EasyKit – Eine allgemeine Methodik für die Entwicklung von Steuerungskomponenten

Dipl.-Inf. Simon Barner, Dipl.-Inf. Michael Geisinger,
Jia Huang, M.Sc., Prof. Dr.-Ing. Alois Knoll
Institut für Informatik VI, TU München
Boltzmannstraße 3, 85748 Garching b. München
Tel. 089/289-18111, Fax. 089/289-18107
{barner, geisinge, huangji, knoll}@in.tum.de

Dipl.-Wirtsch.-Ing. Holger Bönicke, Prof. Dr.-Ing. Christoph Ament
Institut für Automatisierungs- und Systemtechnik, TU Ilmenau
Gustav-Kirchhoff-Straße 1, 98693 Ilmenau
Tel. 03677/69-1508, Fax. 03677/69-1434
{holger.boenicke, christoph.ament}@tu-ilmenau.de

Dr. Jochen Mades
KSB AG - Mechatronik Automation
Johann-Klein-Str 9, 67227 Frankenthal
Tel. 6233/86-3260, Fax. 6233/86-3426
jochen.mades@ksb.com

Dr. Reinhard Pittschellis
Festo Didactic GmbH & Co. KG
Rechbergstraße 3, 73770 Denkendorf
Tel. 0711/3467-1415, Fax. 0711/34754-1415
pitt@de.festo.com

Dipl.-Ing. Gerd Bauer
efm-systems GmbH
Reinsburgstr. 96/1, 70197 Stuttgart
Tel. 0711/65677-112, Fax. 0711/65677-115
bauer@efm-systems.de

Zusammenfassung

Bei der Entwicklung mechatronischer Systeme spielen Entwurf und Implementierung der Steuerungskomponente eine entscheidende Rolle. Hierbei lassen sich die folgenden Trends feststellen: Zentralisierte, oft SPS-basierte Systeme werden vermehrt durch dezentrale Ansätze abgelöst, bei denen die prozessnahe Vorverarbeitung von Sensorsignalen und die Er-

zeugung von Steuersignalen für die Aktorik von effizienten, auf die jeweilige Aufgabe spezialisierten Systemen übernommen werden. Des Weiteren gibt es vermehrt Anstrengungen, die sequenzielle Entwicklung von Maschine, Steuerung und Software aufzubrechen und zu parallelisieren *(Hardware/Software Co-Design)*.

Die *EasyKit*-Methodik ist ein möglicher Ansatz, um diesen Anforderungen gerecht zu werden: Mit einem Elektronik-Baukastensystem steht ein Mittel zur Verfügung, mit dem sich effizient auf die vorliegende Aufgabe zugeschnittene Mikrocontroller-basierte Steuerungen entwickeln lassen, die ohne weitere Beschaltung gängige industrielle Signaltypen verarbeiten und in bestehende Bussysteme und Netzwerke integriert werden können. Die Entwicklung der Software erfolgt mit dem modellgetriebenen Werkzeug *EasyLab* auf hohem Abstraktionsniveau und unabhängig von der verwendeten Hardware-Architektur. Die Modellierung dient als Grundlage für einen vorlagenbasierten Code-Generator sowie eine Simulationskomponente, die den Entwickler bei Planung und Fehlersuche unterstützt. Darüber hinaus erlaubt es eine Debugging-Schnittstelle, Programme, die auf der Zielhardware ablaufen, in Echtzeit zu beobachten und zu manipulieren.

Die universelle Einsetzbarkeit eines Entwicklungsprozesses spielt eine wichtige Rolle für dessen Akzeptanz. Daher zeigen wir anhand von drei Fallbeispielen auf, wie *EasyKit* verwendet werden kann, um sowohl die Entwicklung komplexer industrieller Systeme zu strukturieren als auch die entsprechenden Grundlagen in der Ausbildung anschaulich zu vermitteln.

Schlüsselwörter

Komponentensysteme, Modellgetriebene Entwicklung, Codegenerierung, Didaktikkonzepte für die berufliche und schulische Bildung

1 Einleitung

In mechatronischen Systemen findet zunehmend eine Verlagerung der Auswerte- und Steuerungskomponenten in dezentrale Module statt, die über ein Bussystem verbunden und an eine übergeordnete Steuerung angeschlossen werden können. Gegenüber dem herkömmlichen Ansatz, bei dem die einzelnen Sensoren und Aktoren zumeist analog an Module einer zentralen SPS-Steuerung angebunden sind, bietet ein dezentraler Systemaufbau den Vorteil, dass kleinere (und somit kostengünstigere) auf die jeweilige Aufgabe zugeschnittene Systeme Signalwandlung- und Aufbereitungs- sowie lokale Steuerungsaufgaben in Prozessnähe übernehmen. Während die Verwendung von digitalen Datenbussen sowohl den Verkabelungsaufwand als auch Störeinflüsse reduziert, ermöglichen vorkonfigurierte und vorkalibrierte Komponenten eine verbesserte Wartbarkeit und Austauschbarkeit.

Der Aufbau von Steuerungen durch maßgeschneiderte Systeme erfordert jedoch erhöhte Anforderungen an den Hardware-Entwicklungsprozess, um mit traditionellen, auf Standardkomponenten basierenden Lösungen konkurrieren zu können.

Auch die Programmierung von Steuerungskomponenten stellt eine Herausforderung dar, die in traditionellen Entwicklungsprozessen typischerweise von Hand und für jeden einzelnen Anwendungsfall neu durchgeführt wird. Wegen kürzerer Produktentwicklungszyklen und aus Gründen der Flexibilität werden Entwicklungsprozesse und Werkzeuge benötigt, die auf einem hohen Abstraktionsniveau eine schnellere und zuverlässigere Softwareentwicklung ermöglichen.

Die EasyKit-Entwicklungsmethodik begegnet den genannten Herausforderungen mit einem modularen Ansatz, der die einfache Erweiterbarkeit und Rekonfiguration bestehender Systeme ermöglicht. Dies ist zum einen für den Hardware-Entwicklungsprozess (Kapitel 2) der Fall, dessen unterschiedliche Stufen untereinander softwarekompatibel sind, so dass ab dem ersten Prototypenstadium mit der Softwareentwicklung begonnen werden kann (Hardware-/Software Co-Design). Zum anderen trifft dies auch auf das modellbasierte Entwicklungswerkzeug EasyLab (Kapitel 3) zu, mit dem die benötigte Software mit Hilfe von grafischen Modellen spezifiziert und Code für unterschiedliche Zielplattformen erzeugt werden kann. Die Anwendbarkeit der Methodik wird in Kapitel 4 anhand von drei Fallbeispielen aus dem akademischen, industriellen und didaktischen Umfeld untersucht, bevor in Kapitel 5 ein Resümee gezogen wird.

2 Effiziente Entwicklung von Steuerungen mit dem EasyKit-Baukasten-System

Die Hardware-Entwicklungsmethodik in EasyKit basiert auf der Verwendung von Funktionsgruppen, die in Form physisch verfügbarer Baugruppen oder in Form von Bibliotheken vorliegen können und welche über definierte Schnittstellen miteinander verbunden werden können [VDMA66305].

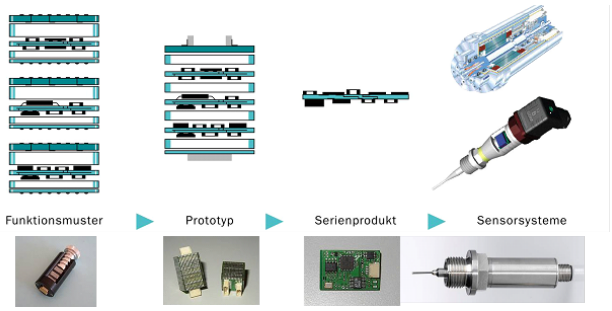

Abbildung 1: Entwicklungsmethodik EasyKit

Um den unterschiedlichen Stufen der Entwicklung vom Funktionsmuster bis zum Serienprodukt Rechnung zu tragen, ist die Entwicklungsmethodik in drei Stufen unterteilt.

- **Stufe 1: Funktionsmuster – Frühe Entwicklungs- und Testphase:** Vorgefertigte und sofort verfügbare Funktionsgruppen werden über definierte Schnittstellen mit Hilfe trennbarer Verbindungen zu kompletten Systemen verbunden, so dass sofort mit der Softwareentwicklung begonnen werden kann.

- **Stufe 2: Prototyp – Entwicklung, Muster-Aufbau sowie Null- und Kleinserien:** Auf Basis vereinfachter Funktionsgruppen aus Stufe 1 werden Komplettsysteme erstellt, die bis hin zu Kleinserien eingesetzt werden können. Der Übergang von Stufe 1 zu Stufe 2 erfordert keinen zusätzlichen Entwicklungsaufwand im Bereich der Elektronik und Software.

- **Stufe 3: Anwendungsspezifisches System – Einsatz für die Serienproduktion:** Durch Nutzung vorhandener Funktionsgruppen können neue, spezifische Designs gleicher Funktion erstellt werden. Die Übernahme der Schaltungen der verwendeten Funktionsgruppen senkt die Kosten der Hardwareentwicklung (die bereits erstellte Software bleibt lauffähig), so dass der leicht erhöhte Materialeinsatz für kleine bis mittlere Serien, wie sie im Anlagenbau meist üblich sind, nicht relevant ist.

2.1 Funktionsträger

Abbildung 2: Funktionsträger

Mit dem Begriff *Funktionsträger* werden elektronische und mechatronische Baugruppen von EasyKit definiert. Diese Funktionsträger stellen in sich abgeschlossene Einzelfunktionen dar, die in Kombination eine übergeordnete Gesamtfunktion erfüllen können, wobei die unterschiedlichen Ausprägungen der Stufen 1 bis 3 möglich sind. Für die Fertigung können verschiedene Technologien und Aufbauformen zum Einsatz kommen – derzeit am weitesten verbreitet ist die Leiterplattentechnologie.

2.2 Bausteine

Abbildung 3: Bausteine

Bausteine sind in sich geschlossene Funktionsträger, die durch Kombination einen direkten Aufbau von Systemen ermöglichen. Da für den Einsatz in Prozessnähe eine kompakte Bauweise und insbesondere für Stufe 1 eine einfache Kombinierbarkeit benötigt wird, wurde auf eine 3D-Aufbauform [LGR97] modifiziert [Bau09]. Der Baustein ist in ein quaderförmiges Gehäuse mit quadratischer Grundfläche gefasst und besteht aus mehreren Ebenen (Funktionsträger und -komponenten), wobei der innere Aufbau dem Hersteller überlassen ist. Die elektrischen Anschlüsse sind auf einen internen Bus geführt, der durch den kompletten Baustein führt. Den Abschluss des Bausteins bildet jeweils eine Boden- und Deckelplatine, die über definierte Schnittstellen an der Ober- und Unterseite verfügen. Die Anschlüsse können in Form von Steckern (Stufe 1) oder in Form von Kontaktierungsflächen (Stufe 2) ausgeführt sein. Im letzteren Fall entsteht so durch abwechselnde Kombination getesteter Einzelfunktionen in Form von Funktionsträgern und Rahmenelementen auf Basis konventioneller Fertigungstechnologien ein kompaktes miniaturisiertes Komplettsystem (Verbindung der Komponenten durch Löten).

2.3 Systemaufbau

Aus den vorhandenen Funktionsträgern lassen sich in Form von Stapeln *Systeme* erstellen. Hierzu stehen Mikrocontroller-Bausteine unterschiedlicher Leistungsfähigkeit (8, 16 oder

32 Bit), Bausteine zur Busanbindung (derzeit: CAN-Bus und Modbus (RS485)) sowie zur Aufbereitung der Eingangsspannungen (von 1,2V bis 35V) auf die erforderlichen Systemspannungen zur Verfügung. Für das Messen, Steuern und Regeln liegen Funktionsgruppen vor, wie etwa analoge und digitale Eingangsstufen (Spannung, Strom), Signalwandlung (10 bis 24 Bit) sowie Ausgangsstufen (z.B. Ansteuerung induktiver Lasten, Spannungs- und Stromausgänge).

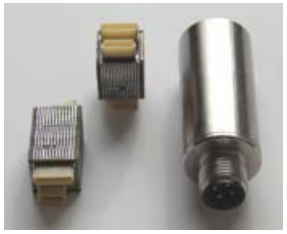

Abbildung 4: Systemaufbau

3 Modellgetriebene Entwicklung von Steuerungssoftware mit EasyLab

Die Software-Entwicklung basiert auf dem Werkzeug *EasyLab* [BGB+08], bei dem die benötigte Software mit Hilfe von Modellen beschrieben wird. Auf deren Basis können Simulationen durchgeführt werden oder lauffähiger Code für das Zielsystem erzeugt werden.

3.1 Modellierung

Das *Gerätemodel* beschreibt die Hardware der Steuerungskomponente und bildet die jeweilige Funktionalität in die Software ab. Hierbei wird eine Steuerungskomponente, die gemäß der in Kapitel 2 beschriebenen Methodik erstellt wurde, durch eine Menge von *Geräten* beschrieben. Ein Gerät kann dabei Untergeräte sowie eine Menge von Softwarekomponenten enthalten, die die Funktionalität des Geräts abbilden. Ein System (Kapitel 2.3) wird also durch ein Gerät beschrieben, das Untergeräte für die jeweils verwendeten Bausteine enthält (Mikrocontroller-Baustein, Treiber für Ein- und Ausgänge, usw.), wobei die Funktionalität der Bausteine durch Funktionsblöcke (s.u.) zugänglich gemacht wird.

Das *Anwendungsmodell* ist ebenfalls hierarchisch aufgebaut und stellt eine funktionale und temporale Beschreibung des Verhaltens mit Hilfe der folgenden Modellelemente dar:

- *Variablen:* Jede der Hierarchieebenen definiert einen neuen Namensraum für Variablen. Globale Variablen sind auf der äußersten Ebene definiert.
- *Ablaufsprache:* Die Ablaufsprache ist an die “Sequential Function Charts” (SFC) der IEC 61131-3 angelehnt und beschreibt Programmzustände und Transitionen zwischen diesen. Jeder Zustand eines Programms der Ablaufsprache hat eine zugeordnete Deadline und referenziert ein Untermodell, das ausgeführt wird, solange sich das Programm in diesem Zustand befindet. Zustände können durch den Sequenz-, den Alternativ-, den Parallel- sowie den Sprungoperator komponiert werden; die Programmausführung beginnt im (eindeutig festgelegten) Startzustand. Zustandsübergange sind mit Transitionsbedingungen annotiert, die aus Booleschen Termen sowie Vergleichen über arithmetischen Termen bestehen.

- *Datenfluss:* Die Datenflusssprache (vgl. Abbildung 8 auf Seite 9) orientiert sich am „Function Block Diagram“ (FDB) der IEC 61131-3 sowie am *Aktor-orientierten* Design [EJL+03]. Ein Datenfluss-Modell ist ein gerichteter Multigraph, in dem jeder Knoten einer Instanz einem bestimmten Funktionsblock-Typen entspricht und in dem Kanten den Datenfluss zwischen Funktionsblock-Instanzen bezeichnen. Ein Funktionsblock wird durch die Menge seiner Eingänge und Ausgänge sowie seiner internen Zustandsvariablen beschrieben. Da jedem Funktionsblocktyp die Aktionen *start*, *step* und *stop* zugeordnet sind und die Ausführungsreihenfolge für (Multiraten)-Datenflussgraphen statisch bestimmt werden kann [LM87, BBH+02], kann die Ausführung wie folgt definiert werden: Sobald der Graph zum ersten Mal ausgeführt wird, werden die s*tart*-Aktionen aller Funktionsblock-Instanzen ausgeführt; gleiches gilt für die *stop*-Aktionen nach der letzten Ausführung. Solange der Graph aktiv bleibt, werden die Funktionsblöcke gemäß des Ablaufplanes ausgeführt, wobei die Ausführung jeweils aus dem Lesen von Eingängen, dem Aufrufen der *step*-Aktion und dem Schreiben der Ausgänge besteht.

3.2 Ausführung

Grundsätzlich lässt sich bei der Ausführung von Modellen zwischen dem Modus unterscheiden, bei dem die Modelle von einer virtuellen Maschine „interpretiert“ werden, sowie dem Modus, bei dem generierter Code nativ auf der Zielplattform abläuft.

3.2.1 Simulation

Simulationen sind ein Mittel zum Nachweis der korrekten Funktionalität von Steuerungskomponenten sowie ein bewährtes Werkzeug während der Entwicklungsphase. Vorgaben für Eingangs- bzw. Sensorwerte können interaktiv während der Simulation verändert werden oder durch ein geeignetes Modell beschrieben werden. Zur Simulation des Effekts von Änderungen von Ausgangs- bzw. Aktuatorgrößen wird ein gesondertes Modell der Regelstrecke benötigt, dessen Erstellung zwar über den Fokus dieser Arbeit hinausgeht, das aber mittels entsprechender Schnittstellen an die Simulation angebunden werden kann.

3.2.2 Codegenerierung

Durch die Verwendung von plattformunabhängigen Modellen in Verbindung mit einem vorlagenbasierten Codegenerator [SSG91] ist es möglich, effizienten und robusten Programmcode für die jeweilige Zielplattform zu erzeugen. Hierzu werden die plattformunabhängigen Spezifikationen des Benutzers schrittweise mit Hilfe von Modell-zu-Modell-Transformationen (wie etwa der Berechnung von Ausführungsplänen) in ein plattformabhängiges Modell umgewandelt, aus dem im letzten Schritt der Code für die Zielarchitektur erzeugt wird. Als Zielsprache wurde *C* gewählt, was einen Kompromiss zwischen Portabilität und Effizienz des erzeugten Codes darstellt, da praktisch für alle relevanten Zielar-

chitekturen entsprechende Compiler zur Verfügung stehen. Die Abbildung des transformierten Modells auf die Zielarchitektur wird durch die beiden folgenden Mittel erreicht:

- *Plattformabhängige Vorlagen:* Für jedes Modellelement (z.B. Funktionsblock-Typ) können unterschiedliche Vorlagen hinterlegt werden, die auf die Besonderheiten der jeweiligen (Hardware- bzw. Software-)Plattform angepasst sind.
- *Laufzeitbibliothek:* Diese Bibliothek abstrahiert gängige Peripherieeinheiten der jeweiligen Zielplattformen hinter einer plattformunabhängigen Schnittstelle (z.B. digitale und analoge E/A, Timer, Kommunikation). Da sich die Codevorlagen auf einer Abstraktionsschicht abstützen, ist es möglich, die Anzahl der plattformabhängigen Codevorlagen so klein wie möglich zu halten, um den Aufwand bei der Portierung des Codegenerators auf eine neue Zielarchitektur zu minimieren.

3.2.3 Echtzeitüberwachung und -parametrisierung

Obwohl die Simulation nicht nur dazu dienen kann, frühzeitig Designfehler zur vermeiden, sondern auch beim Systementwurf hilfreich sein kann (etwa: Dimensionierung von Reglern), basiert diese Technik auf Modellen bzw. Abstraktionen der Umwelt, weshalb weitere Tests von Steuerungskomponenten im realen Umfeld unerlässlich sind. Deshalb ermöglicht es EasyLab, die Ausführung der realen Steuerungskomponente zu visualisieren und erlaubt die Modifikation von Parametern zur Laufzeit (ohne erneute Code-Erzeugung).

Die Visualisierung und Manipulation von Werten ist bei Steuergeräten nicht nur während der Entwicklungsphase, sondern auch zur Wartung und Überwachung von ausgelieferten Systemen über Feldbusse, spezielle Servicegeräte oder eine Mensch-Maschine-Schnittstelle von Bedeutung. Mit sog. *Servicevariablen* stehen hierzu Funktionsblöcke zur Auswahl von Werten zur Verfügung, so dass aus Gründen der Effizienz und Diskretion (interne Größen, sicherheitskritische Parameter) nur auf ausgewählte Werte zugegriffen werden kann. Die Anbindung externer Applikationen erfolgt über eine entsprechende Bibliothek.

3.3 Erweiterbarkeit

Die einfache Erweiterbarkeit und Anpassbarkeit spielt aufgrund der Plattformabhängigkeit des generierten Codes sowie der großen Anzahl möglicher Zielplattformen eine wichtige Rolle. Bei EasyLab ist sowohl die Einbindung neuer *Plattformen* als auch *Werkzeugketten* (Compiler, Linker, Binärdateikonvertierung, Übertragung auf Zielsystem, usw.) möglich. Eine Plattform dient zur Anpassung an die Kombination der jeweiligen Prozessorarchitektur sowie der optional vorhandenen Softwareplattform (Betriebssystem, Middleware) und ist durch benutzerdefinierte Codevorlagen sowie Bibliotheksreferenzen (insbesondere: Laufzeitbibliothek) umgesetzt. EasyLab kann durch benutzerdefinierte Funktionsblock-Typen auch um neue Funktionalität erweitert werden. Die zur Datenübertragung benötigte Kommunikationsschicht (Kapitel 3.2.3) ist abstrahiert und austauschbar.

4 Fallstudien

4.1 Intelligente Pumpe

Die kontinuierlich steigende Funktionsdichte in hydromechatronischen Pumpensystemen [PSO09] wird bei der KSB Aktiengesellschaft über eingebettete Plattformlösungen realisiert. Die enorme Variantenvielfalt im konfigurierbaren Standard-Produktportfolio und die schnelle Bereitstellung von kunden- und auftragsspezifischen Lösungen stellen höchste Ansprüche an die Qualität der entsprechenden Kontrollkomponenten. Um diesen Ansprüchen gerecht zu werden, setzt die KSB AG auf eingebettete Plattformlösungen und auf standardisierte werkzeuggestützte Entwicklungsprozesse.

Die EasyKit-Methodik und Werkzeuge werden diesen Anforderungen sowohl in der Hardware- als auch in der Firmwareentwicklung gerecht. Der produktspezifische Hardwareentwurf erfolgt schnell und einfach mit dem EasyKit-Baukasten, wobei durch das Stecksystem effizient auf Änderungen der Anforderungen reagiert werden kann. Zudem wird durch das 3-Stufen-System sichergestellt, dass produktübergreifend die gleiche Schaltungsvariante einer in Hardware realisierten Funktion verwendet wird.

Mit EasyLab existiert ein modellbasiertes Werkzeug zur Firmwareentwicklung, welches in den KSB-Entwicklungsprozess integriert und mit Hilfe von Servicevariablen (Kapitel 3.2.3) an bestehende Service-Werkzeuge angebunden wurde. Die Erweiterbarkeit von EasyLab (Kapitel 3.3) stellt die Wiederverwendung von Basisfunktionalitäten über alle Produkte hinweg sicher. In der vorliegenden Anwendung wurde eine bestehende Hydraulik-Bibliothek mit ihren speziellen Funktionen wie etwa Förderstromschätzung oder bedarfsabhängige Sollwertanhebung in EasyLab integriert.

Zur Validierung der EasyKit-Methodik wurde in der Fallstudie „intelligente Pumpe“ eine Pumpenauswerteeinheit in Hard- und Firmware aufgebaut, wozu ein Mikrocontroller-Baustein (16 Bit), zwei Bausteine mit analogem Stromeingang (4-20mA), ein Baustein mit analogem Spannungseingang (0-10V) sowie ein Baustein mit analogem Stromausgang (4-20mA) verwendet wurden. Sowohl die Kommunikationsschnittstellen über RS485 (Systembus) und RS232 (Wartung) als auch das LC-Display wurden auf einem Basisboard realisiert (siehe Abbildung 5).

Die Funktionalität die Systems, wie etwa die Berechnung des Differenzdrucks und der reduzierten Förderhöhe, wurden in EasyLab mit der KSB-Hydraulik-Bibliothek zunächst modelliert, per Simulation validiert und nach der Codegenerierung auf dem Zielsystem getestet. Zur Bedienung, Parametrierung und Auswertung wurde eine Anbindung an ein produktübergreifend eingesetztes Service-Werkzeug geschaffen.

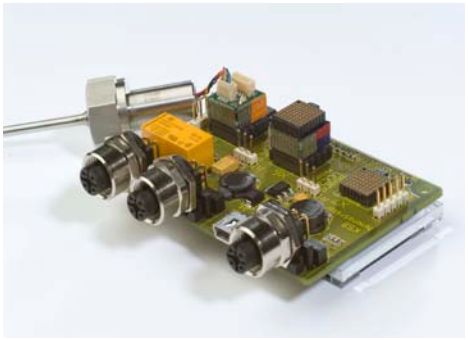

Abbildung 5: Pumpensteuerung (mit LCD auf der Unterseite)

4.2 Pneumatischer Zylinder

Zur Validierung der EasyKit-Methodik wurde eine Positionsregelung für pneumatische Zylinder entwickelt, die i.A. schwer auf Position regelbar sind [Iss08]. Als vorrangiges Problem ist hier der Stick-Slip-Effekt zu nennen, welcher aus der Kombination der Reibung des Kolbens am Gehäuse und der Kompressibilität von Luft resultiert, was mit Hilfe der in Abbildung 6 dargestellten Stribeck-Kurve beschrieben werden kann [Esc94]. In Bereich I wirkt die Haftreibung und es gibt im Zylinder nur elastische Verformungen an der Dichtung zwischen Kolben und Gehäuse. In Bereich II löst sich die Dichtung langsam, so dass eine Mischreibung entsteht. In Bereich III wirkt nur noch die Gleitreibung. Durch den vorher aufgebauten Überdruck bewegt sich der Kolben beim Übergang in den Bereich III mit überproportionaler Geschwindigkeit, was eine Regelung von kurzen Verfahrwegen sehr erschwert, weshalb diese hier nicht betrachtet werden. Der vorliegende Zylinder wurde mit zwei 3/2-Wege-Schaltventilen bestückt, bei denen der Strom zunächst einen Schwellwert überschreiten muss, was das Öffnen und Schließen verzögert (Abbildung 7).

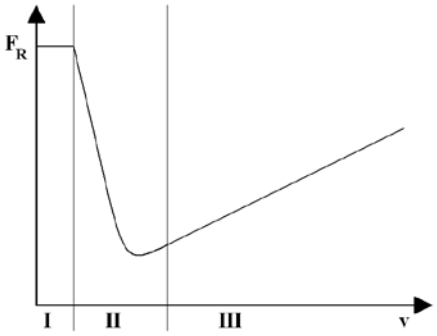

Abbildung 6: Stribeck-Kurve

$$i(t) = I_{max} \cdot \left(1 - e^{-\frac{tR}{L}}\right)$$

Abbildung 7: Stromverlauf

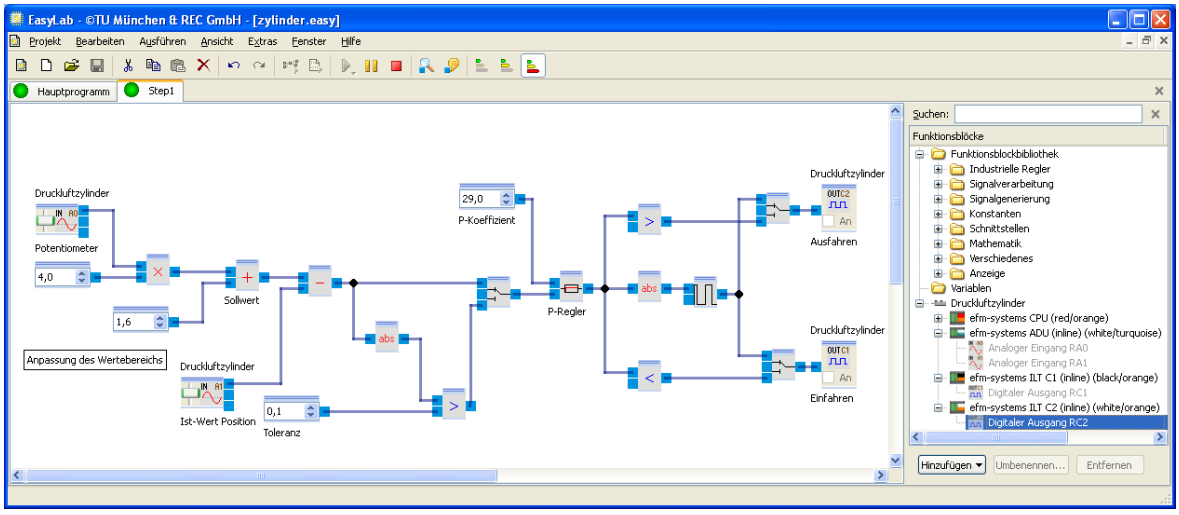

Abbildung 8: Datenflussprogramm zur Positionsregelung (li.) u. Gerätemodell (re. unten)

Zum Aufbau der Steuerelektronik wurde ein Mikrocontroller-Baustein (8 Bit), ein analoger Eingangs-Baustein zur Anbindung eines Positionssensors und eines Potentiometers sowie zwei Bausteine mit je einem Treiber für induktive Lasten verwendet. Zur Positionsregelung werden die Ventile über ein PWM-Signal angesteuert, dessen Tastverhältnis von einem P-Regler erzeugt wird, der mittels Echtzeitüberwachung und -parametrierung (Kapitel 3.2.3) trotz des Stick-Slip-Effekts und der Schaltschwelle der Ventile schnell empirisch

dimensioniert werden kann. Die Positionsregelung in Abbildung 8 besteht aus einem Datenflussprogramm, das – unter Berücksichtigung einer Toleranzgrenze – ein PWM-Signal erzeugt, mit dem das entsprechende Ventil des Zylinders angesteuert wird.

4.3 Didaktikkonzept für allgemeinbildende Schulen

Mikrocontroller, wie sie auch im EasyKit-Baukastensystem zum Einsatz kommen, sind heute allgegenwärtig und finden sich in sehr vielen technischen Geräten. In der technischen Ausbildung an Berufs- und Hochschulen hat das Thema Mikrocontroller daher schon lange seinen festen Platz. Aber auch in allgemeinbildenden Schulen beginnt man sich für das Thema Technik zu interessieren. Dabei finden Mikrocontroller aufgrund ihrer weiten Verbreitung, vielseitigen Anwendbarkeit und nicht zuletzt auch aufgrund relativ geringer Anforderungen an das Budget große Beachtung.

Am Markt verfügbare Mikrocontroller-Entwicklungskits sind sicher preiswert und vielseitig anwendbar, werden in aller Regel aber in C, Basic oder gar Assembler programmiert. Mag dies in der beruflichen oder universitären Ausbildung für die Lernenden noch zu bewältigen und in gewisser Weise sogar sinnvoll sein, so gilt dies in allgemeinbildenden Schulen nicht mehr.

EasyKit und EasyLab ermöglicht nun völlig neue didaktische Konzepte. Ausgehend von der Aufgabe werden die erforderlichen Hard- und Softwarebausteine ausgewählt. Bei der modellbasierten Vorgehensweise wird ein Programm durch Ziehen von Bausteinen auf die Programmieroberfläche und das Einfügen von Kanten erstellt. Durch die Erhöhung des Abstraktionsniveaus kann sich der Lernende anders als in klassischen Programmiersprachen auf die Verarbeitung von Daten und die Problemlösung konzentrieren und muss kaum Rücksicht auf die speziellen Eigenheiten des verwendeten Mikrocontrollers nehmen.

Während die Schüler in C oder Basic nur sehr einfache Aufgaben lösen können, ist es mit EasyLab möglich, auch komplexe Fragestellungen aus der Regelungs-, Signalverarbeitungs- und Nachrichtentechnik zu bearbeiten. Einen besonderen didaktischen Vorteil bietet die in Kapitel 3.2.3 beschriebene Möglichkeit zur Programmüberwachung, womit der Schüler während der Ausführung das Programm beobachten und dadurch eventuelle Fehler sehr schnell finden kann. Die Einstellung von Parametern (z.B. bei Reglern) wird ebenfalls sehr erleichtert (vgl. Kapitel 4.2).

Auch die Simulation (Kapitel 3.2.1) ist in Schulen sehr hilfreich: Die Tatsache, dass die Werte von Eingängen manuell eingestellt werden können und das Verhalten des Programms beobachtet werden kann, ist nützlich, wenn eine Schule aus Kostengründen nicht jedem Schüler einen Mikrocontroller zur Verfügung stellen kann. Die Schüler testen dann ihr Programm zuerst im Simulationsmodus. Erst wenn das Programm in der Simulation zufriedenstellend arbeitet, wird es auf der realen Hardware getestet. Dabei lernen die Schüler auch die für das ingenieurmäßige Arbeiten typische Methode der Simulation kennen.

5 Resümee und Ausblick

Das Ziel der EasyKit-Methodik ist eine Effizienzsteigerung des Entwicklungsprozesses von mechatronischen Steuerungen. Getestete Bausteine liegen in einem Hardware-Baukasten vor, der nicht nur eine schnelle Verbindung und – nach erfolgter Trennung – Wiederverwendung von Komponenten ermöglicht, sondern auch die schrittweise Überführung in ein Serienprodukt mit bedarfsgerechter Fertigungszahl ermöglicht. Ein essenzieller Teil des Ansatzes ist das Modellierungswerkzeug EasyLab, das durch Abstraktion in Form von Geräte- und Anwendungsmodellen den Fokus von der bloßen Implementierung zur eigentlichen Problemstellung verschiebt und das modulare Konzept der Hardware auch auf die Firmware abbildet. Mittels Simulation kann die Funktionsfähigkeit der Firmware auch schon vor dem Vorliegen der endgültigen Version der Hardware nachgewiesen werden (Hardware-Software Co-Design). Wie aus den vorgestellten Fallstudien hervorgeht, ist das Konzept dazu geeignet, Anforderungen aus der industriellen Steuerungsentwicklung gerecht zu werden. Dennoch ist EasyKit aufgrund der einfachen Handhabbarkeit ebenso gut für die berufliche und akademische sowie nicht zuletzt die schulische Ausbildung geeignet.

Literatur

[Bau09] BAUER, G.: 3D Packaging Technologies for PCBs, mst news, 3/2009, S. 22-23.

[BBH+02] BHATTACHARYYA S.S.; BUCK, J.T.; HA, S.; LEE, E.A.: Generating Compact Code from Dataflow Specifications of Multirate Signal Processing Algorithms, Readings in Hardware/Software Co-design. Kluwer Academic Publishers, S. 452-464, 2002.

[BGB+08] BARNER, S.; GEISINGER, M.; BUCKL, C.; KNOLL, A.: EasyLab: Model-based Development of Software for Mechatronic Systems. IEEE/ASME International Conference on Mechatronic and Embedded Systems and Applications, S. 540-545, 2008.

[EJL+03] EKER, J.; JANNECK, J.W.; LEE, E.A.; LIU, J.; LIU, X.; LUDVIG, J.; NEUENDORFFER, S.; SACHS, S.; XIONG, Y.: Taming Heterogeneity – The Ptolemy Approach, Proc. of the IEEE, 2003.

[Esc94] ESCHMANN, R.: Modellbildung und Simulation pneumatischer Zylinderantriebe. Dissertation, Fakultät für Maschinenwesen, Rheinisch-Westfälische Technische Hochschule Aachen, Aachen, 1994.

[Iss08] ISSERMANN, R.: Mechatronische Systeme – Grundlagen, 2. Auflage, S. 483-486, 2008.

[LM87] LEE, E.A.; MESSERSCHMITT, D.G.: Static Scheduling of Synchronous Data Flow Programs for Digital Signal Processing, IEEE Trans. Comp., V. 36, No. 1, S. 24-35, 1987.

[LGR97] LEUTENBAUER, R.; GROSSER, V.; REICHL, H.: Die Entwicklung eines stapelbaren BGA-Package (TB-BGA). Tagungsband SMT/ES&S/Hybrid, Nürnberg, S. 77-84, 1997.

[PSO09] PAULUS, T.; SCHULLERER, J.; OESTERLE, M.: Pumpenintegrierte Prozessregelung: Der intelligente Aktor Pumpe – Stand und Möglichkeiten. apt, 7/2009.

[SSG91] SINGH, A.; SCHAEFFER, J.; GREEN, M.: A template-based Approach to the Generation of Distributed Applications using a Network of Workstations. IEEE Transactions on Parallel and Distributed Systems, Band 2, S. 52-67, 1991.

[VDMA66305] VERBAND DEUTSCHER MASCHINEN- UND ANLAGENBAU, AG MATCH-X: VDMA-Einheitsblatt 66305: Bausteine und Schnittstellen der Mikrotechnik, 2005.

Autoren

Simon Barner schloss 2006 sein Diplom-Studium der Informatik an der TU München ab und ist dort seitdem wiss. Mitarbeiter am Lehrstuhl für Echtzeitsysteme und Robotik.

Michael Geisinger schloss 2007 sein Diplom-Studium der Informatik an der TU München ab. Seitdem ist er dort wiss. Mitarbeiter am Lehrstuhl für Echtzeitsysteme und Robotik.

Jia Huang schloss sein Studium der Kommunikationstechnik an der Shandong Universität, China mit dem Bachelor und an der RWTH Aachen mit dem Master ab. Derzeit ist er wiss. Mitarbeiter am Lehrstuhl für Echtzeitsysteme und Robotik der TU München.

Prof. Dr.-Ing. Alois Knoll schloss 1985 das Studium der Elektrotechnik / Nachrichtentechnik an der Universität Stuttgart ab und wurde 1988 an der Fakultät für Informatik der TU Berlin promoviert, der er bis zur Habilitation 1993 angehörte. Danach war er bis 2001 Professor an der Technischen Fakultät und Direktor der Forschungsgruppe „Technische Informatik“ der Universität Bielefeld. Zwischen 2001 und 2004 war er im Direktorium am Fraunhofer Institut für Autonome Intelligente Systeme tätig, wo er Leiter der Forschungsgruppe „Robotics Construction Kits” war. Seit 2001 ist er Professor an der Fakultät für Informatik der TU München, wo er von 2004 bis 2006 geschäftsführender Direktor der Fakultät war.

Holger Bönicke ist seit 2007 wiss. Mitarbeiter am Fachgebiet Systemanalyse der TU Ilmenau. Hier schloss er 2006 sein Studium mit dem Diplom des Wirtschaftsingenieurs ab.

Christoph Ament ist seit 2005 Professor für Systemanalyse an der TU Ilmenau. Er promovierte 1997 an der Ingenieurwissenschaftlichen Fakultät der Universität Ulm bei Prof. Hofer. Bis 1998 war er Assistent am Lehrstuhl für Robotik der Universität Japan Advanced Institute of Science and Technology in Kanazawa/Japan. Im Anschluss war er bis 2003 technischer Leiter und Oberingenieur im Bereich Messtechnik, Automatisierung und Qualitätswissenschaft am Bremer Institut für Betriebstechnik und angewandte Arbeitswissenschaften (BIBA). Bis zum Ruf auf die TU Ilmenau war er Professor für Systemtheorie am Institut für Mikrosystemtechnik (IMTEK) an der Universität Freiburg.

Dr.-Ing. Jochen Mades ist Leiter Entwicklung Automation–Mechatronik bei der KSB AG in Frankenthal. Nach dem Studium der Elektrotechnik an der TU Darmstadt und der Promotion bei der Infineon Technologies AG im Bereich Mixed-Signal-Simulation war Dr. Jochen Mades Entwicklungsleiter bei der Kobil Systems GmbH.

Dr.-Ing. Reinhard Pittschellis ist seit 2002 Leiter Entwicklung und Produktmanagement bei Festo Didactic GmbH & Co. KG. Nach dem Studium des Maschinenbaus an der TU Braunschweig (1988-1993) wurde er 1998 promoviert.

Gerd Bauer studierte nach seiner Ausbildung zum Werkzeugmacher (1986-1990) Maschinenbau an der TU Ilmenau (1991-1996). Von 1996-2001 war er beim Fraunhofer Institut für Produktionstechnik und Automatisierung an der Entwicklung des Match-X-Baukastens beteiligt. Seit 2002 ist er Gründer und Geschäftsführer der efm-systems GmbH.