7. LuK Kolloquium

11./12. April 2002

Herausgeber: LuK GmbH & Co.
Industriestrasse 3 • D -77815 Bühl/Baden
Telefon +49 (0) 7223 / 941 - 0 • Telefax +49 (0) 7223 / 2 69 50
Internet: www.LuK.de

Redaktion: Ralf Stopp, Christa Siefert

Layout: Vera Westermann

Druck: Konkordia GmbH, Bühl
Das Medienunternehmen

Printed in Germany

Vorwort

Innovationen bestimmen unsere Zukunft. Experten sagen voraus, dass sich in den Bereichen Antrieb, Elektronik und Sicherheit von Fahrzeugen in den nächsten 15 Jahren mehr verändern wird als in den 50 Jahren zuvor. Diese Innovationsdynamik stellt Hersteller und Zulieferer vor immer neue Herausforderungen und wird unsere mobile Welt entscheidend verändern.

LuK stellt sich diesen Herausforderungen. Mit einer Vielzahl von Visionen und Entwicklungsleistungen stellen unsere Ingenieure einmal mehr ihre Innovationskraft unter Beweis.

Der vorliegende Band fasst die Vorträge des 7. LuK Kolloquiums zusammen und stellt unsere Sicht der technischen Entwicklungen dar.

Wir freuen uns auf einen interessanten Dialog mit Ihnen.

Bühl, im April 2002

Helmut Beier

Vorsitzender
der Geschäftsführung LuK Gruppe

Inhalt

Codegenerierung contra Manufaktur

Die Software des elektrischen Zentralausrückers

Reinhard Ludes, AFT Werdohl
Thomas Pfund

Einleitung

Software spielt im Bereich der Fahrzeugtechnik mittlerweile eine erhebliche Rolle. Besonders im Fahrzeug selbst nehmen Anzahl und Bedeutung von mechatronischen Systemen, die einen starken Integrationsgrad von mechanischer und elektronisch / softwarebasierter Funktionalität haben, deutlich zu. Die Software nimmt hier Funktionen im Bereich Steuern und Regeln wahr, oder übernimmt als Automatisierungslösung Funktionen, die zuvor vom Fahrer ausgeübt wurden.

Die Funktionalität der Systeme wird zunehmend komplexer, und diese Komplexität des Systemverhaltens wird zunehmend auf die Software verlagert. Während im Bereich mechanischer und elektronischer Hardware, auch markenübergreifend, immer mehr Plattformen eingesetzt werden, kommt der Software auch zunehmend die Aufgabe der Individualisierung der System- und Fahreigenschaften zu.

Die scheinbar leichte Änderbarkeit der Software unterstützt diesen Trend. Vorschub leisten hier auch leistungsfähige Prozessorsysteme, die nicht nur immer preiswerter werden, sondern immer mehr Funktionalitäten und immer größer werdende Speicherbereiche bieten, eine Versuchung sondergleichen für ambitionierte Entwicklerteams.

Was jedoch häufig übersehen wird, ist die Tatsache, dass immer mehr Aufwand in Spezifikation, Realisierung und Test der Software sowie der Validierung des gesamten mechatronischen Systems am Prüfstand und im Fahrzeugeinsatz investiert werden muss.

Grund hierfür ist, dass die verschiedenen Methoden der Softwareerstellung nicht Schritt gehalten haben mit den Anforderungen, die an Softwaresysteme gestellt werden. Sie haben auch nicht Schritt gehalten mit den Komplexitäten, die tagtäglich von vielköpfigen Entwicklungsmannschaften erzeugt bzw. verursacht werden.

Zwar gab und gibt es bei den Programmiersprachen Fortschritte, eine Reihe von Entwurfsmethoden haben sich etabliert, um den Entstehungsprozess zu optimieren, Spracherweiterungen wie die objektorientierte Programmierung (OOP) etablieren sich auch langsam im Echtzeitbereich, aber als Fakt bleibt: Während die Software erhebliche Beiträge im Bereich der Steuerung, Regelung und Automatisierung leistet, wird sie selbst in der Regel noch per Hand erstellt. Es handelt sich also bei der Softwareentwicklung noch um eine klassische Manufaktur.

Im vorliegenden Projekt, der Entwicklung der Software für einen elektrischen Zentralausrükker (EZA), wurde ein alternativer Weg beschritten. Mit Hilfe der Werkzeugkette TDC, der Kurzname steht für Total Development Chain der

Software in der Automobiltechnik	
Steuerung & Automatisierung von Funktionen	
Im Fahrzeug:	
• **Steuer- / Regelfunktionen**	**Einspritzung, Zündung, Motormanagement, ABS, ASR, ESP, ...**
• **Automatisierung**	**Kupplung (EKM), Getriebe (ASG)**
In Entwicklung & Produktion:	
• **Methoden & Werkzeuge**	**CAD, CAE, CAM**
• **Automatisierung**	**Robotik**

Bild 1: *Software in der Automobiltechnik*

Firma **AFT Atlas Fahrzeugtechnik**, wurde die notwendige Reglersoftware automatisch erstellt. Dies wird im Folgenden beschrieben.

Der Standard-Entwicklungsprozess für Software

Im Folgenden sei kurz der Standard-Prozess für die Entwicklung von Software dargestellt. Selbstverständlich gibt es im Detail viele Varianten, hier seien nur die wesentlichen und generell typischen Schritte erwähnt (Bild 2).

Einer Idee folgend, wird zuerst die Regelstrategie formuliert. Dies kann schriftlich in Form eines Lastenheftes erfolgen. Eine fortgeschrittene Vorgehensweise ist die Spezifikation und allmähliche Detaillierung des Reglerentwurfs im Rahmen eines Simulationsprogramms, wo sich modellbasiert seine Struktur und Parametrierung kontinuierlich verbessert. Gegenüber schriftlich formulierten Spezifikationen und Entwürfen hat diese Vorgehensweise den Vorteil, dass das Reglermodell gut zwischen Fachleuten aus Bereichen wie Systemanalyse, Regelung und Simulation kommuniziert werden kann.

Hat die Regler- bzw. Automatisierungsfunktion, zumindest im Bereich der Simulation, einen hinreichenden Stand erreicht, ist es in klassischer Vorgehensweise jetzt der Softwarefachmann, der im Hinblick auf seine Qualifikation aus dem Bereich Echtzeitsoftware und Mikrocontrollerdesign stammt und letztlich für die Implementierung zuständig ist.

Allerdings ist das grundsätzliche Reglerkonzept für ihn erst der Beginn der Arbeit. Auch er muss erst einmal die an ihn gestellte Aufgabe analysieren und ein entsprechendes Designkonzept für sein Echtzeitsystem erstellen, ehe überhaupt an die Implementierung, d. h. die Übertragung von Funktionalitäten, in den konkreten Code zu denken ist. Grund für diesen Bruch, wie er auch in der Darstellung zu sehen ist, sind unterschiedliche Sichtweisen, die in unterschiedlichen Modellansätzen münden.

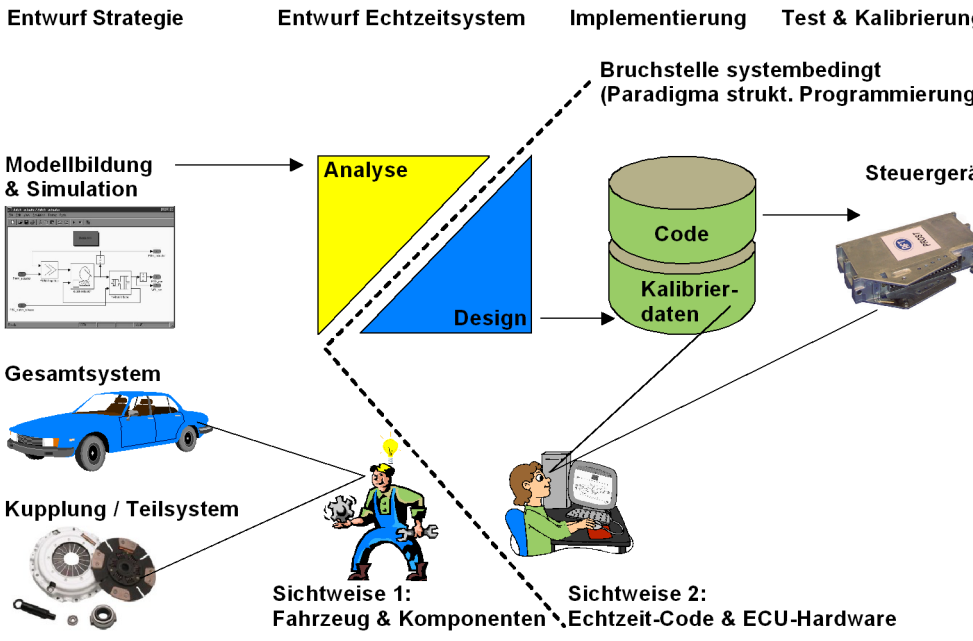

Bild 2: *Standard-Entwicklungsprozesse für Software*

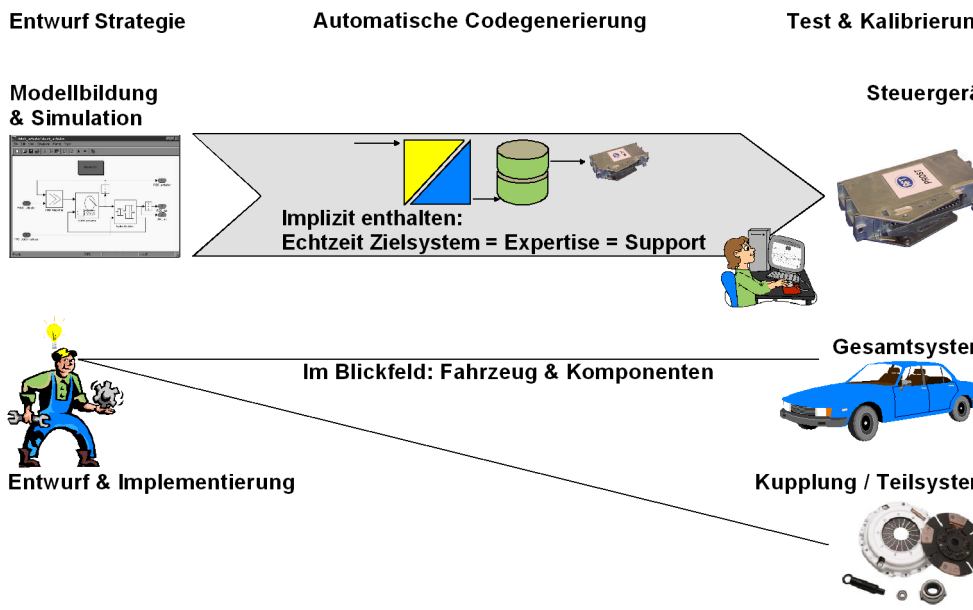

Bild 3: *AFT-TDC im Entwicklungsprozess*

Beim Reglerdesign steht das zu regelnde Objekt, z. B. im aktuellen Fall der Stellmotor als Aktor und die Kupplung als Regelstrecke, im Vordergrund. Daran orientieren sich alle Modellierungen. Beim Echtzeit-Softwaredesign, aber auch bei jedem anderen Design von Software, steht jedoch die Architektur des Rechner- bzw. Mikrocontrollersystems im Vordergrund. Die zugrunde liegenden Modelle orientieren sich damit grundsätzlich an der üblichen Architektur von Rechnerkernen bzw. Prozessoren (Von-Neumann-Maschinen), auch alle üblichen prozeduralen Programmiersprachen sind darauf ausgelegt.

Die Stärke dieses Bruchs ist unterschiedlich und hängt von den verwendeten Sprachen und Methoden ab. Der Bruch ist jedoch grundsätzlicher Natur. Lediglich die objektorientierte Programmierung zeigt Ansätze, diesen zu beheben.

Es gibt mehrere Ansätze, diesen Bruch zu überbrücken. Vor allem hängen die Auswirkungen auf den Entwicklungsprozess davon ab, wie gut die Kommunikation zwischen diesen beiden grundsätzlichen Entwicklertypen funktioniert.

Bei LuK beispielsweise ist es gelungen, beide Qualifikationen gleichzeitig im Team oder beim einzelnen Entwickler zu verankern. Zudem werden Codesegmente aus der Offline-Simulation auf Quellcode-Ebene in die Reglersoftware übernommen.

Codegenerierung im Entwicklungsprozess

Im vorliegenden Projekt wurde der Weg der automatischen Codegenerierung beschritten. Der verwendete Codegenerator setzt auf der modell- bzw. simulationsbasierten Spezifikation des Reglers auf und erzeugt seriennahen Echtzeitcode für den Mikrocontroller im automotiven Steuergerät. Es liegt hier demnach ein lückenloser Weg vor, von der Spezifikation der generellen über die detaillierte Strategie bis hin zum Echtzeitcode im Steuergerät (Bild 3).

Der Ansatz fördert die LuK Philosophie, grundsätzlich nur einen Experten bzw. eine Expertengruppe einzusetzen, um das Gesamtsystem zu entwerfen und zu implementieren: Es ist der

Systemingenieur, der auch in der klassischen Vorgehensweise den Regler entwirft. Im Codegenerator ist implizit das Wissen vorhanden, um einen optimalen Code zu erzeugen. Im Blick sind also von Anfang an die Funktionen, die sich auf das Fahrzeug bzw. auf eine seiner Komponenten beziehen. Der konkret verwendete Prozessor, die verwendete Programmiersprache und das verwendete Betriebssystem treten in den Hintergrund.

Der Bruch in der Spezifikations- bzw. gesamten Entwicklungskette ist überwunden.

Einsatz der TDC beim EZA – Methodik: Die geschlossene Werkzeugkette

Der Einsatz der AFT-TDC bedeutet mehr als nur den Einsatz eines Codegenerators. TDC reflektiert eine heterogene, aber geschlossene Kette von Werkzeugen, die den gesamten Entwicklungsprozess unterstützt. Heterogen, weil unterschiedliche Werkzeuge kombiniert werden, und zwar jeweils die, die für das jeweilige Projekt am besten geeignet sind. Geschlossen, weil die Ergebnisse aus dem Test wieder zurückgekoppelt werden, um das ursprüngliche Modell evolutionär zu verbessern (Bild 4).

Im vorliegenden Projekt wurden eingesetzt: Matlab-Simulink für die Simulation, TargetLink für die Codegenerierung, das AFT-**PROST** als geeignetes Prototyp-Steuergerät, **MARC I** für die Nachkalibrierung in der Erprobungsphase. Bemerkenswert ist auch, dass die Daten, die zur Kalibration, d. h. zur Feinabstimmung, eingesetzt werden, schon in der Simulationssoftware festgelegt werden. Für das Applikationssystem werden zudem Beschreibungsdateien (nach ASAP 2) erzeugt, die automatisch in das Applikationssystem eingelesen werden. Der Testingenieur braucht deshalb keine weiteren Konfigurationen an seinem Applikationssystem vorzunehmen.

Die während der Applikation gewonnenen Daten können wiederum in das Simulationsmodell zwecks Feinabstimmung des Modells eingepflegt werden. Weiterhin können die im Test gewonnenen Messdaten aus dem Applikationssystem oder aus weiteren installierten

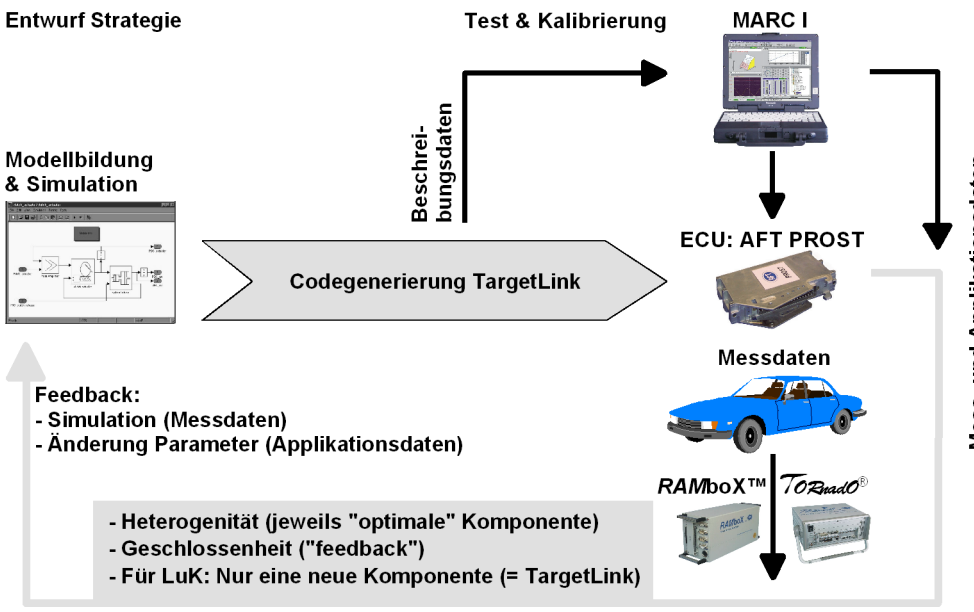

Bild 4: *Charakteristika der AFT-TDC: Beispiel – LuK EZA*

Messsystemen wieder in die Simulation rückgekoppelt werden. Bei LuK stehen zu diesem Zweck grundsätzlich die im Einsatz befindlichen Datalogger ***RAM*boX™** und **TORnadO®** zur Verfügung.

Letztlich war nur der Codegenerator sowie eine Reihe von Erweiterungen im Applikationssystem zusätzlich zu installieren, um die Entwicklungskette wie beschrieben aufzubauen. Alle weiteren Tools waren vorhanden und den Entwicklern vertraut.

Das EZA-Projekt im Spiegel des V-Modells

Wie schon in der vorhergehenden Abbildung angedeutet, spiegelt die TDC eine Entwicklungsmethodik wider, die über die reine Codegenerierung hinausgeht. Es handelt sich um eine simulationsbasierte Entwicklungsmethodik mit ausführbaren Spezifikationen. Die gesamte Kette orientiert sich am V-Modell. Bild 5 zeigt die Umsetzung des V-Modells mit all jenen Komponenten, die bei der AFT in der Softwareentwicklung für Steuergeräte eingesetzt werden.

Charakteristisch für das V-Modell als Grundlage der Entwicklungsstrategie ist Folgendes:

- Jedem Entwurfsschritt auf der linken Seite stehen auf der rechten Seite die entsprechenden Test- und Validierungsschritte gegenüber.
- Zwischen jedem Schritt auf der linken Seite sind Übergaben und Überprüfungen notwendig, um festzustellen, dass die im vorhergehenden Schritt gemachten Vorgaben auch erfüllt werden.
- Daraus folgt auch: Je später ein Fehler erkannt wird, desto mehr Schritte müssen im Rahmen des Redesigns auch wieder durchlaufen werden, d. h. desto aufwändiger und teurer gestaltet sich seine Behebung. Besonders kritisch sind damit nicht oder fehlerhaft übernommene Basis-Spezifikationen, die dann erst im letzten Glied entdeckt werden können.

Auf der linken Seite sind die beiden wesentlichen Entwurfsschritte zu sehen:

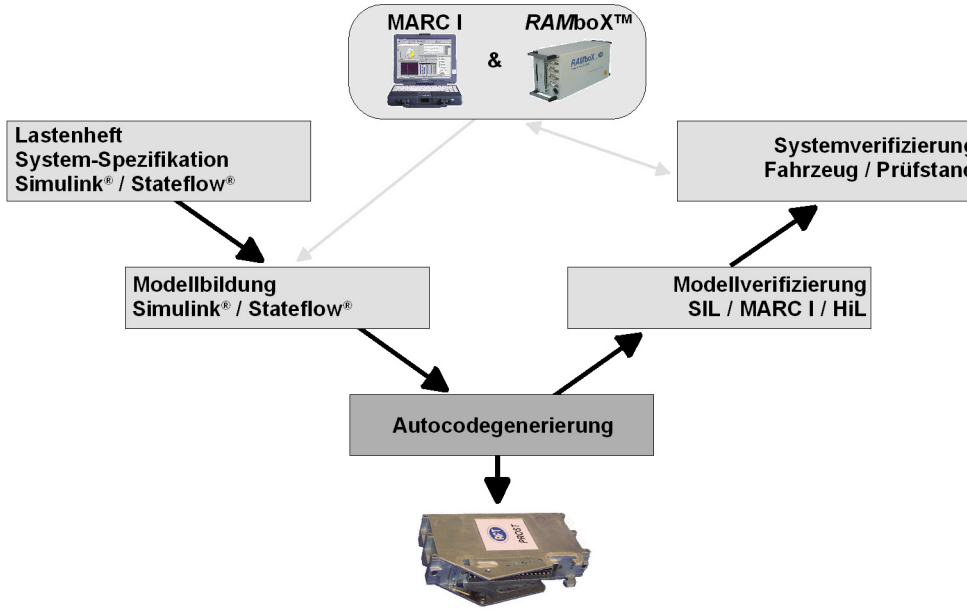

Bild 5: *EZA-Projekt im Spiegel des V-Modells*

- Lastenheftphase – die Systemspezifikation erfolgt zum Zwecke eindeutiger Definitionen z. T. bereits auf der Basis von Simulationsmodellen und Zustandsautomaten.
- Modellbildung – aufbauend auf der Systemspezifikation und evtl. bereits bestehenden Teilmodellen wird eine simulierbare Struktur, eingebettet in ein mehr oder weniger genaues Streckenmodell, erstellt.

In beiden Schritten wird MatLab Simulink eingesetzt, zuerst für die groben Systemstrukturen und Vorgaben, im zweiten Schritt dann ist das gesamte Modell von Regler und Regelstrecke im Detail definiert.

Der Prozess endet in der Spitze des V-Modells mit Autocodegenerierung auf dem jeweiligen Zielsystem. Damit wird die ursprüngliche Reglerspezifikation nicht nur innerhalb einer Simulationsumgebung, sondern auch auf einem Steuergerät als Zielsystem ausführbar. Die gesamte Kette ist durchgängig.

Den beiden Entwurfselementen stehen auf der rechten Seite die entsprechenden Test- und Validierungsschritte gegenüber:

- Modell-Verifikation, d. h. Validierung der grundsätzlichen Reglerfunktion, in der SiL – Simulation (SiL – Software in the Loop) oder ggf. in einem HiL-System (HiL - Hardware in the Loop).
- Gesamt-Systemverifikation auf Prüfstand oder Teststrecke.

Der Regler wird zunächst im Rahmen einer SiL-Simulation getestet. Bei der SiL-Simulation werden alle Schnittstellen zur spezifizierten Software ebenfalls als Modell abgebildet, so dass die Simulation ohne zusätzlichen Hardwareaufwand erfolgen kann. Das System unterstützt dieses Vorgehen dadurch, dass die I/O – Schnittstellen zum Controller – Betriebssystem als Targetlink – Blöcke verfügbar sind und dass eine Simulation sowohl mit Gleitkomma- als auch mit Festkommaarithmetik möglich ist.

Ein HiL – System wird zur Modell-Verifikation in dem Fall zum Einsatz kommen, wenn sich einzelne Komponenten der Regelstrecke nicht oder nur mit sehr großem Aufwand in der Simulationsumgebung abbilden lassen, gleichzeitig jedoch ein Test im Gesamtsystem z. B. auf Grund zu großen Applikationsaufwandes nicht bzw. noch nicht wirtschaftlich ist. Von Fall zu Fall ist auch hier zu entscheiden, welche Teile der Regelstrecke physikalisch vorliegen müssen und welche als Softwaremodell abgebildet werden können.

Aus Sicht des Steuergerätes ist kein Unterschied festzustellen, ob es die reale oder in Echtzeit simulierte Regelstrecke „bedient“. Bemerkenswert ist, dass in diesem simulationsbasierten Entwicklungsprozess auch das für den HiL-Simulator benötigte Streckenmodell schon weitgehend aus dem Schritt Systemspezifikation vorhanden ist. Im beschriebenen Projekt wurden die wichtigsten Teilsysteme an einem Prüfstand installiert (EZA gegen nicht rotierende Kupplung, PROST – Steuergerät und übergeordnetes EKM - Steuergerät).

Die Systemverifikation bezieht sich auf den Gesamtsystementwurf. Wird das Verhalten der Kupplung im Fahrzeug festgelegt, so muss auch die Validierung im Fahrzeug oder in einer fahrzeugadäquaten Systemumgebung enden. Auch im vorliegenden Fall wurde die abschließende Verifizierung im Fahrzeug vorgenommen.

Damit wurden im vorliegenden Projekt alle Stationen des V-Modells durchlaufen.

Zudem wurden Messdaten rückgekoppelt, die in der Modell- und Systemverifizierung sowie im Endtest gewonnen wurden.

Ergebnisse, Teil I: Die Reglerstruktur

Die Spezifikation des Reglers bestand zunächst aus ersten groben Blöcken, die dann durch Verfeinerung und Parametrierung mit „Leben“ gefüllt wurden. Die Blöcke lassen sich direkt mit ihren Verknüpfungen (Signalwegen) als Grafik ausgeben und unterstützen damit eine optimale Spezifikation und Dokumentation:

- die Soll- und Istwertberechnung für den Regler
- der Zustandsregler als Kern
- die Vorsteuerung zur Erhöhung der Verstelldynamik
- die Logik, die unterschiedliche Grundzustände unterscheidet (z. B. Normal-, Notlauf- und Testbetrieb)
- die Nachlauffunktionen, die verschiedene Ein- und Ausschaltprozeduren umfassen
- der Block „Ausgänge setzen“ zur Grenzwertüberprüfung und Anpassung an die Hardware
- der Block „Messwerte auf CAN“ als Kontrollfunktion während der Erprobungsphase

Eine grafische Darstellung wird automatisch erstellt. Die Einbindung an dieser Stelle sprengt jedoch den Rahmen.

In dieser grafischen Darstellung sind prinzipiell die Eingänge des Reglers links, die Reglerausgänge rechts angeordnet. Jeder Block kann auf weitere Details heruntergebrochen werden. Der Regler als Gesamtsystem ist wiederum ein Block im gesamten Offline-Simulationsmodell, in dem die restlichen Komponenten der Regelstrecke abgebildet sind: Endstufe, E-Motor, das Federbandgetriebe und die Kupplung selbst.

Da die Reglerspezifikation letztlich in einem Reglermodell endet, das sowohl in der Simulation „gegen“ das Modell der Regelstrecke laufen kann, als auch nach Codegenerierung im Steuergerät „gegen“ die reale Regelstrecke, handelt es sich um eine ausführbare Spezifikation.

Befindet sich der Regler nach Codegenerierung im realen Steuergerät, so greift die generierte Software auf die gleichen Ein- und Ausgangsschnittstellen zu, wie sie im Modell schon dargestellt wurden. Die Schnittstelle zur Systemsoftware bildet dabei dann das AFT Controller Interface (ACI).

Ergebnisse, Teil II: Einsparpotenzial

Für die Erstellung des Reglers für den EZA waren ursprünglich drei Mannmonate für Entwurf, Simulation, Codierung und Ersttest vorgesehen worden. Das Projekt wurde zwei Wochen nach Projektstart auf die neue Methodik umgestellt. Insgesamt fielen mit der neuen Methodik ebenso drei Mannmonate an.

Es wurde keine Kontrollgruppe definiert, die die gleiche Aufgabenstellung mit der konventionellen Methodik löste. Es besteht allerdings hinreichende Erfahrung mit der Definition und Durchführung derartiger Projekte, so dass in etwa von dem gleichen Arbeitsaufwand ausgegangen werden kann.

Allerdings ist einiger Aufwand durch die Einarbeitung in die neuen Werkzeuge geflossen. Der erst nach zwei Wochen gefällte Entschluss, auf die Methodik zu wechseln, brachte auch einige Stunden Mehraufwand, da die Simulationsmodelle noch an die Echtzeitanforderungen angepasst werden mussten.

Geschätzt wird, dass sich die Entwicklungszeit bei eingearbeiteten Mitarbeitern um ca. 50% auf 1,5 Mannmonate reduziert. Dies ergibt zum einen die Analyse der Stundenschreibung, zum andern zeigt dies die Gegenrechung: Erfahrungsgemäß fallen für die Codierung des Reglers einschließlich Test sowie Konfiguration des Applikationssystems auch 1,5 Mannmonate an, ein Aufwand, der bei der neuen Methodik entfällt.

Der mögliche Ratioeffekt beträgt somit gut 50% und ergibt sich aus dem Wegfall der Erstellung des Reglercodes und durch den Wegfall des Aufwands zur Konfiguration des Applikationssystems, das in der Testphase zur Feinabstimmung eingesetzt wird.

Es muss erwähnt werden, dass die für eine Serienfreigabe notwendigen Validierungsschritte (Funktionsdauerläufe, Codeinspektion usw.) in der Zeitschätzung nicht berücksichtigt wurden. Bei einer solchen Betrachtungsweise würde sich der Ratioeffekt reduzieren.

Sehr wichtig ist, an dieser Stelle festzustellen, dass es sich nicht um eine reine Funktionsdarstellung auf einem automotiven Hochleistungs-Testsystem handelt.

Das Prototyp-Steuergerät AFT-**PROST** beinhaltet einen üblichen, serientauglichen 16-bit Mikrocontroller mit Festkommaarithmetik. Ebenso ist der generierte Code ohne großen Aufwand auf ein konventionelles Seriensteuergerät übertragbar. Damit ist diese Konzeption auf Grund ihrer Seriennähe gegenüber anderen Systemen, die auf Hochleistungs-Fließkomma-Prozessoren beruhen und lediglich eine Funktionsdarstellung im Prototyp-Fahrzeug ermöglichen, weit überlegen.

Antworten auf häufig gestellte Fragen

Woher kommt die Effektivität des Codegenerators? Worauf bezieht sich diese?

Die Effektivität bezieht sich auf die Ausführungszeit des generierten Codes. Sie beträgt laut Herstellerangaben ca. 0,9 bis 1,2 im Vergleich mit handerstelltem Code. Dies ist ein hervorragender Wert, da bisherige Codegeneratoren Code mit zweieinhalb- bis dreifacher Laufzeit generierten. Selbst im „worst-case“ sind 20% zusätzliche Laufzeit absolut akzeptabel. Der Speicherplatzbedarf ist zudem ungefähr vergleichbar mit handerstelltem Code.

Diese Angaben wurden im Rahmen dieses Projektes nicht überprüft, hätte dann doch gleichzeitig die gleiche Aufgabe mit erfahrenen Programmierern gelöst werden müssen. Laufzeiten und Codegröße liegen jedoch im erwarteten Rahmen.

Ursächlich für die Laufzeiteffektivität sind im Wesentlichen drei Punkte.

Zum einen ist es die Benutzung von besonderen Codes des Zielprozessors, die einem C-Programmierer nicht zur Verfügung stehen.

Zum anderen ist im Codegenerator implizit viel Spezialwissen über die effektive Programmierung des Zielprozessors und der eingesetzten Betriebssystemumgebung vorhanden, das von einem Codierer erst mühsam erarbeitet werden muss. Weiterhin wird bei sachgerechter Handhabung der Tools bei den Variablen eine Skalierung auf den verwendeten Wertebereich vorgenommen. Dies ermöglicht den Einsatz von günstigen Multiplikations- und Divisionsoperationen, die ansonsten einen hohen Zeitbedarf aufweisen. Die Skalierung kann durch Angabe des Wertebereiches oder automatisch erfolgen.

Ist der generierte Code lesbar?

Der Code ist lesbar, Variablennamen werden ebenso in veränderter, jedoch lesbarer Form übernommen. Es wird lediglich ein Pre- oder Postfix angehängt, um die jeweilige Datenstruktur mit der übergeordneten Systemstruktur in Verbindung zu bringen.

Allerdings hängt diese Frage sicherlich damit zusammen, dass seitens der Entwickler noch ein gewisses Misstrauen gegenüber dem generierten Code vorhanden ist. Auch die von Hochsprachencompilern erzeugten Codes werden in der Regel nicht mehr einer Überprüfung durch den Entwickler unterzogen.

Wie gehe ich mit Projekten um, die auf vorhandenen Code zurückgreifen?

Die AFT-TDC ist hier im Vergleich mit anderen Entwicklungswerkzeugen besonders vorteilhaft. Es existieren generell drei Quellen (Bild 6):

1. Die Generierung von Code aus den Simulink-Modellen wie oben beschrieben.
2. Die zusätzliche Addition von handgeschriebenem Code.
3. Die Integrationsmöglichkeit von vorhandenem Code aus anderen Projekten.

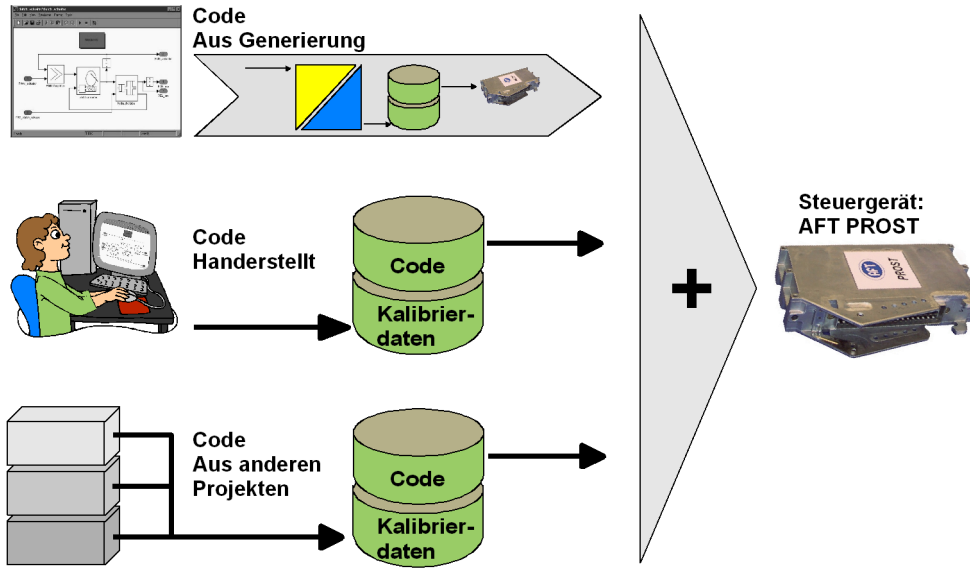

Bild 6: *Vorteil AFT-TDC: Mischen von Code*

In allen Fällen lassen sich, wenn die Konventionen eingehalten werden, auch die Parametrierungen für das Applikationssystem, d. h. die Beschreibungsdaten des Steuergerätes, erzeugen.

Wie weit ist der generierte Code serientauglich?

Wie schon zuvor erwähnt ist der Code seriennah, da er für übliche 16-bit-Mikroprozessoren als Zielsystem generiert wird. Ebenso ist der eingesetzte Funktionsträger, das automotive Prototyp-Steuergerät AFT-**PROST**, seriennah. Die Ergebnisse sind somit in der anschließenden Serienentwicklung weitgehend übertragbar.

Zusammenfassung und Ausblick

Bild 7 fasst noch einmal die generellen Vorteile einer geschlossenen Entwicklungskette sowie die spezifischen Vorteile der AFT-TDC zusammen. Diese Vorteile entspringen den ebenso dargestellten typischen Eigenschaften.

Die Entwicklung der EZA-Software war ein Pilotprojekt, das die Leistungsfähigkeit der AFT-TDC für den Entwurf und die Erstellung von Steuergeräte-Seriencode belegt hat. Bei LuK wird zur Zeit angedacht, nach diesem Vorentwicklungsprojekt auch ein Pilotprojekt in der Serienentwicklung durchzuführen.

Weitere Projekte bei AFT sind in der Vergangenheit erfolgreich abgeschlossen worden. Dazu gehören Projekte für die DaimlerChrysler-Forschung, die die Werkzeuge und Methoden mittlerweile nach entsprechenden Schulungen ihrer Mitarbeiter für die Eigenentwicklung übernommen hat.

Andere Projekte führt AFT als Dienstleister durch, wobei die AFT-TDC als Methode und Werkzeug eingesetzt wird, um Projekte in kurzer Zeit und kosteneffizient durchzuführen. Hierzu gehört auch die Entwicklung einer Steuergerätesoftware für die LuK Fahrzeug-Hydraulik in Bad Homburg.

Zusammenfassung	
Generelle Merkmale TDC:	**Generelle Vorteile TDC:**
• **Durchgängiger simulationsbasierter Entwicklungsprozess vom Entwurf bis zur Implementierung einschließlich aller Testphasen (SiL, HiL, Applikation und Erprobung)** • **Entwurf auf hoher Ebene (modellbasierte Grafik)** • **Ausführbare Spezifikation** • **Implizite Dokumentation** • **Expertise Echtzeitsystem z. T. im Codegenerator**	• **Verkürzung der Entwicklungszeiten** • **Reduzierung möglicher Fehlerquellen** • **Reduzierung der Kosten** • **Verlagerung der notwendigen Expertise zum Systemingenieur** • **Getestete Zwischenergebnisse in jedem Entwicklungsschritt**
Grundmerkmale TDC-AFT:	**Spezifische Vorteile TDC-AFT:**
• **Heterogenes System** • **Zusammenstellung der besten Komponenten, die sich als Standard bewährt haben: MATLAB®/Simulink®, Code-Generator von dSPACE, MARC I, *RAM*boX™, Tornado®**	• **Anpassbarkeit bei sich ändernden Standards** • **Integrationsfähigkeit in bereits bestehende Projekte: Kombination bzw. Mischung von handcodiertem und automatisch erstelltem Code** • **Anpassbarkeit an individuelle Bedürfnisse durch Zusammenstellung für den Kunden** • **Seriennaher Code**

Bild 7: *Zusammenfassung*