Automatische Prüfung der Safety-Kriterien für ASIL D Basissoftware

Gewährleistung von Freedom from Interference in Automotive-Anwendungen

Ulrich Kloidt, Altium Europe GmbH
Rudolf Grave, Elektrobit (EB)

Die Integration sicherheitsrelevanter Anwendungen im AUTOSAR-Umfeld stellt eine Herausforderung dar, da nahezu immer Software mit unterschiedlichen Sicherheitszielen integriert werden muss, kombiniert mit einem Rückwirkungsfreiheitsargument. Zur Unterstützung des Integrators soll hier eine automatisierte Überprüfung vorgestellt werden, die die Entwicklung von Software im Safety-Umfeld wesentlich erleichtert. Diese Vorgehensweise führt zu einer besseren Ausnutzung der Hardware-Ressourcen und reduzierten Entwicklungszeit und damit zu geringeren Kosten.

Einführung

Um die Sicherheit elektronischer Systeme durch Standardisierung zu erhöhen, wurden verschiedene Normen entwickelt. Im Kraftfahrzeugbereich findet beispielsweise die ISO 26262 Anwendung. Diese dient der Gewährleisung der funktionalen Sicherheit eines Systems mit elektrischen/elektronischen Komponenten. Sicherheitsstandards für andere Bereiche werden in Normen wie IEC 61508, ISO 25119 oder EN 50128 definiert.

Traditionell werden sicherheitskritische Softwarefunktionen von einer separaten Einzelprozessor-Steuereinheit (ECU) verarbeitet. Diese Praxis gewährleistet eine Trennung von Funktionen mit unterschiedlichen Sicherheitsintegritätsebenen (SIL), da eine physikalische Trennung besteht und damit eine gegenseitige Beeinflussung ausgeschlossen wird.
Gegenwärtig werden mehrere dieser Einzelprozessor-Steuergeräte durch wenige Mehrkern-Steuergeräte ersetzt. Vorteile dieser Lösung sind eine Reduzierung des Verkabelungsaufwands und auch des Stromverbrauchs. Diese Maßnahme bedeutet aber die parallele Existenz von Funktionen mit unterschiedlichen Sicherheitsanforderungen in einer ECU.

Das ist nach ISO 26262 nur dann zulässig, wenn eine Rückwirkungsfreiheit (Freedom from Interference) zwischen den Softwarekomponenten, die eine unterschiedliche Sicherheitsintegritätsebene bzw. unterschiedliche Sicherheitsanforderungen haben, garantiert ist. Als Beispiel würde sowohl eine Funktion mit ASIL A Anforderungen von einer ASIL C Funktion getrennt als auch zwei ASIL D Funktionen mit inhaltlich unterschiedlichen Sicherheitsanforderungen.

Des Weiteren darf eine sicherheitsrelevante Funktion z.B. nicht durch die Basis-Software beeinflusst werden. Eine Möglichkeit, dieses zu erreichen, ist die Speicherpartitionierung (Software Partitioning), die durch den Einsatz einer Memory Protection Unit (MPU) realisiert wird. Dieser Ansatz ist aufwändig und bedingt eine vorhandene Hardware, auf der die Anwendung getestet werden kann. Effizienter ist es, schon früher in den Entwicklungsprozess einzugreifen und die Einhaltung der Speicherpartitionierung mittels einer Software zu überprüfen.

Rückwirkungsfreiheit (Freedom from Interference)
In der ISO 26262, Teil 1, Definition 1.49 wird die Rückwirkungsfreiheit beschrieben als "Nichtvorhandensein von Kaskadierungsfehlern zwischen zwei oder mehr Elementen, die zur Verletzung einer Sicherheitsanforderung führen können." Zum Beispiel sind Element 1 und Element 2 rückwirkungsfrei, wenn kein Fehler von Element 2 einen Fehler in Element 1 auslösen kann.

Es gibt drei Arten von Interferenzen:
Ausführungszeit
Hierbei muss sichergestellt sein, dass eine sicherheitsrelevante Funktion nicht durch eine andere Funktion unterbrochen wird, wobei diese Unterbrechung dann dazu führen kann, dass die sicherheitsrelevante Funktion nicht innerhalb des vorgegebenen Zeitrahmens bzw. Zeitintervalls ausgeführt werden kann. Dieses wird üblicherweise mittels Hardware- und Software-Watchdogs, die Kontrollfluss-Überwachung und Deadline-Überwachung unterstützen, gewährleistet. Das sind meist Bestandteile eines Echtzeitbetriebssystems.

Speicher
Eine Speicherinterferenz tritt auf, wenn Programmcode oder Programmdaten einer sicherheitsrelevanten Funktion durch andere Funktionen oder externe Fehler modifiziert werden. Speicherzugriffsverletzungen können durch eine Speicherschutzeinheit (MPU) erkannt werden.

Informationsaustausch
Störungen beim Informationsaustausch treten auf, wenn Nachrichten wiederholt werden oder verlorengehen bzw. deren Inhalt verfälscht wird.

Die hier vorgestellte Safety Checker Software hilft, Speicherinterferenzen zu erkennen. Das reduziert sowohl den Entwicklungsaufwand als auch den Bedarf einer Speicherschutzeinheit bei niedrigen Sicherheitsstufen:

- MPUs sind komplex zu konfigurieren. Nach Aktivierung der MPU treten häufig Fehler auf, die auf Flüchtigkeitsfehler beim Codieren oder Konfigurieren zurückzuführen sind. Da jeder einzelne Fehler analysiert werden muss, ist dies eine zeitraubende Aufgabe.
- Es ist sehr aufwändig, eine vollständige Codeabdeckung zu erreichen, um alle möglichen Speicherzugriffsverletzungen zu erkennen [3].
- Fehler, die bei der Konfiguration der MPU gemacht wurden, treten unter Umständen erst im Feld auf, also nach Auslieferung der Software [1,2].
- Der späte Einsatz einer MPU und das aufwändige MPU-Testen verzögert die Fehlererkennung und erhöht die Entwicklungskosten [1,4].

Statische Analyse des Quellcodes als Alternative zur MPU
Um Speicherinterferenzen zu erkennen, kann eine statische Analyse des Quellcodes mittels einer Software durchgeführt werden. Der Code wird dabei nicht auf einer Zielhardware ausgeführt. Das erlaubt eine schnelle Interferenzprüfung nach Quellcode-Änderungen. Die Analyse beinhaltet eine Kontrolle aller möglichen Programmabläufe, wodurch eine hohe Codeabdeckung gewährt wird. Dedizierte

Testfälle werden daher nicht benötigt. C++ Code sowie (Inline-) Assembler Code werden nicht überprüft.
Nachteil dieses Ansatzes ist, dass der Quellcode aller Dateien vorliegen muss, um die Analyse durchzuführen. Das ist zum Schutze des geistigen Eigentums kein gangbarer Weg. Eine Lösung dieses Problems ist, die Analysedaten der einzelnen Dateien in einem verschlüsselten Format abzulegen. Diese Daten werden dann bei der Integration der Software benutzt.

Differenzierung innerhalb einer Sicherheitsintegritätsstufe
Der Hardwareansatz über die MPU erlaubt keine Differenzierung innerhalb einer SIL-Stufe. Der Softwareansatz bietet diese Möglichkeit. Es können beliebig viele Stufen definiert werden, also auch mehrere für eine SIL-Stufe:

```
__SAFETY_CLASS_ATTRIBUTES__
{
        /* Klassenname */
        { 0, "QM" },
        { 1, "ASIL A" },
        { 2, "ASIL B_1" },
        { 3, "ASIL B_2" },
        { 4, "ASIL C" }
        { 5, "ASIL D" }
        { 6, „COM_Buf_B1toA"}
};
```

Anwendungsbeispiel: MCAL-Treiber. Diese dürfen auf Konfigurationsdaten sowohl lesend als auch schreibend zugreifen. Andere ASIL B Funktionen dürfen diese Konfigurationsdaten jedoch nur lesen.

```
__SAFETY_CLASS_ACCESS_RIGHTS__
{
/* Quelle Ziel Zugriffsrechte */
{ ASIL_B_1, COM_Buf_B1toA, W },/* ASIL B_1 Funktion darf in den
Kommunikationspuffer zur ASIL A Partition schreiben */
{ ASIL_B_1, ASIL_A, R },     /* ASIL B_1 Funktion darf ASIL A Daten
                                lesen */
{ ASIL_A, ASIL_B_2, X },     /* ASIL A Funktion darf ASIL B_2
                                Funktion aufrufen */
{ ASIL_B_1, ASIL_B_2, R },   /* ASIL B_1 Funktion darf ASIL B_2
                                Daten lesen */
{ ASIL_B_2, ASIL_B_1, R|W|X }/* ASIL B_2 Funktion darf ASIL B_1
Funktionen aufrufen und hat Vollzugriff auf  ASIL B_1 Daten */
};
```

MCAL-Funktionen und Konfigurationsdaten werden der Klasse ASIL_B_2 zugewiesen, die restlichen ASIL_B Funktionen und Daten der Klasse ASIL_B_1.
Die Zuweisung der Funktionen und Variablen zu den zuvor definierten Sicherheitsintegritätsstufen geschieht mittels einer Tabelle:

```
__SAFETY_CLASS_SELECTIONS__
{
/* Dateiname Name SIL */
/* Alle Funktionen und Variablen aus Datei 'file_1.c' beginnend mit
'f' werden ASIL_B_1 zugewiesen */
```

```
{ "file_1.c", "f*", ASIL_B_1 },
/* Die Variable mit dem Namen 'x' wird ASIL_B_2 zugewiesen */
{ "file_1.c", "x", ASIL_B_2 },
/* Alle Funktionen und Variablen aus Datei 'file_2.c' werden ASIL_A
zugewiesen, 'file_2.c' beinhaltet Funktion 'test' */
{ "file_2.c", "*", ASIL_A }
};
```

Soll ein Adressbereich statt einer Variablen oder Funktion einer SIL-Stufe zugewiesen werden, so wird eine Startadresse, Größe und die gewünschte Sicherheitsstufe des Bereichs angegeben.

Nachdem die statische Analyse beendet wurde, werden Verletzungen der vorgegebenen Zugriffsregeln in einer Diagnosedatei angezeigt:

csaf E497: ["file_1.c" 12] safety violation writing "x" (ASIL B_2) from "f2" (ASIL B_1)
csaf E499: ["file_2.c" 13] safety violation calling "f1" (ASIL B_1) from "test" (ASIL A)

Zur Dokumentation liegt ein Bericht vor, dessen inhaltlicher Detailgrad skalierbar ist. Dieser Bericht besteht aus

- einer Tabelle, in der die Zuweisung der Funktionen und Variablen zu den SIL Klassen aufgeführt ist,
- einer SIL-Klassen-Zugriffsmatrix, die die Interaktionsmöglichkeiten einzelner SIL-Stufen zeigt,
- einem Aufrufdiagramm, das alle Interaktionen der Funktionen und die seitens der Software durchgeführten Prüfungen aufzeigt, sowie
- einer Zugriffstabelle, die zeigt, welche Art von Zugriff auf jede einzelne Funktion und Variable erfolgt.

Praxisanwendung
Zwei Anwendungsszenarien sollen im folgenden Abschnitt dargestellt werden:

- Rückwirkungsfreiheit ohne hardwareunterstützen Speicherschutz für niedrige Sicherheitslevel
- Effiziente Integrationsunterstützung bei Steuergeräten mit Speicherschutz

Ein typisches Szenario ist, Softwareelemente mit niedrigen Sicherheitsanforderungen zusammen mit QM-Software zu integrieren, z.B. um Performance-Einbußen bei einem Partitionswechsel, z.B. durch das Umschalten oder Reprogrammieren der MPU, zu sparen, Legacy Code wiederzuverwenden oder sich eine Speicherschutzeinheit auf dem Prozessor zu ersparen. Wie einleitend angeführt muss nun eine Argumentation für die Rückwirkungsfreiheit geliefert werden. Dies kann durch eine „virtuelle“ Partitionierung im Safety Checker geschehen, indem in der Konfiguration unterschiedliche Partitionen für die zu integrierenden Software Elemente angelegt werden. Dabei bekommt der Safety Checker einen hohen Tool Confidence Level (TCL), da er direkt für die Sicherheitsargumentation benutzt wird.

Ein weiterer Anwendungsfall findet sich bei Systemen mit Speicherschutz. In den frühen Projektphasen ist häufig der Speicherschutz deaktiviert, da zeitnah lauffähige

Funktionen gezeigt werden müssen und am Brettaufbau keine Gefahr für Leib und Leben besteht. In dieser Phase achtet der Entwickler meist nicht auf eine saubere Partitionierung, da diese ja nicht kontrolliert wird. Soll in einer späteren Projektphase der Speicherschutz aktiviert werden, kann dies bis zur vier Wochen Entwicklungszeit kosten, da jeder falsche Speicherschutzzugriff zu einer Exception führt, die dann einzeln analysiert und korrigiert werden muss. Der Einsatz des Safety Checkers insbesondere in frühen Projektphasen kann hier die Entwicklungszeit drastisch reduzieren und die Anwendung auf den Einsatz des Speicherschutzes vorbereiten. Da die Safety-Argumentation in diesem Szenario auf dem verfügbaren Speicherschutz beruht, wird der TCL meist auf den Level 1 gesetzt und erfordert somit keine weiteren Aktivitäten.

Integration des Safety Checkers

Die Integration des Safety Checkers in einer Make-Umgebung ist relativ einfach und kann vom AUTOSAR-Zulieferer vorbereitet werden. Der Aufruf des Safety Checkers erfolgt analog zu dem eines Compilers.

Die Erstellung des Konfigurationsfiles besitzt eine ähnliche Komplexität wie die eines Linkerscripts, allerdings lassen sich bei der Verwendung einer Systembeschreibung, wie sie z.B. bei AUTOSAR vorhanden ist, große Teile generieren. So können z.B. die im Systemmodel hinterlegten Partitionen (OS Application) und das SW-C/ Basic Software Mapping in der AUTOSAR RTE verwendet werden um die `SAFETY_CLASS_ATTRIBUTES` und die `SAFETY_CLASS_SELECTIONS` zu generieren.

Schlussfolgerung

Der Einsatz einer statischen Analysesoftware, um Speicherinterferenzen zu lokalisieren, reduziert den Entwicklungs- und Hardware-Testaufwand mittels einer MPU, da viele Zugriffsverletzungen bereits frühzeitig in der Entwicklungsphase erkannt und behoben werden können. Durch einen hohen Integrationsgrad mit Betriebssystemen wie AUTOSAR und dem dazugehörigen Tooling lässt sich der zusätzliche Aufwand bedingt durch den Einsatz des Safety Checkers sehr stark reduzieren, da benötigte Informationen bereits hinterlegt sind. Die Wunschvorstellung eines Integrators ist der Knopf oder das make-target „make safetycheck“.

Literaturverzeichnis

[1] Boehm, B. W. Understanding and controlling software costs. Journal of Parametrics 8, 32–68 (1988).

[2] http://www.popsci.com/software-rising-cause-car-recalls

[3] Software debugging, testing and verifcation by Hailpern and Santhanam, 2002 (http://www.cs.uleth.ca/~benkoczi/3720/pres/debug-test-verify_hailpern02.pdf)

[4] Tassey, G. The economic impacts of inadequate infrastructure for software testing. National Institute of Standards and Technology, RTI Project 7007, (2002).