aussage, dass es sehr schwer sein dürfte, ohne die Unterstützung durch Fehlersimulation eine Selbsttest-Software mit einem Diagnosedeckungsgrad von 90% zu erstellen.

Autor

Dr. Jörg Koch ist Experte im Functional Safety Competence Centre der Firma Renesas Electronics Europe und betreut firmenweit Projekte der Funktionalen Sicherheit. Er nimmt außerdem regelmäßig als Gast an den Sitzungen des DKE/GK 914 "Funktionale

Sicherheit elektrischer, elektronischer und programmierbarer elektronischer Systeme (E, E, PES) zum Schutz von Personen und Umwelt" teil. Nach dem Studium der Physik an der RWTH Aachen promovierte er an der Ruhr-Universität Bochum im Bereich Halbleiterphysik.



Kontakt
Internet: www.renesas.com
Email: joerg.koch@renesas.com