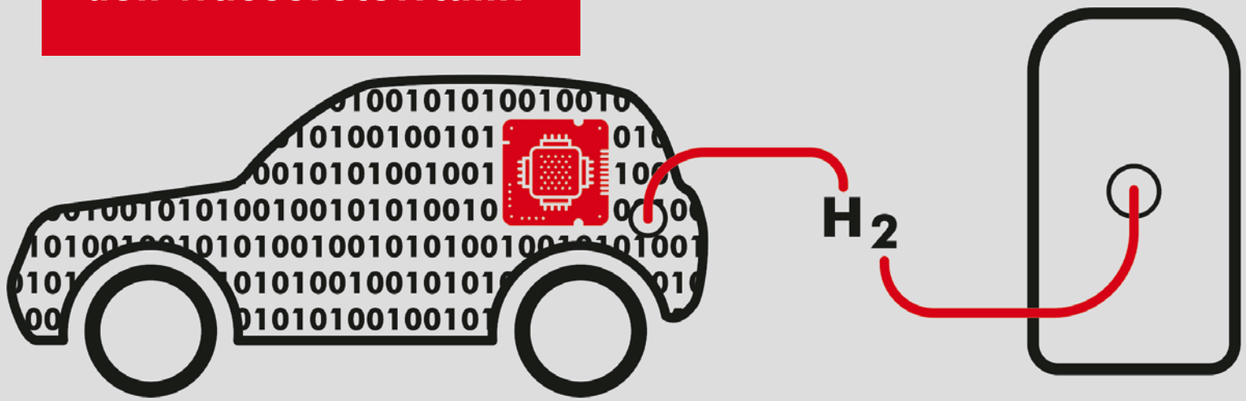
Rapid Safety für den Wasserstofftank



© Hakawut Potjanarit | 123RF.com

AUTOR

Christian Herbst
ist Projektleiter Automotive bei der Assystem Germany GmbH in München.

Mit Software Coded Processings können Sicherheitsanforderungen bis ASIL D ausschließlich mit einem diversitären Softwarekanal umgesetzt und damit kostenintensive Hardwareredundanz reduziert werden. Assystem erläutert dieses Verfahren am Beispiel der Wasserstofftank-ECU für eine elektrifizierte Prototypenflotte und zeigt neue Anwendungsfelder.

SOFTWARE CODED PROCESSING IN DER PRAXIS

Funktionale Sicherheit ist ein integraler Bestandteil jeder Neuentwicklung von sicherheitsrelevanten Steuergeräten. Eine Grundvoraussetzung, um sichere Systeme auch mit unsicheren Standardkomponenten (CPUs, SoCs) entwickeln zu können, ist bislang deren redundante Auslegung. Eine mehrkanalige Hardware auf Basis von Multiapplikationssystemen und Mehrkernarchitekturen ist ein weit verbreiteter Ansatz. Software Coded Processing ermöglicht es hingegen, Sicherheitsanforderungen bis ASIL D ausschließlich mit einem diversitären Softwarekanal zu realisieren. Dies führt zu einer deutlichen Reduzierung oder sogar Verzicht von kostenintensiver Hardwareredundanz. Das Verfahren wird nachfolgend am Beispiel der Wasserstofftank-ECU für eine elektrifizierte Prototypenflotte erläutert und abschließend ein Ausblick auf neue Anwendungsfelder gegeben.

BEISPIEL WASSERSTOFFTANK-ECU

Um das Steuergerät für den Wasserstofftank im Fahrzeug kosteneffizient, flexibel und mit den geforderten Sicherheitszielen zu realisieren, setzten die Protypen-Entwickler auf ein einkanaliges Rapid-Prototyping-Control-System, das ohne typische Safety-Module wie zum Beispiel eine Memory-Protection-Unit auskommt. Eine Herausforderung lag darin, dass dadurch Hardware wie Basissoftware nicht an die gestellten Sicherheitsanforderungen angepasst werden konnten. Der Nachweis der funktionalen Sicherheit – eine Grundvoraussetzung, um die Fahrzeuge für den Straßenbetrieb zuzulassen – musste daher ausschließlich in der Applikationssoftware geführt werden. Einfache Integration der Sicherheitsmaßnahmen in die vorhandene einkanalige Zielhardware und Infrastruktur sowie die Koexistenz von sicherheits- und nicht sicherheitsrelevanten Funktionen innerhalb der Applikationssoftware lauteten die wesentlichen Anforderungen. Hohe Flexibilität und Portierbarkeit der Implementierung sowie die Eignung für Funktionen mit hoher Kritikalität und die Verwendung einer modellbasierten Serien-Toolchain waren ebenfalls gefordert.

Korrektes Verhalten	21 + 6 = 27
Operandenfehler	Veralteter oder verfälschter Wert des Operanden/verfälschter Operand: 21 + 8 = 29
Operatorfehler	Durch verfälschten Programmzähler /Instruktion wird eine falsche Instruktion ausgeführt: 21 + 6 +23 = 50
Berechnungsfehler	Die Operation erzeugt ein falsches Ergebnis: 21 + 6 = 35

TABELLE 1 Programmausführungsfehler (© Assystem Germany)

FEHLERFREIHEIT ALS VORAUSSETZUNG

Um mit Software Coded Processing (SCP) die geforderten Sicherheitsziele einzuhalten und Safety-Mechanismen effizient zu realisieren, sind einige Rahmenbedingungen zwingend erforderlich. So müssen die sicherheitsgerichteten Anteile der (Applikations)Software hinsichtlich Programmier- und Ausführungsfehlern fehlerfrei sein. Typische Ursachen für Ausführungsfehler sind zufällige, permanente oder transiente Hardwarefehler wie Hardware-Aging oder Bit-Flips im Speicher oder in der CPU. Zugleich stellen auch systematische Fehler im Hardwaredesign oder Software-Interference-Fehler potenzielle Safety-Risiken dar. Propagieren sich diese Fehler in die Programmausführung führt diese zu einem Fehlverhalten des Systems bis hin zur Entstehung einer Gefahrensituation. Bei der Software führen diese Fehler dann zu korrupten Daten (Operandenfehler), unerwarteten Operationen (Operatorfehler) oder einfach zu falschen Ergebnissen (Berechnungsfehler). **TABELLE 1** zeigt typische Beispiele für diese Fehlerklassen.

Ist die Software frei von systematischen Fehlern (Programmierfehlern), wird zum Nachweis der funktionalen Sicherheit zusätzlich eine Sicherung der Daten und Operationen gegen zufällige Hardwarefehler benötigt. Das System kann so kritische Fehler erkennen und mit Schutzreaktionen abfangen.

HÄRTUNG DER DATEN ALS METHODIK

Beim Software Coded Processing wird die fehlende Hardwareredundanz durch diversitäre Redundanz in der Software kompensiert. Sie basiert auf der Idee des 1989 veröffentlichten Vital Coded Processors (VCP) [1]. VCP war der erste Ansatz, der Coded Processing zu großen Teilen ausschließlich in Software implementiert. Die ersten Praxiseinsätze erfolgten beispielsweise bei automatischen und halbautomatischen Zugführungssystemen, wie im Nahverkehrssystem der Linie RER A in Paris.

Die Härtung der Daten (Variablen und Konstanten) gegen Hardwarefehler und Software-Interferenz basiert auf der Transformation der Funktionswerte in redundante, ANBD-kodierte Werte.

BILD 1 zeigt die Kodierung der Daten durch Hinzufügen von redundanten Informationen.

Der Faktor A muss eine Primzahl sein und bestimmt wichtige sicherheitsrelevante Kenngrößen wie den Hammingabstand und die Restfehlerwahrscheinlichkeit P=1/A der Software. Er dient der Erkennung von Bitfehlern (Bitkipper). Mit der variablenspezifischen Signatur B wird die Korrektheit der Speicheradresse der benutzten Variablen sichergestellt (Adressierungsfehler). Die Detektion von veralteten, nicht aktualisierten Daten erfolgt mit dem Zeitstempel D, zum Beispiel einem Task-Cycle-Counter. Da die Rechenoperationen immer direkt auf den kodierten Werten (hier: x_c und y_c) stattfinden, ist für jede Rechenoperation ein kodierter Operator nötig. Würden die Daten vor der Operation dekodiert werden, lägen wieder die ungesicherten Werte vor. Eine Addition von kodierten Werten ist wie folgt definiert:

GI. 1 $z_c = x_c + (c)\, y_c$

GI. 2 $z_c = x_c + y_c + (B_z - B_x - B_y) - D$

Die Kodierung der übrigen grundlegenden arithmetischen und logischen Operatoren erfolgt ähnlich. Wie das Beispiel der Addition – der einfachsten arithmetischen Operation – zeigt, erhalten die Operationen durch die Kodierung eine deutlich komplexere Struktur.

Wie kann nun mit diesen Maßnahmen eine ungewollte Manipulation der Daten oder eine fehlerhaft ausgeführte Berechnung erkannt werden? Eine Möglichkeit der Fehlerüberprüfung ist der Vergleich des Ergebnisses der kodierten Operation

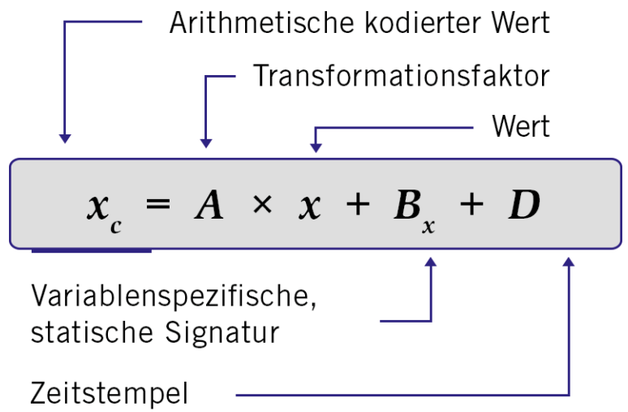


BILD 1 Transformationsvorschrift für Daten (© Assystem Germany)

(zc) mit dem Ergebnis des redundanten (unkodierten) Kanals:

Gl. 3 $z_c = A \cdot (x+y) + B_z + D$

Wahlweise kann die Korrektheit der Daten mit der Modulo-Verifikation nachgewiesen werden. Es liegt kein Fehler vor, wenn das Ergebnis einer Operation ein Vielfaches des Transformationsfaktors A ist:

Gl. 4 $(z_c - B_z - D) \bmod A == 0$

Mit arithmetisch kodierten Daten ist es also möglich, sicherheitskritische Daten und deren Verarbeitung innerhalb der ECU abzusichern. Die dabei erzielbaren Fehlererkennungsraten erfüllen die Anforderungen bis ASIL D. [2]

ARITHMETISCHE KODIERUNG

Die interne, statische Architektur der Safety-Software wird bei der arithmetischen Kodierung ausschließlich von Komponenten auf Applikationsebene und deren Schnittstellen definiert. Sie basiert auf einer logischen Kapselung der technischen Funktionen. Diese wurden als strukturell getrennte Softwarekomponenten implementiert. Die Basissoftware realisiert nur den Zugriff auf die externen Schnittstellen der ECU und stellt einige Low-Level-Funktionen zur Signalkonditionierung zur Verfügung. Im Folgenden werden die sicherheitsgerichteten Merkmale der Applikationsebene detaillierter erläutert.

Die Überwachungsfunktionen wurden aus dem Sicherheitskonzept des Systems abgeleitet und zweikanalig, das heißt redundant diversitär, implementiert. Der erste Softwarekanal dieser Zusatzfunktionen ist unkodiert in den ungesicherten, funktionalen Teil der Softwarekomponenten zusammen mit den zugehörigen Nutzfunktionen integriert.

Tritt ein unzulässiger Systemzustand ein, aktiviert die entsprechende Überwachungsfunktion über den Pfad der Nutzfunktion den sicheren Zustand. Unabhängig vom Systemzustand meldet die Sicherheitsfunktion kontinuierlich ihren Status an den zweiten Kanal.

Der zweite Softwarekanal wurde in der zentralen Safety-Komponente der Überwachungsfunktionen gekapselt, **BILD 2**, Bereich 2. Sie enthält die ANBD-kodierten Daten und Operationen sowie diversitäre, funktional identische Vergleichs- und Überwachungsfunktionen. Die Ergebnisse dieser Funktionen werden ebenfalls an den Safety Voter übermittelt. Dieser konsolidiert und plausibilisiert die Ergebnisse der beiden Überwachungskanäle. Unabhängig von den Nutzfunktionen und dem unkodierten Kanal wird der gewünschte sichere Zustand über einen redundanten Abschaltpfad ausgelöst.

Neben dem redundanten zweiten Kanal und der Koordination der Ergebnisse stellt die Safety-Komponente Dienste für standardisierte Fehlerreaktionen zur Verfügung. Dazu gehören der Übergang in den sicheren beziehungsweise degradierten Zustand, die Heilung von Fehlerzuständen, die Ansteuerung der Fehlfunktionsanzeige sowie ein einfacher Historienspeicher.

Zur Absicherung der vollständigen Wirkkette der Überwachungsfunktionen wird die Integrität der sicherheitsrelevanten, externen Schnittstellen durch klassische Safety-Engineering-Techniken überprüft. Dazu gehören diversitäre Eingangsdaten und End-to-End-Protection der Bus-Signale ebenso wie ein zusätzlicher Abschaltpfad der Aktuatoren und die Programmablaufkontrolle mit einem Watchdog.

Eine neuentwickelte Safety-Bibliothek reduzierte den Modellierungsaufwand für die Safety-Komponente und half fehlerträchtige, parallele Programmierung zu vermeiden. Auf Basis einer Analyse der Sicherheitsfunktionen konnten die benötigten kodierten, arithmetischen und logischen Operationen identifiziert werden. Diese Low-Level-Funktionen wurden unabhängig von der Funktionsentwicklung modelliert, verifiziert und in eine Bibliothek ausgelagert. Der Entwickler erhält dadurch vordefinierte Prozeduren und Funktionen, die den Aufwand für die Verarbeitung von ANBD-kodierten Daten erheblich reduzieren. Für die Verwendung der Bibliothek sind keine Kenntnisse des VCP-Ansatzes und der zugrundeliegende Mathematik erforderlich.

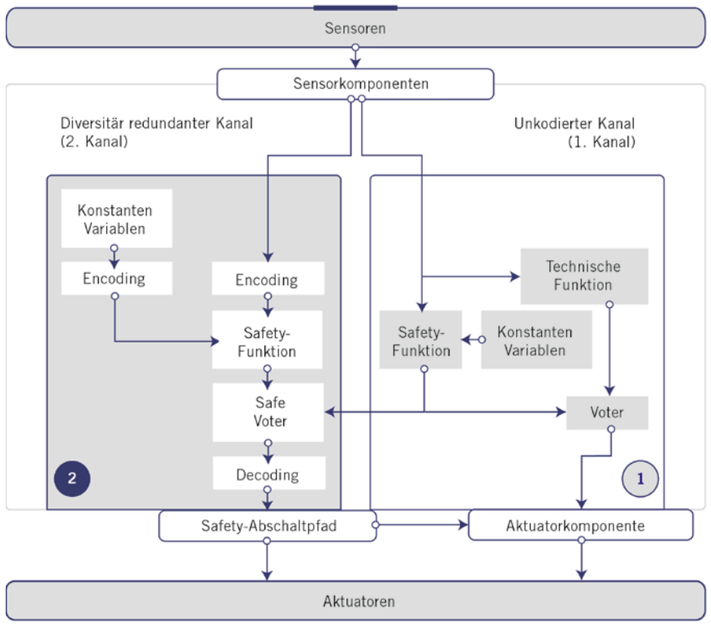


BILD 2 High-Level Softwarearchitektur (© Assystem Germany)

FAZIT UND AUSBLICK

Software Coded Processing ist ein vielversprechendes Verfahren, um Safety-Mechanismen sicher und kosteneffizient zu realisieren. Mit der beschriebenen Architektur können diese genügend unabhängig von Nutz- und anderen Sicherheitsfunktionen agieren. Weitere Maßnahmen zur Isolation der sicherheitskritischen Anteile von der Standardsoftware, zum Beispiel Partitionierung der Software auf unterschiedliche Tasks oder Rechnerkerne, sind nicht notwendig. Eine unerwünschte, intransparente und gefährliche Fehlerpropagation wird zuverlässig verhindert. Der Overhead, der durch die Replikation der Überwachungsfunktionen, **BILD 1** und Gl. 2 entsteht, führte auch im Projekt nicht zu einer signifikanten Verschlechterung des Laufzeitverhaltens der Software. Durch die gezielte Anwendung des Konzepts konnten im Vergleich zu anderen Safety-Architekturen viele Standard-Safety-Checks entfallen. So wurde der Mehraufwand an Laufzeit und Ressourcen größtenteils kompensiert.

SCP beschränkt sich jedoch nicht alleine auf die Softwareentwicklung unter Prototypen-Bedingungen. Aus technoökonomischer Sicht besitzt das Verfahren eine Reihe von Merkmalen, die es auch für die Serienentwicklung softwareintensiver Systeme attraktiv macht. Gegenüber applikationsspezifischen oder hochspezialisierten Mikrocontrollern lassen sich auf Basis von Software Coded Processing zuverlässige und sichere Systeme mit Standardprozessoren (CPUs, SoCs) aufbauen. Dies reduziert, insbesondere bei hohen Stückzahlen im Automobilsektor, die Hardwarekosten drastisch. Im Vergleich zu Fehlerkorrekturverfahren wie Error Correcting Codes (ECC) oder Cyclic Redundancy Checks (CRC), die Daten nur im RAM/ROM absichern, bietet das Verfahren darüber hinaus eine End-to-End-Protection über den gesamten Speicherbereich einschließlich CPU-Register, Cache, RAM/ROM und Pipelines, **BILD 3**.

Weitere Anforderungen zur funktionalen Sicherheit wie zum Beispiel Opcode-Diagnose und Program Flow Monitoring können ebenfalls erfüllt werden. Da keine Hardwarespezifischen Low-Level-Treiber, Funktionsbibliotheken oder Tools notwendig sind, wird eine sehr hohe Portierbarkeit der Applikation erreicht. Durch die aktuellen Fahrzeuginnovationen, zum Beispiel bei Fahrerassistenzsystemen bis hin zum hochautomatisierten Fahren, rückt noch eine weitere Eigenschaft der vorgestellten Architektur in den Fokus: Das hohe Maß an Isolation, bei gleichzeitiger starker Integration der einzelnen Komponenten in Form einer engen Vernetzung. Damit ist der Datenaustausch zwischen den Softwaremodulen bei komplexen Steuerungs- und Interaktionsmustern deutlich einfacher umzusetzen. Software Coded Processing bietet damit eine praxistaugliche Alternative für sicherheitsgerichtete Systeme ohne kostenintensive Hardwareredundanz.

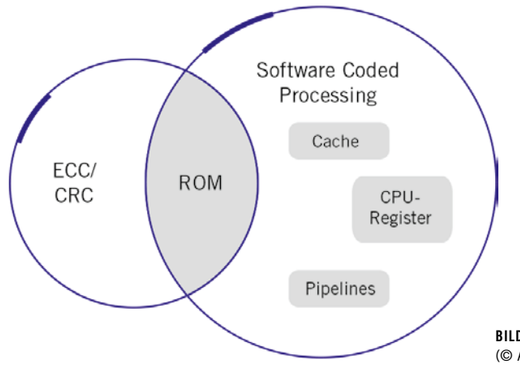


BILD 3 Abgesicherte Datenbereiche (© Assystem Germany)

LITERATURHINWEISE

[1] Forin P.: Vital Coded Microprocessor: Principles and Application for various Transit Systems. Proc. IFAC-GCCT, Paris, France, 1989

[2] Ulbrich, P. et al.: Eliminating Single Points of Failure in Software-Based Redundancy. Ninth European Dependable Computing Conference, 2012

[3] ISO 26262-1:2011, Road Vehicles – Functional safety, First Edition 2011

DIESER BEITRAG IST IM E-MAGAZIN VERFÜGBAR UNTER:
www.emag.springerprofessional.de/atz