OPTIMIERUNG DES STEUERELEKTRONIK-ENTWICKLUNGSABLAUFS DURCH AUTOMATISIERTE SOFTWAREENTWICKLUNG MIT HILFE VON VHDL-AMS

Sven Slawinski, Lutz Zacharias, Mirko Bodach, Thomas Barucki
Westsächsische Hochschule Zwickau, Dr.-Friedrichs-Ring 2A, 08056 Zwickau

Dieses Paper beschreibt die Erstellung und Anwendung eines Werkzeuges für die modellgestützte Softwareentwicklung zum automatisierten Steuerelektronikentwurf mit Hilfe von VHDL-AMS. Als Applikationen dienen hierbei verteilt-alternative Elektroenergieanlagen. Es werden die verschiedenen Teile eines solchen Entwurfsprozesses dargestellt. Das Hauptaugenmerk des Beitrags liegt auf der Funktionsweise des Codegenerators, welcher aus der Hardwarebeschreibungssprache VHDL-AMS den entsprechenden C-Code zur Programmierung der Steuerungselektronik generiert. Das diesem Bericht zugrundeliegende Vorhaben wurde mit Mitteln des Bundesministeriums für Bildung, und Forschung unter dem Förderkennzeichen 1767X09 gefördert. Die Verantwortung für den Inhalt dieser Veröffentlichung liegt beim Autor.

This paper describes the development and use of a tool for model-driven, automated software development to control electronics design by using VHDL-AMS. As test applications serve alternative electrical power systems. The different parts of such a design process will be described. The paper focuses on the way of the code generator operation, which transfers the hardware description language VHDL-AMS into the corresponding C code for programming the control electronics. The project underlying of this report was supported by the Federal Ministry for Education and Research under grant number 1767X09. The responsibility for the content of this publication is located on side of the author(s).

1. Einleitung

1.1 Ausgangssituation

Die Steuerung und Regelung von modernen Elektroenergieanlagen wird heutzutage fast ausschließlich mittels Software realisiert. Derzeit ist der hardwarenahe Software-Entwicklungsprozess für elektronische Steuergeräte und zeitdiskrete Regelungen noch häufig von ermüdender Handarbeit geprägt und entsprechende C- oder gar Assemblerroutinen müssen manuell erstellt werden. Software-Tests werden in vielen Fällen mit Hilfe von Testwerkzeugen (Emulatoren) und vordefinierten Testmustern realisiert, was auch die möglichen Szenarien einschränkt.
Um ein allgemeines Funktionsverständnis zu erreichen, werden häufig mathematische Funktionsmodelle der enthaltenen Komponenten in Simulationen entwickelt und erprobt. Somit ist der Schritt zur modellbasierten Software-Entwicklung naheliegend.
Ausgangspunkt modellbasierter Softwareentwicklung bilden heute meist Blockdiagrammbeschreibungen (z.B. Matlab® & Simulink® von MathWorks®), welche den abzubildenden technischen Prozess auf einer sehr hohen (oft rein regelungstechnischen) Abstraktionsebene widerspiegeln und damit dessen hier zu betrachtenden elektronischen bzw. mechatronischen Charakter nur ansatzweise reflektieren.
Eine Alternative zu Blockdiagramm-Simulationen bietet VHDL-***AMS*** [2], [4], [10], [15], [18] eine standardisierte, nicht proprietäre Hardwarebeschreibungssprache (IEEE 1076.1), mit der digitale, analoge und gemischt analog-digitale Systeme, die aus elektrischen und nichtelektrischen Komponenten aufgebaut sind, modelliert und simuliert werden können. VHDL-AMS erfüllt Anforderungen, wie sie insbesondere vermehrt in der regenerativen Energietechnik, der Automobilindustrie, in Luft- und Raumfahrt, oder auch im Bereich der Medizintechnik auftreten. Ein wesentlicher Vorzug von VHDL-AMS liegt darin, dass sowohl die Steueralgorithmen als auch die anzusteuernde Hardware (Regelstrecke) damit komfortabel beschrieben werden können [6], [11], [12]. Weiterhin sind der plattformunabhängige Standard und die daraus resultierende universelle Modellportierbarkeit ein wichtiger Faktor zur Flexibilisierung des Entwicklungsablaufes.

1.2 Ziel

Das Ziel besteht darin, einen Designflow mit entsprechenden Werkzeugen zu entwickeln, welcher es dem Anwender erlaubt, ohne den Code für die Zielhardware selbst (‚von Hand') programmieren zu müssen, ein komplexes technisches System zu steuern und zu regeln. Nach Aufbau und Test der VHDL-AMS Modelle in einem Simulationssystem sind die Ansteueralgorithmen zu extrahieren und durch einen Codegenerator automatisiert in einen, auf der Zielhardware lauffähigen C-Code zu übersetzen, um diesen zur Ansteuerung des Zielsystems zu implementieren.

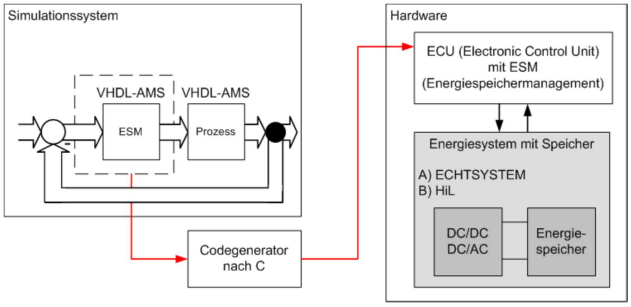

Bild 1: Systemkonfiguration

Die vornehmliche Aufgabe im hier behandelten Projekt ist die Umsetzung der oben erläuterten automatischen Code-Generierung für Testapplikationen aus

dem Bereich dezentral-regenerativer Energieversorgungssysteme (siehe dazu auch Bild 1).Als geeignete Zielhardware dient im vorliegenden Fall ein duales Speichersystem für die regenerative Energietechnik. Es besteht aus einem Kurzzeitspeicher (Superkondensatorpackage), einem Batteriesystem und leistungselektronischen Komponenten mit einem Speichermanagement, welches durch die direkte Anbindung an die Simulation erprobt werden soll. Dabei ist die Möglichkeit vorgesehen, die Testapplikationen virtuell mit Hardware-in-the-Loop (HIL) abzubilden.

2 Umsetzung

2.1 Aufbau des Systems aus VHDL-AMS Blöcken

Aufgrund des enormen Entwicklungs- und Testaufwandes für den Code-Generator erfolgt zu Beginn die Entwicklung verschiedenster, einfacher VHDL-AMS-Einzelmodelle. Diese werden innerhalb des verwendeten Systemsimulators Portunus (Hersteller: Adapted Solutions) zur dauerhaften Verwendung als Bibliothek angelegt. Komplexere Simulationsmodelle, welche dann das jeweils gewünschte Verhalten nachbilden, lassen sich aus diesen Einzelmodell-Grundbausteinen einfach und flexibel aggregieren, anordnen und verbinden [1].

2.2 Aufbau der automatischen Code-Generierung

Eine nicht zu unterschätzende Tatsache ergibt sich durch den enormen Sprachumfang, den VHDL-AMS bietet und welcher äquivalent in C-Code abgebildet werden muss. Des Weiteren ist es nötig, auf dem Zielsystem eine Realtime-Umgebung zu schaffen, die eine definierte Codeausführung ermöglicht. Aus diesen Gründen wird anfangs ein reduzierter Sprachumfang unterstützt und zunächst mit den unter Gliederungspunkt 2.1 erwähnten, überschaubaren Einzelmodellen getestet.
Gemäß adäquater Prinzipien der Informatik erfolgt die Übersetzung von einer Quellsprache (VHDL-AMS) zu einer Zielsprache (C-Code) mit Hilfe eines sogenannten Compilers [3], [14], [16] dessen prinzipielle Funktionalität Bild 2 illustriert. Das sogenannte Frontend, welches die Quelldatei einliest und die Zwischenform erzeugt, besteht im Wesentlichen aus einem LEX & YACC-basierten Parser. LEX realisiert dabei die lexikalische Analyse und logische Zerlegung des Quelltextes in sogenannte Tokens, YACC übernimmt die syntaktische und semantische Analyse des entstandenen Symbolstromes. Die Semantikanalyse basiert auf einer BNF-ähnlichen (Backus-Naur-Form, zur Darstellung kontextfreier Grammatiken) Grammatik der Quellsprache. Unter Verwendung von Flex zur Behandlung der LEX-Datei und Bison für die YACC-Datei wird in einer Linuxumgebung der Parser erzeugt [5], [9], [17]. Damit geschieht dann in einem nächsten Schritt nach dem Einlesen und der Prüfung auf Korrektheit über einen Syntaxbaum die Erstellung einer allgemeinen Zwischenform.

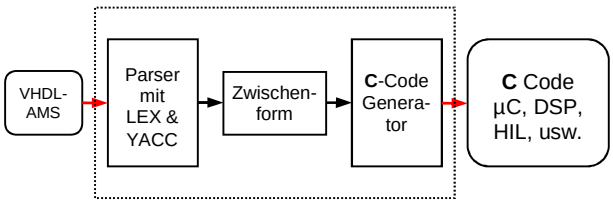

Bild 2: Schematische Darstellung der Codeübersetzung

Daraus wiederum wird durch Ablaufen des Syntaxbaumes anschließend im Backend mittels Codegenerator der dem VHDL-AMS-Modell entsprechende C-Code für die jeweilige Zielhardware generiert.
Da jede Zielhardware (wie z.B. PowerPC, verschiedenste 8-32 Bit µC, DSP, usw.) jeweils andere Hardwarevoraussetzungen sowie Ausstattungen bieten und somit systemspezifische Eigenheiten zur Programmierung erfordern, muss für jedes Zielsystem ein eigenes angepasstes Backend erstellt werden. In diesem Zusammenhang ist auch zu beachten, dass eine definierte Abarbeitungszeit und deren Überwachung mittels Echtzeitumgebung erforderlich sein kann.

2.3 Ergebnis

Erste Ergebnisse liegen in Form eines bereits funktionsfähigen Entwicklungswerkzeug- Prototypen vor, welcher nach Angabe entsprechender Parameter (z.B. Abtast-zeit, Dateinamen) ausgeführt wird. Damit ist es u.a. möglich, eine VHDL-AMS Quelldatei mit eingeschränktem Sprachumfang in C-Code für ein HIL-System von dSpace [7], [8] zu übersetzen. Der generierte Code enthält das gewünschte Verhalten und nutzt mit der angegebenen Abtastzeit die Echtzeitfunktionen des HIL-Systems.
Somit konnte die im Simulationssystem mit Hilfe von VHDL-AMS erstellte Anwendung über den entwickelten automatischen Softwaregenerator – welcher den Steuerungsalgorithmus auf ein HIL-System übersetzt – real nachgebildet werden. Dabei sollte (wie in Bild 3 dargestellt) ein Quelle/Last-System durch gezielte Ansteuerung über das HIL-System in der Lage sein, einen Energiespeicher, wie z.B. einen Superkondensator oder wie hier eine Batterie, nachzubilden. Damit kann die Funktion entsprechend nachgeschalteter Hardware getestet werden, ohne dass die Nachteile eines realen Speichersystems beachtet werden müssen (langwierige Ladevorgänge, Tiefentladungen, Überspannung, usw.).

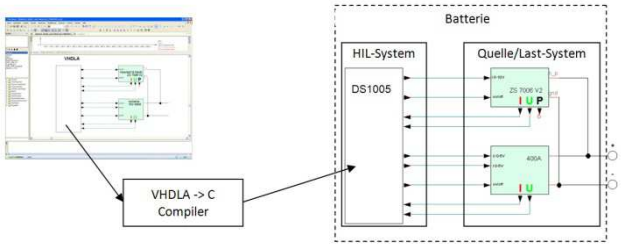

Bild 3: Designflow und beispielhafte Anwendung

Einsatzmöglichkeiten finden sich beispielsweise in der Überprüfung der Funktionsweise von bidirektional wirkenden DC/DC Konvertern für Photovoltaik

Anwendungen. Im derzeitigen Entwicklungsstand des Softwaregenerators ist es lediglich nötig, im entstandenen C- Code einige Programmzeilen zu ergänzen, um das Ansprechen der speziellen AD/DA-Wandler - welche für die Ein/Ausgänge erforderlich sind - zu ermöglichen. In Bild 4 sind die ermittelten Ergebnisse aus Messung einer realen 12 V 11 Ah Batterie, des Simulationsmodelles und des Quelle/Last-Systems mit Ansteuerung durch das dSpace-System vergleichend dargestellt. Darin ist zu erkennen, dass sich das Verhalten des VHDLA-Modells und des automatisch erzeugten C-Codes nicht unterscheiden. Die deutliche Stufung der Messwerte des Quelle/Last-Systems entsteht durch die Abbildung der möglichen 120 V auf den mit 12 Bit aufgelösten 5 V Ansteuereingang der Quelle.

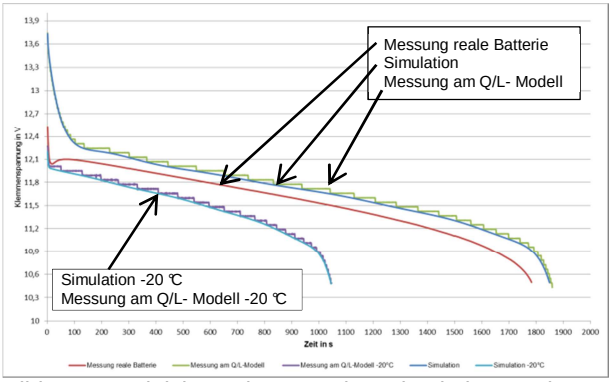

Bild 4: Vergleich reale Batterie, Simulation und Quelle/Last-Modell

3 Zusammenfassung

Zur Realisierung eines durchgängigen Designflows zum Steuerelektronikentwurf ist es gelungen, einen funktionsfähigen Entwicklungswerkzeug- Prototypen zu erstellen. Dieser Prototyp ermöglicht es, VHDL-AMS – aktuell noch mit eingeschränkter Sprachunterstützung – in C-Code zu übersetzen. Der dabei entstandene Programmcode kann direkt mit dem PowerPC-Compiler [13] des HIL-Systems auf die Hardware übertragen werden. Er ist auf diesem Echtzeitsystem lauffähig und bildet das in VHDL-AMS erstellte Verhalten korrekt nach. Somit ist es also gelungen, aus dem VHDL-AMS-Simulationsmodell heraus automatisch C-Code für eine Zielhardware zu erzeugen, ohne dass hierfür umfassende Programmierkenntnisse nötig sind.

4 Ausblick

Im weiteren Fortgang dieser Arbeit ist die Erweiterung des VHDL-AMS Sprachumfanges, welcher in C-Code übersetzt werden kann, eine Hauptaufgabe. Zusätzlich sollen die möglichen unterstützten Zielhardwareplattformen um verschiedenste µC und DSP Familien ergänzt und erweitert werden. Ebenfalls kann eine Ausgabe von ANSI-C implementiert werden um allgemeingültigen Programmcode zur Weiterverarbeitung zur Verfügung zu stellen. Um eine nutzerfreundliche Verwendung des Werkzeuges zu ermöglichen, soll eine grafische Benutzeroberfläche erstellt werden oder es erfolgt die direkte Integration in ein Simulationssystem.

Literaturverzeichnis

[1] Adapted Solutions / Portunus, Übersicht zu VHDL-AMS, Chemnitz, 2009

[2] Alain Vachoux.: VHDL-AMS Instant IEEE, 2006.

[3] Alfred V. Aho, Monica S. Lam, Ravi, Ravi Sethi, Jeffrey S. Ullman, Compiler Prinzipien Techniken und Werkzeuge, PEARSON EDUCATION DEUTSCHLAND GMBH, 2008.

[4] Andreas Mäder: VHDL Kompakt, Technical report, Universität Hamburg, 2005.

[5] Charles Donnelly, Richard Stallman, Bison The Yacc-compatible Parser Generator, November 2009.

[6] Dr. -Ing. Reinhold Vahrmann, Hardware Beschreibungssprachen, Technical report, FH Heilbronn, 1998.

[7] dSPACE, DS1005 PPC Board RTI Reference Release 6.3, dSPACE GmbH, November 2008

[8] dSPACE, DS1005 PPC Board RTILib Reference Release 6.3, dSPACE GmbH, November 2008

[9] Helmut Herold, lex & yacc Die Profitools zur lexikalischen und syntaktischen Textanalyse, ADDISON-WESLEY, 2003.

[10] IEEE, IEEE Standard VHDL Analog and Mixed-Signal Extensions, IEEE, The Institute of Electrical and Electronics Engineers, Inc. 3 Park Avenue, New York, NY 10016-5997, USA, 1999.

[11] Joachim Haase, Simulation mit VHDL-AMS, Technical report, Fraunhofer-Institut für Integrierte Schaltungen Außenstelle EAS Dresden, 2001.

[12] Joachim Haase, P. Schwarz, P. Trappe, W. Vermeiren, Erfahrungen mit VHDL-AMS bei der Simulation heterogener Systeme. Technical report, Fraunhofer-Institut für Integrierte Schaltungen, Außenstelle EAS Dresden, 2000.

[13] Mentor Graphics®, Microtec® C/C++ Compiler User's Guide and Reference for the PowerPC Family Software Version 3.3, Mentor Graphics Corporation, August 2007

[14] Prof. Dr. Jürgen Scholz, Compiler Fachhochschule Augsburg, 2008.

[15] Synplicity, Inc., VHDL handbook, Technical report, Synplicity, Inc., 2007.

[16] U. Kastens, Handbuch der Informatik Übersetzerbau, Oldenbourg Verlag München Wien, 1990.

[17] Vern Paxon, Flex a fast scanner generator. University of California, 1995.

[18] Y. Herve, VHDL-AMS Anwendung und Industrieller Einsatz, Oldenburg Verlag München Wien, 2002.