17 Software-Entwicklung

17.1 Entwicklungsprozesse, Methoden und Werkzeuge

17.1.1 Einleitung

Bis vor einigen Jahren wurde Software in der Automobilindustrie kaum als eigenständiges Thema wahrgenommen. Stattdessen wurde sie als Bestandteil des Bauteils „Steuergerät“ quasi „nebenher“ mitentwickelt. Inzwischen ist eine starke Zunahme des Software-Anteils an den Herstellkosten eines Fahrzeugs und der Trend zur Realisierung gerade auch wettbewerbsdifferenzierender Funktionen durch Software unverkennbar. Ursachen dafür finden sich sowohl in neuen oder verschärften Anforderungen des Gesetzgebers, zum Beispiel im Bereich der Emissionsreduzierung, als auch in den zunehmenden Komfort- und Sicherheitsansprüchen des Autokäufers. Als Folge fällt der Software-Entwicklung mittlerweile eine Schlüsselrolle innerhalb der bisher eher maschinenbaugeprägten Fahrzeugentwicklung zu. Der Entwicklungs-, Integrations- und Testaufwand steigt überproportional zu der Menge der Software und stellt sich zunehmend als eigenständiges Problem heraus, das der Innovationsfähigkeit des Automobilherstellers Grenzen setzt. Bei sinkenden Markteinführungszeiten und unverändert hohen Qualitätsansprüchen stehen Fahrzeughersteller und -zulieferer vor der Herausforderung, die interdisziplinären Entwicklungsprozesse und -methoden unter Einbeziehung der Software-Entwicklung anzupassen und durch geeignete Werkzeuge zu unterstützen.

Die Komplexität der Software-Entwicklung für Automobil-Anwendungen kann unter anderem an folgenden „Kenngrößen“ festgemacht werden: Anzahl der elektronischen Steuergeräte im Fahrzeug, Software-Umfang und Speicherkapazität der einzelnen Steuergeräte, Anzahl und Bandbreite der verwendeten Kommunikationsbusse sowie Umfang der hierüber ausgetauschten Daten. Viele für den Fahrer wahrnehmbare Funktionen wie Abstandsregelung, Überlagerungslenkung oder adaptives Kurvenlicht beruhen auf der Vernetzung der Steuergeräte, auch über bisher getrennte Bereiche im Fahrzeug hinweg. Der erhöhte Vernetzungsgrad erzeugt Abhängigkeiten zwischen den einzelnen Steuergeräten und den darauf implementierten Software-Modulen, bewirkt eine Fortpflanzung von Fehlern durch das Gesamtsystem und erfordert eine Gesamtsicht auf das Funktionsnetzwerk sowie den Test von Steuergeräte-Netzen zusätzlich zum Test der Einzelsteuergeräte.

Die Kernanforderung an den Software-Entwicklungsprozess besteht also darin, die Komplexität der einzelnen Steuergeräte mit ihren steigenden Software-Umfängen sowie das Zusammenspiel der Steuergeräte in einem hochgradig vernetzten Gesamtsystem zu beherrschen. Die bisher eingesetzte Entwicklungsmethodik wird als unzureichend für steuergeräte- und domänenübergreifende Fahrzeugfunktionen betrachtet, insbesondere aufgrund der fehlenden Funktionsnetzwerksicht. Zu den häufig genannten Problemfeldern gehören auch die Schnittstellen in der Zusammenarbeit zwischen Fahrzeughersteller und Zulieferer. Aufgrund der späten Systemintegration und des insgesamt noch zu wenig systematischen Testens in den frühen Entwicklungsphasen werden Fehler im Zusammenspiel der Steuergeräte sehr spät entdeckt. Daraus resultieren Software-Änderungen kurz vor Produktionsstart, deren Folgen und Seiteneffekte oft nicht mehr überblickt werden können. Nicht zuletzt werden erhebliche Anforderungen an das Software-Variantenmanagement gestellt, die aus der Vielzahl der Fahrzeug-, Motor-, Getriebe- und Ausstattungsvarianten resultieren. Bereits entwickelte und getestete Software-Komponenten werden noch in zu geringem Maß wiederverwendet.

All dies führt zu langen Projektlaufzeiten, hohen Entwicklungskosten und gefährdet die Qualität der elektronischen Systeme im Fahrzeug. Der Einsatz von Modellierungs-, Simulations- und Code-Generierungswerkzeugen auf Basis definierter und gelebter Entwicklungsprozesse kann als Grundvoraussetzung betrachtet werden, um diese Herausforderungen in den Griff zu bekommen.

17.1.2 Prozessstandards und Software-Qualitätsmodelle

Zur Beschreibung von Software-Entwicklungsprozessen wird häufig das V-Modell herangezogen ([1], [2]), das zwar die in der Realität unterschiedlichen Vorgehensweisen meist nicht exakt wiedergeben kann, aber doch eine Art „normierte“ Diskussionsgrundlage bietet. Auch der Automobilelektronik-Entwicklungsprozess kann anhand eines solchen V-Modells veranschaulicht werden (**Bild 17-1**), wobei zu berücksichtigen ist, dass die Elektronikentwicklung das gesamte elektrische und elektronische System des Fahrzeugs umfasst, einschließlich Steuergeräte-Hardware, Kabelbaum, Busverbindungen, Sensoren, Aktoren usw. Aus der engen Verzahnung von Hardware- und Software-Entwicklung resultieren einige Besonderheiten.

Durch Analyse der Anforderungen erhält man im ersten Schritt eine Spezifikation auf der Ebene des gesamten elektronischen Fahrzeugsystems.

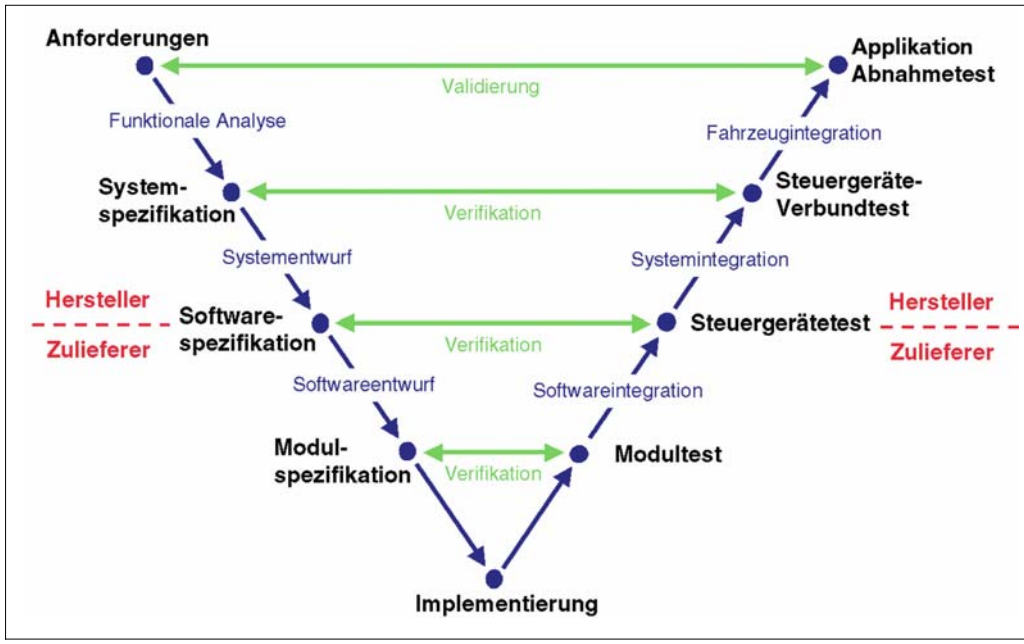


Bild 17-1: V-Modell des Software-Entwicklungsprozesses für Automobil-Anwendungen

Aus dem Systementwurf ergibt sich die Spezifikation der einzelnen Hardware-Komponenten (Steuergeräte, Sensoren, Aktoren, Busse) und der auf diesen Einheiten zu realisierenden Software-Funktionen. In der klassischen Aufgabenteilung findet dieser Schritt noch beim Fahrzeughersteller statt. Die Aufgabe des Zulieferers besteht anschließend in der Umsetzung der Hardware- und Software-Spezifikationen, wobei der Bereich der Hardware in **Bild 17-1** nicht weiter dargestellt ist. Auf Software-Ebene ergeben sich durch weitere Zerlegung der Software- und schließlich der Modulentwurf für ein Steuergerät. Auf dem rechten Ast des V-Modells findet eine zunehmende Integration der Software und Hardware statt. Damit einher gehen die Testschritte vom Modultest (oder „Unit Test“) bis hin zur abschließenden Integration im Fahrzeug und der damit verbundenen Steuergeräte-Applikation.

Neben dem eigentlichen Entwicklungsprozess werden unterstützende Prozesse wie Anforderungs- und Konfigurationsmanagement, Management-Prozesse wie Projekt- und Qualitätsmanagement sowie Kunden- und Lieferantenprozesse benötigt. All diese Prozesse müssen im Hinblick auf die Qualitätssicherung einer ständigen Bewertung und Verbesserung unterliegen. Dazu werden in der Automobilindustrie vor allem zwei Software-Qualitätsmodelle herangezogen: SPiCE [3] und CMMI ([4], [5]). Die in der „Herstellerinitiative Software“ (HIS, [6]) zusammengeschlossenen deutschen Automobilhersteller fordern von ihren Zulieferern ein Assessment gemäß einem der beiden Qualitätsmodelle. Dazu wurde eine Auswahl der relevanten Prozesse spezifiziert [7]. Die grundsätzliche Vorgehensweise bei der Reifegradbestimmung und Prozessverbesserung soll nachfolgend am Beispiel von SPiCE aufgezeigt werden. Einen Vergleich mit CMMI findet man in der Literatur [8].

Die Reifegradbestimmung mit Hilfe eines Assessments soll einem Auftraggeber nicht nur eine zuverlässige Basis für die Auswahl von Zulieferern bieten, sondern auch die Grundlage zur Durchführung von Prozessverbesserungen bei allen Beteiligten bilden, indem sie eine vollständige Beschreibung des gegenwärtigen Zustands liefert. Aus ihr lassen sich dann Stärken, Schwächen und Risiken einzelner Prozesse erkennen, woraus wiederum Prioritäten und Maßnahmen zur Prozessverbesserung abgeleitet werden können.

Diese Ziele werden von SPiCE durch die Vorgabe eines Prozessreferenzmodells und eines Assessment-Prozesses erreicht. Das Prozessreferenzmodell beschreibt einzelne Arbeitspraktiken, die als unverzichtbar für einen guten Software-Entwicklungsprozess angesehen werden. Das Modell schreibt keine einzelnen Techniken oder Methoden für einen Prozessschritt vor, sondern legt nur fest, dass bestimmte Prozessschritte existieren müssen und gibt einige Anforderungen an diese vor. Die jeweilige Organisationseinheit definiert für sich ein kompatibles Prozessmodell, das den Anforderungen des Prozessreferenzmodells entspricht. Das kompatible Modell ist dann die Grundlage des Assessments. Dessen Ergebnis ist eine Bewertungsmatrix, auch „Prozessprofil“ genannt.

Die Bewertung nach SPiCE erfolgt in zwei Dimensionen: in Prozessen und in Fähigkeiten. Jeder Prozess wird einzeln nach seinen Fähigkeiten bewertet. Ein generisches, zu SPiCE kompatibles Prozessreferenzmodell kann der Norm ISO/IEC 12207 [9] entnommen werden. In dieser Norm sind die Prozesse in verschiedene Gruppen aufgeteilt, darunter die Prozessgruppe Engineering, die aus folgenden Prozessen besteht:

ENG.1 Development
ENG.1.1 System requirements analysis and design
ENG.1.2 Software requirements analysis
ENG.1.3 Software design
ENG.1.4 Software construction
ENG.1.5 Software integration
ENG.1.6 Software testing
ENG.1.7 System integration and testing
ENG.2 System and software maintenance

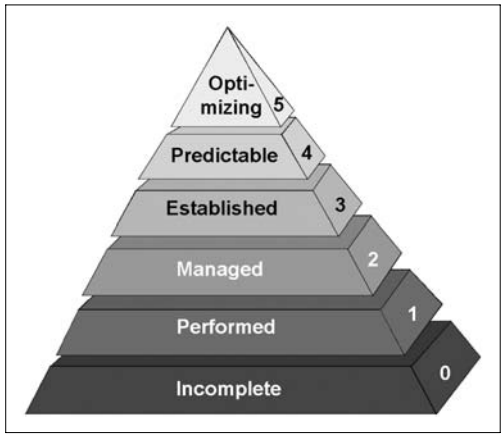


Bild 17-2: Capability Levels nach SPiCE

Orthogonal zu der Prozessdimension steht die Dimension der Fähigkeiten. Die Bewertung erfolgt anhand der in **Bild 17-2** dargestellten *Capability Levels*. Jede der Ebenen beschreibt einen Satz von Prozessattributen, die zusammen genommen einen bedeutenden Schritt in Richtung eines reiferen Entwicklungsprozesses bedeuten. Ein Assessor „misst" die Fähigkeiten eines Prozesses mittels sogenannter Indikatoren. Dabei kann es sich um nachvollziehbare Basistätigkeiten wie den Gebrauch eines Konfigurationsmanagement-Systems, um Eingabedokumente in den Prozess, zum Beispiel Anforderungsspezifikationen, und um Ausgabedokumente wie Testreports handeln. Entsprechend den Erkenntnissen, die der Assessor aus den Indikatoren ziehen kann, vergibt er für jeden Prozess pro Prozessattribut einen Erfüllungsgrad. Die ausgefüllte Bewertungsmatrix stellt dann das Prozessprofil dar (**Bild 17-3**). Hieraus kann ein Gesamtreifegrad für die bewertete Organisationseinheit abgeleitet werden.

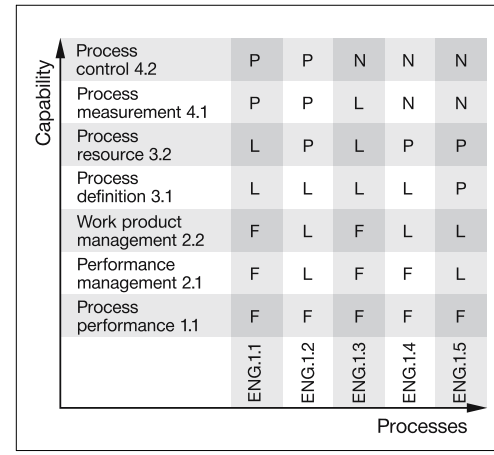


Bild 17-3: Einfaches Beispiel für ein Prozessprofil mit den Erfüllungsgraden N (Not), P (Partially), L (Largely) und F (Fully)

Da im Automobil zunehmend sicherheitskritische Systeme zu entwickeln sind, stellt sich auch hier die Frage nach entsprechenden Normen und Prozessen. Sicherheitsnormen definieren einen Katalog von Maßnahmen, die eingehalten werden müssen, um die angestrebte Sicherheit zu erreichen. Die Maßnahmen können generell in drei Kategorien unterteilt werden: die Auswahl von Entwicklungsmethoden und Werkzeugen, die Implementierung sowie die Verifikation und Validierung des Systems. Die verfügbaren Normen unterscheiden sich bezüglich ihrer Blickwinkel auf das zu entwickelnde System. Einige befassen sich mit der Entwicklung des Gesamtsystems. Dazu gehört z.B. die in der Automobilelektronik bisher verwendetete IEC 61508 [10]. Diese Norm ist ein anwendungsbereichsunabhängiger Standard für elektrische und elektronische Sicherheitssysteme, der die Entwicklung spezifischer Normen erleichtern soll und übergangsweise in Bereichen angewendet wird, in denen es noch keine spezifische Normen gibt. Teil 3 der Norm beschäftigt sich mit Anforderungen an die Softwareentwicklung. Andere Normen wie die RTCA DO-178B [11] und die DoD-2167A behandeln ausschließlich die Entwicklung der Software, sind dafür aber detaillierter und damit konkreter.

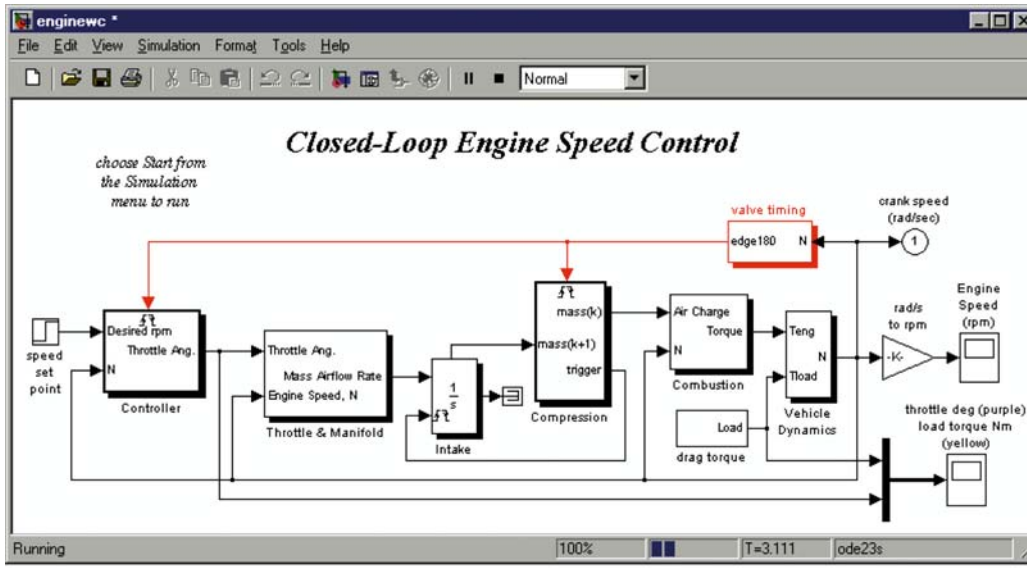
Auch im Bereich der sicherheitskritischen Systeme bieten die modellbasierte Entwicklung und die automatische Code-Generierung weitreichende Möglichkeiten zur Beherrschung von Komplexität und zur Effizienzsteigerung. Mit ihrer verstärkten Nutzung ist die Frage nach der Qualifizierung der Entwicklungswerkzeuge für solche Entwicklungsprojekte und die Anwendbarkeit der bisher existierenden Normen verbunden. Dies betrifft analog zur Zertifizierung von Compilern insbesondere den Code-Generator, der das Modell in C-Code übersetzt [12]. Die auftretenden Fragestellungen

sind bisher nicht abschließend beantwortet, werden aber derzeit in den einschlägigen Fachkreisen der Automobilindustrie diskutiert, so dass in absehbarer Zukunft mit allgemein akzeptierten Vorgehensweisen zu rechnen ist. Im Rahmen der FAKRA (Fachausschuss Kraftfahrzeuge) wurde der Entwurf einer sektorspezifischen Ausprägung der IEC 61508 für den Automobilbereich erarbeitet („Automotive 61508"). Dieser wurde Anfang November 2005 als Working Draft ISO/WD 26262 mit dem Ziel der Standardisierung an die ISO übergeben und enthält u.a. Teile zur Softwareentwicklung sowie zu übergreifenden Aktivitäten wie die Zertifizierung von Entwicklungswerkzeugen [13].

17.1.3 Modellbasierte Funktionsentwicklung

Branchenexperten haben in den letzten Jahren ermittelt, dass zwischen 15 und 40 % aller Software-Fehler in der Automobilelektronik ihre Ursache in unvollständigen und mehrdeutigen Spezifikationen hatten. Zwischen 40 und 60 % aller Probleme entstanden durch Programmierfehler während der Software-Implementierung, davon die Hälfte aus sukzessiven Änderungen. Mit der Verwendung mathematischer Modelle und hochwertiger Beschreibungsformen wie Blockdiagrammen – die letztlich ausführbare (simulierbare) Spezifikationen sind – lässt sich ein Großteil der durch mehrdeutige und unvollständige Spezifikationen hervorgerufenen Probleme ausräumen. Verständlichkeit und Transparenz innerhalb des Entwicklungsprozesses steigen für alle Beteiligten erheblich.
Aus diesem Grund hat sich die modellbasierte Funktionsentwicklung in den letzten zehn Jahren in weiten Bereichen der Automobilindustrie etabliert, häufig zunächst „nur" mit dem Ziel, die Software-Funktionen der elektronischen Steuergeräte eindeutig spezifizieren und die Spezifikation durch Simulation überprüfen zu können. Der modellbasierte Funktionsentwurf erfolgt typischerweise als grafische Blockdiagrammbeschreibung von Steuerungs- und Regelungsalgorithmen in einer entsprechenden Modellierungs- und Simulationsumgebung wie zum Beispiel MATLAB, Simulink und Stateflow (**Bild 17-4**). Modelle können auf allen Ebenen des Software-Entwurfs eingesetzt werden. Das heißt, sie können Funktionen auf System-, Steuergeräte- oder Modulebene beschreiben. Durch Simulation, die in Form des Rapid Control Prototypings (Abschnitt 17..2) auch im realen Fahrzeug oder am Prüfstand unter Echtzeitbedingungen erfolgen kann, werden Entwurfsfehler bereits in einem frühen Stadium aufgedeckt. Test- und Validierungswerkzeuge bieten die Möglichkeit, Funktionsmodelle – zum Teil automatisiert – auf Korrektheit bezüglich der Anforderungsspezifikation zu überprüfen. Die Vorteile wie verringerte Entwicklungszeit und Entwicklungskosten sowie verbesserte Qualität liegen auf der Hand.

Die modellbasierte Funktionsentwicklung wird seit einigen Jahren durch die automatische Code-Generierung für Seriensteuergeräte unterstützt. Code-Generatoren wie TargetLink erzeugen aus einem Blockdiagramm hocheffizienten und zuverlässigen C-Code in Produktionsqualität. Die automatische Code-Generierung steht derzeit an der Schwelle zum produktiven Einsatz mit entsprechenden Rollouts in großen Entwicklungsprojekten (Abschnitt 17.3). Damit ist eine enge Kopplung zwischen Funktionsmodellierung und Software-Erstellung für das in großen Stückzahlen gefertigte Endprodukt (Steuergerät oder Fahrzeug) möglich, so dass der modellbasierte Entwurf eine neue, zentralere Rolle im Entwicklungsprozess des Automobils einnehmen kann und muss. Die Herausforderung besteht in der optimalen Integration der modellbasierten Funktionsentwicklung einschließlich der automatischen Code-Generierung in die Prozesse der Serienentwicklung bei Herstellern und Zulieferern. Besondere Bedeutung gewinnen in diesem Zusammenhang das systematische Management von Anforderungen und deren Verknüpfung mit dem Testprozess, die Kombination des modellbasierten Entwurfs mit klassischen Methoden des Software-Designs sowie die durchgängige Anwendung der Simulation auf der Basis von Architektur-, Software- und Funktionsmodellen. Auch Themen wie Konfigurations-, Varianten- und Datenmanagement müssen im Zusammenhang mit Funktionsmodellen neu betrachtet werden.
Für den Test der Steuergeräte-Funktionen wird ebenfalls seit vielen Jahren die Hardware-in-the-Loop-Simulation (HIL-Simulation) erfolgreich eingesetzt. Ihr Anwendungsgebiet reicht vom frühen Test einzelner Funktionen (Modultestebene) bis hin zum Freigabetest des Gesamtsystems (Abschnitt 17.4). Die HIL-Simulation begleitet damit den gesamten Bereich der Software-, System- und Fahrzeugintegration auf dem rechten Ast des V-Modells (**Bild 17-1**). In der modellbasierten Funktionsentwicklung können Testaktivitäten auch schon in früheren Entwicklungsphasen einsetzen, das heißt, in den linken Teil des V-Modells verlegt werden. Konkret bedeutet dies, dass Spezifikationen, die in Form simulierbarer (ausführbarer) Funktionsmodelle vorliegen, weit vor der Implementierung systematisch getestet werden können (Abschnitt 17.5).
Das Gesamtverhalten eines Fahrzeugs wird letztlich nicht nur durch die eigentlichen Software-Funktionen im Steuergerät beeinflusst, sondern hängt auch maßgeblich von der Einstellung der Steuer- und Regelparameter ab. Durch Abstimmung dieser Parameter im Rahmen der Steuergeräte-Applikation (Abschnitt 17.6) werden die Software-Funktionen so angepasst, dass das Gesamtverhalten die Lastenheftvorgaben erfüllt. In einem modellbasierten Entwicklungsprozess sind die dazu notwendigen Mess- und Applikationswerkzeuge eng mit anderen Tools, beispielsweise Modellierungswerkzeugen und Datenmanagementsystemen, zu koppeln.



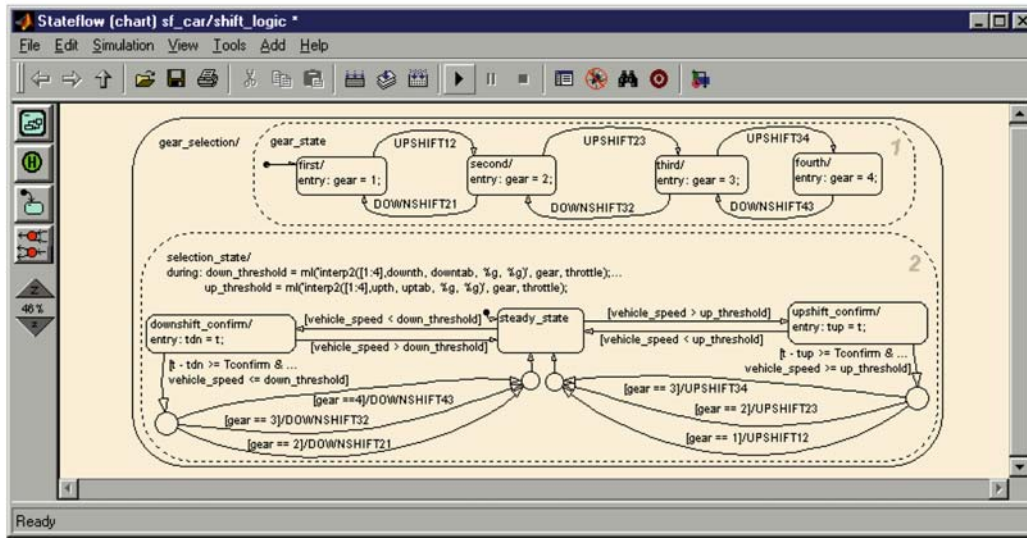


Bild 17-4: Funktionsmodellierung mit Block- und Zustandsdiagrammen

17.1.4 Software-Komponenten und architekturbasierter Entwurf

Eine deutliche Effizienzsteigerung der Software-Entwicklung erreicht man durch einen hohen Grad der Wiederverwendung von Software-Komponenten, wodurch die Entwicklungs- und vor allem die Testaufwände reduziert werden. Als Software-Komponenten werden einzelne Module bezeichnet, die unabhängig von der Umgebung wie Hardware und Betriebssystem eingesetzt und über vordefinierte Schnittstellen mit anderen Komponenten gekoppelt werden können, wodurch sich – baukastenähnlich – die Gesamtfunktion ergibt.

Ein typisches Wiederverwendungsszenario findet sich bei einem Zulieferer, der eine bestimmte Software-Komponente mehrfach für verschiedene Steuergeräte und Kunden verwenden will.

Verfügen Software-Komponenten über standardisierte Schnittstellenbeschreibungen, wie dies in der AUTOSAR-Initiative [15] angestrebt wird, sind weitergehende Wiederverwendungsszenarien denkbar. Automobilhersteller können zum Beispiel höherwertige Funktionen selbst entwickeln und als

Software-Komponente einem oder – im Rahmen verschiedener Baureihen – auch mehreren Zulieferern zur Integration in ein Steuergerät übergeben. Auch die Austauschbarkeit von Software-Komponenten wird dadurch erleichtert. Begleitet werden solche Bemühungen von Standardisierungsaktivitäten für nichtwettbewerbsdifferenzierende Basisfunktionen wie Betriebssysteme und I/O- oder Kommunikationstreiber [6], [14].

Weiterhin bilden komponentenbasierende Ansätze die Grundlage für die durchgängige Erfassung und Planung des Gesamtsystems. Damit kann ein explizites softwaretechnisches Design des zu entwickelnden Systems etabliert werden. Dies umfasst zum einen die bereits erwähnte Verschaltung der Software-Komponenten zur funktionalen Architektur des Systems. Diese kann für eine virtuelle Systemintegration einschließlich einer frühen ersten Überprüfung der funktionalen Eigenschaften des Gesamtsystems verwendet werden. Durch die Festlegung der Software-Schnittstellen wird außerdem die spätere Integration von separat entwickelten Systembestandteilen wie Software-Komponenten, Treibern oder auch ganzen Steuergeräten erleichtert.

Zum anderen umfasst das System-Design die Beschreibung der verwendeten Hardware-Architektur, also der Steuergeräte, Sensoren, Aktoren, Busse und Netzwerk-Topologie, auf welche die funktionale Architektur abgebildet wird. Die Verbindung zwischen hardwareunabhängigen Software-Komponenten und spezifischer Hardware wird durch die Generierung von Verbindungscode (Middleware, Runtime Environment) hergestellt. Spezifiziert man in einem solchen System-Design zusätzlich das Verhalten der einzelnen Software-Komponenten – wie in Abschnitt 17.1.3 beschrieben – mit Hilfe von Modellen, zum Beispiel in MATLAB, Simulink und Stateflow, können die modellbasierte Funktionsentwicklung mit klassischen Methoden des Software-Architekturentwurfs verbunden und die Vorteile aus beiden Welten optimal genutzt werden. Beispiele für Architekturbeschreibungen sind in der Literatur zu finden ([15], [16], [17], [18], [19]).

Literatur zu Abschnitt 17.1

[1] *Dröschel, W.* (Hrsg.): Das V-Modell 97. Oldenbourg Verlag, München, 1999.

[2] *Fennel, H.; Judaschke, U.; Weis, O.*: Automotive Open System Architecture. Elektronik Automotive, Ausgabe 3/2004, WEKA Fachzeitschriften-Verlag, Poing.

[3] ISO/IEC TR 15504:1998, Information technology – Software Process Assessment. ISO/IEC, Switzerland, 1998.

[4] *Paulk, M.; Curtis, B.; Chrissis, M.; Weber, C.*: Capability Maturity Model for Software (Version 1.1). SEI Technical Report, CMU/SEI-93-TR-024, ESC-TR-93-177, 1993.

[5] *Paulk, M.; Weber, C.; Garcia, S.; Beth, M.; Bush, C. M.*: Key Practices of the Capability Maturity Model SM, Version 1.1. SEI Technical Report, CMU/SEI-93-TR-025, ESC-TR-93-178, 1993.

[6] Herstellerinitiative Software (HIS), www.automotive-his.de.

[7] *Hehn, U.*: Softwaretest und -prüfung im Zeichen von SPICE. Automotive Electronics and Systems, Ausgabe 6/2003, Carl Hanser Verlag, München.

[8] *Thomsen, T.*: Integration of International Standards for Production Generation. SAE Technical Paper Series 2003-01-0855, SAE World Congress 2003.

[9] ISO/IEC 12207:1995, Information technology – Software Life Cycle Processes. ISO/IEC, Switzerland, 1998.

[10] IEC 61508 Functional Safety of Electrical/Electronic/Programmable Electronic Safety Related Systems. IEC, 1998.

[11] RTCA/DO-178B Software Considerations in Airborne Systems and Equipment Certification. RTCA Inc., 1992.

[12] *Beine, M.; Otterbach, R.; Jungmann, M.*: Development of Safety-Critical Software Using Automatic Code Generation. SAE Technical Paper Series 2004-01-0708, SAE World Congress 2004.

[13] *Conrad, M.; Dörr, H.*: Einsatz von modell-basierten Entwicklungstechniken in sicherheitsrelevanten Anwendungen: Herausforderungen und Lösungsansätze. Dagstuhl-Workshop, Modellbasierte Entwicklung eingebetteter Systeme II, Informatik Bericht TU Braunschweig 2006-01, 2006

[14] OSEK/VDX, www.osek-vdx.org.

[15] AUTOSAR, www.autosar.org.

[16] ITEA Projekt EAST-EEA, www.east-eea.net.

[17] EU Projekt EASIS, www.easis.org.

[18] *Eppinger, K.; Berentroth, L.*: Plattform versus Flexibilität: Die Siemens VDO EMS 2 Plattform Architektur. VDI Berichte Nr. 1789, 2003.

[19] *Kraft, D.; Lapp, A.; Schirmer, J.*: Elektrik/Elektronik-Architektur – Die Herausforderung für die Automobilindustrie. VDI Berichte Nr. 1789, 2003.

17.2 Rapid Control Prototyping

17.2.1 Überblick

Dieser Abschnitt gibt einen Überblick über den Stand der Technik des Rapid Control Prototyping (RCP) und stellt aktuelle Weiterentwicklungen dieser Technologie vor. Mit Rapid Control Prototyping können neue Konzepte und Ideen für regelungs- und steuerungstechnische Funktionen in frühen Phasen der Entwicklung an der realen Regelstrecke effizient getestet und verifiziert werden. Die Methode ist weithin anerkannt und trägt entscheidend zur Beschleunigung des modellbasierten Software-Entwicklungsprozesses (Abschnitt 17.1.3) für Fahrzeugsteuergeräte bei ([1], [2], [8]). Ausgangspunkt für Rapid Control Prototyping sind grafische Repräsentationen (Modelle) der zu entwickelnden Funktionen, die häufig schon zuvor in einer Software-Simulation getestet wurden. Um diese Modelle in der realen Umgebung testen zu können, muss eine Anwendung mit geeigneten I/O-Schnittstellen aus dem Modell generiert, auf einer Echtzeit-Hardware zur Ausführung gebracht und mit dem zu steuernden oder zu regelnden System verbunden werden.

In frühen Entwicklungsphasen stehen für den Regelungstechniker typischerweise noch nicht die Einzel-

heiten der späteren Software-Implementierung auf dem Seriensteuergerät im Vordergrund, auch wenn hier schon Modellierungsrichtlinien eingehalten werden sollten, um die spätere Generierung von effizientem Seriencode zu ermöglichen (Abschnitt 17.3).

Durch die Verwendung von RCP-Systemen im Entwurfsprozess sind geringe Iterationszeiten garantiert. Am Ende der RCP-Phase steht dann eine sogenannte „ausführbare Spezifikation" in Form des grafisch modellierten Reglers zur Verfügung, die als Referenz für weitere Simulationen dient und in der nachfolgenden Implementierungsphase die Basis für die automatische Seriencode-Generierung bildet.

Typische RCP-Systeme sollten in Bezug auf Rechenleistung, Speicherkapazität sowie Leistungsfähigkeit der I/O und Instrumentierungsmöglichkeiten zum Messen und Applizieren keinen wesentlichen Einschränkungen unterliegen. Hier haben sich PowerPC-basierte Prozessoren auch unter rauen Umgebungsbedingungen bewährt, die aufgrund ihrer Architektur auch komplexe Algorithmen in Mikrosekunden verarbeiten können. Weiterhin bieten sie genügend Reserven zum Messen von Signalen und zum Verstellen von Parametern, um das Verhalten des Reglers zu analysieren. Neben flexiblen, modularen Systemkonfigurationen, die je nach Anwendung und Aufgabenstellung individuell mit leistungsfähigen Prozessor- und I/O-Karten zum Beispiel zur Verarbeitung analoger, digitaler oder auch Bus-Signale ausgestattet werden können, stehen auch kompakte, fest vorkonfigurierte Systeme zur Verfügung. Letztere werden häufig direkt im Fahrzeug unter rauen Umgebungsbedingungen an Stelle des Seriensteuergeräts eingesetzt und müssen über entsprechende elektrische und mechanische Eigenschaften verfügen. Die dazugehörige Software-Werkzeugkette kann wie folgt charakterisiert werden:

- standardisierte grafische Beschreibungssprache zur Implementierung der Regelalgorithmen,
- automatische Code-Generierung für den in grafischer Form (Block- und Zustandsdiagramme) modellierten Regler inklusive Taskhandling, Echtzeitbetriebssystem und I/O,
- möglichst umfangreiche Unterstützung von Standard-I/O, zum Beispiel Analog-, Digital- und Bus-Signale, sowie anwendungsspezifischer I/O, zum Beispiel für Motoranwendungen,
- einfach integrierbare und parametrierbare I/O-Schnittstellen zum Modell,
- intuitiv bedienbare und leistungsfähige Experimentierumgebung zum Visualisieren, Messen von Signalen und Verstellen von Parametern zur Laufzeit.

Generell wird beim RCP zwischen zwei Ansätzen unterschieden: Fullpass und Bypassing. Beim Fullpass sind alle Aktoren und Sensoren direkt mit dem RCP-System verbunden. Dieses ersetzt das Steuergerät komplett und hat die vollständige Kontrolle über die Regelstrecke (Abschnitt 17.2.2). Bypassing stellt einen effizienten Ansatz zur Entwicklung neuer Steuergeräte-Funktionen sowie zur Optimierung bereits existierender Reglerfunktionen dar. Beim Bypassing werden Teilfunktionen aus dem Steuergerät auf das RCP-System ausgelagert, das seinerseits über eine Echtzeitschnittstelle mit dem Steuergerät synchronisiert wird und Daten austauscht (Abschnitt 17.2.3). Das RCP beschränkt sich dabei aber nicht nur auf die Entwicklung einzelner Reglerfunktionen in isolierten Steuergeräten, sondern wird sich auch immer auf vernetzte Funktionen erstrecken, die in einem Steuergeräte-Verbund realisiert werden sollen (Abschnitt 17.2.5).

Bild 17-5 gibt einen Überblick über die verschiedenen Systemkategorien und zeigt deren Verwendung beim Fullpass und Bypassing.

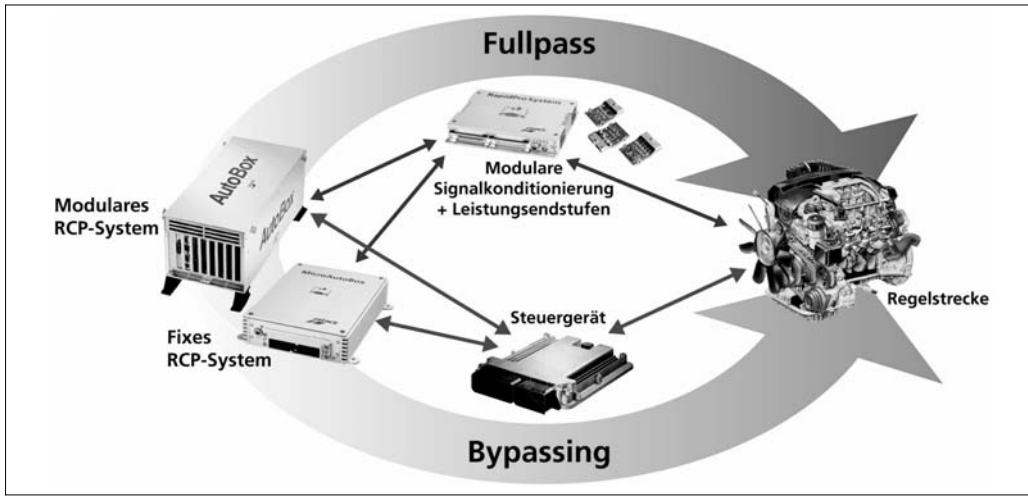


Bild 17-5: Überblick über die verschiedenen Systemkategorien und deren Verwendung beim Fullpass und Bypassing

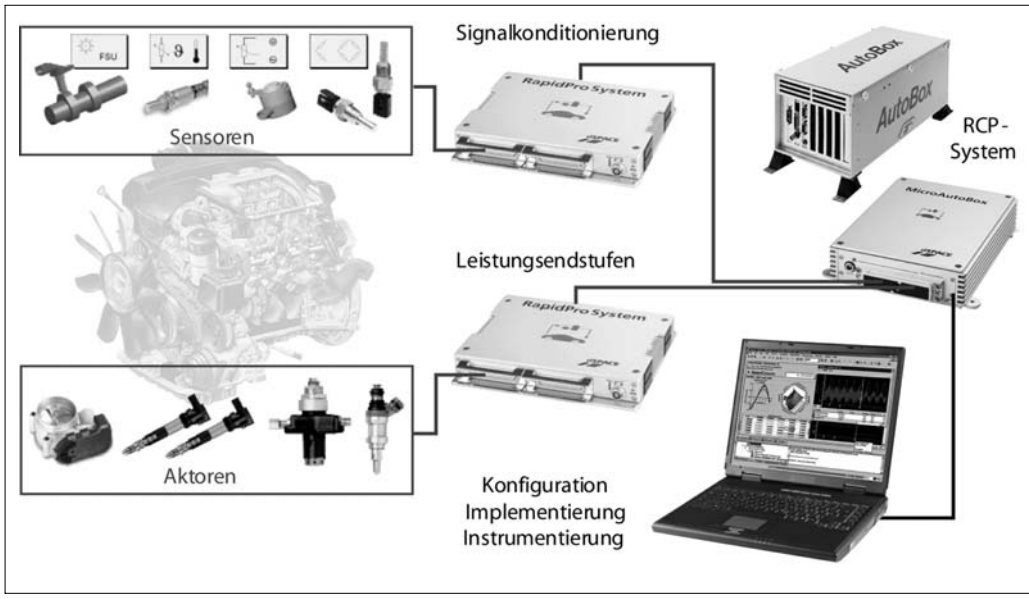


Bild 17-6: Flexible Signalkonditionierung und Leistungsendstufen für RCP-Systeme

17.2.2 Fullpass

Die Signale der meisten Sensoren und Wandler müssen aufbereitet werden, bevor ein RCP-System diese präzise erfassen und verarbeiten kann. Die dazu notwendige Signalkonditionierung beinhaltet z.B. Funktionen wie Schutzbeschaltung, Verstärkung, Abschwächung, Filterung und galvanische Trennung. Aktoren wie elektrische Motoren, Ventile, Relais oder ohmsche Lasten benötigen dagegen Leistungsendstufen wie Vollbrücken und High- oder Low-Side-Treiber mit umfangreichen Schutz- und Fehlererkennungsmechanismen für beispielsweise Übertemperatur, Kurzschluss, Unterspannung und Überstrom.

Aufgrund der großen Vielfalt der in der Fahrzeugtechnik verwendeten Sensoren und Aktoren muss die notwendige Signalanpassung ein hohes Maß an Flexibilität bereitstellen. Wesentliche Vorteile bietet hier ein modularer Ansatz, der es erlaubt, eine auf die jeweilige Aufgabenstellung zugeschnittene Lösung zu realisieren. Zusätzlich kann die Anpassungsfähigkeit durch Hardware- und Software-Konfigurierbarkeit der Module weiter erhöht werden. Mit Verfügbarkeit und Wiederverwendbarkeit derartiger kommerzieller Produkte können so projektspezifische Entwicklungen und Implementierungen vermieden und damit Zeit und Kosten gespart werden. Ein solches flexibles Konzept, bestehend aus Signalkonditionierung und Leistungsendstufen sowie RCP-Systemen, ist in **Bild 17-6** dargestellt.

Mit zunehmender Anzahl der verwendeten Sensoren und Aktoren und steigender Funktionalität muss die Elektronik einerseits größere Datenmengen verarbeiten, andererseits wird die Erfassung und Generierung der I/O-Signale, wie beispielsweise in Motoranwendungen, immer komplexer. In solchen Fällen werden häufig Master-Slave-Konzepte verwendet, in denen zusätzlich zum Hauptprozessor (Master) ein für solche Anwendungen geeigneter Mikrocontroller als reiner I/O-Controller (Slave) eingesetzt wird. Dieser erfasst und generiert komplexe I/O-Signale, zum Beispiel von Kurbelwellen- und Nockenwellensensor für Zündung und Einspritzung, unabhängig vom Hauptprozessor und von der zeitdiskreten Abarbeitung des Regelalgorithmus.

Der I/O-Controller kann dabei entweder in das RCP-System selbst integriert sein, oder aber auch als zusätzliche externe Einheit an das RCP-System angekoppelt werden.

In beiden Fällen erfolgt der Datenaustausch zwischen Master und Slave über eine schnelle Kopplung wie zum Beispiel ein Dual-Port-Memory. Bei räumlicher Trennung ist zusätzlich eine Kommunikationsschnittstelle zwischengeschaltet. Diese sollte nicht nur unempfindlich gegen Störungen sein, sondern auch über eine hohe Bandbreite und geringe Latenzen verfügen.

Die klare Trennung zwischen I/O und Modellberechnung führt zu einer Entlastung des Hauptprozessors und erhöht damit die Leistungsfähigkeit des Gesamtsystems.

In der modellbasierten Entwicklungsumgebung erfolgt die Unterstützung der Slave-I/O genau wie die restliche I/O in der Regel durch Bereitstellung von grafischen I/O-Blöcken. Diese sollten sowohl um-

fangreiche Basis-I/O, zum Beispiel A/D-, D/A-, PWM- oder Bit-I/O, als auch komplexe anwendungsspezifische I/O-Funktionalitäten unterstützen, die sonst nur durch sehr aufwendige Handprogrammierung realisiert werden könnten. In Motoranwendungen ist dies beispielsweise die Erfassung und Generierung von winkelbasierten Signalen.
In Fällen, in denen Hardware-Kosten eine übergeordnete Rolle spielen, oder wenn der Fokus bereits während der Prototyping-Phase auf der späteren Serienimplementierung liegt, wird oft auf Serien-Hardware- und -Software-Komponenten zurückgegriffen. Zentrales Element dieser Systeme ist ein kostengünstiger eingebetteter Mikrocontroller, wie er auch in Serien-Steuergeräten eingesetzt wird. Dieser enthält neben Mikroprozessor und Speicher auch die notwendige I/O-Funktionalität auf einem Chip, bietet aber nicht die Rechenleistung und Speicherkapazität der im Vorangegangenen diskutierten RCP-Systeme. Des Weiteren sind Umfang und Leistungsfähigkeit der I/O durch den verwendeten Mikrocontroller begrenzt. Allerdings verfügen derartige Systeme gegenüber Seriensteuergeräten in der Regel über mehr externen Speicher und oft auch über eine leistungsfähigere Schnittstelle zum Messen und Applizieren (Abschnitt 17.6). Außerdem sind Signalkonditionierung und Leistungsendstufen oft wie bei Seriensteuergeräten integriert. Um die Hardware einfach und schnell optimal an die jeweilige Anwendung und Aufgabenstellung anpassen zu können, ist auch hier ein modularer und konfigurierbarer Aufbau wünschenswert. So können individuelle kompakte Systeme kosteneffizient aufgebaut werden. Aufgrund der geringeren Rechenleistung und des gegebenenfalls geringeren Speichers des Mikrocontrollers sollte der zu implementierende Code in Bezug auf Laufzeit und Speicherbedarf optimiert sein. Heutige Produktions-Code-Generatoren erfüllen diese Anforderungen. Weiterhin bieten diese den Vorteil, sowohl das beim Prototyping eingesetzte Modell als auch den Code-Generator selbst für die spätere Serienimplementierung direkt weiterzuverwenden, was wiederum Kosten und Zeit spart und die Durchgängigkeit verbessert.

17.2.3 Bypassing

Im Gegensatz zu Fullpass-Anwendungen, bei denen das RCP-System das Steuergerät vollständig ersetzt, werden beim Bypassing nur einzelne Teilfunktionalitäten, die modifiziert oder neu hinzugefügt werden sollen, auf das RCP-System ausgelagert. Alle anderen Funktionen werden unverändert auf dem Steuergerät ausgeführt. Dabei bleiben die ursprüngliche I/O, das Betriebssystem, die Diagnose und das Netzwerk-Management erhalten. Das verwendete Steuergerät kann zum Beispiel der Vorgänger eines neu zu entwickelnden Steuergeräts sein. Damit kann auf bewährte Serien-Hardware und bereits verifizierte Funktionsmodule zurückgegriffen werden, die nicht erst aufwendig auf dem RCP-System reproduziert werden müssen. Der Funktionsentwickler kann sich somit voll auf seine eigentliche Aufgabe – die Entwicklung neuer Funktionen – konzentrieren.
Während der Entwicklung und Verifikation einer Funktion werden häufig zusätzliche Prozessinformationen benötigt. Deshalb sollten in der Experimentierumgebung sowohl Mess- und Verstellvariablen der Bypassing-Funktion des RCP-Systems als auch steuergeräteinterne Variablen zeitsynchronisiert dargestellt werden können. Für das Bypassing werden zur Instrumentierung daher häufig Applikationssysteme eingesetzt (Abschnitt 17.6).
Typische Bypassing-Schnittstellen sind in **Bild 17-7** dargestellt. Im Wesentlichen unterscheiden sich diese durch Datentransferrate und Latenzen, Wiederverwendbarkeit und Standardisierungsgrad sowie dem Anpassungsaufwand der Software und der Hardware. Im Folgenden wird auf die hardware- und softwaretechnische Realisierung dieser Schnittstellen näher eingegangen.

17.2.3.1 Hardwaretechnische Realisierung

Im Wesentlichen kommen für den Datenaustausch zwischen RCP-System und Steuergerät folgende typische Hardware-Schnittstellen zum Einsatz, die zum Teil auch für die Steuergeräte-Applikation (Abschnitt 17.6) verwendet werden:

Mikrocontroller-Bus basierte Schnittstelle
In Anwendungen, in denen leistungsfähige Steuergeräte mit beispielsweise 32-Bit-Mikrocontrollern und hohen Taktfrequenzen zum Einsatz kommen, spielen Latenzzeiten und Datendurchsatz eine bedeutende Rolle. Hier werden unter anderem auf Dual-Port-Memory (DPMEM) basierende Lösungen verwendet, die diese Anforderungen optimal erfüllen. Da das DPMEM direkt mit dem Mikrocontroller-Bus (Adress-, Daten- und Steuerbus) verbunden wird, bedeutet dies immer einen spezifischen Eingriff in die Steuergeräte-Hardware. Das DPMEM selbst ist üblicherweise Bestandteil eines sogenannten Plug-on-Device (POD), das sowohl die notwendigen Pegelanpassungen als auch eine leistungsfähige Kommunikationsschnittstelle zum RCP-System beinhaltet. Das Plug-on-Device muss für die am Steuergerät gegebenen Umgebungsbedingungen ausgelegt sein und wird häufig sogar direkt im Steuergeräte-Gehäuse integriert. Mit solchen auf einem Mikrocontroller-Bus basierenden Schnittstellen können zum Beispiel Übertragungszeiten unter 20 µs für die Übertragung von 20 Byte Daten zwischen Steuergerät und RCP-System erreicht werden.

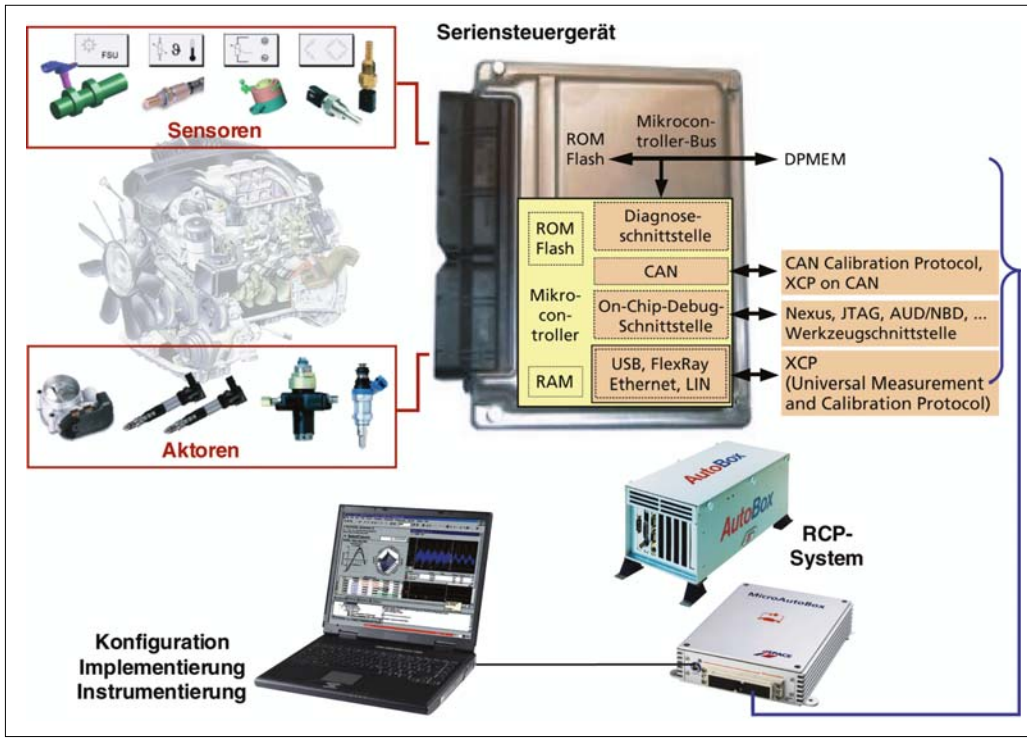


Bild 17-7: Typische Bypassing-Schnittstellen; die Abkürzungen werden im Text erklärt (Abschnitte 17.2.3.2 und 17.2.3.2)

On-Chip-Debug-Schnittstellen

Mikrocontroller in künftigen Steuergeräten mit Taktfrequenzen über 100 MHz und zunehmender I/O-Funktionalität führen dazu, dass der externe Adress-, Daten- und Steuerbus des Mikrocontrollers nicht mehr oder nicht mehr exklusiv für den Anschluss von DPMEM-basierten Bypass-Schnittstellen zur Verfügung steht. On-Chip-Debug-Schnittstellen wie Nexus [3], NBD/AUD oder OCDS/JTAG gewinnen in diesem Zusammenhang mehr und mehr an Bedeutung, insbesondere für anspruchsvolle Bypass-Aufgaben, bei denen auf dem Steuergerät vorhandene Busschnittstellen wie CAN keine ausreichende Bandbreite bieten.

Busschnittstellen

Eine weitere Möglichkeit besteht darin, bereits auf dem Mikrocontroller und auf dem RCP-System vorhandene Busschnittstellen wie CAN für das Bypassing zu benutzen. Ein Eingriff in die Steuergeräte-Hardware oder zusätzliche Hardware ist somit nicht erforderlich. Dies führt zu Kosten- und Aufwandsersparnis. Allerdings ist bei dieser Methode die Datentransferrate deutlich geringer als bei den vorangegangenen Lösungen.

17.2.3.2 Softwaretechnische Realisierung

Unabhängig von der eigentlichen Bypassing-Hardware-Schnittstelle ist eine Modifikation des Steuergeräte-Codes notwendig, um eine Funktion auf das RCP-System auszulagern. Hierzu werden so genannte Freischnitte (spezifische Code Patches) an entsprechenden Stellen in den bestehenden Steuergeräte-Code eingebracht. Aufgabe dieser Code Patches ist es, dem RCP-System benötigte Eingangsgrößen verfügbar zu machen und die Ausgangsgrößen des RCP-Systems wieder in den Programmablauf des Steuergeräts einzubringen.

Das Einfügen des Freischnitts erfordert in der Regel genaue Kenntnis über die Strukturen der Steuergeräte-Software, so dass diese Aufgabe üblicherweise vom Steuergeräte-Zulieferer übernommen wird. Der eingeführte Freischnitt sollte für den Anwender transparent sein und möglichst geringe Auswirkungen auf das Laufzeitverhalten des Steuergeräte-Codes haben.

In **Bild 17-8** ist als Beispiel ein DPMEM-basiertes Bypassing-Szenario abgebildet. Das Steuergerät liest die Sensorsignale der Regelstrecke ein und berechnet gegebenenfalls erste steuergeräteinterne Funktionen. Die für die neu zu entwickelnde Funktion benötigten Eingangsgrößen werden in das DPMEM übertragen (Code Patch #1).

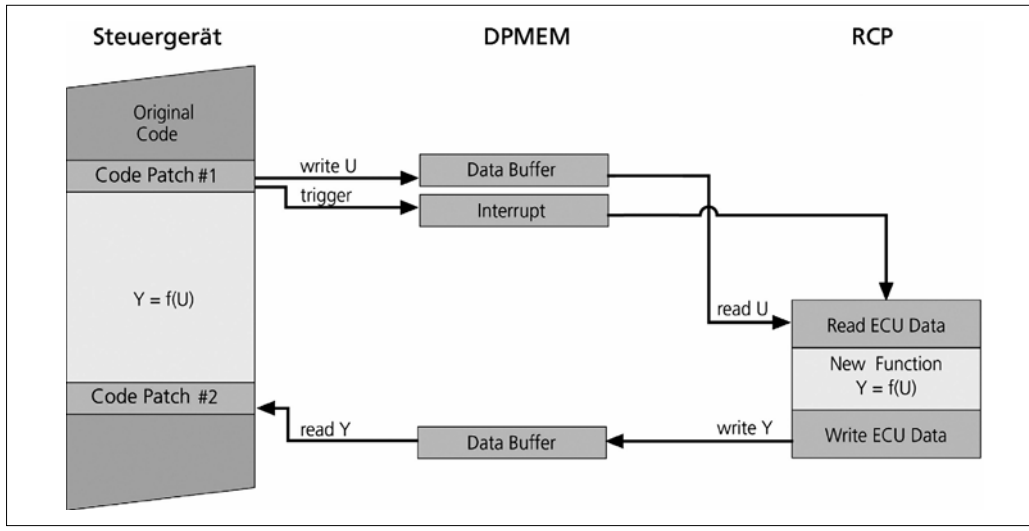


Bild 17-8: DPMEM-basiertes Bypassing-Szenario

Nach vollständiger Übertragung erzeugt das Steuergerät einen Interrupt am RCP-System, das daraufhin die Werte aus dem DPMEM einliest. Anschließend wird auf dem RCP-System die neue Funktion berechnet. Die Ergebnisse werden zurück in das DPMEM übertragen, vom Steuergerät ausgelesen (Code Patch #2) und weiterverarbeitet.

Ist eine möglichst kleine Latenzzeit bei der Datenübertragung ausschlaggebend, sollten die nötigen Freischnitte fest und statisch in den Steuergeräte-Code eingebracht werden. Der Nachteil eines solchen festcodierten Freischnitts liegt jedoch darin, dass dieser modifiziert werden muss, wenn im Laufe der Funktionsentwicklung erkannt wird, dass zum Beispiel ein Eingangssignal fehlt oder ein weiteres Ausgangssignal zum Steuergerät übertragen werden muss. Da dafür der Steuergeräte-Code angepasst werden muss, was häufig nur mit Unterstützung des Steuergeräte-Zulieferers möglich ist, sind automatisch mehrere Personen in die Realisierung einer solchen Modifikation eingebunden. Dieses führt in der Regel zu einem erneuten Abstimmungsaufwand, zusätzlichen Verzögerungen und letztlich zu Kosten.

Will man diese Nachteile umgehen oder als Fahrzeughersteller nahezu unabhängig vom Steuergeräte-Zulieferer arbeiten können, muss die Bypass-Schnittstelle zuverlässig konfigurierbar sein. Das heißt, Eingangs- und Ausgangssignale des Bypassing-Modells müssen flexibel über das RCP-System ausgewählt werden können.

Um dies zu erreichen, ist es möglich, einmalig sogenannte Services anstelle der festcodierten Code Patches im Steuergerät zu implementieren. Diese können dann mit dem RCP-System flexibel aktiviert, deaktiviert oder parametriert werden, ohne den Steuergeräte-Code neu übersetzen zu müssen. Ist ein Service deaktiviert, so darf er nahezu keinen Einfluss auf das Laufzeitverhalten des Steuergeräte-Codes haben. Aufgrund der geringen Auswirkungen und aus Gründen der Aufwandsminimierung verbleiben derartige Bypassing-Services sogar häufig im späteren Seriencode.

Steigender Kostendruck auf die Steuergeräte-Entwicklung hat in letzter Zeit verstärkt dazu geführt, dass die Fahrzeughersteller generische, wiederverwendbare Lösungen basierend auf standardisierten Schnittstellen und Protokollen den kundenspezifischen Bypass-Lösungen vorziehen. Die XCP-Aktivitäten (Universal Measurement and Calibration Protocol, XCP) des ASAM e.V. [5] sind ein wichtiger Schritt in Richtung eines allgemein anerkannten Protokolls für die Steuergeräte-Applikation und Funktionsentwicklung. Dieses Protokoll ist unabhängig vom physikalischen Übertragungsmedium und kann an zahlreiche Schnittstellen wie CAN, FlexRay, Ethernet und USB angepasst werden.

17.2.3.3 Steuergeräteinternes Bypassing

Beim steuergeräteinternen Bypassing wird im Gegensatz zu den zuvor beschriebenen externen Bypassing-Lösungen die Bypassing-Funktion nicht mehr auf einem RCP-System, sondern direkt auf dem Steuergerät implementiert und ausgeführt. Zunächst ist die vorliegende Funktion per Handcodierung oder unter Anwendung eines Seriencode-Generators in C-Code zu realisieren, mit Hilfe der Steuergeräte-Entwicklungsumgebung zu übersetzen (Compiler und Linker)

und in einem ungenutzten Speicherbereich des bestehenden Steuergeräts einzufügen. Indem die Bypass-Services im ursprünglichen Steuergeräte-Code die Eingangs- und Ausgangssignale der Funktion sowie die internen Sprungbefehle zu der neuen Funktion bereitstellen, kann diese im sogenannten steuergeräteinternen Bypass ohne Verwendung zusätzlicher Hardware ausgeführt werden.
Bei einer solchen Implementierung auf einem Seriensteuergerät spielt Code-Effizienz eine wichtige Rolle, da anders als bei RCP-Systemen Speicher und Rechenleistung begrenzt sind. Der wesentliche Vorteil beim internen Bypassing ist die Target-Nähe und die Kostenersparnis, da die Steuergeräte-Hardware unverändert übernommen werden kann und keine RCP-spezifische Software und Hardware erforderlich ist.

17.2.4 Mischformen von Fullpass und Bypassing

Reicht für die zu entwickelnde Funktion die I/O-Funktionalität des Steuergeräts nicht aus, besteht beim Bypassing mit einem RCP-System weiterhin die Möglichkeit, weitere Sensoren und Aktoren wie im Fullpass additiv anzuschließen (**Bild 17-6**). Diese Mischform kommt oft zum Tragen, wenn die zu entwickelnde Funktion bislang nicht benötigte Messwerte verlangt, zum Beispiel weitere Temperaturen oder aber wenn bisher nicht vorgesehene Aktoren angesteuert werden sollen.
Da die I/O des Steuergeräts in der Regel nicht oder nur mit sehr großem Aufwand modifiziert werden kann, liegt der gleiche Fall vor, wenn bisher eingesetzte Sensoren und Aktoren gegen leistungsfähigere oder aber kostengünstigere Varianten mit modifizierten Schnittstellen ersetzt werden sollen.

17.2.5 RCP in verteilten Systemen

Heutige Regelfunktionen sind nicht nur auf einzelne isolierte Steuergeräte beschränkt, sondern erstrecken sich auch auf vernetzte Funktionen, die in einem Steuergeräte-Verbund realisiert werden. Die Regelfunktionen sind dabei häufig von Zustandsgrößen wie Statusinformationen oder auch Sensorgrößen anderer Steuergeräte abhängig.
Hierbei werden für die Kommunikation je nach Anforderung unterschiedliche Fahrzeugbussysteme eingesetzt. Dabei hat sich der CAN als weltweiter Standard durchgesetzt, da die Eigenschaften dieses Bussystems die Anforderungen von Regel- und Steuersystemen im Bereich Antriebsstrang und Karosserieelektronik besonders gut erfüllen. In kostensensitiven Anwendungen hat sich der LIN als geeignete Lösung erwiesen. In verteilten, geschlossenen Fahrzeugregelsystemen für sicherheitskritische Anwendungen kommen dagegen zeitgesteuerte Bussysteme, insbesondere der FlexRay [6] zum Einsatz. Im Gegensatz zu ereignisgesteuerten Systemen, in denen die Aktivitäten durch zeitlich nicht vorhersehbare Ereignisse ausgelöst werden, müssen in zeitgesteuerten Systemen die Aktivitäten vorab zu bestimmten Zeitpunkten definiert (geplant) werden. Zur Entwicklung solcher vernetzter Funktionen, die in einem Steuergeräte-Verbund realisiert werden, muss das RCP-System sowohl die entsprechenden Bussysteme als auch die dazu notwendigen Protokolle und gegebenenfalls das Entwurfsverfahren unterstützen.

Literatur zu Abschnitt 17.2

[1] *Hanselmann, H.*: Development Speed-Up for Electronic Control Systems, Convergence International Congress on Transportation Electronics, Dearborn, 1998.

[2] *Hanselmann, H.*; *Schütte, F.*: Control System Prototyping and Testing with Modern Tools, PCIM, Nürnberg, 2001.

[3] The Nexus 5001TM Forum, www.nexus5001.org.

[4] *Rolfsmeier, A.*; *Richert, J.*; *Leinfellner, R.*: A New Calibration System for ECU Development, SAE World Congress, Detroit, 2003.

[5] ASAM e.V., www.asam.net.

[6] FlexRay Konsortium, www.flexray.com.

[7] *Otterbach, R.*, *et al.*: Rapid Control Prototyping – neue Möglichkeiten und Werkzeuge, VDI-Berichte 1828, 2004.

[8] *Otterbach, R.*; *Schütte, F.*: Effiziente Funktions- und Software-Entwicklung für mechatronische Systeme im Automobil. 2. Paderborner Workshop „Intelligente mechatronische Systeme“, 2004.

17.3 Automatische Seriencode-Generierung

17.3.1 Motivation und Nutzen

Die Software-Entwicklung für Steuergeräte wird in zunehmendem Maße mit modellbasierten Entwicklungswerkzeugen durchgeführt. Steht die Spezifikation unter Zuhilfenahme dieser Werkzeuge fest, beginnt die Arbeit der Codierung. Die grafisch beschriebenen Funktionen müssen nun in C-Code umgewandelt und auf dem Steuergerät implementiert werden. Für diese Aufgabe werden vermehrt automatische Code-Generatoren eingesetzt.
Entgegen den ersten Generationen von Code-Generatoren sind die Methoden der Code-Erzeugung heute so ausgereift, dass sie in der Lage sind, die Brücke vom modellbasierten Funktionsentwurf hin zur Seriensoftware-Entwicklung zu schlagen. Gute Code-Generatoren reduzieren dabei nicht nur die Entwicklungszeit, sondern sie bringen auch eine erhöhte Software-Qualität hervor und verhindern menschliche Programmierfehler.

Zur Zeit werden automatische Code-Generatoren primär für neue funktionale Software-Bestandteile, also für die Steuerungs- und Regelungsfunktionen, in elektronischen Steuergeräten eingesetzt. Die hardwarenahe Software und die Software-Infrastruktur werden manuell erstellt. Da der Funktionscode in Steuergeräten einen Großteil der Software einnimmt, hat die Verwendung automatischer Code-Generatoren in der Automobilelektronik sehr an Bedeutung gewonnen. Sie ermöglichen sowohl eine Erhöhung der Code-Effizienz und -Qualität in Steuergeräten als auch eine Verbesserung der Sicherheit elektronischer Steuergeräte.

17.3.2 Anforderungen und Werkzeugeigenschaften

Der modellbasierte Funktionsentwurf und damit verbunden die automatische Code-Generierung ist im Rapid-Control-Prototyping (Abschnitt 17.2) schon seit einigen Jahren eine etablierte Technologie. Die Tatsache, dass die Seriencode-Generierung erst jetzt an der Schwelle zum breiten produktiven Einsatz steht, hat mit den besonderen Eigenschaften zu tun, die von einem Seriencode-Generator gefordert werden.

Insbesondere sind sowohl die Anforderungen an den generierten Code selbst als auch die an die Einbindung in den Seriensoftware-Entwicklungsprozess zu nennen. Darüber hinaus müssen der Code-Generator sowie der Werkzeughersteller für den Einsatz in Serienprojekten Anforderungen hinsichtlich Qualität, Support und Wartung erfüllen.

Code-Effizienz

Seriencode-Generatoren müssen automatisch Code mit äußerst hoher Effizienz erzeugen. Mangelnde Effizienz war der Hauptgrund, warum Code-Generatoren zwar im Bereich des Rapid Control Prototypings, jedoch nicht für die Seriencode-Generierung eingesetzt wurden.

Um einen Mangel an Effizienz auszugleichen, kam und kommt der Einsatz von schnelleren, mit mehr Speicher bestückten Prozessoren aus Kostengründen nicht in Frage.

Lange Zeit mochten manche Software-Spezialisten nicht glauben, dass Seriencode-Generatoren einmal so gut wie menschliche Programmierer sein oder diese sogar übertreffen würden. Schlechte Erfahrungen mit früheren Code-Generatoren haben diese Skepsis gefördert. Heute können Seriencode-Generatoren sehr wohl die Effizienz von handprogrammiertem Code für die funktionalen Bestandteile von Steuergeräten erreichen.

Denn während der Seriencode-Generator zu jeder Zeit den Code optimiert, haben menschliche Programmierer oft nicht die Zeit, um nach wiederholten Modelländerungen immer wieder den Code zu optimieren.

Es gibt Code-Generatoren, die für verschiedene Compiler-Mikrocontroller-Kombinationen jeweils automatisch das optimale Codierungsmuster auswählen können. Das kann geschehen, ohne vom ANSI-C-Standard abzuweichen oder, falls erlaubt, abweichend vom Standard. Dabei können zur Verfügung stehende Compiler-Erweiterungen genutzt oder auch vereinzelte und gut begründete Einfügungen in Assembler-Sprache vorgenommen werden. Dies geschieht jedoch nur, wenn die Effizienz des Codes dadurch erheblich gesteigert werden kann.

Durch solche „Target-Optimierungen“ werden besonders für kleinere Mikrocontroller mit beschränkten Ressourcen sowie für Prozessoren mit spezieller Hardware, zum Beispiel integrierten Leistungsmerkmalen für digitale Signalverarbeitung, hervorragende Optimierungsergebnisse erzielt.

Vergleichstests haben gezeigt, dass sich die Effizienz von automatisch generiertem Code hinsichtlich Laufzeit und Speicherbedarf ohne weiteres mit handprogrammiertem Code messen kann. Oft wurden sogar die Ergebnisse der handcodierten Software übertroffen (**Tabelle 17-1**).

Tabelle 17-1: Effizienz von automatisch generiertem TargetLink-Code im Vergleich mit handprogrammiertem Code für Antriebsstrang- und Chassis-Anwendungen [Quelle: Delphi]

ROM	RAM	Stack	Geschwindigkeit
0,96–1,1	0,97–1,2	1,2–1,25	0,75–1,2

Festkomma-Unterstützung

Auch bei der Festkomma-Implementierung bietet ein guter Seriencode-Generator dem Anwender umfangreiche Unterstützung. Der Übergang von Fließkomma- zu Festkomma-Arithmetik ist auch für den erfahrenen Entwickler ein aufwendiger und fehleranfälliger Schritt.

Der Seriencode-Generator versetzt den Anwender insbesondere durch automatische Skalierungsoptionen in die Lage, auch für große Modelle schnell zu einer ersten Skalierung zu kommen. Eine Simulation des generierten Codes gibt dem Anwender die Möglichkeit, direkt Festkomma-Implementierung und Fließkomma-Verhalten des Modells miteinander zu vergleichen. Quantisierungseffekte können so einfach analysiert werden. Überlauferkennung und entsprechende Warnungen während der Simulation und der Code-Generierung bieten zusätzliche Hilfestellungen. Um die notwendige Effizienz zu erreichen, ist ein Code-Generator mit dedizierter Festkomma-Code-Generierung unabdingbar.

Software-Konfiguration und -Integration
Automatisch generierter Funktionscode wird in hardwarenahe Software und in die Gesamt-Software-Infrastruktur des Steuergeräts integriert. Etablierte Software-Funktionen, zum Beispiel existierende Look-up-Tabellenfunktionen, müssen in den automatisch generierten Code integriert werden.
Die Code-Partitionierung muss vom Anwender frei vorgegeben werden können, um Funktionen wiederverwenden und Multitasking durchführen zu können. Das Layout des Codes muss nach bestimmten Projekterfordernissen und vorhandenen Konventionen beeinflussbar sein. Variablen müssen für die Steuergeräte-Applikation in bestimmten Speicherbereichen abgelegt werden.
All dies verlangt nach einem Maximum an Flexibilität und umfangreichen Konfigurationsmöglichkeiten bei der Code-Generierung.
Geeignete Seriencode-Generatoren erfüllen diese Anforderungen. Die freie Vergabe von Variablen- und Funktionsnamen, die Vorgabe der Software-Struktur, die vollständige Spezifikation von Funktionsschnittstellen und die Beeinflussung der Code-Ausgabeformatierung sind möglich. Die grafische Oberfläche ist auf die Bedürfnisse des Seriensoftware-Entwicklers abgestimmt.

Lesbarkeit
Schließlich ist auch die Lesbarkeit des generierten Codes von Bedeutung. Die Seriencode-Generierung ist eine relativ junge Technologie, weshalb das Vertrauen in die eingesetzten Code-Generatoren noch wachsen muss. Daher sind heute Code-Inspektionen als endgültige Qualitätskontrolle üblich. Aber auch wenn diese Code-Inspektionen zukünftig mehr und mehr abnehmen und verschwinden werden, bleibt die Lesbarkeit und Nachvollziehbarkeit des erzeugten Codes für dessen Integration, das Debuggen im Fehlerfall und das tägliche Arbeiten mit dem Code wichtig.
Zahlreiche erfolgreich abgeschlossene Serienprojekte in den letzten Jahren haben gezeigt, dass diese Anforderungen an den generierten Code von geeigneten Seriencode-Generatoren erfüllt werden [1], [2].

17.3.3 Einbindung in den Entwicklungsprozess

Für einen Gewinn bringenden Einsatz ist die automatische Seriencode-Generierung optimal in den Entwicklungsprozess zu integrieren. Die Unterstützung von Entwicklerteams, ein projektweites Management von Steuergeräte-Variablen, die Anbindung an Versionskontroll- und Konfigurationsmanagementsysteme, die Handhabung von Varianten und die Automatisierbarkeit von Arbeitsschritten sind zentrale Aspekte, um die Seriencode-Generierung in hohem Maße produktiv einsetzen zu können.

Datenmanagement und Variantenhandhabung
Ein so genanntes Data Dictionary erlaubt ein modellübergreifendes Management von Steuergerätedaten. Durch eine enge Kopplung an den Code-Generator können diese Daten einfach und komfortabel für die Code-Generierung herangezogen werden. Über geeignete Import- und Export-Filter und eine dem Anwender zugängliche Programmierschnittstelle kann der Anschluss an bereits existierende Datenbanklösungen erfolgen. Letztlich ist so die Verwendung bestehender Steuergerätedaten durch den Seriencode-Generator möglich. Darüber hinaus bietet ein Data Dictionary Unterstützung für den Umgang mit Datensatz-Varianten.

Versionskontroll- und Konfigurationsmanagementsysteme
Eine Anbindung an Versionskontroll- und Konfigurationsmanagementsysteme ist möglich, wenn beispielsweise alle anfallenden Daten in separaten Dateien im Dateisystem abgelegt werden. Eine weitergehende Unterstützung und hochwertigere Anbindung, welche die einfache Verwaltung von Code-Generierungsprojekten ermöglicht, steht jedoch noch aus.

Verifikation und Validierung des erzeugten Codes
Ein wesentlicher Vorteil der modellbasierten Entwicklung ist die Möglichkeit der frühen Verifikation durch Simulation. Eine konsequente Ausnutzung der in der verwendeten Entwicklungsumgebung zur Verfügung stehenden Simulationsmöglichkeiten ermöglicht dem Anwender eine schnelle und einfache entwicklungsbegleitende Überprüfung der erzielten Ergebnisse sowie der vorgenommenen Änderungen und Anpassungen.
In verschiedenen Modi können das Funktionsmodell, der generierte Code auf dem Entwickler-PC und der generierte Code auf einem Evaluierungsboard simuliert werden. Die Überprüfung kann somit schrittweise erfolgen (**Bild 17-9**):

- Die Simulation des Funktionsmodells als ausführbare Spezifikation wird üblicherweise als Model-in-the-Loop-Simulation bezeichnet (**Bild 17-9a**). Die Model-in-the-Loop-Simulation liefert Referenzergebnisse, die mit den Ergebnissen der Simulation des generierten Codes verglichen werden können.
- Zur Simulation des generierten Codes auf dem Entwickler-PC, die als Software-in-the-Loop-Simulation bezeichnet wird (**Bild 17-9b**), wird der generierte Code mit einem Host-Compiler übersetzt.
- Die Simulation des erzeugten Codes auf einem Evaluierungsboard, das typischerweise den im Steuergerät verwendeten Prozessor enthält, wird als Processor-in-the-Loop-Simulation bezeichnet (**Bild 17-9c**). Der erzeugte Code wird mit dem im Projekt verwendeten Target-Compiler übersetzt.

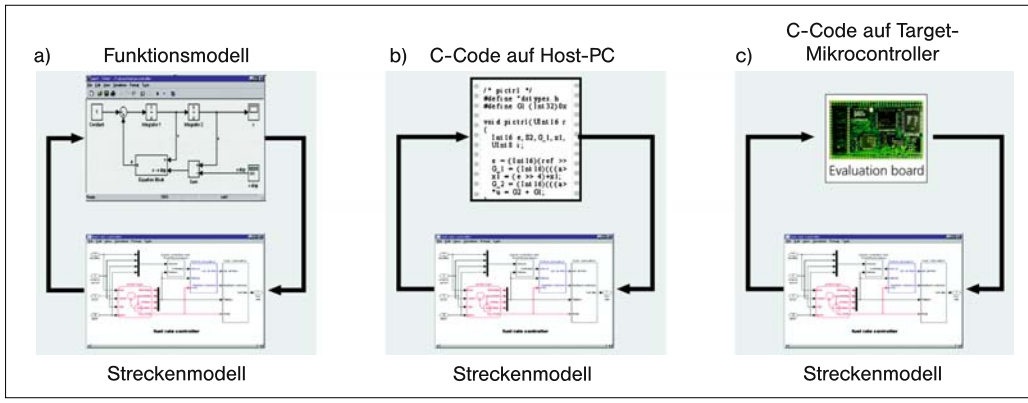


Bild 17-9: Simulationen ermöglichen eine schnelle Überprüfung der Arbeitsergebnisse und unterstützen den Steuergeräte-Test: a) Model-in-the-Loop, b) Software-in-the-Loop, c) Processor-in-the-Loop

Ursprünglich bezeichneten die Begriffe „Model-in-the-Loop", „Software-in-the-Loop" und „Processor-in-the-Loop" nur den Test im geschlossenen Regelkreis (Loop, siehe Bild 17-9). Mittlerweile wird die Bezeichnungsweise sinngemäß auch für reine Steuerungsanwendungen (ohne geschlossenen Regelkreis, „Open Loop") verwendet. Geeignete Werkzeuge bieten die Möglichkeit, in allen unterstützten Simulationsmodi Signalverläufe aufzuzeichnen. Diese können gespeichert und direkt miteinander verglichen werden. Sie geben direktes, visuelles Feedback und ermöglichen weitergehende Analysen. Auch die Möglichkeit zur Ermittlung der Code-Überdeckung (Code Coverage) in Software-in-the-Loop- und Processor-in-the-Loop-Simulationen wird heute bereits unterstützt.

Die Bereitstellung dieser Möglichkeiten ist ein wichtiger Schritt auf dem Weg zur angestrebten Qualitätsverbesserung, Fehlerreduzierung und Zeitersparnis durch den Einsatz von automatischer Code-Generierung in Verbindung mit modellbasierter Entwicklung. Idealerweise erfolgen dann auch systematische Tests in der gleichen Umgebung. Model-in-the-Loop-, Software-in-the-Loop- und Processor-in-the-Loop-Simulation bilden dabei unter anderem die Grundlage für Back-to-Back-Tests (Abschnitt 17.5) und schlagen somit die Brücke zum systematischen Software-Test.

Software-Dokumentation und Beschreibungsdatei für die Steuergeräte-Applikation

In den Aufgabenbereich des Code-Generators fällt auch die auf die Bedürfnisse des Software-Entwicklers abgestimmte Dokumentation des erzeugten Codes sowie die Generierung von Beschreibungsdateien für die Steuergeräte-Applikation. Damit werden bei Einsatz eines entsprechenden Code-Generators das Modell als ausführbare Spezifikation, der erzeugte Code, die Dokumentation und die Datei mit Beschreibungsdaten zur Steuergeräte-Applikation in jedem Stadium der Entwicklung automatisch konsistent gehalten.

Automatisierung von Arbeitsschritten

Die Automatisierung von Arbeitsschritten in einem iterativen Prozess hilft Flüchtigkeitsfehler zu vermeiden und sorgt somit für eine höhere Qualität und Zeitersparnis. Modellierungsumgebung und Code-Generator bieten eine dokumentierte Programmierschnittstelle. Durch die Erstellung von Skripten durch den Anwender können somit immer wiederkehrende manuelle Arbeitsschritte automatisiert und letztlich auch reproduzierbar gemacht werden. Der Aufbau einer auf die Anwenderbedürfnisse abgestimmten, skriptbasierten Werkzeugkette ist damit möglich.

17.3.4 Unterstützung relevanter Standards

Unabdingbar für den Aufbau einer durchgängigen Werkzeugkette, ein reibungsloses Zusammenspiel der eingesetzten Werkzeuge und eine nahtlose Integration in den Entwicklungsprozess sind klar definierte und idealerweise standardisierte Schnittstellen. Standards unterstützen und vereinfachen die Wiederverwendbarkeit von einzelnen Software-Modulen, die Integration der Funktionssoftware in das Steuergerät sowie das Zusammenspiel von verschiedenen im Entwicklungsprozess eingesetzten Werkzeugen und Software-Anteilen [3].

ANSI-C

Ein für die Seriencode-Generierung relevanter Standard ist naturgemäß die Norm der Ausgabesprache. In der Regel ist dies die Sprache C, die durch die ISO/IEC 9899 international genormt wurde und identisch mit der ANSI X3.159 ist. Die Einhaltung des ANSI-C-Standards ermöglicht die projektübergreifende Wiederverwendung sowie die Portierung von Software-Funktionen auf andere Prozessorplattformen.

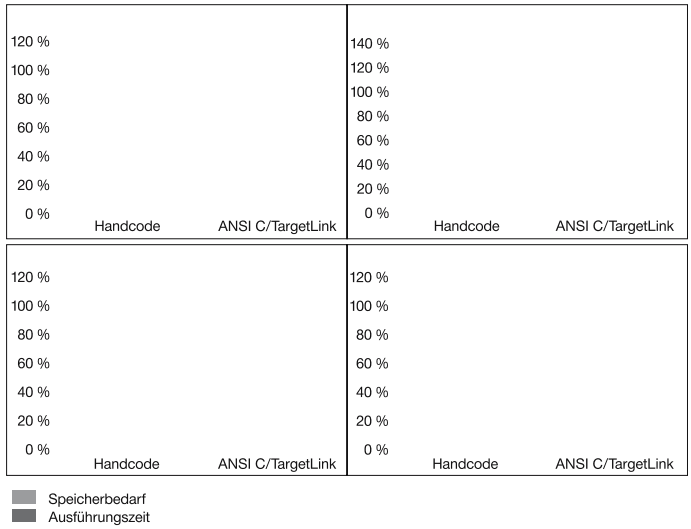


Bild 17-10:
Portierung einer ESP-Implementierung – Vergleich von handprogrammiertem ANSI-C-Code und automatisch mit TargetLink generiertem ANSI-C-Code für vier verschiedene Mikrocontroller

Da hinsichtlich der Portabilität das Ziel darin bestehen sollte, Modelle zu portieren und nicht den Code selbst, spielen Abweichungen vom ANSI-C-Standard für die automatische Code-Generierung eigentlich keine große Rolle. Solche Abweichungen entsprechen zwar nicht den im Allgemeinen vorzufindenden bisherigen Programmierrichtlinien, jedoch wurden diese zur Unterstützung der menschlichen Programmierer eingeführt. Der Code kann aber zu jeder Zeit passend für einen neuen Mikrocontroller oder eine neue Steuergeräte-Umgebung wieder generiert werden („Just-in-Time-Generierung"). Es kann dann sogar Code für einen Festkomma-Controller erzeugt werden, wenn vorher für einen Fließkomma-Controller entwickelt wurde und umgekehrt.
Ein japanischer Automobilelektronikzulieferer hat eine Evaluierung des Seriencode-Generators TargetLink mit Blick auf die Portierung von Reglerfunktionen zwischen verschiedenen Mikrocontrollern durchgeführt (**Bild 17-10**). Für die mit Simulink und Stateflow beschriebenen Funktionen eines elektronischen Stabilitätsprogramms (ESP) wurde mit TargetLink Code für vier verschiedene Mikrocontroller erzeugt. Selbst ohne zielprozessorspezifische Optimierung konnten hinsichtlich der Effizienz des Codes exzellente bis akzeptable Ergebnisse erreicht werden. Der zeitliche Aufwand reduzierte sich hierbei um 75 % gegenüber einer manuellen Portierung.

MISRA C
Es gibt eine Reihe von die ANSI-C-Norm ergänzenden Standards und Publikationen, die dem Zweck der Qualitätssicherung dienen. Die von der britischen Motor Industry Software Reliability Association (MISRA) herausgegebenen „Guidelines for the Use of the C Language in Vehicle-based Software" [4], allgemein unter dem Schlagwort MISRA C bekannt, haben sich zu einer in der Automobilindustrie viel beachteten Norm für die Verwendung von C-Code in Steuergeräte-Projekten bis zu Safety Integrity Level 3 entwickelt. Diese Richtlinien wurden ebenfalls mit Blick auf den menschlichen Programmierer eingeführt. Nicht alle Regeln sind daher auch für Code-Generatoren sinnvoll. Vor diesem Hintergrund wird von den Fahrzeugherstellern und Zulieferern eine MISRA-C-Version für Code-Generatoren angestrebt.
Viele Seriencode-Generatoren halten einen Großteil der MISRA-Regeln ein. Abweichungen von der Norm sind ausdrücklich erlaubt, solange diese technisch begründet und dokumentiert sind. Mit Bereitstellung eines Dokuments, in dem die Abweichungen zu MISRA dargestellt sind, lassen sich Seriencode-Generatoren bereits heute MISRA-konform einsetzen.

OSEK
Die direkte Unterstützung des von der Automobilindustrie initiierten und mittlerweile weltweit akzeptierten Betriebssystemstandards OSEK/VDX [5] durch Code-Generatoren ermöglicht die einfache Integration und Wiederverwendbarkeit von Funktionscode-Modulen. Schon auf Modellebene können entsprechende Einstellungen für eine OSEK/VDX-konforme Code-Generierung vorgenommen werden. Die Informationen zu verwendeten oder zu verwendenden Betriebssystemobjekten und Einstellungen werden über OIL-Dateien (OSEK Implementation Language) ausgetauscht. Für eine Integration des erzeugten Codes mit dem Betriebssystem sind dann keine manuellen Nacharbeiten und Anpassungen mehr notwendig.

MSR- und AUTOSAR-Software-Beschreibungen
Eine geeignete Dokumentation von Modell und erzeugtem Code beinhaltet eine vollständige Beschreibung der Schnittstellen des generierten Codes, der dazugehörigen Dateien und Funktionen, der globalen Variablen, der applizierbaren Parameter und der messbaren Signale.
Ein existierender Standard für eine auf die Bedürfnisse des Software-Entwicklers abgestimmte XML-basierte Beschreibung des erzeugten Codes ist die vom MSR-Konsortium (Manufacturer Supplier Relationship) [6] erarbeitete MSR-Software-DTD (Document Type Definition). Auch im Rahmen von AUTOSAR [7] entsteht eine solche XML-basierte Software-Beschreibung.

ASAM-MCD 2MC
Eine weitere Pflichtübung eines Code-Generators ist die enge Verknüpfung mit Applikationssystemen. Vom Anwender spezifizierte Variablen oder Parameter sollen in diesen Systemen zur Verfügung stehen. Dieses wird dem Applikationssystem über eine Beschreibungsdatei nach dem ASAM-MCD-2MC-Standard [8] bekannt gegeben. Alle wichtigen Applikationssysteme unterstützen heute diesen Standard und der Code-Generator kann die Datei parallel zum generierten Code erzeugen.

17.3.5 Qualität und Support

Die Entwicklung sicherheitsrelevanter Systeme ([10], [11], [12]), Fragen der Produkthaftung sowie die immensen Kosten und der damit verbundene Image-Schaden einer Rückrufaktion rücken den Aspekt der Qualitätssicherung in den Mittelpunkt (Abschnitt 17.1) [9]. Auch die Software-Verifikation, wie in Abschnitt 17.3.3 beschrieben, ist ein wichtiger Ansatzpunkt, um die geforderte Produktqualität sicherstellen zu können.
Die Einhaltung von Produktentwicklungszielen und Time-to-Market-Anforderungen verlangen darüber hinaus nach schnellem und kompetentem Support. Ebenso ist eine langfristige Unterstützung und Pflege auch von nicht mehr aktuellen Code-Generator-Versionen zu gewährleisten, um einen Umstieg in laufenden Serienprojekten zu vermeiden.
Für den Anwender ist der Zugriff auf die Beschreibung von aktuell bekannten Werkzeugproblemen hilfreich, um zu verhindern, dass man sich an bereits bekannten Problemen aufhält, für die eine Umgehungsmöglichkeit existiert.

Literatur zu Abschnitt 17.3

[1] *Weckenmann, H.*: DaimlerChrysler setzt bei Motorsteuerungen auf TargetLink, dSPACE NEWS, 2/2003.
[2] *Diehm, J.*; *Günther, S.*, Dr.: Schneller ans Ziel: TargetLink bei Conti Temic, dSPACE NEWS, 2/2003.
[3] *Thomsen, T.*: Integration of International Standards for Production Code Generation. SAE Technical Paper Series 2003-01-0855, SAE World Congress 2003.
[4] MISRA Guidelines for the Use of the C Language in Vehicle-Based Systems, April 1998.
[5] OSEK/VDX, www.osek-vdx.org.
[6] Manufacturer Supplier Relationship (MSR), www.msr-wg.de.
[7] AUTOSAR, www.autosar.org.
[8] ASAM e.V. Association for Standardization of Automation- and Measuring Systems, www.asam.de.
[9] *Beine, M.*; *Otterbach, R.*; *Jungmann, M.*: Development of Safety-Critical Software Using Automatic Code Generation. SAE Technical Paper Series 2004-01-0708, SAE World Congress 2004.
[10] *Alaoui, A.*: C Code Reaches New Heights at Nord-Micro, dSPACE NEWS, 1/2002.
[11] *Tschiskale, E.*: Active Control Retractor with TargetLink at TRW, dSPACE NEWS, 1/2003.
[12] *Dick, W.*; *Holle, M.*: Entwicklung einer Überlagerungslenkung, ATZ 5/2003.

17.4 Hardware-in-the-Loop-Simulation

17.4.1 Motivation und Nutzen

Mit Hilfe der Hardware-in-the-Loop-Simulation (HIL) wird die Steuergeräte-Funktionalität inklusive der Diagnosefunktionalität getestet. Die HIL-Simulation hat sich inzwischen als integraler Bestandteil des Steuergeräte-Entwicklungsprozesses der Zulieferer und der Automobilhersteller etabliert. Damit wurde im Elektronik-Entwicklungsprozess ein weiterer Meilenstein zur Erhöhung der Qualität im Fahrzeug durch Eliminierung von Fehlern verankert. Der Test am HIL-Simulator reicht vom frühen Funktionstest einzelner Funktionen bis hin zum Freigabetest des Gesamtsystems. Es gibt hier keinen wesentlichen Unterschied in Bezug auf die Anwendung: Steuergeräte für den Antriebsstrang, für die Fahrdynamik und für den Bereich Innenraum, Komfort, Sicherheit werden durch HIL-Simulation getestet. HIL-Simulation ist heute Bestandteil der Freigabevoraussetzung des Fahrzeugs.

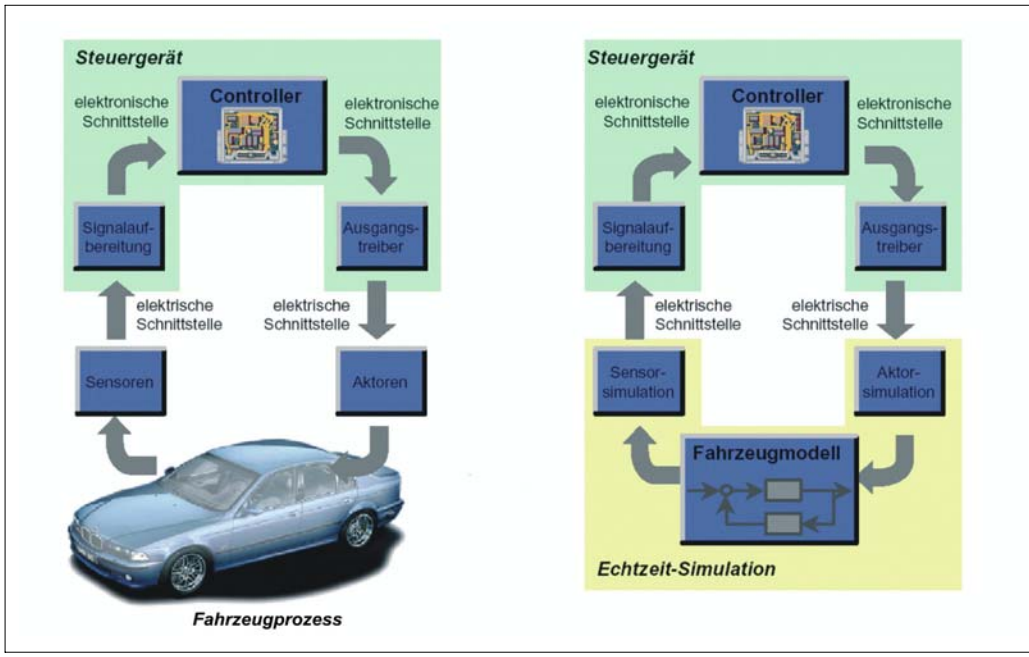


Bild 17-11: Signalfluss im realen Fahrzeug (links) und in der HIL-Simulation (rechts)

Die HIL-Simulation ermöglicht den Test von Steuergeräten in einer realistischen Testumgebung während der Steuergeräte-Entwicklung, ohne dass die Umgebung (Fahrzeug, andere Steuergeräte, angeschlossene Aktoren und Sensoren) real vorhanden ist. Statische und dynamische Vorgänge der Umgebung des Steuergeräts werden in Echtzeit simuliert. Testobjekt und Simulationsumgebung bilden einen geschlossenen Kreislauf (**Bild 17-11**).

Die Ausgangssignale des Steuergerätes sowie seine Bussignale werden in Echtzeit vermessen. Nach einer eventuellen Simulation der Aktoren werden die Signale als Eingangsgröße dem Echtzeitsimulationsmodell der Regelstrecke zugeführt. Basierend auf dem Zustand dieses Streckenmodells werden die Sensorsignale erzeugt, die das Steuergerät für seinen Betrieb erwartet. Parallel dazu werden die Bussignale (CAN, LIN, FlexRay, MOST, seriell...) erzeugt, die im wirklichen Fahrzeug von anderen Steuergeräten gesendet werden. Diese Simulation des Kommunikationsnetzwerks nennt man auch Restbussimulation.

Folgende Unterschiede bestehen zwischen HIL und anderen Testmethoden: Reine Brettaufbauten und Fahrversuche sind teuer, aufwendig und nicht automatisierbar und können damit auch nicht reproduziert werden. Das Testen von Grenzbereichen im Fahrversuch ist gefährlich. Entgegen dem Testen auf dem Prüfstand erlaubt das Testen mit HIL ein beliebiges Einstellen von Betriebspunkten, zum Beispiel die Ausnutzung des kompletten Motordrehzahlbereichs. Zumindest für die Reproduzierbarkeit kann mit HIL eine deutlich höhere Präzision erreicht werden. Kosten werden durch Einsparen von Testfahrzeugen reduziert. Letztlich werden Regressionstests durch Automatisierung erst möglich. Des Weiteren erlaubt eine Automatisierung einen Test rund um die Uhr und an sieben Tagen in der Woche (24/7-Test).

17.4.2 Rollenverteilung im Test von Steuergeräte-Software

Eine Aufgabe der Automobilhersteller ist, das Zusammenspiel von Steuergeräten verschiedener Zulieferer im Gesamtsystem zu überprüfen (Steuergeräte-Verbundtest). Zu diesem Zeitpunkt des Tests sollten die Einzelsteuergeräte bereits durch die Zulieferer auf Erfüllung der Spezifikation getestet worden sein (Abschnitt 17.1.2).

Bisher haben die Automobilhersteller häufig auch Funktionstests der Steuergeräte übernommen. Um Aufwand und damit Zeit und Kosten zu sparen, muss sichergestellt sein, dass der Hersteller nicht erst zum Zeitpunkt des Verbundtests noch Fehler im Komponentenbereich aufdeckt. Deshalb findet derzeit ein intensiver Austausch bezüglich der zukünftigen Rollenverteilung und deren Verankerung in den jeweiligen Entwicklungsprozessen statt.

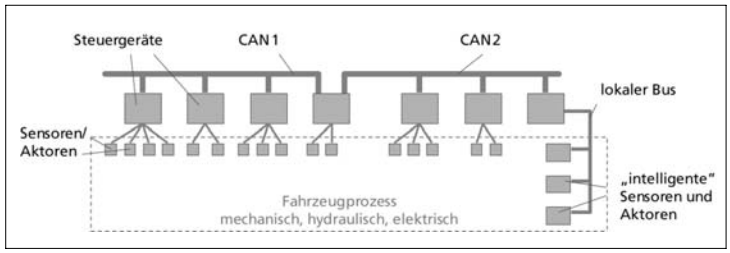


Bild 17-12:
Schematische Darstellung der Steuergeräte-Schnittstellen auf Signal- und Busebene

Tabelle 17-2: Rollenverteilung im Entwicklungsprozess bei der HIL-Anwendung

Wer?	Funktionsentwickler	Steuergeräteverantwortlicher	Systemverantwortlicher
Wo?	Steuergerätehersteller (evtl. auch Automobilhersteller)	Automobilhersteller oder auch Steuergerätehersteller	Automobilhersteller
Phase	Steuergeräte-Funktionstest	Test eines einzelnen Steuergerätes	Test des Steuergeräte-verbundes
Ziel	Validierung der Regel-strategie	Software-Integrationstest Abnahme- und Freigabetest	Test verteilter Funktionen Test vernetzter Systeme
Charakteristika der Tests	Regelstrategie Diagnoseverfahren Entwicklungssteuergerät	Gesamtfunktionalität des Steuerge-rätes Gesamter Betriebsbereich Kritische Fahrzustände Diagnosefunktionen Messungen aus Fahrversuch Restbussimulation	„Zusammenspiel" Busverhalten Netzwerkmanagement Stromaufnahme

Für den Test auf Komponentenebene beim Zulieferer muss der Fahrzeughersteller unter Umständen eine Restbussimulation der übrigen Steuergeräte zur Verfügung stellen (**Bild 17-12**).
Im Folgenden werden die Rollenverteilung in den verschiedenen Entwicklungsphasen und die sich daraus ergebenden Anforderungen an das HIL-System erläutert [4]. **Tabelle 17-2** gibt einen Überblick.

Funktionsentwickler beim Zulieferer (oder beim Fahrzeughersteller)
Der Funktionsentwickler testet in diesem Schritt einzelne Funktionen und die Regelstrategien. Typischerweise ist die Funktionsabnahme das Ziel dieser Phase. Daraus ergeben sich folgende Anforderungen an das HIL-System:

- Häufig ist es üblich, dass Zulieferer einen Regelalgorithmus in verschiedenen Steuergeräten wiederverwenden. Das erfordert eine flexible Simulator-Hardware, die leicht an die entsprechenden Steuergeräte-Varianten angepasst werden kann. Eine entsprechende Verwaltung der HIL-Software-Komponenten wie Teilmodelle und Testskripte ist notwendig.
- Außerdem ist eine flexible, interaktive Bedienbarkeit der Simulator-Software ist gewünscht.
- Üblicherweise ist nur ein geringer Grad an Automatisierung nötig (Testskripte werden oft erst parallel zur Steuergeräte- und Funktionsentwicklung erstellt).

Da die Steuergeräte-Diagnose von Signalwerten abhängig ist, die erst zu einem späteren Zeitpunkt appliziert werden (können), findet in dieser Phase der Funktionstest häufig an einem Steuergerät mit deaktivierter Diagnose statt.
Um Aufwand zu sparen, sollten Tests, die erfolgreich während der Funktionsentwicklung durchgeführt wurden, nicht in der Integrationstestphase wiederholt werden müssen.

Steuergeräte-Verantwortlicher beim Fahrzeughersteller (oder Zulieferer)
Sobald die Funktionen zusammen mit den unteren Software-Schichten (Betriebssystem, I/O-Treiber) auf der Ziel-Hardware integriert sind, muss das Steuergerät makroskopische Tests durchlaufen. Dazu gehören auch Tests auf überlappenden administrativen Ebenen, zum Beispiel der Umgang mit dem Diagnosespeicher.
Entweder führt der Hersteller oder der Zulieferer den Freigabetest für das Steuergerät durch. Ziel ist, das komplette Steuergerät einschließlich Diagnosen ohne Fehler freizugeben.

In dieser Phase ist automatisiertes Testen unverzichtbar. Ein interaktiver Einsatz ist nur für die Testentwicklung erforderlich oder dann, wenn die Ursache für einen entdeckten Fehler gefunden werden soll.
Setzen Hersteller Steuergeräte unterschiedlicher Zulieferer mit identischer Funktionalität ein (Second-Source-Prinzip), müssen sie die Tests für beide Systeme am besten mit Hilfe von Regressionstests durchführen. Beim Steuergeräte(integrations)test kommt der Administration der Software-Komponenten des Simulators (Teilmodelle, Testskripte usw.) noch größere Bedeutung zu als während des Funktionstests.

System-Verantwortlicher beim Fahrzeughersteller
Wie bereits erwähnt, sollten Tests, die bereits auf Komponentenebene durchgeführt wurden, aus Effizienzgründen nicht beim (Sub-)Systemtest wiederholt werden müssen. Bei der Durchführung der Freigabetests mit HIL für die komplette Fahrzeugelektronik liegt der Schwerpunkt explizit auf den Tests der verteilten Funktionen und der Buskommunikation. In diesem Zusammenhang wird auch das Netzwerkmanagement getestet. Testziel ist ein einschließlich der Diagnosen fehlerfreies Gesamtsystem für die HIL-Freigabe.
Häufig müssen hier verschiedene Fahrzeug-Konfigurationen getestet werden. Länderspezifische Varianten sowie verschiedene Ausstattungsvarianten und Sondermodelle machen den Umgang mit unterschiedlichen Konfigurationen erforderlich. Folglich werden kombinatorische Tests gebraucht. Das HIL-System muss eine Vielzahl an Streckenmodellen und I/O-Kanälen sowie die Buskommunikation unterstützen.
An dieser Stelle ist automatisiertes Testen unverzichtbar. Für das System ausgelegte Tests können leicht für alle Steuergeräte- und Fahrzeugvarianten wiederholt werden. Je höher der Automatisierungsgrad, desto höher ist auch die Testabdeckung.
Ein weiterer wichtiger Aspekt ist, dass Tests, die während der Serienentwicklung die Funktionalität verifiziert haben, nach Beginn der Serienproduktion auch dafür eingesetzt werden können, Gewährleistungsansprüche zu untersuchen.
Es gilt an dieser Stelle zu erwähnen, dass bei der Simulation immer ein Restrisiko durch Ungenauigkeiten der Simulationsmodelle und durch nicht berücksichtigte Situationen bleibt. Risiken durch Vernachlässigung in der Modellbildung sind beim Systemtest im Fahrzeug nicht vorhanden. Fahrversuche bleiben aus diesem Grund unverzichtbar. Die endgültige Freigabe eines kompletten Systems muss deshalb im Fahrzeug erfolgen [3].

17.4.3 Komponenten eines HIL-Simulators

Trotz unterschiedlicher Detailanforderungen abhängig von den Anwendungsfeldern (Abschnitt 17.4.2) sind wesentliche Komponenten des HIL-Simulators immer in ähnlicher Ausprägung vorhanden. Im Folgenden werden die einzelnen Komponenten eines HIL-Simulators und die Anforderungen an sie genauer beschrieben [6]. **Bild 17-13** zeigt alle wesentlichen Komponenten eines HIL-Simulators für ein einzelnes Steuergerät.

17.4.3.1 Hardware

Echtzeitprozessoren
Die HIL-Simulation mit ihren immer komplexeren und genaueren Modellen (siehe Abschnitt 17.4.3.2) stellt besonders hohe Anforderungen an die Rechenleistung der Echtzeitprozessoren. Üblicherweise werden die Modelle mit einer Schrittweite von 1 ms gerechnet, um die Anforderungen an das Echtzeitverhalten zu erfüllen. In Motorsportanwendungen liegen die Rechenzeiten mit 50 bis 250 µs vielfach noch darunter [9]. Da in den letzten Jahren die Komplexität der benötigten Modelle ständig gestiegen ist, ist parallel auch der Bedarf an Rechenleistung gewachsen. Neben den seit langem eingesetzten PowerPC-Prozessoren mit RISC-Technologie gewinnen in letzter Zeit PC-Prozessoren auch in der HIL-Technologie eine immer stärkere Bedeutung, weil sie ihnen in Bezug auf die Rechenleistung inzwischen fast ebenbürtig sind. Für die Verwendung noch komplexerer Modelle oder für die Verkopplung mehrerer einzelner Simulatoren zu einem Verbundsystem lassen sich die einzelnen Prozessoren zu einem verteilten Multiprozessorsystem zusammenschließen. Auf diese Weise werden heute HIL-Simulatoren für Steuergeräte-Verbünde zum Test von 50 oder mehr Steuergeräten realisiert ([7], [8], [10]).

Erzeugung und Erfassung von Signalen
Für die Nachbildung von Sensorsignalen und die Erfassung von Aktoransteuersignalen vom Steuergerät sind I/O-Komponenten aus der allgemeinen Mess- und Regelungstechnik oft ungeeignet. Spezielle Anforderungen ergeben sich für die Erzeugung und Erfassung von Signalen im Antriebsstrang in Echtzeit. Hier ist es die Kombination von winkelbasierten und zeitbasierten Signalen, die besondere Vorkehrungen erfordert:

- Zünd- und Einspritzsignale müssen sowohl bezüglich der Kurbelwellenposition als auch in Bezug auf die Zeitdauer genau erfasst werden.
- Signalformen für Positionsgeber müssen beliebig vorgebbar sein und ebenfalls mit einer hohen Genauigkeit winkelsynchron generiert werden.
- Klopfsignale, Ionenstrom- oder zukünftig Zylinderinnendrucksensorsignale müssen zeitbasiert und teilweise auch winkelbasiert generiert werden. Besondere Charakteristiken der Signale (Dämpfung, Abhängigkeiten von der aktuellen Motorlast usw.) müssen dabei online im Simulationsraster geändert werden können.

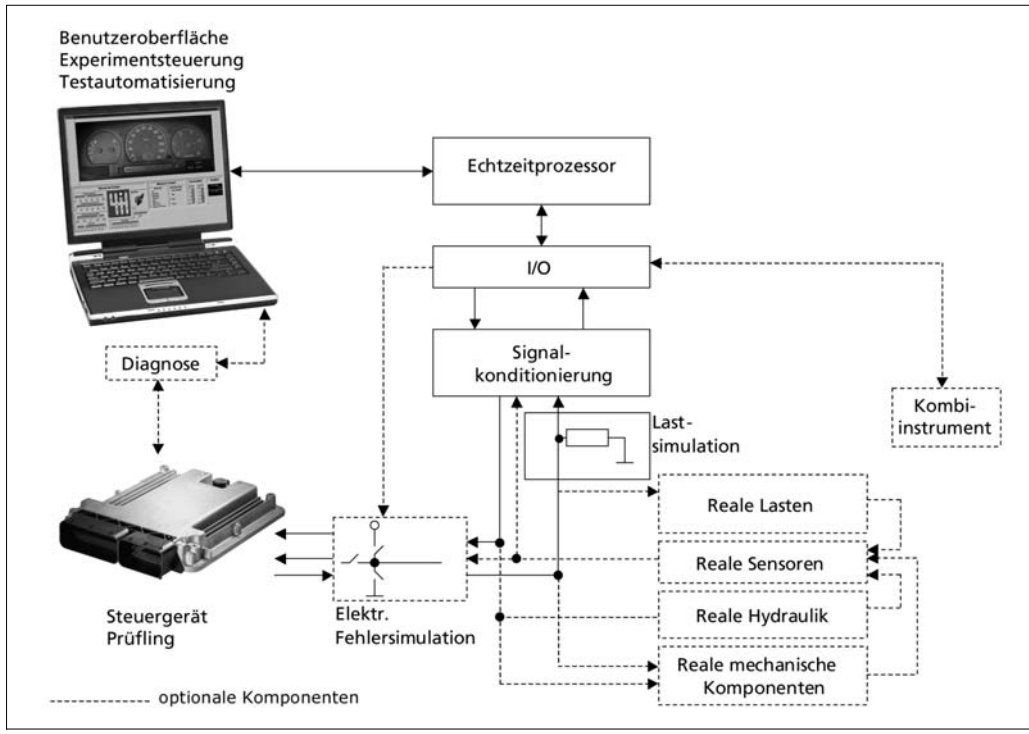


Bild 17-13: Komponenten eines HIL-Simulators

Mit speziellen Hardware-Architekturen (FPGA oder Slave-DSP-Prozessoren) können diese Anforderungen erfüllt werden. Auflösungen in der Größenordnung von 0,01 Grad Kurbelwelle über den gesamten Drehzahlbereich von 0 bis 30 000/min sind dabei heute kein Problem mehr.

Darüber hinaus ist die Unterstützung aller Bussysteme im Kfz aus dem Echtzeitsystem heraus grundlegend. Momentan die größte Bedeutung hat der CAN-Bus in verschiedenen Ausprägungen, zum Beispiel high-speed, low-speed, single-wire fault tolerant oder 24 V trailer. Für den HIL-Betrieb einzelner oder weniger Steuergeräte ist es zunächst wichtig, die vom Steuergerät gesendeten CAN-Botschaften in Echtzeit, typischerweise im 1-ms-Rhythmus, einzulesen. Zur Simulation der Umgebung der zu testenden Steuergeräte ist es jedoch auch bedeutsam, den CAN-Busverkehr der nicht real existierenden Steuergeräte in Echtzeit nachzubilden und mit dem Testobjekt zu koppeln (Restbussimulation). Sollen mehrere oder gar alle Steuergeräte eines CAN-Busses im Verbund getestet werden, kann es für eine Reihe von Tests erforderlich sein, CAN-Botschaften zwischen real existierenden Steuergeräten zu manipulieren. Dies kann mit Hilfe eines CAN-Fehler-Gateways realisiert werden.

Neben dem CAN finden in der Fahrzeugindustrie auch weitere Bussysteme zunehmend Verwendung. Diese müssen natürlich auch von den HIL-Systemen unterstützt werden. Außer dem LIN-Bus (besonders im Innenraum-Komfortbereich) sind das vermehrt auch zeitgesteuerte Busse wie FlexRay sowie der MOST-Bus aus dem Bereich Infotainment.

Signalkonditionierung

Die Hauptaufgabe der Signalkonditionierung besteht darin, die unterschiedlichen Signalpegel zwischen Steuergeräten und den I/O-Boards anzupassen. Eingelesene analoge Signale müssen in der Regel in ihrer Amplitude reduziert werden. Da manche echte Sensoren nur sehr kleine Signalspannungen ausgeben, muss für ihre Simulation durch das HIL-System die Ausgangsspannung des I/O-Kanals entsprechend reduziert werden, ohne dass die Auflösung verloren geht. Grundsätzlich ist besonders bei analogen Signalen darauf zu achten, dass die Signale differentiell zum Steuergerät geführt werden. Damit lassen sich Fehler aufgrund von Masseverschiebungen vermeiden. Bei digitalen Signalen muss die Signalkonditionierung die Charakteristik der realen Signale nachbilden. So müssen beispielsweise die digitalen Ausgänge des HIL-Systems zwischen Open Collector, Pull-down und Pull-up umkonfiguriert werden können. Weiterhin dient die Signalkonditionierung auch der Filterung analoger Signale sowie dem Schutz der I/O-Karten vor

Überspannung und -strom. Sehr spezielle Sensoren, zum Beispiel lineare Breitband-λ-Sonden oder LVDT-Sensoren (siehe Abschnitt 11.1), deren vollständige Nachbildung in Software schwierig ist, erfordern besondere Signalkonditionierungsschaltungen. In diesen Schaltungen wird das zum Teil komplexe Verhalten der Sensoren direkt in Hardware nachgebildet, wobei wesentliche Kenngrößen über Software in Echtzeit veränderbar sind. Bei einigen I/O-Boards, die speziell für den Einsatz in HIL-Systemen entwickelt worden sind, ist die Signalkonditionierung bereits integriert. Für den Fall der zur I/O additiven Signalkonditionierung sollte sie möglichst modular, flexibel und erweiterbar ausgeführt werden.

Lastsimulation
Jedes Steuergerät überwacht seine Ausgänge dahingehend, ob elektrische Fehler wie z.B. Kurzschlüsse oder Kabelbrüche vorliegen. Damit im Test des Normalbetriebs des Steuergeräts kein Kabelbruch erkannt wird, muss an jeden Steuergeräte-Ausgang eine passende Echtlast oder Lastsimulation angeschlossen werden. Je nach Empfindlichkeit der jeweiligen Diagnose gibt es hier verschiedene Ansätze: Im einfachsten Fall, der heute immer noch der überwiegende Fall ist, reicht es, eine im Vergleich zur realen Last hochohmige Ersatzlast anzuschließen. Dies kann oft direkt auf den so genannten Lastkarten erfolgen. Hier ist im Normalfall auch keine Kühlung erforderlich. Überprüft das Steuergerät an einem kritischen Ausgang auch den fließenden Strom, so muss im HIL-Simulator zumindest eine elektrisch äquivalente Last verbaut und angeschlossen werden. Das kann beispielsweise ein auf einem Kühlkörper montierter Lastwiderstand sein. In besonderen Fällen ist es notwendig, die realen Lasten zu verbauen. Ein Beispiel hierfür sind stromgeregelte Hydraulikventile, deren Induktivität nur schwer nachzubilden ist. Die Ansteuersignale werden vom HIL-Simulator in der Regel direkt an der Last vermessen und als Eingangsgrößen des Echtzeitsimulationsmodells verwendet. In Fällen von stromgeregelten Lasten muss im Simulator der Laststrom mit einem geeigneten Wandler (Hallsensor oder Shunt-Widerstand) zunächst in eine Spannung gewandelt und gefiltert werden, bevor er eingelesen werden kann.

Elektrische Fehlersimulation
Neben der Untersuchung der Steuergeräte-Funktionen im Normalbetrieb ist die Untersuchung der Diagnosefunktionen ein wichtiger Einsatzbereich für die HIL-Simulation. Hierbei wird zunächst geprüft, ob ein Fehler, beispielsweise ein defekter Sensor, richtig erkannt wird. Anschließend wird getestet, ob das Steuergerät die zugehörige Notlauffunktion aktiviert, also zum Beispiel einen Ersatzwert für den ausgefallenen Sensor berechnet. Voraussetzung für diese Tests ist eine leistungsfähige, automatisierbare elektrische Fehlersimulation. Sie sollte zumindest folgende Fehlerfälle abdecken können: Leitungsunterbrechung, Kurzschluss nach Masse, Kurzschluss zur Batterie sowie Kurzschlüsse zwischen Steuergeräte-Pins. Für die Nachbildung schlechter Kontakte kann es erforderlich sein, veränderbare Leitungswiderstände zwischen zwei Steuergeräte-Pins oder zwischen einem Steuergeräte-Pin und einer Referenz (Masse, Batteriespannung) zu schalten. Wackelkontakte an Schaltereingängen können mit hochfrequenten Pulsmustern simuliert werden.

Steuergeräte mit integrierter Sensorik
Manche Steuergeräte haben Sensoren im Steuergeräte-Gehäuse integriert, die für einen Betrieb des Steuergeräts stimuliert werden müssen. Beispielsweise gibt es Getriebesteuergeräte, die direkt im Getriebe verbaut sind. Sie detektieren die Eingangs- und Ausgangsdrehzahlen mit integrierten Hall-Sensoren. In Steuergeräten für die elektrische Parkbremse sind Neigungssensoren integriert. In diesen Fällen können Sensorsignale vom HIL-System nicht direkt elektrisch stimuliert werden. Das Gleiche gilt für Sensoren, die mit dem Steuergerät über ein geheimes Protokoll kommunizieren. Dies ist zum Beispiel bei Crash-Sensoren oder beim Geschwindigkeitssensor eines Fahrtenschreibers der Fall.
In diesen Fällen gibt es zwei Möglichkeiten, diese Sensoren in der HIL-Simulation zu behandeln:
Eine Möglichkeit ist, den Sensor mit der echten physikalischen Größe zu stimulieren. So kann ein integrierter Drehzahlsensor durch ein magnetisches Wechselfeld angeregt werden, das durch Elektromagnete erzeugt wird. Ist dieses nicht möglich, muss durch ein Öffnen („Cracken“) der Steuergeräte oder der Sensoren auf die elektrische Schnittstelle zugegriffen werden. Dies ist beispielsweise bei den Beschleunigungssensoren eines Airbag-Steuergeräts ein geeigneter Weg.

17.4.3.2 Software

Von der Software-Seite her betrachtet unterscheidet man beim HIL-System die Echtzeit-Software auf der einen und die Experimentier- und Testautomatisierungssoftware auf der anderen Seite (**Bild 17-14**).

Echtzeit-Software allgemein
Die Software, die auf dem Echtzeitprozessor (Abschnitt 17.4.3.1) ausgeführt wird, ist üblicherweise C-Code, der aus blockorientierten grafischen Simulationsumgebungen heraus mit automatischer Code-Generierung erzeugt wird. In der Regel laufen die Programme in einem Echtzeit-Betriebssystem oder zumindestens in einem speziellen Echtzeit-Rahmen (Echtzeit-Frame). Dieser sollte echtes Multitasking bieten sowie Timer-, Software- oder Hardware-Interrupts unterstützen.

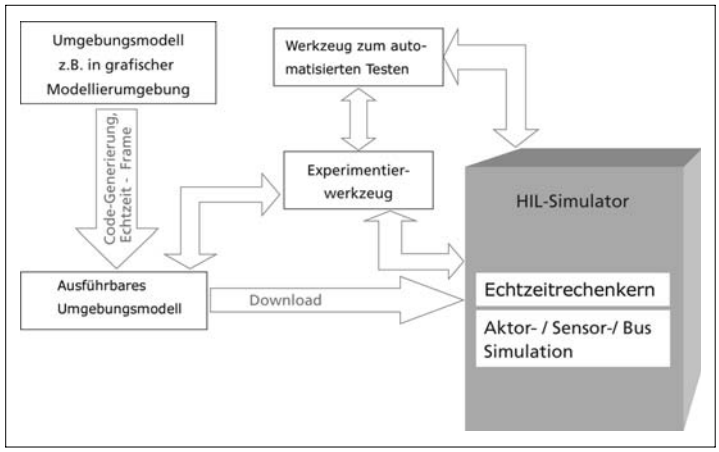


Bild 17-14:
Software-Komponenten in der HIL-Simulation. Die unbeschrifteten Pfeile markieren einen Datenaustausch

Neben den eigentlichen Streckenmodellen werden auf dem Echtzeitprozessor auch die kompletten I/O-Funktionen inklusive der Restbussimulation berechnet. Selbstverständlich lässt sich auch die gesamte Inter-Prozessor-Kommunikation für einen eventuellen Multiprozessorbetrieb über eine grafische Benutzeroberfläche für den Anwender transparent implementieren [5].

Echtzeitmodelle
Die Anforderungen an mathematische Streckenmodelle sind je nach Anwendungsbereich sehr verschieden. Für Funktions- und Diagnosetests sollten die Modelle so genau wie nötig sein, um das Steuergerät in allen Betriebszuständen mit plausiblen Signalen zu versorgen, das heißt, alle Regelkreise so zu schließen, dass die Steuergeräte-Diagnose keine Fehler erkennt. Mit steigender Empfindlichkeit der Diagnosefunktionen wachsen auch die Anforderungen an die Modellkomplexität. So erfordern die OBD-II-Funktionen der neuesten Motorsteuergeräte eine relativ genaue dynamische Modellierung der abgasführenden Komponenten inklusive Temperaturen und Drücken sowie ein präziseres Katalysatormodell, als es noch vor wenigen Jahren notwendig war.
Noch komplexer werden die Modelle, wenn am HIL-Simulator nicht nur Funktionen oder die Eigendiagnose getestet werden sollen, sondern auch eine Vorapplikation der Steuergeräte-Parameter durchgeführt werden soll. In diesem Fall steigt natürlich auch der Aufwand für die Parametrierung der Modelle.

Bediensoftware
Die manuelle Bedienung des Testsystems, die Visualisierung und Speicherung der Echtzeitgrößen sowie das Verstellen von Modellparametern erfolgt gewöhnlich vom Host-PC aus. Speziell für den Test fahrdynamischer Funktionen oder der Adaptive-Cruise-Control-Funktionalität reichen rein numerische Anzeigen oder einfache Plotterinstrumente nicht mehr aus. Hier bietet ein 3D-Echtzeitanimationswerkzeug eine wesentlich bessere Möglichkeit, die Bewegung des Fahrzeugs zu visualisieren und auf einen Blick zu beurteilen.
Erst durch eine systematische und vollständige Automatisierung aller ihrer Komponenten kann die HIL-Simulation ihren vollständigen Nutzen ausspielen (Abschnitt 17.5.8).

17.4.4 Einbindung in den Elektronik-Entwicklungsprozess

Der erfolgreiche Test am HIL-Simulator ist inzwischen für viele Fahrzeughersteller Bestandteil der Freigabevoraussetzung des Fahrzeugs.

Aufgabenverteilung zwischen Fahrzeughersteller und Zulieferer
Bevor ein neuer Steuergeräte-Stand durch den Zulieferer an den Fahrzeughersteller geliefert wird, ist dieser Stand durch den Zulieferer in einem Komponententest abzusichern. Die anschließende Integration der von externen Lieferanten entwickelten Komponenten in ein (Sub-)System ist eine kritische Phase im Entwicklungsprozess. Missverständnisse und fehlerhafte Spezifikationen können dazu führen, dass Fehler erst beim Fahrzeughersteller während der Tests des Sub- oder Gesamtsystems gefunden werden [2]. Eigentliches Ziel des Systemtests beim Fahrzeughersteller ist jedoch lediglich die Absicherung von Funktionen, die über mehrere Bussysteme hinweg implementiert sind. Denn diese verteilten Funktionen – zentraler Algorithmus, Sensoren und Aktoren sowie Bedien- und Anzeigeelemente sind in verschiedenen Steuergeräten untergebracht – können nur im Zusammenspiel aller beteiligten Steuergeräte abgesichert werden.

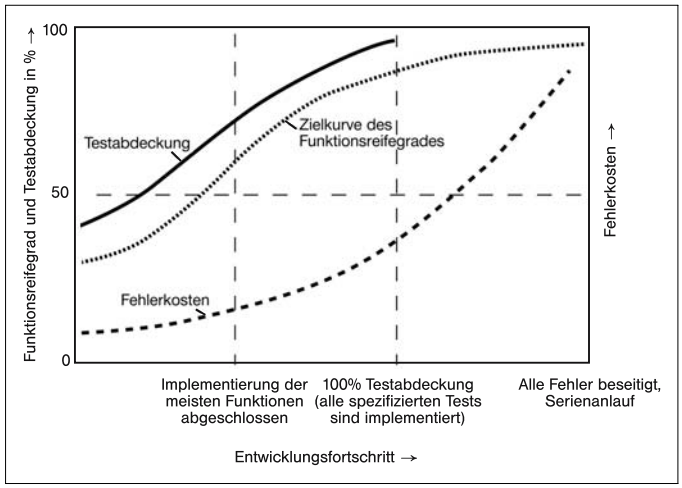


Bild 17-15:
Funktionsreifegrad und Testabdeckung im Elektronik-Entwicklungsprozess

Setzen Zulieferer und Fahrzeughersteller kompatible Testsysteme ein, erlaubt dies, Tests und Testergebnisse zu vergleichen und auszutauschen. Gemeinsame Konzepte vermeiden unnötige Redundanzen und führen zu erhöhter Testfallabdeckung durch klare Aufteilung von Zuständigkeiten.

Ziele

Die Kosten je Fehler steigen, je später im Entwicklungsprozess der Fehler entdeckt wird (**Bild 17-15**). Je früher die Probleme erkannt und beseitigt werden, desto schneller kann ein hoher Reifegrad des Fahrzeugs während der Entwicklung erreicht werden. Durch den konsequenten Einsatz von HIL-Testsystemen zum frühest möglichen Zeitpunkt im Entwicklungsprozess kann ein möglichst hoher Reifegrad erreicht werden [2].

Voraussetzung für eine schnelle Fehlerbehebung ist die enge Verzahnung von Entwicklung, Inbetriebnahme und Verifikation der Steuergeräte. So müssen zum Beispiel bei neuen Software-Ständen Informationen zu den implementierten Änderungen bekannt gegeben werden, so dass die Tests bei Bedarf entsprechend erweitert werden.

HIL-Prozess

Für eine effiziente Einbindung in den Gesamtentwicklungsprozess ist die HIL-Simulation als Teilprozess zu betrachten. Dieser Teilprozess beginnt zunächst mit einer Spezifikation des benötigten Systems auf Basis der zu dem Zeitpunkt vorliegenden Kenntnis über das Fahrzeug und die verbauten Steuergeräte. Nach dem vorbereitendenden Aufbau aller steuergeräteunabhängigen Komponenten kann die Anpassung des Simulators an das Steuergerät ohne Zeitverlust erfolgen, sobald die Spezifikationen existieren. Dies beinhaltet zum Beispiel die Implementierung von Lastsimulationen. Iterationen bei geänderten Steuergeräte-Spezifikationen bleiben möglich. Nach der Inbetriebnahme des Simulators mit der vorgesehenen Anzahl von Steuergeräten ist der Simulator funktionsbereit und die Prüflinge können getestet werden.

Bei einer Steuergeräte-Aktualisierung erfolgt (nach eventuell vorangegangener Adaption des Simulators) eine erneute Inbetriebnahme des Simulators, bevor Regressionstests durchgeführt werden.

Testablaufpläne und Testfälle werden basierend auf der Steuergeräte- oder Funktionsspezifikation entwickelt. Neben den Lastenheften dienen vor allem bereits implementierte Testfälle von Vorgängermodellen sowie die Erfahrung der Entwickler als Grundlage. Rückmeldungen aus der Erprobung oder aus Werkstätten fließen ebenfalls ein. Erst in der zweiten Hälfte des Entwicklungszyklus wird eine hohe Testabdeckung erreicht (**Bild 17-15**).

Integration von HIL-Simulationen in den Entwicklungs- und Testprozess

Die Fahrzeug-Serienentwicklung beginnt in der Regel spätestens 30 Monate vor dem geplanten Produktionsstart. Die Software-Entwicklung beginnt oft schon mit der Funktionsspezifikation und deshalb bereits innerhalb einer der Serienentwicklung vorgelagerten Design- und Vorbereitungsphase. Der HIL-Test ist in der Software-Entwicklung ein fester Bestandteil des Steuergeräte-Integrationstests. Die Fahrzeughersteller, die die HIL-Simulation derart in ihrem Prozess verankert haben, haben ständig eine Vielzahl von HIL-Simulatoren für den Subsystem- und Systemtest für die aktiv in der Entwicklung befindliche Produktpalette im Einsatz. Diese Systeme arbeiten 24 Stunden an sieben Tagen in der Woche.

Bild 17-15 zeigt den angestrebten Funktionsreifegrad über die Dauer eines Entwicklungszyklus. Der Funktionsreifegrad beschreibt den Prozentsatz der getesteten Spezifikationsanforderungen, die einen Test

erfolgreich bestanden haben. Bei Serienanlauf dürfen keine Fehler mehr vorhanden sein (100 % Reifegrad). Nachdem alle spezifizierten Tests bezogen auf die definierten Anforderungen implementiert sind (100 % Testabdeckung), kann eine hohe Automatisierung erreicht werden. Nun werden alle Fehler sukzessive eliminiert, bis zum Zeitpunkt des Serienanlaufs keine Fehler mehr bekannt sind.

Ankopplung an das Anforderungsmanagement
Werkzeuge wie DOORS oder TestDirector werden häufig genutzt, um Anforderungen und Spezifikationen zu sammeln und zu verwalten. Testspezifikationen und Ergebnisse können ebenfalls sehr gut damit verwaltet werden. Gleiches gilt für Testpläne und Testausführung aus den Werkzeugen heraus. Eine direkte Anbindung der HIL-Testumgebung ist deshalb ein konsequenter Schritt innerhalb der Prozessintegration.

Literatur zu Abschnitt 17.4

[1] *Lamberg, K.*: Methodik zur systematischen Bereitstellung von HIL-Testsystemen für Kfz-Steuergeräte, Dissertation, Herbert Utz Verlag, 2001.

[2] *Honisch, A.*; *Hutter, A.*; *Schmid, H.*: Vollautomatisierter Test von Steuergeräte-Netzwerken, ATZ-MTZ extra Mercedes-Benz A-Klasse, 2004.

[3] *Schäuffele, J.*; *Zurawka, T.*: Automotive Software Engineering, ATZ-MTZ-Fachbuch, Wiesbaden, 2003.

[4] *Köhl, S.*; *Jegminat, D.*: How to Do Hardware-in-the-Loop-Simulation Right, SAE-Paper No. 2005-01-1657, Detroit, USA, 2005.

[5] Solutions for Control, Catalog 2005, www.dspace.de, Paderborn, Deutschland, 2004

[6] *Schütte, H.*; *Plöger, M.*; *Diekstall, K.*: *Wältermann, P.*; *Michalsky, Th.*: Testsysteme im Steuergeräte-Entwicklungsprozess, Automotive Electronics, S. 16–21, März 2001

[7] *Gehring, J.*; *Schütte, H.*: Automated Test of ECUs in a Hardware-in-the-Loop Test Bench for the Validation of Complex ECU Networks, Proc. of the SAE World Congress, Detroit, USA, März 2002

[8] *Köhl, S.*; *Lemp, D.*; *Plöger, M.*: Steuergeräte-Verbundtests mittels Hardware-in-the-Loop Simulation, S. 948–955, ATZ, Wiesbaden, Deutschland, Oktober 2003

[9] *Wältermann, P.*; *Michalsky, T.*; *Held J.*; Hardware-in-the-Loop Testing in Racing Applications, Proc. of the SAE Motorsports Engineering Conference and Exhibition, Paper Nummer: 2004-01-3502, Dearborn, USA, Dezember 2004

[10] *Wältermann, P.*; *Schütte, H.*; *Diekstall, K.*; Hardware-in-the-Loop Test verteilter Kfz-Elektroniksysteme; S. 416-425, ATZ, Wiesbaden, Deutschland, Mai 2004

17.5 Software-Testen

17.5.1 Grundbegriffe des Testens

Testen ist eine Entwicklungsaktivität mit dem Ziel, Fehler zu finden. Somit dient Testen im Allgemeinen dazu, die Qualität eines Systems zu erhöhen. Testen ist erlernbar und kann auch auf sehr komplexe Systeme angewendet werden. Obwohl Testen recht aufwändig sein kann, zum Beispiel bei den Anschaffungskosten für ein HIL-Simulationssystem, stellt sich das Kosten-Nutzen-Verhältnis in der Regel sehr positiv dar. Nichtsdestotrotz kann die Korrektheit oder Fehlerfreiheit eines Systems damit nicht vollständig nachgewiesen werden.
Entsprechend dem jeweiligen Testziel wird zwischen Validierung und Verifikation unterschieden. Validierung ist der Test gegenüber den Anforderungen an das zu entwickelnde System. Anforderungen sind das Ergebnis der Anforderungsanalyse (Abschnitt 17.1). Anforderungen können funktional, also auf den eigentlichen Zweck des Systems ausgerichtet, oder nichtfunktional sein, z.B. hinsichtlich Performance, Echtzeit, Speicherverbrauch usw. Die Verifikation meint den Test eines Objekts (Testobjekts) gegenüber seiner Spezifikation. Das Testobjekt ist dabei Teil des zu entwickelnden Gesamtsystems. Die Spezifikation beschreibt die technischen Anforderungen an das Objekt. Das sind häufig die Schnittstellen und das gewünschte Verhalten des Objekts an diesen Schnittstellen. Das Testobjekt kann auf allen Systemebenen (Unit, Modul, Komponente, Subsystem) liegen, je nachdem, auf welcher Ebene die Spezifikation vorliegt und überprüft wird. Die Verifikation selbst ist jedoch kein Nachweis der allgemeinen Fehlerfreiheit gegenüber den Systemanforderungen, da bereits die Spezifikation fehlerhaft sein kann.

17.5.2 Klassifikation der Testmethoden

Testmethoden lassen sich nach dem in **Bild 17-16** gezeigten Schema klassifizieren. Grundsätzlich wird zwischen statischem und dynamischem Testen unterschieden. Der wesentliche Unterschied besteht darin, dass das Testobjekt beim dynamischen Testen ausgeführt wird, beim statischen jedoch nicht. Eine weitere Methodenklasse bildet die formale Verifikation.
Im Folgenden wird nur auf das dynamische Testen und die formale Verifikation eingegangen, da diese im Rahmen der modellbasierten Entwicklung primär zum Einsatz kommen.

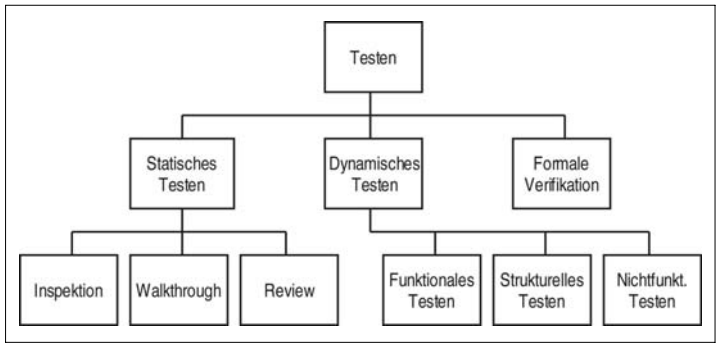


Bild 17-16:
Klassifikation von Testmethoden

17.5.3 Funktionales Testen

Beim funktionalen Testen, also dem Testen des funktionalen Verhaltens des Testobjekts, wird das Testobjekt üblicherweise mit vorgegebenen Signalverläufen (Testdaten) stimuliert und das Antwortverhalten wird erfasst und hinsichtlich des Sollverhaltens (Erwartungswerte) bewertet. Dabei werden keinerlei Informationen über den internen Aufbau des Testobjekts benötigt und verwendet. Daher wird funktionales Testen auch als Black-Box-Testen bezeichnet. Beispiele für die Anwendung des funktionalen Testens sind Requirements-Based Testing, Äquivalenzklassentests (Klassifikationsbaummethode) und Back-to-Back-Tests.

Klassifikationsbaummethode

Die Klassifikationsbaummethode ist eine Black-Box-Methode, bei welcher der Eingaberaum des Testobjekts (Menge der Eingabewerte und -kombinationen) unter bestimmten Aspekten zerlegt wird. Die entstehenden Partitionen, die sogenannten Klassifizierungen, werden wiederum in Äquivalenzklassen unterteilt. Ziel der Partitionierung ist es, die einzelnen Klassen so zu wählen, dass sie sich einheitlich (uniform) in Bezug auf die Aufdeckung potenzieller Fehler verhalten. Das heißt, das Testobjekt verhält sich für alle Werte einer Klasse entweder korrekt oder fehlerhaft (Uniformitätshypothese) [1].

Bild 17-17 zeigt einen Klassifikationsbaum. Der Name des Testobjekts (hier „VDC“) bildet die Wurzel des Baums, die Eingangssignale definieren die Klassifikationen unterhalb des Wurzelknotens. Darunter finden sich die Äquivalenzklassen. Basierend auf der so erfolgten Partitionierung des Eingaberaums lassen sich nun Testsequenzen definieren. Die Testsequenzen ergeben sich durch gezielte Kombination einzelner Klassen: Jede Zeile der Kombinationstabelle beschreibt einen Testschritt innerhalb der Testsequenz.

Grenzwertanalyse

Die Grenzwertanalyse hat zum Zweck, das Testobjekt mit Daten zu stimulieren, die unzulässige oder andere kritische Eingabegrößen darstellen könnten. Basierend auf der Zerlegung des Eingaberaums mit Hilfe der Klassifikationsbaummethode wird das Testobjekt beispielsweise mit den Werten, die die Äquivalenzklassen begrenzen, und Werten in deren Nähe stimuliert. Auch die Stimulation des Testobjekts mit Werten außerhalb des zulässigen oder spezifizierten Wertebereichs gehört zur Grenzwertanalyse. Auf diese Weise lässt sich das Verhalten des Testobjekts in Bereichen testen, die bei der Spezifikation übersehen wurden oder die typische Fehlerquellen darstellen.

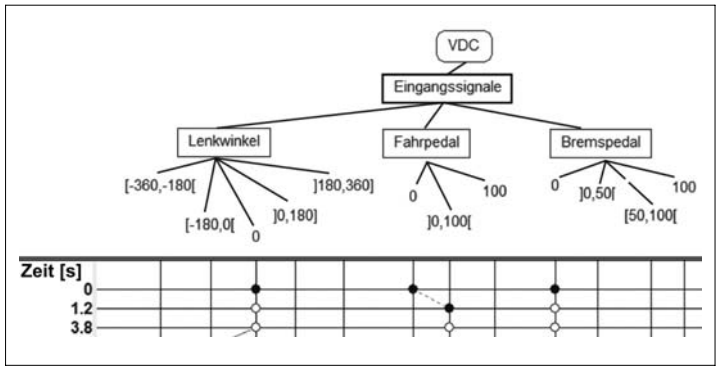


Bild 17-17:
Klassifikationsbaum für das Testobjekt „VDC“

Back-to-Back-Tests
Back-to-Back-Tests nutzen aus, dass die Spezifikation eines zu testenden Systems – z.B. Programmcode – in einer modellbasierten Entwicklung als ausführbares Simulationsmodell vorliegt. Das Modell und der Programmcode werden mit denselben Testdaten stimuliert. Das Ausgangsverhalten des Programmcodes bezogen auf die Testdaten lässt sich dann unmittelbar mit dem der ausführbaren Spezifikation vergleichen (siehe **Bild 17-18**). Die Tatsache, dass das Verhalten des Programmcodes direkt mit dem der Spezifikation verglichen wird, macht deutlich, dass es sich bei der Anwendung von Back-to-Back-Tests in der Regel um eine Verifikation handelt.

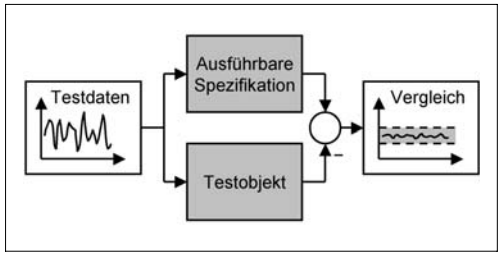


Bild 17-18: Prinzip des Back-to-Back-Tests

Für den Vergleich lassen sich unterschiedliche Kriterien heranziehen. Meistens wird überprüft, ob sich das Differenzsignal innerhalb eines definierten Toleranzbandes bewegt. Dabei können sowohl absolute als auch relative Toleranzen von Interesse sein. Modernere Auswertealgorithmen können sogar zeitliche Verschiebungen oder Amplitudenabweichungen getrennt voneinander erkennen und bewerten [2].

17.5.4 Strukturelles Testen

Im Gegensatz zum funktionalen Testen werden beim strukturellen Testen Informationen über den internen Aufbau (Struktur) und die Eigenschaften des zu testenden Systems berücksichtigt und für die Entwicklung von Testfällen verwendet. Aus diesem Grund wird strukturelles Testen auch als White-Box-Testen bezeichnet. Durch die Kenntnis des internen Aufbaus eines Systems ist es zum Beispiel möglich, gezielt Testfälle zu definieren, die dazu führen, dass bestimmte Strukturelemente des Testobjekts wie Anweisungen und Verzweigungen durchlaufen werden. Dadurch lassen sich bestimmte Teile des Testobjekts gezielt anregen und testen.

Modell- und Code-Überdeckung
Umgekehrt ist es möglich, während der Durchführung von Tests die Häufigkeit zu erfassen, mit der bestimmte Systemteile durchlaufen wurden. Werden derartige Messungen auf Modellebene durchgeführt, spricht man von Modellüberdeckung (Model Coverage); auf Programmcode-Ebene von Code-Überdeckung (Code Coverage, Abschnitt 17.3.3). Mit Hilfe von Modell- und Code-Überdeckungsmessungen lassen sich zum Beispiel Modell- und Programmteile identifizieren, die trotz einer Vielzahl von Tests gar nicht ausgeführt wurden. Für diese Teile können dann gezielt weitere Testfälle definiert werden. Im Einzelfall kann es sogar sein, dass bestimmte Modell- und Programmteile entfernt werden können, wenn sie nicht erreichbar und somit überflüssig sind (Dead Code).

Automatische Testvektorgenerierung
Bei der automatischen Testvektorgenerierung werden Informationen zum internen Aufbau des Testobjekts systematisch ausgenutzt, um automatisiert Testfälle zu erzeugen, so dass ein bestimmtes Kriterium erreicht wird. Beispiele für derartige Kriterien sind vorgegebene Überdeckungsmaße (Model oder Code Coverage) oder das Erreichen eines bestimmten Systemzustands (Drive to State oder Drive to Transition, Drive to Property). Dabei besteht ein Testfall immer aus Anregungssituationen und erwartetem Ausgangsverhalten.
Der Einsatz der automatischen Testvektorgenerierung ist insbesondere in Zusammenhang mit Back-to-Back-Tests (s.o.) sinnvoll, bei denen das Ausgangsverhalten des Testobjekts mit den Ausgangssignalen seiner Spezifikation für eine Vielzahl automatisch generierter Anregungssituationen verglichen werden kann. Auf diese Weise kann beispielsweise das Verhalten manuell oder automatisch erzeugter Codes mit dem Verhalten des zugrunde liegenden Simulationsmodells verglichen werden. In diesen Fällen spricht man von dem Modell auch als einem „Testorakel“.

17.5.5 Nichtfunktionales Testen

Nichtfunktionales Testen bedeutet, die nichtfunktionalen Anforderungen an ein System zu testen. Nichtfunktionale Anforderungen sind Eigenschaften, die ein System erfüllen muss, die jedoch nicht primär auf die eigentliche Funktion des Systems – den Systemzweck – ausgerichtet sind. Nichtfunktionale Anforderungen gemäß ISO/IEC 9126 sind zum Beispiel Anforderungen hinsichtlich Zuverlässigkeit, Benutzbarkeit, Effizienz, Änderbarkeit und Übertragbarkeit. Möglichkeiten, nichtfunktionale Anforderungen zu testen, sind beispielsweise Stresstests: Dabei wird untersucht, wie sich ein Steuergerät bei sehr hoher Last (hohes Datenaufkommen) auf dem CAN-Bus verhält.

17.5.6 Formale Verifikation

Während die Validierung und die Verifikation im obigen Sinne bestimmte Anregungssituationen erzeugen, um bestimmte Systemeigenschaften zu testen, dient die formale Verifikation dem Nachweis bestimmter Eigenschaften des Systems. Damit ist es zum Beispiel möglich, die Korrektheit eines Systems, das heißt, die Erfüllung der Anforderungen im mathematischen Sinne zu beweisen. Der Unterschied zum dynamischen Testen liegt demnach in der Vollständigkeit und Konsistenz der Methode.
Für die formale Verifikation existieren zwei grundlegende Ansätze: Model Checking und Theorem Proving, die sich unter anderem in dem Grad der möglichen Automatisierung, den notwendigen Vorkenntnissen und der Art der verifizierbaren Modelle unterscheiden. Notwendig für die formale Verifikation ist, dass das zu testende System in einer formalen Repräsentation vorliegt. Beispiele für formale Repräsentationen sind Blockdiagramme, Zustandsautomaten oder auch Programm-Code.
In der Praxis der Entwicklung von Software im Automobil ist die Anwendung der formalen Verifikation noch eher selten. Sie wird vorwiegend bei der Entwicklung sicherheitskritischer Software gemäß ISO 61508 (Abschnitt 17.1.2) eingesetzt.

17.5.7 Testen in der Funktionsentwicklung

Für das Testen innerhalb der Funktionsentwicklung für Automobilelektronik stehen eine Reihe von Werkzeugen zur Verfügung, welche die unterschiedlichen Testansätze und -methoden unterstützen. So existiert beispielsweise für die modellbasierte Entwicklung auf Basis von Simulink, Stateflow und TargetLink das Testwerkzeug MTest. Mit Hilfe des darin enthaltenen Klassifikationsbaumeditors können Funktions- und Reglermodelle per Äquivalenzklassentest validiert werde [3]. Die Unterstützung der unterschiedlichen Testmodi von der Model-in-the-Loop- über die Software-in-the-Loop- bis hin zur Processor-in-the-Loop-Simulation (Abschnitt 17.3) ermöglicht den direkten Vergleich der Testobjekte in den unterschiedlichen Prozessphasen (Back-to-Back-Tests, Verifikation).
Während der Testdurchführung auf Modellebene oder auf Code-Ebene können unterschiedliche Modell- und Code-Überdeckungsmaße ermittelt werden, die für das White-Box-Testen notwendig sind. **Tabelle 17-3** enthält exemplarische Beispiele für das Testen in der Funktionsentwicklung zum Einsatz kommender Werkzeuge und deren methodischer Zuordnung.

Tabelle 17-3: Methodische Zuordnung von Testwerkzeugen (Beispiele)

Werkzeug	Methodenzugehörigkeit
MTest	Äquivalenzklassentest, Grenzwertanalyse, Requirements-based Testing, Black-Box-Testing
Model-, Software-, Processor-in-the-Loop	Back-to-Back-Tests
Simulink Model Coverage	Strukturelle Tests (White-Box-Testing)
TargetLink Code Coverage	Strukturelle Tests (White-Box-Testing)
Embedded Validator	Formale Verifikation

17.5.8 Test von Steuergeräten

Für den Test von Steuergeräten und Steuergeräte-Prototypen innerhalb der Entwicklung hat sich die Hardware-in-the-Loop-Technologie (Abschnitt 17.4) mittlerweile etabliert. Immerhin lassen sich mit Hilfe der HIL-Simulation bis zu 90 % aller im Fahrversuch auftretenden Fehler nachbilden. Die zentrale Herausforderung für einen effizienten Einsatz von HIL-Systemen liegt dabei in der automatisierten Durchführung von Tests. Die dazu notwendigen Testspezifikationen werden ausgehend von den Steuergeräte-, Funktions- und Systemspezifikationen entwickelt [4]. Beim HIL-Testen von Steuergeräten handelt es sich daher üblicherweise um Requirements-Based Testing und somit um funktionale, also Black-Box-Tests.

Rollenverteilung
Um die Komplexität im Spannungsfeld Steuergeräte-Entwicklung – HIL-Simulation zu beherrschen, existiert häufig eine klar getrennte Aufgabenverteilung zwischen System- bzw. Funktionsverantwortlichem, Testsystemexperten, Testentwickler und Testanwender. Der Testsystemexperte ist oft schon an der Projektierung des Testsystems und dessen Inbetriebnahme beteiligt. Häufig ist er auf Seiten des HIL-Testsystemlieferanten zu finden. Zunehmende Bedeutung bekommen die Rollen des Testentwicklers und des Testanwenders. Der Testentwickler erarbeitet gemeinsam mit dem System- oder Funktionsverantwortlichen die Testspezifikation und den Testplan. Darauf basierend erstellt der Testentwickler die eigentliche Testimplementierung und stellt diese für die Testdurchführung bereit. Die Durchführung der Tests obliegt dem Testanwender. Für eine bestimmte Testaufgabe stellt dieser die notwendigen Tests zusammen, parametriert sie gegebenenfalls und führt sie durch.

Testbeschreibungen
In Verbindung mit der HIL-Simulation sind die Testaufgaben sehr vielfältig und erfordern daher flexible und vielseitige Möglichkeiten der Testbeschreibung. Aus diesem Grund sind in den vergangenen Jahren eine ganze Reihe von Werkzeugen und Lösungen entstanden, die unterschiedliche Formen der Testbeschreibung ermöglichen.
Sehr weite Verbreitung im Zusammenhang mit dem automatisierten Testen mit Hilfe der HIL-Simulation finden Skriptsprachen. Für die Skriptsprache Python [5] existiert beispielsweise eine Vielzahl von Bibliotheken (APIs) zur Automatisierung von HIL-Simulatoren. Testskripts können in Python implementiert werden und greifen über diese APIs auf die verschiedenen Testsystemkomponenten wie Echtzeitmodell, Fehlersimulationshardware, Diagnose- und Applikationssystem zu.
Die Anwendung von Skriptsprachen ist äußerst flexibel, erfordert jedoch einen vergleichsweise hohen Einarbeitungsaufwand. Daher sind Skriptsprachen häufig nur für Testentwickler geeignet, nicht für Testausführer. Aus diesem Grund sind verschiedene Lösungen entstanden, die einen einfacheren Benutzerzugang für die spezifische Testart bereitstellen sollen. Auf diese Weise muss sich ein Testausführer nicht mit der Skriptsprache an sich befassen, sondern kann seine Tests über Benutzeroberflächen mit entsprechenden Eingabemöglichkeiten komfortabel parametrieren.
Doch auch die Testentwicklung muss heute nicht mehr ausschließlich skriptbasiert erfolgen. Moderne Werkzeuge wie AutomationDesk bieten zusätzlich die Möglichkeit der grafischen Testbeschreibung, wie sie auch aus der Unified Modelling Language (UML, [6]) bekannt ist.

Regressionstests
Die Häufigkeit von Änderungen, die sich durch eine Vielzahl neuer Hardware- und Software-Stände der Einzelsteuergeräte ergibt, macht es notwendig, Tests wiederholt durchzuführen. Diese wiederholten Tests werden als Regressionstests bezeichnet. Mit Hilfe von Regressionstests lässt sich feststellen, ob neue Hardware- oder Software-Versionen zusätzliche, bisher nicht aufgetretene Fehler enthalten, die sich beispielsweise durch die Behebung älterer Fehler eingeschlichen haben. Gerade die exakte und automatisierbare Reproduktion von Testszenarien macht den wesentlichen Effizienzgewinn durch Einsatz der HIL-Simulation aus.

17.5.9 Testmanagement

Schon für einzelne Funktionen, aber auch für ein einzelnes Steuergerät sind in der Regel Hunderte bis Tausende von Tests durchzuführen, beim Test von Steuergeräte-Verbünden geht die Zahl der Tests in die Zehntausende. Diese Tests laufen normalerweise innerhalb einiger Stunden oder einiger Tage automatisch ab (häufig nachts). Diese immense Anzahl von Tests macht zusammen mit den Testdaten und den Testergebnissen ein Testmanagement notwendig [7]. Ein solches Testmanagement schafft einen einheitlichen und strukturierten Zugang zu der Vielzahl von Tests. Gleichzeitig erfolgt durch ein Testmanagement eine Zuordnung der Testdaten und der erzeugten Testergebnisse zu den Tests. Auf Basis der Testergebnisse lassen sich schließlich Testreports in verschiedenen Formaten und mit unterschiedlichen Detaillierungsgraden erzeugen. Solche Reports dienen der nachhaltigen Dokumentation von Testergebnissen, zum Beispiel für entsprechende Freigabeentscheidungen.

Literatur zu Abschnitt 17.5

[1] *Grochtmann, M.*; *Grimm, K.*: Classification Trees for Partition Testing. Software Testing, Verification and Reliability, 3, 63–82, 1993.
[2] *Wiesbrock, H.-W.*; *Lim, M.*: Automatische Signalanalyse – Bessere Qualität durch automatisierte Testauswertung. Elektronik Automotive 2/2004.
[3] *Lamberg, K.*; *Beine, M.*; *Conrad, M.*; *Eschmann, M.*; *Fey, I.*; *Otterbach, R.*; Model-based testing of embedded automotive software using MTest. SAE-Paper No. 2004-01-1593, Detroit, USA, 2004.
[4] *Sax, E.*; *Schmerler, S.*: Qualitätssteigerung in der KFZ-Steuergeräte-Entwicklung. Test Guide Design & Verification, 2004.
[5] Python, www.python.org.
[6] UML Resource Page, www.uml.org.
[7] *Lamberg, K.*; *Richert, J.*; *Rasche, R.*: A New Environment for Integrated Development and Management of ECU Tests. SAE 2003-01-1024, Detroit, USA, 2003.

17.6 Steuergeräte-Applikation

17.6.1 Einführung

Im Entwicklungsprozess moderner Steuergeräte stellt die Bedatung der Steuergeräte-Software einen wesentlichen Meilenstein dar. Dieser Vorgang wird als Steuergeräte-Applikation oder -Kalibrierung bezeichnet und erfolgt in der Regel erst relativ spät im Entwicklungsprozess, wenn der Motor oder das Fahrzeug bereits als Prototyp zur Verfügung steht.
Moderne Kraftfahrzeuge werden in zunehmendem Maße von elektronischen Steuer- und Regelsystemen bestimmt. Steigende Anforderungen an das Verbrauchs-, Leistungs- und Emissionsverhalten sowie höhere Kundenerwartungen bezüglich Komfort, Sicherheit und Variantenvielfalt führen zur Entwicklung immer komplexerer Steuergeräte-Algorithmen und damit zu einer rasanten Zunahme des Software-

Umfangs. In gleichem Maße gestaltet sich die Applikationsaufgabe immer anspruchsvoller und aufwendiger.
Neben den eigentlichen Software-Funktionen beeinflusst die Einstellung der jeweiligen Steuer- und Regelparameter (die sogenannten Kennwerte, Kennlinien und Kennfelder) maßgeblich das Gesamtverhalten eines Fahrzeugs. Ziel der Steuergeräte-Applikation ist es, die Software-Funktionen durch Abstimmung dieser Kenngrößen an Motor- oder Fahrzeugvarianten anzupassen, so dass das Gesamtverhalten die Lastenheftvorgaben erfüllt. Die Algorithmen der Steuer- und Regelfunktionen sind dabei fest in der Steuergeräte-Software einprogrammiert, lediglich die Kenngrößen, auch Applikationsparameter genannt, werden in dieser Entwicklungsphase in ihrem Wert verändert.
Auf Steuergeräteebene erfolgt somit die Variantenbildung bei Kraftfahrzeugen, also die Kombination von verschiedenen Motor-, Getriebe-, Fahrwerk- und Karosserieoptionen, über eine unterschiedliche Bedatung der Applikationsparameter. Dabei spielen ebenfalls länderspezifische Kriterien wie Abgasrichtlinien oder präferierte Fahrstrategien eine wesentliche Rolle.
Die Steuergeräte-Applikation erfolgt üblicherweise beim Automobilhersteller oder bei beauftragten Entwicklungsdienstleistern durch Experten und Regelungstechniker, so genannte Applikateure, und nicht durch Software-Entwickler.

17.6.2 Applikation von Motorsteuerungen

Bedingt durch wachsende Komplexität und Variantenvielfalt moderner Motoren sowie durch steigende Diagnoseanforderungen gestaltet sich die Applikationsaufgabe bei Motorsteuergeräten sehr anspruchsvoll. Motorsteuerungen mit über zehntausend Applikationsparametern, darunter mehrere hundert, teilweise über tausend Kennlinien und Kennfelder, sind heute bereits Realität. Eine typische Bedatung durchläuft dabei verschiedene Applikationsphasen [1].
In der ersten und sehr wichtigen Phase erfolgt die Grundbedatung des Steuergeräts, die so genannte stationäre Grundabstimmung am Motorprüfstand, um einen sicheren und zuverlässigen Betrieb des Motors zu gewährleisten. Mit modernen Prüfstandsystemen und Parameteroptimierungsverfahren lässt sich dabei ein Großteil der Applikationsaufgaben automatisieren. Aufgrund der hohen Anzahl von Freiheitsgraden moderner Antriebskonzepte werden zur Beherrschung der Komplexität zunehmend mathematisch-statistische Ansätze im automatisierten Prüfstandbetrieb eingesetzt [1], [2].
Die Ergebnisse aus der stationären Grundabstimmung am Motorenprüfstand sind zwar häufig hinsichtlich bestimmter Kriterien optimiert, weisen aber bezüglich Fahrbarkeit meist noch große Defizite auf. Die Grundbedatung wird daher in der zweiten Phase an die Fahrzeugapplikateure weitergegeben. Diese müssen aufgrund der mangelhaften Fahrbarkeit in die Grundbedatung eingreifen und verlassen dadurch zwangsläufig das zuvor gefundene stationäre Optimum. In den weiteren Phasen sind das Emissions- und Verbrauchsverhalten des Motors sowie diagnoserelevante Funktionen, zum Beispiel zur On-Board-Diagnose (OBD), anzupassen. Abschliessend erfolgt die Validierung der Abstimmung im Flottenversuch.
Üblicherweise werden 30 bis 50 % aller Einstellarbeiten am Prüfstand und entsprechend 50 bis 70 % im Fahrzeug durchgeführt [3]. Bei einer Gesamtapplikation ist es dabei wichtig, die richtige Reihenfolge in der Abstimmung der einzelnen Software-Funktionen einzuhalten. Der Zeitaufwand für den Applikationsprozess kann dabei je nach Antriebskonzept und verwendeten Applikationsmethoden bis zu 27 Monate betragen.

17.6.3 Software-Stand und Beschreibungsdateien für Steuergeräte

Ein Software-Stand für ein elektronisches Steuergerät besteht in der Regel aus dem Programmstand, also den eigentlichen Software-Funktionen, und dem Datenstand, den Applikationsparametern. Der Datenstand wird häufig auch als Datensatz bezeichnet. Programmstand und Datenstand sind bei Seriensteuergeräten im nichtflüchtigen Lesespeicher (ROM oder Flash) abgelegt. Software-Stände moderner Motorsteuerungen erfordern Flash- oder ROM-Speichergrößen von typischerweise 1 bis 2,5 MByte. Dazu wird üblicherweise das interne Flash des Mikrocontrollers und, falls erforderlich, ein externer ROM- oder Flash-Baustein verwendet. Der Datenstand befindet sich dabei in einem bestimmten, vom Programmstand getrennten Adressbereich des Steuergeräts.
Dem Applikateur ist in der Regel nicht bekannt, wie die teilweise sehr komplexen Software-Funktionen im Steuergerät im Detail funktionieren. Der Software-Stand umfasst daher zusätzlich eine Software-Dokumentation in Form von Blockschaltbildern oder verbalen Beschreibungen über die in der Steuergeräte-Software realisierten Fahrzeugfunktionen. Diese Dokumentation gibt zudem Aufschluss über die Bezeichner der Applikationsparameter, deren Bedeutung sowie den Ablauf der Funktionsbedatung.
Darüber hinaus steht eine steuergerätespezifische Datenbeschreibung zur Verfügung. Darin sind Informationen zu den in der Steuergeräte-Software verfügbaren Applikations- und Messgrößen, deren Speicheradressen und Konvertierungsmethoden sowie Speicherlayout und Datenstrukturen abgelegt. Zusätzlich werden an dieser Stelle die für die Mess- und

Applikationsschnittstelle erforderlichen Kommunikationsparameter festgehalten.
Steuergeräte-Beschreibungsdateien entstehen im Rahmen des Software-Entwicklungsprozesses und liegen typischerweise im standardisierten Format ASAM-MCD 2MC vor [4]. Dieses Format kann in der Regel von Mess- und Applikationswerkzeugen eingelesen und interpretiert werden. Es standardisiert somit den Austausch von Steuergeräte-Beschreibungsdateien innerhalb der Prozesskette zwischen den einzelnen Entwicklungspartnern.
Ergänzend stehen häufig Dateien zur Verfügung, die die Signale auf dem Fahrzeugbus sowie die Vernetzung der Steuergeräte beschreiben.

17.6.4 Mess- und Applikationssysteme

Ein Mess- und Applikationssystem besteht im Wesentlichen aus dem Mess- und Applikationswerkzeug auf einem PC, den entsprechenden Schnittstellen zum Steuergerät und zum Fahrzeugbus sowie der Zusatzmesstechnik (siehe **Bild 17-19**).
Mess- und Applikationswerkzeuge werden während der Steuergeräte-Entwicklung zu sehr unterschiedlichen Aufgabenstellungen herangezogen. Zu den Hauptaufgaben zählen das Verstellen von Steuergeräte-Parametern, die Messdatenaufzeichnung und das Speichern der Verstellergebnisse in Datenständen. Darüber hinaus dient das Werkzeug häufig auch zum Verwalten, Vergleichen und Zusammenführen von Datenständen, zur Auswertung und Dokumentation von Verstell- und Messergebnissen sowie zur Steuergeräte-Flash-Programmierung.
Prinzipiell kann man zwei Anwendungsszenarien unterscheiden: die manuelle Durchführung von Mess- und Verstellaufgaben, wie es in der Regel im Fahrzeug praktiziert wird, und die automatisierte Steuergeräte-Applikation, zum Beispiel am Prüfstand.
Bei der Kennfeldbedatung am Prüfstand wird das Mess- und Applikationswerkzeug vorwiegend automatisiert betrieben, so dass der Prüfstandfahrer und der Applikateur in erster Linie mit dem Automatisierungssystem und den dort zur Verfügung gestellten Komponenten zur automatisierten Parameterverstellung und -optimierung arbeitet. Die Kopplung erfolgt in der Regel über eine Automatisierungsschnittstelle gemäß dem Standard ASAM-MCD 3MC (ASAP3) oder ASAM-MCD 3 [4].
Üblicherweise wird für den gesamten Applikationsprozess, also von der Vorbereitung im Büro, der Inbetriebnahme im Labor, der Grundapplikation am Prüfstand bis hin zur Serienabstimmung im Fahrzeug ein und dasselbe Mess- und Applikationssystem verwendet.

17.6.4.1 Verstellen und Messen

Steuergeräte-Funktionen sind von der Struktur her fest im Programmstand des Steuergeräts abgebildet. Um das Gesamtverhalten des Fahrzeugs entsprechend den Anforderungen anzupassen, müssen die Parameter der Steuergeräte-Funktionen, also die Kennwerte, Kennlinien und Kennfelder verändert werden. Die Parameterwerte werden dabei in einem Datensatz oder Datenstand abgelegt.

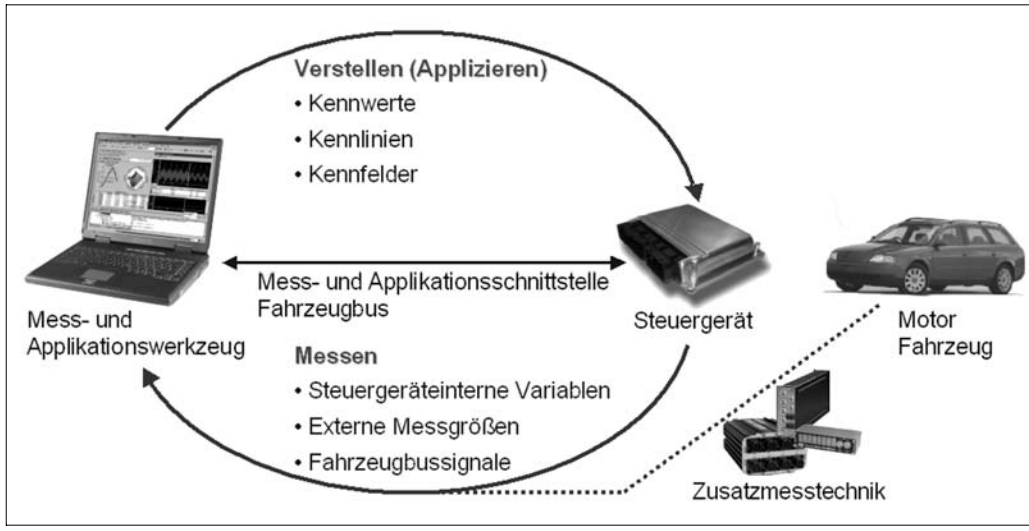


Bild 17-19: Mess- und Applikationssysteme

Software- und Datenstände liegen üblicherweise als Hex-Code vor. Dieses Format kann vom Applikateur nicht unmittelbar interpretiert werden, so dass es in eine lesbare Form zu überführen ist. Diese Aufgabe übernehmen Mess- und Applikationswerkzeuge, indem die Parameterwerte und Messvariablen an den Speicheradressen im Code in physikalische Größen umgewandelt werden.
Die auf dem Steuergerät zugänglichen Verstell- und Messgrößen sowie deren Umrechnungsformeln und Einheiten sind in der Steuergeräte-Beschreibungsdatei gemäß ASAM-MCD 2MC abgelegt. Zudem kann die Datei gemäß ASAM-MCD 2MC eine Kennung enthalten, die eine Konsistenzüberprüfung zwischen Beschreibungsdatei und Software- bzw. Datenstand durch das Applikationssystem erlaubt.
Für die Verstellaufgabe stehen verschiedene Instrumente zur Verfügung, zum Beispiel Editoren für Kennlinien und Kennfelder, die sowohl eine numerische als auch ein grafische Verstellung erlauben.
Zur Überprüfung der Wirksamkeit und Gültigkeit von Parameterverstellungen müssen steuergeräteinterne Variablen und Fahrzeugdaten bei laufender Ausführung der Software-Funktionen, des sogenannten Fahrprogramms, erfasst werden. Dies erfordert häufig den Einsatz von Zusatzmesstechnik zum Aufzeichnen weiterer physikalischer Größen wie Temperaturen, Drücken, Spannungen oder Abgaswerten. Daneben wird die Kommunikation auf dem Fahrzeugbus erfasst.
Das Mess- und Applikationswerkzeug hat dabei die Aufgabe, sämtliche Messgrößen zeitlich zueinander korreliert darzustellen und aufzuzeichnen. Für eine unmittelbare Bewertung der Messungen ist es zudem erforderlich, sämtliche Messsignale als physikalische Größen in einem Zeitschrieb oder einer numerischen Anzeige zu visualisieren und grundlegende Auswerteoptionen zur Messdatenanalyse zur Verfügung zu stellen.

17.6.4.2 Offline- und Online-Applikation

Bei der Offline-Applikation werden Parameteränderungen zunächst nur im Mess- und Applikationswerkzeug auf dem PC durchgeführt und erst anschließend in Form eines neuen Daten- oder Software-Stands auf das Steuergerät geladen oder in das Flash programmiert.
Die Online-Applikation hingegen setzt voraus, dass die zu applizierenden Parameter vom ROM- oder Flash-Bereich des Steuergeräts in einen vom Software-Programm nicht verwendeten, freien RAM-Bereich kopiert werden. Zudem muss durch Modifikation der Steuergeräte-Software oder -Hardware der Mikrocontroller auf die Daten im RAM zugreifen, um während des Fahrprogramms Verstellungen wirksam durchzuführen. Der entsprechende RAM-Bereich im Steuergerät wird häufig als Calibration RAM oder Cal-RAM bezeichnet.
In einigen Fällen kann aufgrund der eingeschränkten Größe des Cal-RAM-Bereichs der Datenstand nicht vollständig kopiert werden, so dass während der Online-Verstellung nur auf eine Untermenge aller Applikationsparameter zugegriffen werden kann.
Der Datenstand im Cal-RAM des Steuergeräts trägt häufig die Bezeichnung „Arbeitsseite“, der Datenstand im ROM oder Flash-Speicher die Bezeichnung „Referenzseite“. Das Konzept von Arbeits- und Referenzseite erlaubt es, Datenstände während des laufenden Fahrprogramms konsistent umzuschalten. Anwendungen dafür sind zum Beispiel die Vorgabe von Sollwertsprüngen, der unmittelbare Vergleich von Parametereinstellungen oder das Zurückschalten auf einen „sicheren“ Datensatz bei plötzlichen Ausnahmesituationen.

17.6.4.3 Steuergeräte-Schnittstellen zum Messen und Applizieren

Für Mess- und Applikationsaufgaben stehen unterschiedliche Schnittstellen im Steuergerät zur Verfügung (**Bild 17-20**).
Grundsätzlich sind für eine geeignete Auswahl verschiedene Kriterien zu berücksichtigen. Dazu zählen der Messdatendurchsatz, die Größe des geforderten Cal-RAM-Speichers, das Kaltstartverhalten, die mechanischen und elektrischen Voraussetzungen des Steuergeräts sowie die Umgebungsbedingungen am Verbauort des Steuergeräts. Darüber hinaus spielen die Kosten für die Mess- und Applikationsschnittstelle und der damit verbundene Integrationsaufwand eine wichtige Rolle.

Der Speicheremulator, zum Beispiel der Generic Memory Emulator (GME) von der Firma dSPACE, bietet in Bezug auf Datendurchsatz, Kaltstartfähigkeit und Größe des Cal-RAMs die höchste Leistungsfähigkeit. Er wird direkt mit dem Mikrocontrollerbus des Steuergeräts verbunden, was einen im Vergleich zu seriellen Schnittstellen hohen Integrationsaufwand im Steuergerät bedeutet. Der Speicheremulator wird ausschließlich während der Entwicklungsphase verwendet.
Häufig steht der Mikrocontrollerbus zur Ankopplung externer Werkzeugschnittstellen nicht oder nur eingeschränkt zur Verfügung. In diesem Fall muss der Zugriff auf das Steuergerät über eine serielle Schnittstelle erfolgen. Ein grundsätzliches Unterscheidungsmerkmal ist dabei, ob die Schnittstellentechnologie auch im Seriensteuergerät zur Verfügung stehen soll, wie zum Beispiel die Diagnoseschnittstelle. Typische Implementierungen dafür sind das Keyword Protocol (KWP) 2000, basierend auf der K-Leitung [5] oder dem CAN [6] und zukünftig dessen Weiterentwicklung unter der Bezeichnung Unified Diagnostic Services (UDS) [7].

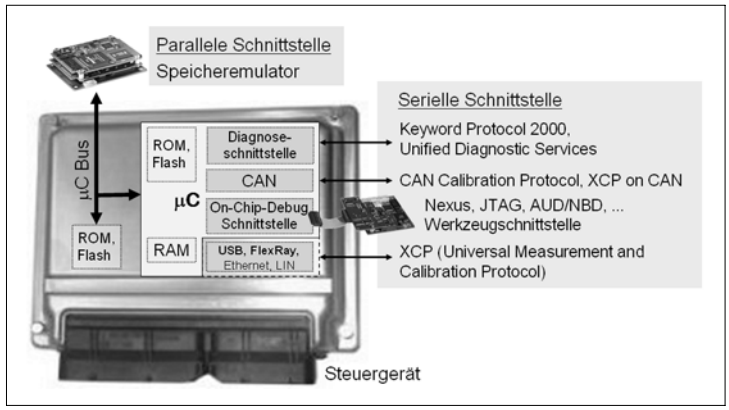


Bild 17-20:
Steuergeräte-Schnittstellen zum Messen und Applizieren; die Abkürzungen werden im Text erklärt

Moderne Prozessorarchitekturen bieten zunehmend leistungsfähige On-Chip-Debug-Schnittstellen wie Nexus, JTAG oder AUD/NBD für die Software-Entwicklung an [8]. Diese Schnittstellen stehen in der Regel nur in der Entwicklungsphase im Steuergerät zur Verfügung und eignen sich gleichermaßen für Software-Debugging, Applizieren, Messen, Bypassing und Flash-Programmierung. Insbesondere die Bemühungen im Rahmen des Nexus-5001-Forums, einen prozessor- und herstellerübergreifenden On-Chip-Debug-Standard zu etablieren, lassen für die Zukunft erwarten, gleiche Werkzeugschnittstellen in unterschiedlichen Projekten einsetzen zu können [9].

In heutigen Fahrzeugen findet eine Vielzahl von Steuergeräten unterschiedlicher Hersteller Anwendung. Die Steuergeräte können sich dabei durch verschiedenste Schnittstellen auszeichnen. Aus wirtschaftlichen Überlegungen sind Automobilhersteller mehr und mehr daran interessiert, standardisierte Lösungen einzusetzen. Neben dem Nexus-Standard rücken daher Mess- und Applikationsschnittstellen auf Basis standardisierter Protokolle zunehmend ins Blickfeld. Das *Universal Measurement and Calibration Protocol* XCP stellt die Weiterentwicklung des bereits etablierten Protokollstandards *CAN Calibration Protocol* CCP dar [4]. Durch die konsequente Trennung von Protokoll- und Transportschicht werden dabei künftige Kommunikationsstandards im Fahrzeug berücksichtigt. In heutigen Entwicklungssteuergeräten werden vorwiegend Implementierungen auf CAN (XCP on CAN) und USB (XCP on USB) zur Steuergeräte-Applikation eingesetzt. Daneben existieren XCP-Implememtierungen auf Ethernet und LIN, die schwerpunktmäßig als Messschnittstelle genutzt werden [10]. Mit der Etablierung von FlexRay in zukünftigen Steuergeräten wird auch eine XCP-Unterstützung zum Messen und Applizieren für diese Kommunikationstechnologie an Bedeutung gewinnen [11].

Grundsätzlich ist man bei seriellen Applikationsschnittstellen auf das im Steuergerät zur Verfügung gestellte Cal-RAM beschränkt. Dieses befindet sich häufig im On-Chip-RAM des Mikrocontrollers und ist kleiner als der komplette Datenstand, was eine entsprechende Einschränkung für die Applikation bedeutet. Bei modernen Mikrocontrollern in Seriensteuergeräten beträgt die Größe des Cal-RAMs in der Regel nur wenige Kilobytes. Daneben bieten einige Halbleiterhersteller Prozessor-Varianten mit deutlich größerem On-Chip-Speicher speziell für die Steuergeräteentwicklung an. Steht der externe Mikrocontrollerbus zur Verfügung, kann alternativ ein zusätzlicher RAM-Baustein für die Applikationsphase im Steuergerät verbaut werden.

17.6.5 Ausblick in die Zukunft

Bedingt durch den Wettbewerbsdruck in der Automobilindustrie und den daraus resultierenden Forderungen nach einer Reduktion von Entwicklungskosten und Entwicklungszeiten ist man zunehmend bestrebt, den Applikationsprozess effizienter zu gestalten.

Ein Aspekt in diesem Zusammenhang ist das Bestreben, die Anzahl der Versuchsfahrzeuge zu verringern. Daraus folgt, dass möglichst viele Applikationsaufgaben, die heute noch im Fahrzeug durchgeführt werden, auf den Prüfstand oder in frühere Entwicklungsphasen verlagert werden müssen und die im Fahrzeug absolut notwendige Arbeit effizienter ablaufen muss. Ansätze dazu sind die Applikation von fahrbarkeitsrelevanten Kenngrößen an dynamischen Prüfständen, eine verstärkte Automatisierung von Standardaufgaben sowie die Software-Bedatung mit Hilfe der Motor- und Fahrzeugsimulation.

Ein weiteres Ziel ist es, im Rahmen der Applikation möglichst seriennahe Steuergeräte am endgültigen Verbauort im Fahrzeug einzusetzen, um Zusatzauf-

wände für die Bereitstellung spezieller Applikationssteuergeräte einzusparen.

Gesamtkosten und Wirtschaftlichkeit werden auch bei Mess- und Applikationssystemen an Bedeutung gewinnen. Der Einsatzbereich der Werkzeuge wird sich daher nicht mehr auf die reine Steuergeräte-Bedatung beschränken, sondern auch auf Bereiche wie Funktionsentwicklung, Steuergeräte-Diagnose und Validierung erstrecken.

Darüber hinaus ist eine engere Integration der Mess- und Applikationswerkzeuge in den Gesamtentwicklungsprozess zu erwarten. Dazu zählen Aspekte wie die Verzahnung mit Modellierungswerkzeugen, die Ankopplung an Datensatz-Managementsysteme und die Verwendung von gleichen Steuergeräte-Beschreibungsdaten für Applikation und Diagnose. Daneben gewinnt die geführte Funktionsbedatung und die engere Integration der Steuergeräte-Dokumentation in den Applikationsablauf an Bedeutung.

Mit der zunehmenden Verteilung einzelner Funktionen auf mehrere Steuergeräte im Fahrzeug wird darüber hinaus eine stärkere Funktions-orientierte Arbeitsweise in den Werkzeugen gefordert werden.

Literatur zu Abschnitt 17.6

[1] *Lütkemeyer, G.*; *Von Reth, A.*; *Kinoo, B.*; Weinowski, R.: Effektive Strategien für Motorsteuerungsapplikationen, MTZ 7-8/2002.

[2] *Buschbaum, G.*; *Röpke, K.*: Optimierung der Motorenentwicklung durch neue Wege in der Entwicklungssystematik, Automotive Engineering Partners 4/2002.

[3] *Miller, G.*; *Hall, K.*; *Willis, W.*; *Pless, W.*: The Evolution of Powertrain Microcontrollers and its Impact on Development Processes and Tools, White Paper, The Nexus 5001TM Forum.

[4] ASAM e.V., Association for Standardisation of Automation and Measuring Systems, www.asam.net.

[5] ISO International Organization for Standardization: ISO 14230 – Road vehicles – Diagnostic systems – Keyword Protocol 2000.

[6] ISO International Organization for Standardization: ISO 15765 – Road vehicles –Diagnostics on Controller Area Networks (CAN).

[7] ISO International Organization for Standardization: ISO 14229-1 – Road vehicles – Unified Diagnostic Services.

[8] *Rolfsmeier, A.*; *Leinfellner, R.*; *Limberg, H.*; *Franzen, O.*: Future-proof Interfaces for ECU Calibration and Function Development, JSAE Annual Congress, Japan, 2004.

[9] The Nexus 5001TM Forum, www.nexus5001.org.

[10] LIN Local Interconnect Network, www.lin-subbus.org/

[11] FlexRayTM, www.flexray-group.com.