Methode zur applikationsspezifischen Absicherung der Basissoftware von AUTOSAR Steuergeräten

Norbert Englisch, Toni Helfrich, Wolfram Hardt

Fakultät für Informatik
Technische Universität Chemnitz
Straße der Nationen 62
09111 Chemnitz
norbert.englisch@informatik.tu.chemnitz.de
toni.helfrich@informatik.tu-chemnitz.de
hardt@cs.tu-chemnitz.de

Abstract: Dieses Paper stellt eine Methode zur Absicherung der Basissoftware von AUTOSAR Steuergeräten vor. Dabei wird die Basissoftware auf die Funktionalität getestet, die sie für die Kundenfunktionalität, also die Applikationsschicht, zur Verfügung stellen muss. Der Test wird im Steuergerät durch ein Complex-Device-Driver-Modul durchgeführt. Ein externer Tester steuert automatisiert das Complex-Device-Driver-Modul, indem er eine generische Testbotschaft über einen angeschlossenen Bus sendet. Eine erste automatisierte Implementierung wurde auf Basis des CAN-Kommunikation-Stack entwickelt. Das Konzept kann alle Basissoftwaremodule auf ihre Aufgaben testen und Fehler in der Konfiguration und in der Programmierung der Basissoftware aufdecken.

1 Motivation

Ein AUTOSAR konformes Steuergerät unterteilt sich in die drei Schichten Basissoftware, Runtime Environment und Applikationsschicht [Au11a]. Durch diese funktionale Trennung und der damit verbundenen Teilung von Aufgaben in der Entwicklung – auf verschiedene Firmen oder Entwicklerteams – stellen sich der Absicherung eines Steuergerätes als Ganzes neue Herausforderungen. So kann bei einem Blackbox Test eines Steuergerätes nur schwer ein Rückschluss auf eine Fehlerursache gezogen werden. Neben dieser Tatsache birgt auch die Konfiguration von Modulen neue Fehlerquellen. Durch die Konfigurationsmöglichkeiten können neben Programmierfehlern auch Konfigurationsfehler auftreten. Um Fehlerquellen zu erkennen oder zu lokalisieren, reicht ein reiner Blackbox Test eines AUTOSAR Steuergerätes selten aus. Vielmehr müssen die Schichten des Steuergerätes auch separat betrachtet und getestet werden. Das hier vorgestellte Konzept betrachtet zunächst die Basissoftware eines AUTOSAR Steuergerätes, welche die Basis für die Kundenfunktionalität liefert.

erschienen im Tagungsband der INFORMATIK 2011
Lecture Notes in Informatics, Band P192
ISBN 978-3-88579-286-4
weitere Artikel online:
http://informatik2011.de/519.html

2 Stand der Technik

In der Praxis richtet sich der Fokus des Tests von AUTOSAR Steuergeräten auf den Bereich der Applikationsschicht, da in dieser die eigentliche Kundenfunktionalität implementiert ist. Die Funktionalität der Basissoftware wird dabei meist vorausgesetzt oder nur in den Modulen überprüft, zu denen Client Server Verbindungen aus der Applikationsschicht bestehen [We11]. Die Runtime Environment bzw. deren Generatoren werden meist auf Konformität zur AUTOSAR Spezifikation überprüft, jedoch nicht auf Applikationskonformität. Nur wenn alle drei Schichten eines AUTOSAR Steuergerätes richtig funktionieren, wird die erwünschte Funktionalität eines sichergestellt.

Im Bereich der Basissoftware beschränkt sich der aktuelle Forschungsstand auf den Blackbox Test in den verschiedenen ICC Klassen (nach [Au11b]). Die Entwickler der Basissoftwaremodule arbeiten zusätzlich mit Whitebox und Greybox Test. Ein Ansatz für einen applikationsspezifischen Test, der die finale Konfiguration beinhaltet und abtestet, wurde in [En09] beschrieben. Allerdings hat diese Methode entscheidende Nachteile. So wird eine Softwarekomponente erzeugt, die die Basissoftware aus der Applikationsschicht heraus testet. Demzufolge muss die RTE des Steuergerätes neu erzeugt werden. Zudem wurde vorausgesetzt, dass es keine weiteren Softwarekomponenten in der Applikationsschicht gibt, außer der Testkomponente.

3 Konzept

Da die Basissoftware den Grundstein für ein AUTOSAR konformes Steuergerät legt, gilt ihr im Bereich der Absicherung von Funktionalitäten ein besonderes Augenmerk. Aus Kunden- und OEM-Sicht ist eine korrekte und zuverlässige Kundenfunktionalität wichtig. Aus diesem Grund befasst sich dieses Konzept mit einer applikationsspezifischen Absicherung der Basissoftware – also die korrekte Funktionalität der Basissoftware in Bezug auf die Applikationen.

Um die Basissoftware applikationsspezifisch zu untersuchen ist es nicht ausreichend nur das Interface auf Korrektheit zu überprüfen. Neben der Konfiguration ist die Ausführung und Stimulation von applikationsspezifischen Werten und die folgende Reaktion der Basissoftwaremodule wichtig. Da die Basissoftware allein nicht ohne Gegenstelle im Steuergerät von außerhalb getestet werden kann, zeigt dieses Konzept eine Lösung, die mit Hilfe eines Complex-Device-Driver-Moduls den Test realisiert.

Das Konzept ist für alle bisher veröffentlichten AUTOSAR Versionen entworfen und umsetzbar.

ISBN 978-3-88579-286-4

3.1 Absicherung der Basissoftware

Die Basissoftware besteht aus einer Vielzahl von Modulen, die alle eine bestimmte Aufgabe übernehmen und wovon nicht immer alle benötigt werden. Die verwendeten Module müssen aber, je nach verwendeter Hardware oder gewünschten Funktionen, unterschiedlich konfiguriert werden [KF09]. Dies ermöglicht sehr viele unterschiedliche Konfigurationen und macht eine Fehlersuche sehr schwer. Dieses Konzept soll Mittel und Wege finden, um die Fehlersuche zu beschleunigen und zu vereinfachen.

Die Absicherung der Basissoftware soll dadurch gewährleistet werden, dass jedes Modul auf korrekte Funktionalität geprüft wird, also genau die Funktionalität, die es für die Applikationsschicht zur Verfügung stellen muss. Der Test soll im Steuergerät durchgeführt werden, indem ein Complex-Device-Driver-Modul (Testmodul) in das Steuergerät integriert wird und das zu testende Basissoftwaremodul durch Stimulation und Auswertung der darauf folgenden Reaktion überprüft (siehe Abbildung 1). Das Testmodul wird durch einen externen Tester gesteuert. Dieser definiert die Art und den Testzeitpunkt und wartet auf ein Testergebnis.

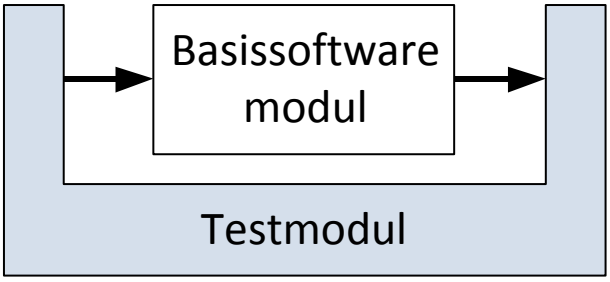

Abbildung 1: Basissoftwaremodul als System under Test

Der durchzuführende Test soll automatisiert erstellt werden. Dies kann durch eine automatisierte Analyse der Konfigurationsdateien der Basissoftwaremodule geschehen. Durch eine Analyse der Schnittstellen der Basissoftwaremodule werden die für die Applikation notwendigen Funktionalitäten analysiert und entsprechende Testfälle im Testmodul definiert.

In Abbildung 2 ist der Workflow des Konzeptes zu sehen. Grundlage für die Kommunikation zwischen dem Testmodul und dem eigentlichen Tester ist eine generische Testbotschaft, mit deren Hilfe Steuerbotschaften und Testergebnisse versendet werden können. Zur Vereinfachung wird vorausgesetzt, dass diese generische Testbotschaft während der Entwicklungsphase eines Steuergerätes bereits in der Netzwerkkommunikation vordefiniert ist. Im zweiten Schritt können die Konfigurationsdateien der Basissoftware analysiert werden und an Hand dieser die applikationsspezifischen Testfälle generiert werden. Diese generierten Testfälle werden dann durch einen Konverter in C-Code konvertiert und als Testmodul in die Basissoftware integriert. Im Folgenden werden auf Basis der generierten Testfälle des Testmoduls, synchronisierte Steuersignale für den Tester erzeugt.

ISBN 978-3-88579-286-4

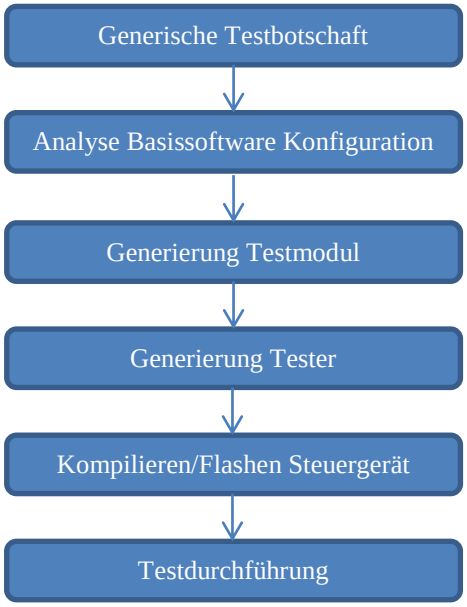

Abbildung 2: Workflow für die Erzeugung von Testfällen der applikationsspezifischen Absicherung der Basissoftware

3.2 Funktionalität des Testfallgenerator

Der Testfallgenerator hat die Aufgabe Testfälle für das Testmodul, also das Complex-Device-Driver-Modul, zu erzeugen und die Steuersignale des Testers zu erzeugen. Der Testfallgenerator arbeitet automatisiert – er analysiert die Konfiguration der Basissoftware, also alle modulspezifischen Konfigurationsdateien. An Hand der Konfiguration werden Testfälle erzeugt, da die Konfigurationsdateien die Schnittstellen der Basissoftwaremodule beschreiben und damit das erwünschte und applikationsspezifische Verhalten. Zusätzlich zu der automatisierten Erzeugung ist auch das manuelle Definieren von Testfällen vorgesehen.

Auf Basis der erzeugten Testfälle für die Basissoftware werden darauffolgend die Testfälle für den externen Tester erzeugt. Das notwendige Minimum an Testfällen ist ein Steuersignal für den Start eines Tests und ein Steuersignal, welches die Ergebnisse des durchgeführten Tests enthält. Die konkrete Auswertung der Testergebnisse kann dann auf Testerseite ausgeführt werden. Die systeminternen Testfälle können für verschiedene Testdurchführungswerkzeuge erzeugt werden, was durch einen Testkonverter erreicht wird (siehe Abbildung 3).

ISBN 978-3-88579-286-4

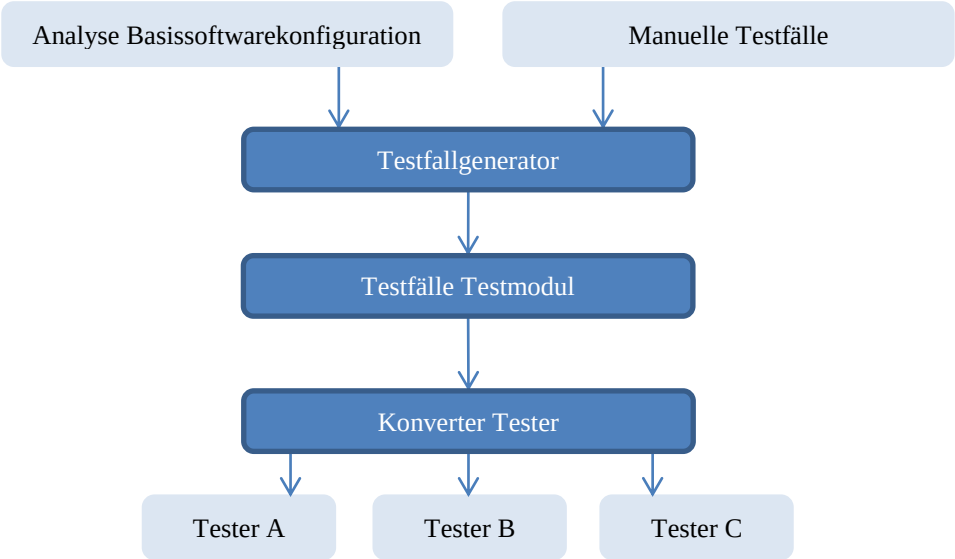

Abbildung 3: Funktionsweise Testfallgenerator

3.3 Schwerpunktsetzung des Konzeptes

In diesem Konzept wird der Schwerpunkt auf die Absicherung der Basissoftware gelegt, die für konkrete Applikation konfiguriert und kompiliert wurde. Auf die Absicherung der Applikationsschicht wird nicht näher eingegangen, denn dafür existieren schon diverse kommerzielle Programme, die den Entwickler bei der Fehlersuche unterstützen.

Die RTE wird nicht betrachtet. Da das Konzept nur innerhalb der Basissoftware arbeitet, ist eine Kommunikation mit höheren Schichten nicht vorgesehen bzw. nötig. Ein wichtiger Punkt der applikationsspezifischen Absicherung ist, dass diese ohne Neugenerierung der RTE auskommt. Ein Test der RTE ist allerdings möglich, beispielsweise durch einen Parser, der das Modell und den erzeugten RTE-Code gegeneinander prüft, ob alle Verbindungen korrekt erzeugt wurden [He11].

Das Absichern und Testen von bereits vorhanden Complex-Device-Driver-Modulen kann in diesem Konzept nicht pauschal betrachtet werden, da diese keine vorgeschriebenen Schnittstellen haben müssen. Für den Test solcher Module wurde im Konzept das Erstellen manueller Testfälle im Testfallgenerator vorgesehen.

4 Realisierung

Nachfolgend werden die grundlegenden Ansätze erklärt, die für die Umsetzung des Konzeptes verwendet werden. Das Testmodul ist so realisiert, dass keine Änderungen an der vorgegebenen Konfiguration der Basissoftwaremodule benötigt werden und somit im normalen Betrieb getestet werden kann.

4.1 Die Umsetzung des Konzeptes am Beispiel des Kommunikations-Stacks

Für die sukzessive Umsetzung des Konzeptes, wurde das vorgestellte Konzept am Beispiel des Kommunikations-Stacks grundlegend implementiert. Dadurch beziehen sich die erzeugten Testfälle auf die Kommunikation. Die wichtigste Aufgabe, die ein Kommunikations-Stack zu erfüllen hat, ist das Empfangen bzw. Senden von Signalwerten zur Applikationsschicht bzw. den physischen Bus. Darum wird die Umsetzung des Konzeptes auf diese Grundfunktionalitäten abgebildet. Wie in Abbildung 4 dargestellt ist, sind die in AUTOSAR spezifizierten Kommunikations-Stacks ähnlich aufgebaut. Darum wurde der CAN-Kommunikations-Stack (CAN-Stack) repräsentativ für den kompletten Kommunikations-Stack ausgewählt.

Die Minimalkonfiguration des CAN-Stack benötigt die vier Basissoftwaremodule CAN-Driver in der MCAL-Schicht, CAN-Interface in der EAL-Schicht, PDU-Router und COM-Modul in der Services-Schicht. Alle Module bieten durch ihre API verschiedene Kontrollmechanismen an, mit deren Hilfe das Complex-Device-Driver-Testmodul die Funktionalität des jeweiligen Basissoftwaremoduls testen kann.

Die Kommunikation der CAN-Stack-Module untereinander ist fest vorgegeben und bietet keine Möglichkeiten den Datenaustausch zu überwachen, daher ist ein Lesen der zu sendenden Daten nur auf dem CAN-Bus und das Auslesen der empfangenen Daten nur im obersten Basissoftwaremodul des Stacks möglich, dem COM-Modul.

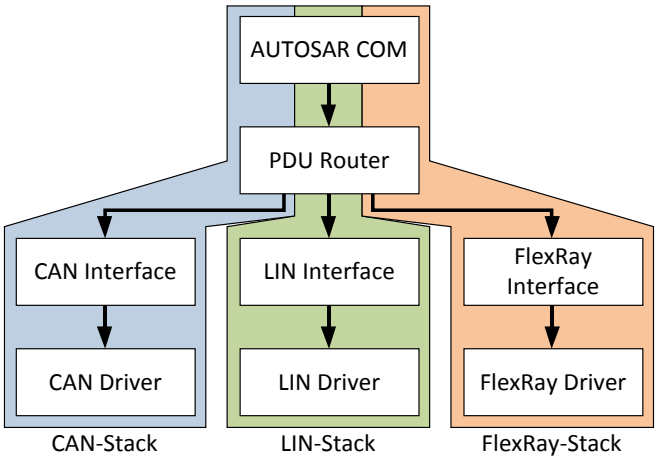

Abbildung 4: Einteilung des Kommunikations-Stack in Bussysteme

4.1.1 Senden von CAN-Nachrichten

Jedes der beteiligten CAN-Stack-Basissoftwaremodule verfügt über eine Senden-API-Funktion, die je nach Modul, unterschiedliche Parameter benötigt. Die Parameter legen fest wie die Daten im CAN-Stack weitergegeben werden und sind in der Basissoftwarekonfiguration des Steuergerätes festgelegt. Der Stack-Aufbau erfordert ein schrittweises Vorgehen beim Test der Module. Da ein direktes Auslesen der Daten aus einem Modul nicht möglich ist, können die zu sendenden bzw. gesendeten Daten nur direkt auf dem CAN-Bus gelesen werden. Das CAN-Driver-Modul ist der Zugang zum CAN-Bus und wird von allen anderen Modulen des CAN-Stack verwendet und muss daher zuerst getestet werden. Daraus ergibt sich eine Bottom-Up-Teststrategie, da die Module im Stack sukzessive von unten nach oben getestet werden. Wurde ein Modul fehlerfrei überprüft kann es als mögliche Fehlerquelle ausgeschlossen werden und das nächste Basissoftwaremodul kann getestet werden. Wie in Abbildung 5 dargestellt ist, wird zuerst das CAN-Driver-Modul getestet, indem eine generische Testbotschaft mittels der Senden-API-Funktion über den CAN-Bus versendet wird. Kommt keine oder eine fehlerhafte Nachricht beim Tester an, kann mit der Fehlersuche in dem CAN-Driver-Modul begonnen werden. Nachdem der Test erfolgreich war, wird das CAN-Interface-Modul getestet, es wird wiederum eine generische Testbotschaft versendet. Das CAN-Interface-Modul übergibt die Daten einem CAN-Driver-Modul, das die Daten dann versendet. Kann eine CAN-Nachricht nicht versendet werden, ist somit sichergestellt, dass sich der Fehler nur in dem aktuell betrachteten Modul befindet, da alle vorhergehenden fehlerfrei funktioniert haben. Dieses Testvorgehen setzt sich über das PDU-Router-Modul bis zum COM-Modul fort. Kann die Funktionalität für das COM-Modul erfolgreich gezeigt werden ist der Test für das Senden abgeschlossen.

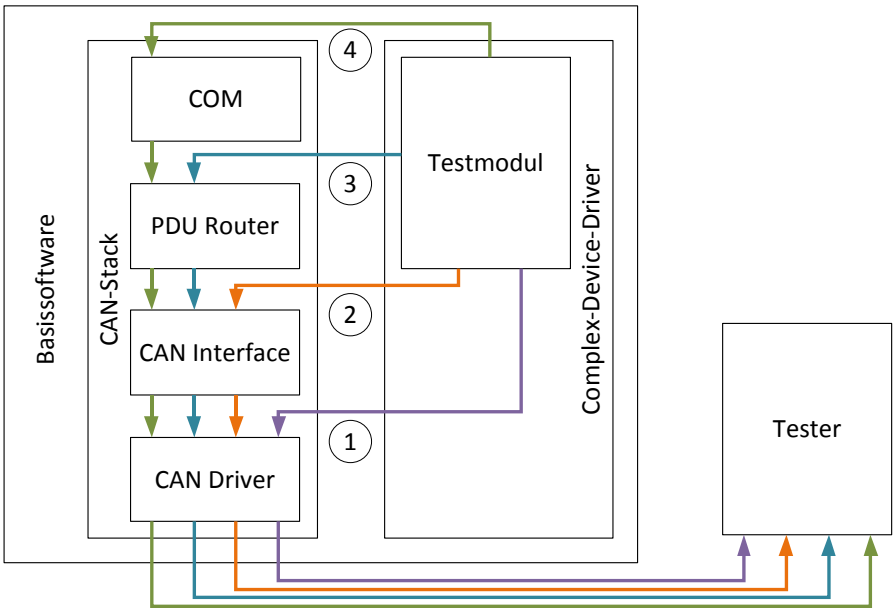

Abbildung 5: Testen der CAN-Stack-Module für das Senden

4.1.2 Empfangen von CAN-Nachrichten

Das Empfangen einer CAN-Botschaft ist über Callback-Funktionen realisiert, hierbei werden die empfangenen Daten von einem Modul an das unmittelbar darüber liegende direkt übergeben. Daher ist es nicht möglich die Daten einzusehen ohne in die Module selbst einzugreifen. Das COM-Modul bietet als einziges Modul des CAN-Stack eine API-Funktion mit dessen Hilfe empfangene Daten jederzeit gelesen werden können, ohne dass Änderungen an der Konfiguration nötig sind. Für das Empfangen wird daher ein Top-Down-Testansatz verwendet, bei dem zuerst das COM-Modul getestet werden muss. Hierbei werden die für die Applikation relevanten Signale über die Callback-Funktion an das Modul übertragen und mit der Lesen-API-Funktion empfangen. Werden die Daten gar nicht oder fehlerhaft übertragen liegt ein Fehler vor, der vor der Weiterführung des Tests behoben werden muss. Anschließend werden sukzessive die darunterliegenden Module des Stacks überprüft, dies ist in Abbildung 6 dargestellt. Mit diesem Ansatz ist es möglich im Fehlerfall die Fehlersuche auf ein bestimmtes Modul oder Funktion einzugrenzen. Da alle vorherigen Module bereits getestet wurden, kann der Fehler nur im aktuell geprüften Modul liegen. Ist der Test erfolgreich wird mit dem nächsten Modul im CAN-Stack fortgefahren.

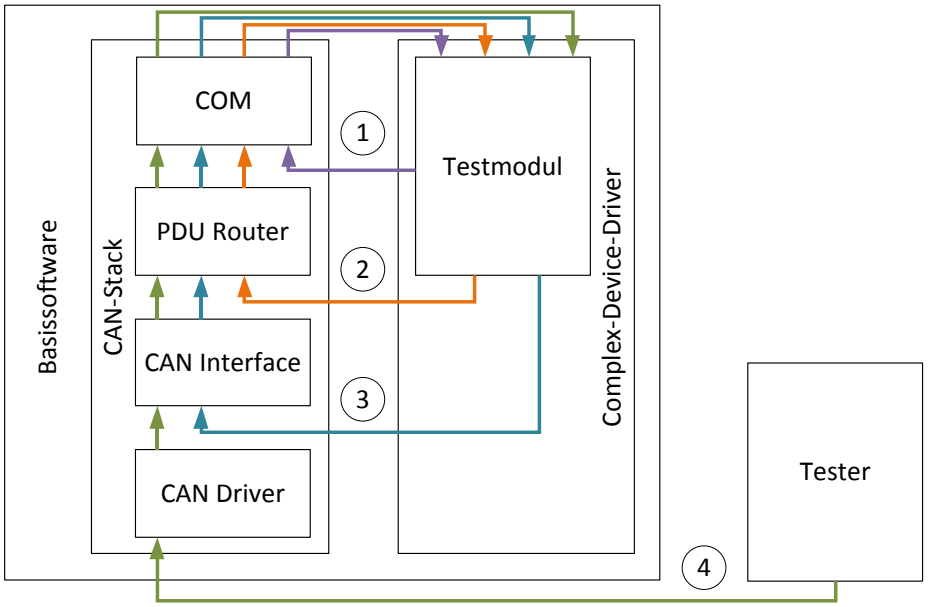

Abbildung 6: Testen der CAN-Stack-Module für das Empfangen

ISBN 978-3-88579-286-4

4.1.3 Versionsinformationen

Ein Ziel des vorgestellten Konzeptes ist es, dass von möglichst vielen Basissoftwaremodulen Gemeinsamkeiten gefunden werden, für die Testfälle erstellt werden können. Das heißt, dass Gemeinsamkeiten in den Schnittstellen gefunden werden sollen. Diese Übereinstimmung, über die fast alle Basissoftwaremodule verfügen, ist die Versionsinformations-API. Mit diesen Funktionen kann die Version eines Moduls ermittelt werden, um die Kompatibilität der Module untereinander prüfen zu können. Diese Schnittstelle wird dazu verwendet um festzustellen ob ein bestimmtes Modul korrekt in die Basissoftware integriert wurde, sofern diese API-Funktion in der Modulkonfiguration aktiviert ist.

4.2 Werkzeugkette

Die vorgestellte Umsetzung wurde auf der AUTOSAR Version 2.1 realisiert. Das Konzept kann auch auf den neueren Versionen 3.1 oder 4.0 umgesetzt werden.

Ziel des vorgestellten Konzeptes ist es, einen hohen Automatisierungsgrad in der Absicherung der Basissoftware zu erreichen. Dies war ein wichtiger Fakt bei der Wahl der Werkzeuge zur Realisierung des Konzeptes (siehe Abbildung 7). Weiterhin war eine mögliche Austauschbarkeit der Werkzeuge ein wichtiger Punkt.

Durch eine Marktrecherche wurde sich für das Werkzeug **SystemDesk** von der Firma dSpace, für die Erstellung von Softwarekomponenten bzw. Complex-Device-Driver-Module entschieden. Für die Konfiguration der Basissoftware wird das **tresos Studio** von Elektrobit genutzt. Das tresos Studio bietet keine Möglichkeit für eine Automatisierung. Allerdings werden durch das implementierte Konzept die notwendigen Umstellungen für den Test der Basissoftware auf ein Minimum reduziert. Die erstellte und flashbare Datei wird mit dem **CodeWarrior** von Freescale auf die Zielhardware geflasht. Der für das Konzept notwendige Tester wird durch das Werkzeug **ECU-TEST** von der Firma TraceTronic abgedeckt.

Um die Basissoftware abzusichern wir im SystemDesk automatisiert eine Softwarekomponente erstellt, die keinen Kontakt zur RTE hat. Die Softwarekomponente wird einem Task zugewiesen [Ds08]. Das tresos Studio wird zur Integration der Konfigurationsdateien für die Basissoftwaremodule und das Erstellen eines elf-Files benutzt.

ISBN 978-3-88579-286-4

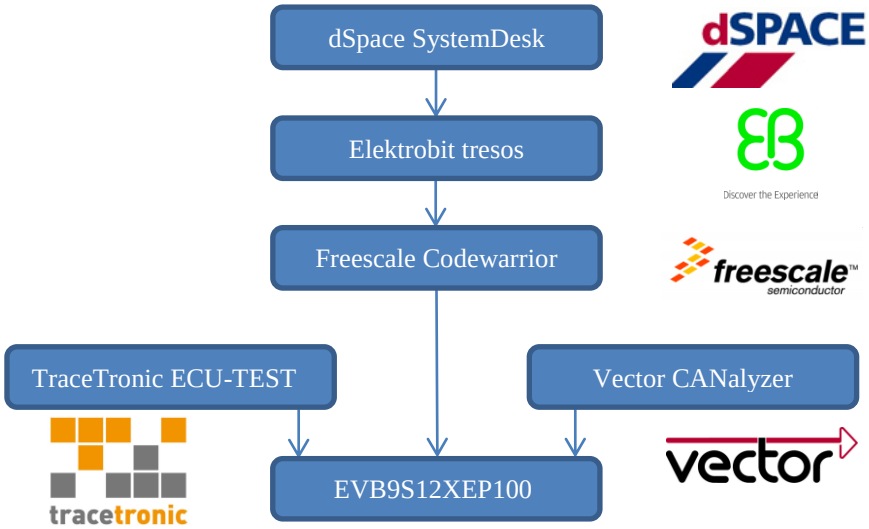

Abbildung 7: Werkzeugkette für die Realisierung des Konzeptes (AUTOSAR 2.1)

5 Ergebnisse

Die Basissoftwarekonfiguration liegt in XML-Form vor und kann somit schnell und einfach mit einem XML-Parser analysiert werden. Die benötigten Daten, wie PDU-ID, Signal-ID, CAN-Controller-Hardware-Handle, CAN-ID, CAN-DLC und Baudrate werden extrahiert und dem Testfallgenerator übergeben. Der Testfallgenerator erzeugt aus den gesammelten Daten die Testfälle und daraus die Konfiguration für den Tester und den Quellcode für das Testmodul, um die gewünschten Funktionen der Basissoftwaremodule zu testen.

Um Daten aus dem COM-Modul auslesen und mittels den Callback-Funktionen senden zu können, werden die Signal-ID und PDU-ID der CAN-Botschaften aus der Konfiguration der Basissoftware benötigt.

Mit Hilfe der Signal-ID kann ein bestimmtes Signal (recvSignalId) durch die Funktion Com_ReceiveSignal() gelesen werden. Die Daten werden der Konfiguration unter Com→ComSignal entnommen.

```
// Read Data from AUTOSAR COM Module
recvReturn = Com_ReceiveSignal(recvSignalId, &recvData);
```

Die Signal-ID (sendSignalId) wird ebenfalls dafür verwendet um im COM-Modul den Empfang eines Signals simulieren (dies entspricht Schritt 1 in Abbildung 6). Die Signal-IDs für das Senden und Empfangen sind nicht voneinander abhängig.

```
// AUTOSAR Com Module RxIndication
Com_RxIndication(sendSignalId, &sendData);
```

Um den Empfang einer Botschaft im PDU-Router zu simulieren, wird die PDU-ID (sendPduId) benötigt, die ebenfalls der Konfiguration der Basissoftware entnommen wird (Schritt 2 in Abbildung 6) unter PduR→Routing Table→PduRRoutingPath→Source PDU ID.

```
// Pdu Router Module RxIndication
PduR_CanIfRxIndication(sendPduId, &sendData);
```

Die Callback-Funktion des CAN-Interface-Moduls erfordert die CAN-ID, Länge der CAN-Botschaft (DLC) und die ID des CAN-Controllers (HRH), der für den Empfang der Nachricht verantwortlich ist (Schritt 3 in Abbildung 6). Diese Daten werden der Konfiguration entnommen: CanIf→CanIfDriverConfig→CanIfHrhConfig enthält dabei die ID des CAN-Controllers (canHrh) und Can→CanIfInitConfig→CanIfRxPduConfig die ID-, ID-Typ- und DLC-Informationen der CAN-Botschaft (canId und canDlc). Der ID-Typ wird für den Tester im nächsten Schritt benötigt.

```
// Can Interface Module RxIndication
CanIf_RxIndication(canHrh, canId, canDlc, &sendData);
```

Im letzten Schritt versendet der Tester eine CAN-Botschaft mit der definierten CAN-ID, ID-Typ, DLC und den Testdaten an das Steuergerät (Schritt 4 in Abbildung 6). Die benötigte Baudrate wird der Konfiguration des CAN-Moduls unter Can→CanController→CanControllerBaudRate entnommen.

Die Konfiguration und Ausführung des Tests der Senderichtung erfolgt analog.

Die Ergebnisse der Testvorgänge werden mit einer generischen CAN-Testbotschaft an den Tester übermittelt. Die Testbotschaft kann direkt über den CAN-Driver versendet werden und setzt daher eine korrekte Konfiguration dieses Moduls voraus (wird im ersten Testschritt des CAN-Stack getestet). Exemplarisch kann die Testbotschaft die Nummer des getesteten Moduls, einen Fehlercode und zusätzliche Informationen beinhalten die eine Fehlersuche vereinfachen.

Abschließend wertet der Tester die Botschaften aus und liefert im Fehlerfall genaue Aussagen darüber, welche Basissoftwaremodule fehlerhaft sind.

ISBN 978-3-88579-286-4

6 Zusammenfassung und Ausblick

Das vorgestellte Konzept zeigt eine Möglichkeit die Basissoftware eines AUTOSAR Steuergerätes in Bezug auf die Applikationsschicht abzusichern. Dafür wird automatisiert ein Complex-Device-Driver-Modul auf Basis der Konfigurationsdateien, also den Schnittstellendefinitionen, erstellt und in die Basissoftware hinein kompiliert. Mit Hilfe dieses Konzeptes lassen sich sämtliche Module der Basissoftware abtesten. An der Technischen Universität Chemnitz wurde das Konzept exemplarisch am CAN-Kommunikations-Stack realisiert. Durch diese Unterstützung in Test und Absicherung konnten verschiedene Migrationsprojekte im Bereich Forschung und Lehre beschleunigt und zum Ziel geführt werden.

Die bereits bestehende Realisierung soll nun um weitere Module aus der Basissoftware erweitert werden. Damit soll mittelfristig eine Absicherung der kompletten Basissoftware erreicht werden. Die Implementierung beschränkt sich aktuell auf die Basissoftware Version 2.1. Das Konzept ist auf alle bisher veröffentlichten AUTOSAR Versionen anwendbar. Die Migration der bereits bestehenden Implementierung auf AUTOSAR 3.1 ist ebenfalls vorgesehen.

Literaturverzeichnis

[Au11a] AUTOSAR GbR: Layered Software Architecture. http://autosar.org/download/R4.0/AUTOSAR_EXP_LayeredSoftwareArchitecture. [Aufruf: 15.April 2011].

[Au11b] AUTOSAR GbR: AUTOSAR BSW & RTE Conformance Test Specification Part 1: Background. http://autosar.org/download/R4.0/AUTOSAR_PD_BSWCTSpecBackground.pdf. [Aufruf: 15.April 2011].

[Ds08] dSpace: SystemDesk Guide. Paderborn, 2008.

[En09] Englisch, N.: Methode zur Absicherung der Basissoftware von AUTOSAR Steuergeräten. Technische Universität Chemnitz, Diplomarbeit, 2009.

[He11] Helfrich, T.: Entwurf und Implementierung eines Konzeptes für den Test horizontaler Schichten der Basissoftware von AUTOSAR Steuergäten. Technische Universität Chemnitz, Technischer Bericht, 2011.

[KF09] Kindel, O., & Friedrich, M.: Softwareentwicklung mit AUTOSAR: Grundlagen, Engineering, Management in der Praxis. dpunkt Verlag, 2009.

[We11] Weinhold, F.: Testkonzept für AUTOSAR Softwarekomponenten. Technische Universität Chemnitz, Masterarbeit, 2011.

erschienen im Tagungsband der INFORMATIK 2011
Lecture Notes in Informatics, Band P192
ISBN 978-3-88579-286-4
weitere Artikel online:
http://informatik2011.de/519.html