Online generierte VHDL-Module für arithmetische Operationen

Christian Meier
Prof. Dr. Jürgen Bäsig
Prof. Dr. Hubert Karl

Georg-Simon-Ohm-Fachhochschule Nürnberg

Fachbereich
Elektrotechnik Feinwerktechnik Informationstechnik

Abstract

Diese Veröffentlichung zeigt eine Möglichkeit auf, wie wiederverwendbare VHDL-Module eingesetzt werden können, damit die Implementierungszeit reduziert und die Flächennutzung bei Multiplikationen und Divisionen mit Konstanten optimiert wird. In einer Untersuchung bezüglich der typischen Eigenschaft, dass durch Vergrößern des Näherungsfehlers bei einer Multiplikation mit einer Konstanten eine Flächenreduktionen erfolgt, wurde die Synthese der genäherten und der exakten Multiplikation exemplarisch auf zwei FPGAs und einem ASIC durchgeführt. Der entwickelte Algorithmus liefert die Aufbauvorschrift für die Näherung der Konstanten, die unmittelbar in eine entsprechende Hardwarerealisierung umgesetzt werden kann. Die Konvergenz dieses Algorithmus wurde mathematisch bewiesen und in der Programmiersprache Perl implementiert.
Die Leistungsfähigkeit der automatisch generierten Operationen wurde durch eine regelungstechnische Anwendung verdeutlicht. Hierfür wurde ein Stabwagen-System, eine instabile Strecke, mittels eines digitalisierten Reglers ausgeregelt. Anschließend wurde der Regler in VHDL implementiert und mit einer Systemsimulation, die die digitalisierte Strecke sowie die Modelle der AD-/DA-Umsetzer einschließt, simuliert und anschließend am Stabwagen verifiziert.

ISSN 1616-0762 Sonderdruck Schriftenreihe der Georg-Simon-Ohm-Fachhochschule Nürnberg Nr. 19, Nürnberg 2003

Einleitung

Der Entwurf von Systemen - bestehend aus Hardware und Software - auf einem Chip wird nur dann kosteneffizient durchführbar sein, wenn der weitaus größte Teil auf Basis einer vorgegebenen Plattform wiederverwendet werden kann (IP-Reuse). Voraussetzung für die Wiederverwendung von Hardware ist zum einen der Einsatz einer Hardwarebeschreibungssprache wie z.B. VHDL[1)] und zum anderen die automatisierte Generierung von z.B. arithmetischen Operationen, wie man sie in der digitalen Signalverarbeitung, Regelungstechnik und Bildverarbeitung findet, um nur einige Anwendungsgebiete herauszugreifen.

Eine Multiplikation mit einer festen Gleitkommazahl kann in einer Float-Point-Unit ausgeführt werden. Der Nachteil dieser Float-Point-Unit ist, dass sie eine sehr große Chip-Fläche benötigt [1]. Die Multiplikation oder die Division von Integerzahlen lassen sich mit Hilfe von Schiebeoperationen durchführen. Durch geschicktes Unterteilen des Datenbus in Vor- und Nachkommastellen, lassen sich Multiplikationen mit Gleitkommazahlen mit einem endlichen Fehler annähern. Diese Näherungen können vor allem bei Algorithmen, wie z.B. in einem *Finite Impulse Response Filter*, eingesetzt werden, da bei diesen Filtern kleinere Abweichungen in ihren Koeffizienten keine großen Änderungen in ihrem Verhalten bewirken. So zeigt es sich, dass bei der Multiplikation mit einer approximierten Konstanten der Flächenbedarf von der Genauigkeit der Näherung abhängt. Bild 1 zeigt, dass die benötigte Fläche bei einer exakten Multiplikation am größten ist. Dies wird umso deutlicher, je breiter der Eingangsdatenbus wird. Zudem wirkt sich die Anzahl der benötigten Operationen direkt auf den Flächenbedarf aus. So sind die Näherungen, die mit Additionen und Subtraktionen erfolgen, meist die mit dem geringsten Flächenbedarf, da sie weniger Operationen benötigten um die Konstante anzunähern. Diese Ergebnisse sind auf beide Architekturen anwendbar und somit ist dies eine typische Eigenschaft dieses Näherungsverfahrens. Die Reduktion des Flächenbedarfs ist bei ASICs[2)] größer, als bei FPGAs[2)] . Der Grund liegt an den schon voroptimierten Strukturen in einem FPGA, welche bei einer Multiplikation mit einer Konstanten zum Einsatz kommen.

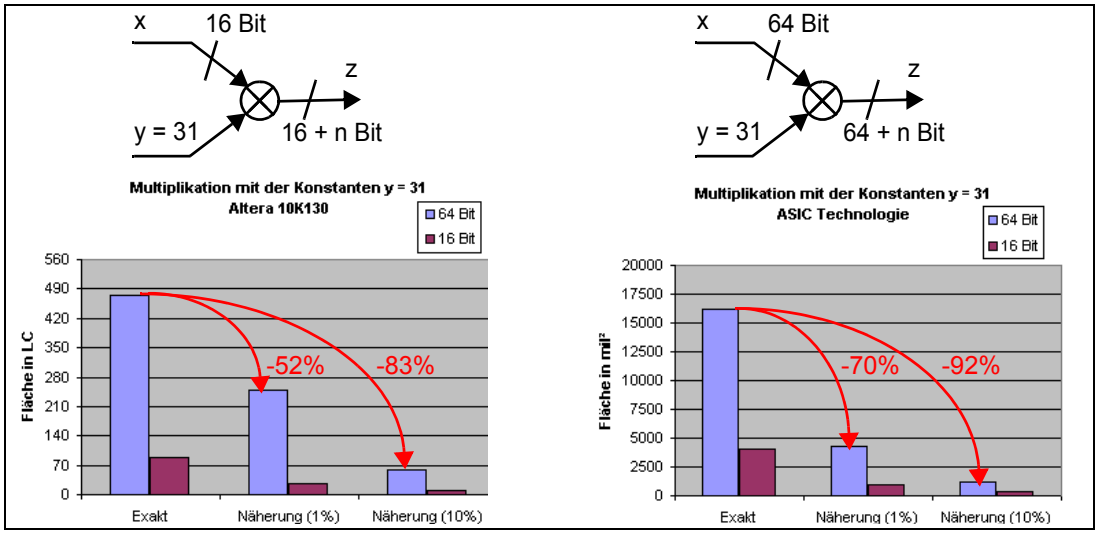


Bild 1: Flächenreduktion durch Verwendung von Näherungen

In einem Forschungsprojekt [2] wurde eine dynamische Webanwendung mit dem Namen CoreGenerator entwickelt. Mit dieser Anwendung können vorgefertigte parametrisierbare Module für die effiziente Hardwarerealisierung in einem ASIC bzw. FPGA über das Internet geladen werden. Für die Generierung der Multiplikation und der Division von Gleitkommazahlen wurde der CoreGenerator um die Kategorie „Arithmetical Operations“ erweitert. Die hierarchische Parametrisierung der Cores geschieht über mehrere Stufen. Am Ende steht die Generierung und das Laden der dynamisch erzeugten Dateien. Ein Core besteht je nach Kategorie aus Core-Dateien, Compilierungs-Vorgaben, Testbenches und Datenblättern.

[1)] VHDL: Very high speed integrated circuits Hardware Description Language
[2)] ASIC: Application Specific Integrated Circuit
FPGA: Field Programmable Gate Array

Bei der Entwicklung des CoreGenerators wurden Java-Scripts, Java-Appletts, Java Server Pages eingesetzt. Der Approximations-Algorithmus für die Generierung der Module für die Multiplikation und die Division von Gleitkommazahlen wird mit Hilfe eines Perl-Scripts realisiert.

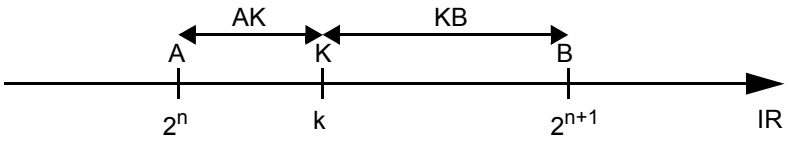
Der Approximations-Algorithmus

Der Approximations-Algorithmus ist das Kernstück des Perl-Scripts. Er nähert die Konstante mit möglichst wenigen Schiebeoperationen an. Das Ergebnis des Algorithmus ist die Aufbauvorschrift, mit der die Multiplikation als Näherungslösung in VHDL realisiert werden kann.

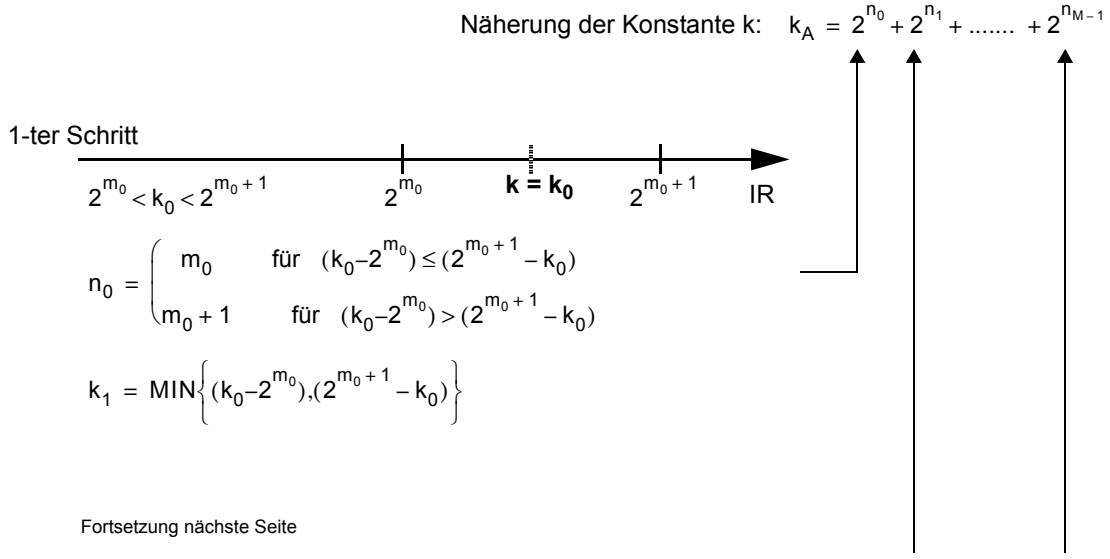
Näherungsverfahren sind Optimierungsprobleme, die mit der Suche nach dem Minimum bzw. Maximum gelöst werden. Die vorgegebene Konstante k wird mit dem Wert k_A solange angenähert, bis

$|k - k_A| < \varepsilon$ ε: vorgegebene Fehlerschranke

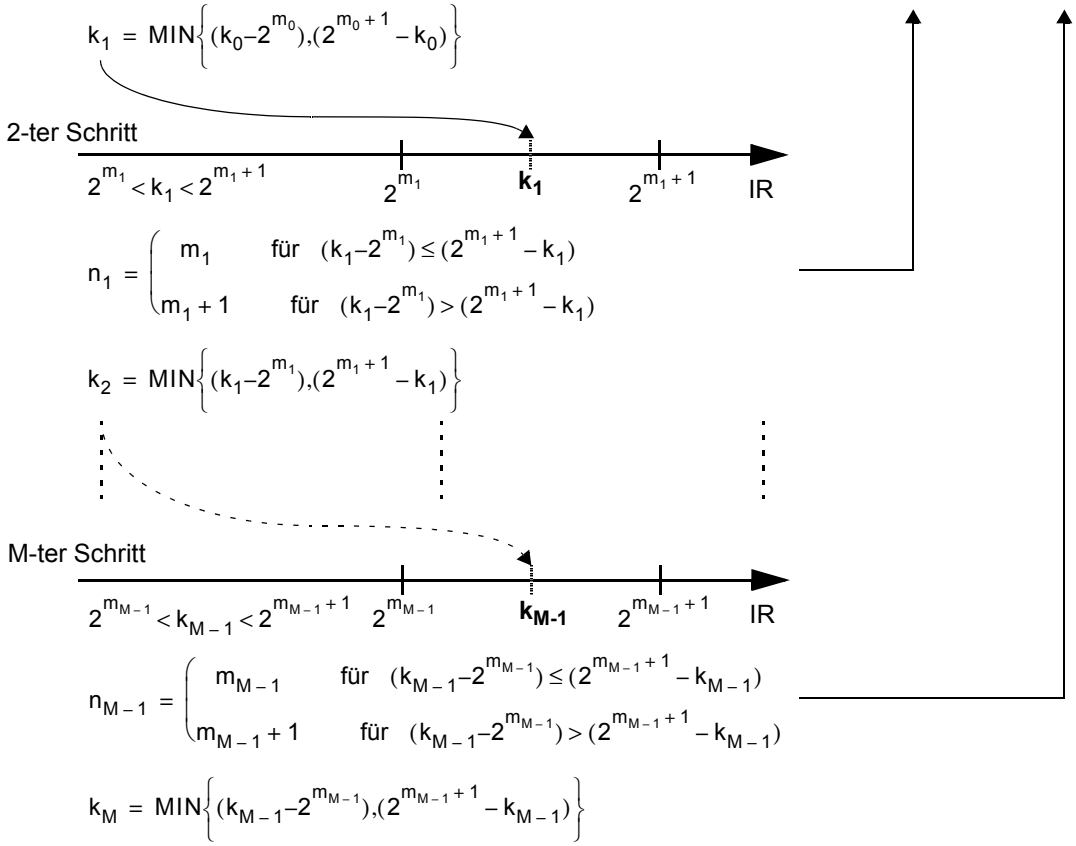
Da bei der vorliegenden Problemstellung ein eindimensionales Optimierungsproblem vorliegt, wird das Verfahren der Intervallschachtelung [3] zur Suche nach dem globalen Minimum eingesetzt. Eine beliebige reelle Zahl k wird mit den Intervallschranken AB, wobei $A = 2^n$ und $B = 2^{n+1}$ entspricht, umschlossen. Anschließend wird das Minimum der beiden Strecken gesucht. Das Minimum entspricht der Länge des kürzesten Streckenabschnittes AK bzw. KB. Dieser kürzeste Abstand wird als der neue Suchwert eingesetzt. Der neue Wert AK bzw. KB wird nun durch erneute Eingrenzung wiederum von den neuen Intervallschranken A'B' umschlossen. Dieser Vorgang wiederholt sich, bis der Wert innerhalb der vorgegeben Fehlertoleranz liegt.



Das folgende Beispiel soll den Algorithmus noch einmal erläutern. Dabei wird o.B.d.A. angenommen, dass die Konstante k > 0 sei.



Fortsetzung nächste Seite



Abbruchkriterium: $\left| k - (2^{n_0} + 2^{n_1} + ... + 2^{n_{M-1}}) \right| < \varepsilon$

Im Folgenden wird gezeigt, dass der Approximations-Algorithmus bei jeder Konstanten |k| > 0 konvergiert. Es ist zu zeigen, dass

$2^{n_0} > 2^{n_1} > ... > 2^{n_{M-1}}$ und damit $n_0 > n_1 > ... > n_{M-1}$ ist.

Aufgrund des Verfahrens der Intervallschachtelung kann angenommen werden, dass folgende Relationen bestehen:

$$MIN\{(k_{\nu-1} - 2^{m_{\nu-1}}), (2^{m_{\nu-1}+1} - k_{\nu-1})\} > MIN\{(k_\nu - 2^{m_\nu}), (2^{m_\nu+1} - k_\nu)\} > MIN\{(k_{\nu+1} - 2^{m_{\nu+1}}), (2^{m_{\nu+1}+1} - k_{\nu+1})\}$$

Unter der Annahme, dass $k_\nu - 2^{m_\nu} < 2^{m_\nu+1} - k_\nu$ und $k_{\nu+1} - 2^{m_{\nu+1}} < 2^{m_{\nu+1}+1} - k_{\nu+1}$ ist, folgt,

dass $k_\nu - k_{\nu+1} > 2^{m_\nu} - 2^{m_{\nu+1}}$ ist. Unter Berücksichtigung, dass $k_\nu > k_{\nu+1}$ gilt, ist $2^{m_\nu} - 2^{m_{\nu+1}} > 0$.

Somit muss - für alle möglichen Kombinationen - gelten, dass $2^{n_\nu} > 2^{n_{\nu+1}}$ ist.

In der folgenden Grafik wird die Arbeitsweise des Approximations-Algorithmus innerhalb von Intervallen bzgl. der Konstanten und des rel. Fehlers veranschaulicht.

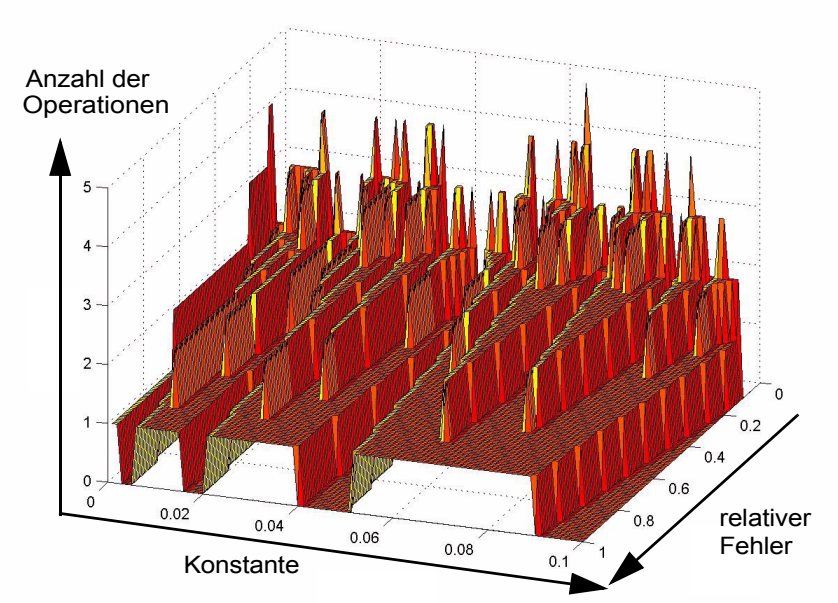


Bild 2: Approximationsergebnis für die Anzahl der Operationen als Funktion der Konstanten und des relativen Fehlers

In der Abbildung ist zu erkennen, dass bei Zahlen einer 2-er Potenz für die Näherung keine Operationen benötigt werden, da diese Zahlen durch alleiniges Schieben realisiert werden können. Als Operation im Sinne der Hardwarerealisierung werden nur Additionen und Subtraktionen gezählt. In der Grafik lässt sich weiterhin erkennen, dass die Anzahl der Operationen stark vom Fehler abhängt; werden größere Fehler zugelassen, so sind weniger Operationen nötig.

Einbindung in den CoreGenerator

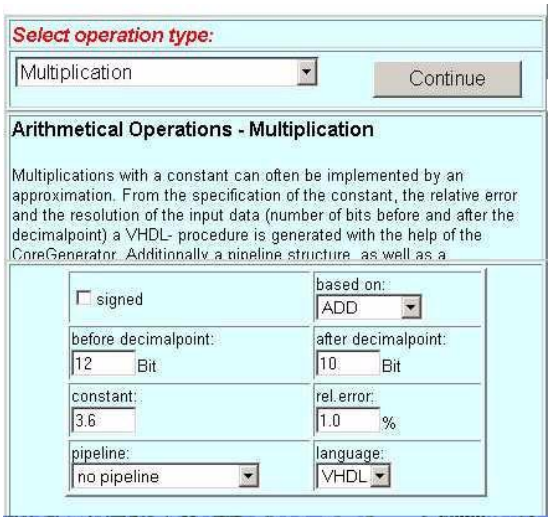


Bild 3: Benutzeroberfläche: Eingabefeld der „Arithmetical Operation - Multiplication“

Für die Generierung und das Laden der Cores ist eine Servlet-Klasse zuständig. Das Servlet erstellt zunächst ein neues Verzeichnis, in dem die generierten Dateien abgelegt werden sollen. Anschließend werden die Dateien durch Aufruf eines Perl-Scripts erzeugt. Nachdem das Script beendet ist, werden die Unterverzeichnisse und Dateien im Zielverzeichnis gepackt und an den Webbrowser weitergeleitet. Dort kann der Benutzer über einen Dialog die Zip-Datei abspeichern. Nachdem die Zip-Datei abgeschickt wurde, wird das Verzeichnis auf dem Webserver mitsamt dem Inhalt wieder gelöscht.
Das Perl-Script berechnet aus den Übergabewerten zuerst die Approximation der Konstanten. Die Konstante wird durch eine Reihenfolge von Schiebeoperationen, die addiert bzw. subtrahiert werden, angenähert. Anschließend wird mit diesen Informationen die VHDL-Procedure erzeugt. Das Perl-Script ist beendet, nachdem eine VHDL-Struktur und das Datenblatt erzeugt und in den von Java vorgegebenen Ordner abgelegt sind. Die Java-Anwendung komprimiert die Dateien und stellt sie zum Abspeichern bereit. Außerdem entnimmt die Java-Anwendung aus einem - vom Perl-Script erzeugten - log-file Informationen und gibt die Anzahl der Operationen und den erreichten Fehler in einem Ausgabefenster zurück.

Ein Anwendungsbeispiel: Stabwagen-System

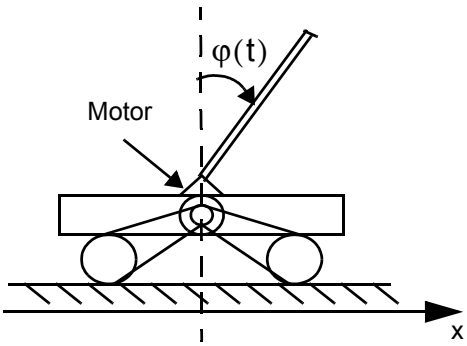


Bild 4: Stabwagen

Das Stabwagen-System wurde an der Technischen Universität Berlin Anfang der 70er Jahre im Labor für Regelungstechnik entwickelt. Dieses instabile System kann mit Hilfe eines wiederum instabilen Reglers betrieben werden. Durch die Zusammenschaltung dieser beiden Systeme entsteht ein stabiles Gesamtsystem.
Der momentan verwendete analoge Regler soll durch einen digitalen Regler ersetzt werden. Die Aufgabe des Reglers besteht darin, die Stellgröße so zu verändern, dass der Wagen den Stab in der senkrechten Lage hält. Als Regelgröße wird der Stabwinkel φ gegen die Senkrechte verwendet, der auf Null ausgeregelt werden soll. Der Stellwert als Ausgangsgröße des Reglers wird in die Leistungseinheit eingespeist. Diese steuert den Scheibenläufermotor, welcher dann den Wagen bewegt.

Der Regler wird mit Hilfe des Wurzelortskurvenverfahrens (WOK) entwickelt. Zur Verifikation wird *Simulink*, ein Simulationstool von *Matlab*, verwendet. Der Regler wird mit Hilfe der Multiplikationen des CoreGenerators angenähert. Diese Näherungen werden wieder in die Wurzelortskurve und in *Simulink* eingesetzt und verifiziert. Ist die Annäherung der einzelnen Multiplikationen in der Wurzelortskurve und in *Simulink* stabil, wird der Regler in VHDL implementiert. Anschließend wird dieser mit Hilfe einer Systemsimulation innerhalb einer Testbench verifiziert. Die vorgegebene Hardware besteht aus einem AD-Umsetzer, der den Winkel des Stabes über ein Potentiometer einliest. Ein FPGA führt die gesamte Steuerung der AD- und DA-Umsetzer und den Regelungsalgorithmus aus. Die Ausgabe des berechneten Stellwertes erfolgt über einen DA-Umsetzer an die Leistungseinheit.

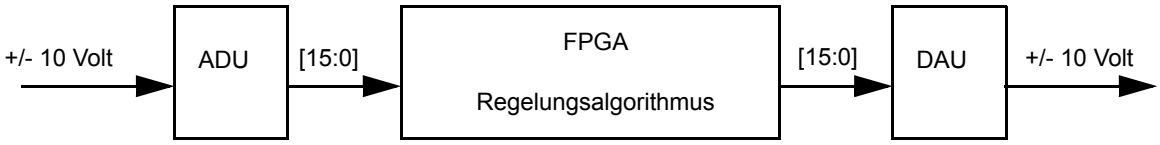


Bild 5: Datenfluss des digitalen Reglers

Beschreibung der Strecke

Ausgehend von den systembeschreibenden Differentialgleichungen [4], ergibt sich für das mathematische Modell des Stabwagens folgendes Strukturbild:

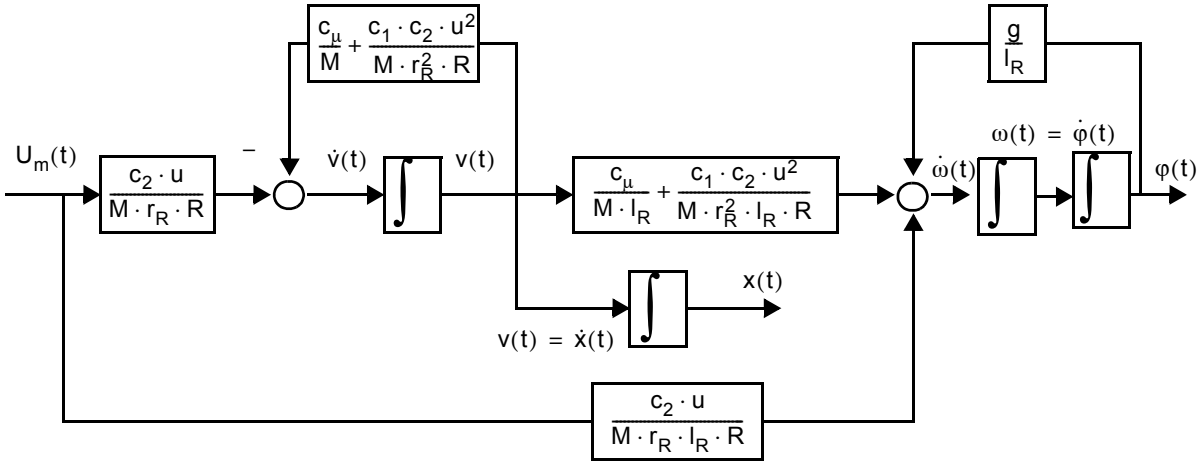


Bild 6: Strukturbild des mathematischen Modells des Stabwagens

Ausgehend von der zugehörigen Übertragungsfunktion $G_S(s)$, welche aus dem Strukturbild ermittelt wurde, wird die diskrete Übertragungsfunktion der Strecke bestimmt (Abtastzeit: 5 ms):

$$G_S(z) = \frac{0,0002489z^2 - 1,064 \cdot 10^{-6}z - 0,0002479}{z^3 - 2,987z^2 + 2,975z - 0,9872}$$

Der approximierte Regler

Die Reglersynthese gelingt mit Hilfe der Wurzelortskurvenmethode. Im analogen Fall ist bei der Wurzelortskurve in der s-Ebene zu beachten, dass diese Kurve, ab einem bestimmten Verstärkungsfaktor (Krs) auf der linken Seite der imaginären Achse liegt. Nachdem ein entsprechendes Krs gefunden wurde, wird der Regler diskretisiert und die Wurzelortskurve in der z-Ebene gezeichnet. Liegt die WOK, für ein bestimmtes Krz innerhalb des Einheitskreises, ist auch das diskrete Regelsystem stabil. Der letzte Schritt ist die Verifikation mit *Simulink*, bei der der komplette Versuchsaufbau incl. der AD/DA-Umsetzer in die Simulation mit einbezogen wird.

Für die realisierte Reglerstruktur im analogen Fall ergibt sich folgende Darstellung:

$$G_R(s) = Krs \cdot \frac{10}{2,57} \cdot \frac{s+1}{s-1} \cdot \frac{s+2,57}{s+10}$$

Bei der Zusammenschaltung der diskretisierten Strecke mit dem diskretisierten Regler ergibt sich folgende Wurzelortskurve:

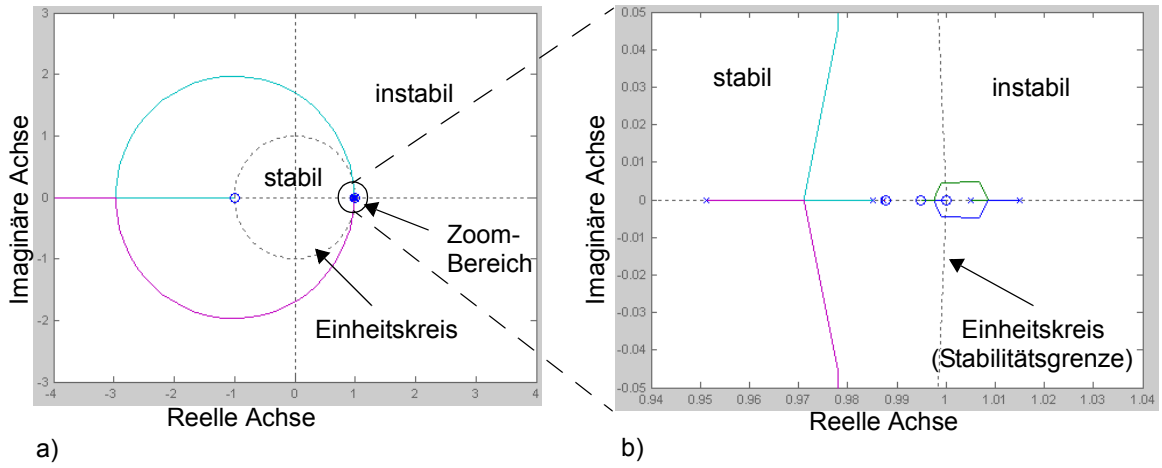


Bild 7: a) Wurzelortskurve, b) Wurzelortskurve, vergrößert

Im Bild 7b ist zu erkennen, dass der Stabilitätsbereich des Systems beschränkt ist. Um ein stabiles System zu erhalten, muss der Verstärkungsfaktor Krz zwischen 1,2 und 25 liegen.

Nachdem der Regler in den Bildbereich der z-Transformation transformiert und mit Hilfe der Wurzelortskurve ein passender Verstärkungsfaktor Krz gefunden wurde, werden die Konstanten der z-Übertragungsfunktion nacheinander angenähert und mit *Simulink* verifiziert, bis das Gesamtsystem stabil ist. Die erste stabile Regler-System-Kombination ist bei einem relativen Fehler von 0.01% gegeben.

Die z-Übertragungsfunktion des Reglers sieht dann wie folgt aus:

$$G_R(z) = \frac{7,6796875z^2 - 15,22265625z + 7,544921875}{z^2 - 1,9560546875z + 0,9560546875}$$

Umsetzung in VHDL

Für die Aufteilung von Kontrollpfad und Datenpfad wurde eine hierarchische Dekomposition gewählt. Dies hat eine klare Aufgaben- und Signaltrennung zur Folge, denn die Steuersignale werden im Kontrollpfad erzeugt und überwacht. Im Datenpfad wird je nach Steuersignal der Datenfluss gebildet.

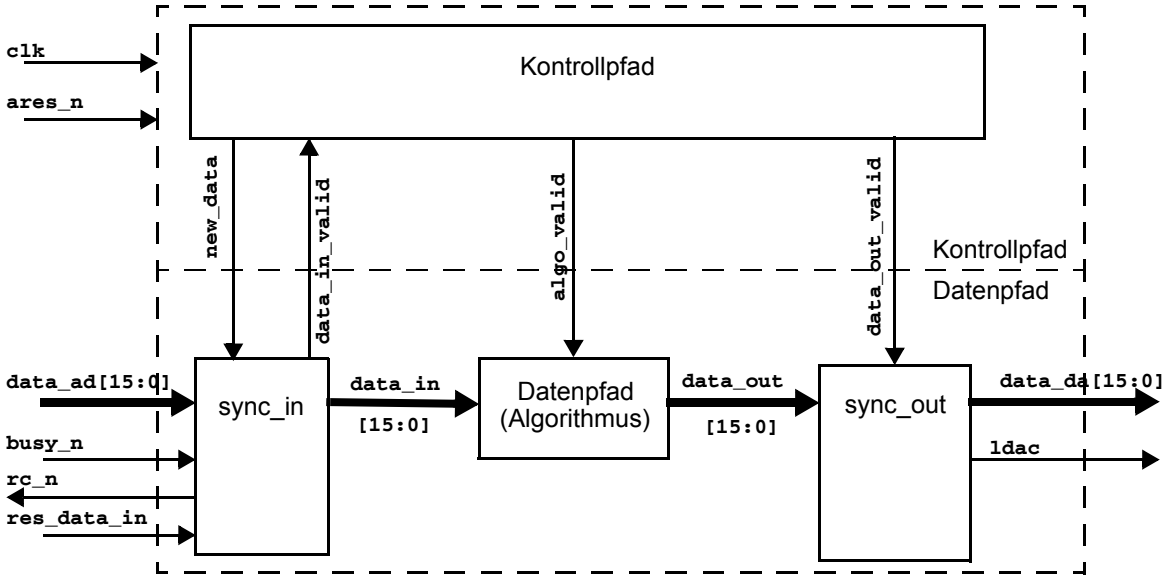


Bild 8: Gewählte Architektur für die Implementierung

Die Kontrollstruktur ist als Zustandsautomat realisiert. Sie fordert das Modul *sync_in* zum Einlesen der Daten auf. Ist dies geschehen, berechnet der Algorithmus den neuen Stellwert. Anschließend wird dieser neue Wert über das Modul *sync_out* ausgegeben.

Simulation der VHDL-Beschreibung

Die Systemsimulation erfolgt mit Hilfe einer VHDL-Testbench, die der *Simulink* Simulation nachempfunden ist. Die Testbench kann die Verhaltens- und die RTL-Beschreibung verifizieren.

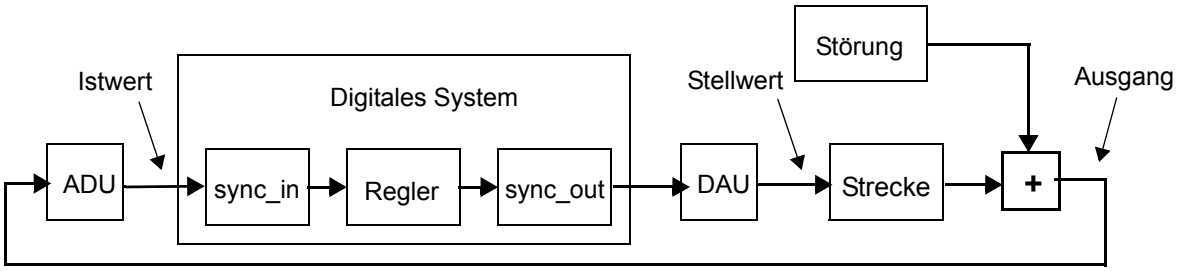


Bild 9: Aufbau der Testbench

Das Modul Strecke enthält die Verhaltensbeschreibung der diskretisierten Strecke. Die Störung erzeugt einen einzelnen Impuls zu einem festen Zeitpunkt. Der Regler beinhaltet, wie auch die *Simulink* Simulation, die Modelle der AD-/ DA-Umsetzer. Diese Modelle haben die Funktion, einen 16 Bit-Datenbus in eine Float-Zahl (DAU), bzw. eine Float-Zahl in einen 16 Bit-Datenbus (ADU) zu konvertieren. Diese Float-Zahl entspricht dem analogen Spannungswert.

Der Ausgang muss sich gleich der *Simulink* Simulation nach einer kurzen Störung wieder auf Null einschwingen.

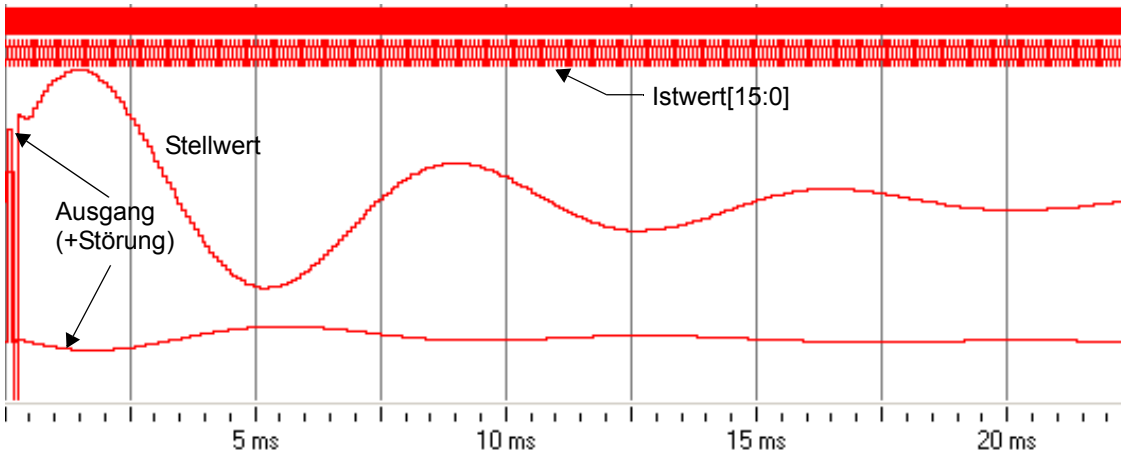
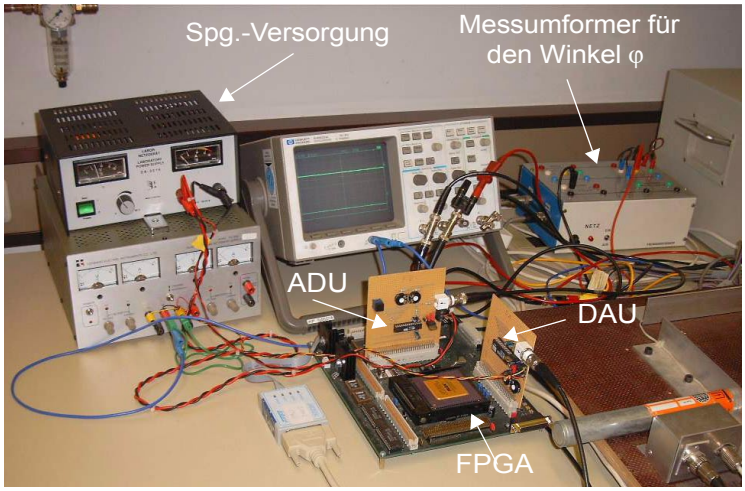


Bild 10: Signalverlauf der Gesamtsystemsimulation

FPGA - Realisierung des Reglers

Um ein stabiles Gesamtsystem zu realisieren, wurde der rel. Fehler der Konstanten auf 0.01% festgelegt. Der dabei erreichte Grenzzyklus[1)] beträgt etwa 35 cm. Der analoge Regler, der im Leistungsteil enthalten ist, hat - zum Vergleich - einen Grenzzyklus von wenigen Zentimetern. Realisiert wurde der Regler auf einem *Rapid Prototyping Board*, welches mit einem 10K130VGC599-3 Baustein von Altera bestückt ist. Der implementierte Regler benötigt 6656 LogicCells (29%) und hat eine max. Systemfrequenz von 2,71 MHz. Aufgrund der automatisch generierten Module für die Multiplikation/Division wird eine Verkürzung der Entwicklungszeit um ca. 60 % erreicht.

a)



b)

Bild 11: a) Versuchsaufbau, b)ausgeregelter Stabwagen

[1)] Grenzzyklus entspricht der Strecke, die der Wagen benötigt, um den Stab in der Senkrechten zu halten

Danksagung

Das dieser Veröffentlichung zugrunde liegende Vorhaben wurde mit Mitteln der Staedtler Stiftung vom 01.03.2002 - 31.12.2002 gefördert. Die Verantwortung für den Inhalt dieses Berichts liegt bei den Autoren.

Literatur

[1] Westphal, D.: Entwicklung einer Floating-Point-Unit als wiederverwendbaren VHDL-Core, Diplomarbeit am Fachbereich Nachrichten- und Feinwerktechnik der Georg-Simon-Ohm-Fachhochschule Nürnberg, WS 2001/2002.

[2] Bäsig, J. / Melnyk, V.: Data Encryption Standard (DES) CoreGenerator - Effiziente Hardwarerealisierung von Verschlüsselungsalgorithmen für ASICs und FPGAs, Schlussbericht, BMB+F FKZ 1705200, 01.10.2000 - 31.01.2002

[3] Krabs, W.: Einführung in die lineare und nichtlineare Optimierung für Ingenieure, B. G. Teubner Verlag, Stuttgart, 1983, S.83.

[4] Zill, T.: Zustandsregelung und Zustandsbeobachtung, Diplomarbeit am Fachbereich Nachrichten- und Feinwerktechnik der Georg-Simon-Ohm-Fachhochschule Nürnberg, WS 1989/1990.