MPC

MULTI PROJEKT CHIP GRUPPE
BADEN-WÜRTTEMBERG

Herausgeber: Hochschule Ulm **Ausgabe:** 51 **ISSN** 1868-9221 **Workshop:** Pforzheim Februar 2014

Cooperating Organisation
Solid-State Circuit Society Chapter
IEEE German Section

Inhaltsverzeichnis

Tagungsband zum Workshop der Multiprojekt-Chip-Gruppe Baden-Württemberg

Die Deutsche Bibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie.

Herausgeber: Gerhard Forster, Hochschule Ulm, Prittwitzstraße 10, D-89075 Ulm

Diesen Workshopband und alle bisherigen Bände finden Sie im Internet unter:

http://www.mpc.belwue.de

Design and Simulation of a Low-Power Analog-to-Digital Converter for Ultra-Thin Chips

Alexander Frank

***Abstract*—The ultra-thin chips manufactured using IMS Chipfilm™ technology are mechanically flexible integrated circuits. The stress in active area of the flexible chips leads to change in electrical behavior of devices. Thus, stress-insensitive circuits are needed to be developed in order to guarantee a functional chip. The proposed analog-to-digital converter does not change its main behavior under mechanical stress. This is achieved by using a conventional ADC topology in addition with a stress-insensitive capacitive array for the DAC. The conversion process is carried out by the successive approximation technique. The presented ADC converter occupies an area of 460 µm × 860 µm in the 0.5-µm CMOS Gate Forest™ technology. Furthermore, the resolution is 8-bit at a sampling rate of 100 kSPS. The power consumption is 615 µW at a supply voltage of 5 V. Its input full-scale range is between 0 V and 5 V, the INL and DNL are +/- 0.4 LSB and the SFDR is 56 dB, respectively. All results are verified by simulations for a stress magnitude from -200 MPa to + 200 MPa.**

***Index Terms*—Analog-to-digital converters, successive approximation, ultra-thin chips.**

I. INTRODUCTION

Today's electronic circuits and their consumer products have reached new applications outside their conventional operational area. The trend is towards portable and low-power circuits which are directly added to this particular operational area in order to process the data without loss of quality and information. In most cases these systems have to be robust against compression, stretching and twisting. However, this flexibility requires particular substrate for the integrated circuits. The drawback of the commonly used technologies is the relatively large thickness of the substrate. In order to make the substrate thinner, one solution is provided by the Institut für Mikroelektronik Stuttgart (IMS). Instead of using the conventional back-thinning technique, IMS produces ultra-thin chips by the so-called IMS Chipfilm™ technology. With this technology process, wafer thicknesses less than 20 µm are achievable as shown in Figure 1 [1]. A subsequent adhesion on a flexible plastic foil makes this system deformable and robust at the same time. But, the resulting flexibility yields to a changed electrical behavior of the integrated circuit processed on this chip. Thus, one of the greatest challenges of this work is to make the integrated circuits fully functioning under mechanical stress.

Alexander Frank, frank@ims-chips.de is with the Institut für Mikroelektronik Stuttgart, Allmandring 30a, 70569 Stuttgart.

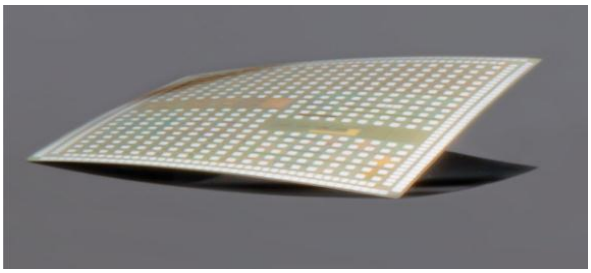

Figure 1: Ultra-thin silicon chip (thickness < 20 µm). The chip is fabricated with the Chipfilm™ technology of IMS CHIPS [1].

In this work an 8-bit analog-to-digital converter is designed which does not change its conversion result under applied mechanical stress.

II. GATE FOREST™ TECHNOLOGY

The mixed-signal Gate Forest™ chips from IMS-Chips provide a huge amount of core cells in a fixed layout in a 0.5-µm CMOS technology. The so-called master chips offer a number of standard die sizes with different numbers of digital and analog core cells. Specific customer designs can be realized by processing the different metallization layers on the top of these preprocessed master wafers. A small volume fabrication can be made in a low-cost manner, because of the reduction to just a customer-specific backend-of-line process. The produced ultra-thin Gate Forest™ chips are fabricated in such a way that the current flow through the most devices is orientated in the [110] crystallographic direction. Any deformation of the ultra-thin chips produces a mechanical stress σ inside these chips (Figure 2).

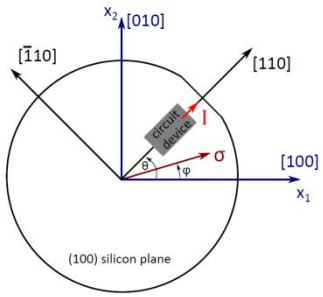

Figure 2: Silicon wafer with the oriented devices. The angle φ of the stress σ is zero in x axis.

The physical effect is well known as the piezoresistive effect and it contributes to a change in the electrical behavior of the different devices depending on their orientation and material. In MOSFETs the applied stress results in a change in the mobility, hence alters the channel current. This current change can be expressed for the [110] orientated MOSFETs by

$$\frac{\Delta I_{D[110]}}{I_{D[110]}} = -\frac{\pi_{11}+\pi_{12}}{2}(\sigma_1+\sigma_2) - \pi_{11}\sigma_3$$

where π_{11} and π_{12} are the piezoresisitve coefficients of the anisotropic silicon and σ_1, σ_2, σ_3 the stress vectors [2]. This leads to a current change of about +/- 10 % in a stress range of -200 MPa to +200 MPa depending on the MOSFET types and stress direction. The *resistor* in integrated circuits is typically made of anisotropic polysilicon. Thus, the resistance change isn't depending on the direction of the applied stress. The resistance change can be expressed by

$$\frac{\Delta R_{poly}}{R_{poly}} = G\frac{\sigma}{E}$$

where G is the gauge factor and E the Young's module which both describe the material's dependency on stress. The change in resistance is typically about +/- 2 % for a stress magnitude of +/- 200 MPa. Hence, resistors as well as MOSFETs are basic circuit elements which are affected by stress and therefore are not suitable candidates for a DAC as a main conversion element in a stress insensitive ADC without compensation circuits.

Another circuit element under investigation is the *capacitor*. Measurements have shown that the capacitance almost doesn't change if any stress is applied either in parallel or perpendicular to the capacitor plates [3]. In the Gate Forest™ technology, sixteen unit capacitors per slot can be used. Each of them is constructed with a poly-poly process for the capacitor plates. In order to design a stress insensitive and also low-power ADC its main conversion part should be built up with the capacitor.

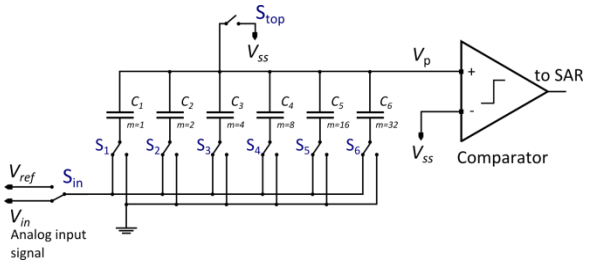

Figure 3: Conventional 6-bit example of the analog-to-digital converter with a binary-weighted DAC [4].

III. ADC Design

A. ADC Conversion Topology

The proposed ADC topology is based on the successive approximation technique. This topology requires only a few basic circuit elements such as a capacitive digital-to-analog converter as the main conversion block, various switches, a comparator and a successive approximation register (SAR) with a control unit (Figure 3) [4]. The complete circuit has a good susceptibility against mechanical stress, because of the used capacitive DAC as the main block. The entire conversion process can be divided into three parts: the sample-mode, the hold-mode and the successive approximation mode.

Each conversion starts with the sample-mode while the input switch S_{in} is connected to the analog input signal V_{in}, the top-plate switch S_{top} is closed and all other bottom-plate switches S_1 to S_6 are also connected to V_{in}. Following the sample mode, a hold mode is started. At this step, all switches S_1 to S_6 are connected to the ground potential, while the top switch S_{top} is opened. At the same time the input switch S_{in} changes its connection to the reference voltage V_{ref}. In this way, a classical sample-and-hold circuit is avoided.

The next step is to open the top switch S_{top} while connecting all bottom-plate switches to the ground potential. The resulting stored charge on the top plate is then proportional to the input voltage and produces a voltage V_p at the comparator as

$$V_p = -V_{in} + V_{ss}$$

After the hold-mode, successive approximation steps must be done in order to get the digital code. It begins by testing the value of the MSB bit. Hence, the bottom plate switch S_1 is connected through S_{in} to V_{ref}. The redistributed charge results in a changed voltage at the input V_p to

$$V_p = -V_{in} + V_{ss} + \frac{V_{ref}}{2}$$

and yields to the corresponding decision of the comparator either

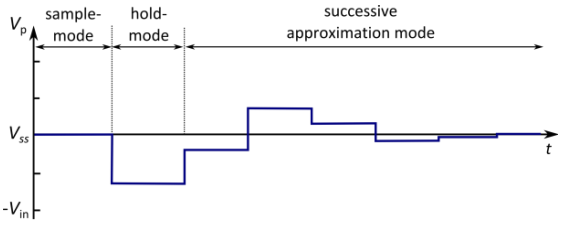

Figure 4: Successive approximation of a conversion sequence.

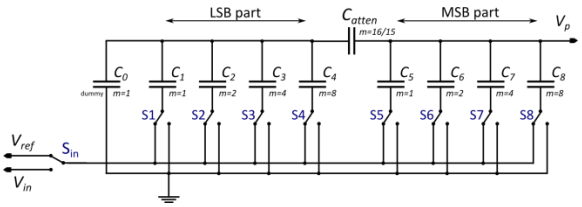

Figure 5: Proposed digital-to-analog converter (DAC) which is divided into two Sub-DACs.

$$V_p \geq V_{ss} \longrightarrow V_{comp} = 0$$

or

$$V_p < V_{ss} \longrightarrow V_{comp} = 1$$

The whole sequence is repeated in this manner with all subsequent bits until V_p reaches V_{ss} as can be seen by [4]

$$V_p = -V_{in} + V_{ss} + \left(S_8 \frac{V_{ref}}{2} + S_7 \frac{V_{ref}}{4} + S_6 \frac{V_{ref}}{8} \ldots\right) = V_{ss}$$

The corresponding output curve is shown in Figure 4.

A benefit of the charge redistribution technique is the slight dependence of the conversion result against the load impedance at the output node V_p of the DAC. Any parasitic capacitors at this node do not distort the conversion result, because the charge is the same in the final configuration as it was in the sample mode. Without the charge redistribution technique the parasitics would produce a gain error which is higher than 1 LSB.

B. ADC Circuit Parts

In this section the different circuits of the proposed ADC are discussed into more detail.

1) Digital-to-Analog Converter

The digital-to-analog converter is conventionally built up with binary weighted capacitors such as described in [5]. The drawback of this structure is the high number of required unit capacitors. In order to reduce the necessary unit capacitors, a Sub-DAC structure is implemented. Therefore, the 8-bit DAC is divided into an LSB-DAC and an MSB-DAC which results in smaller number of 32.1 unit cells compared to

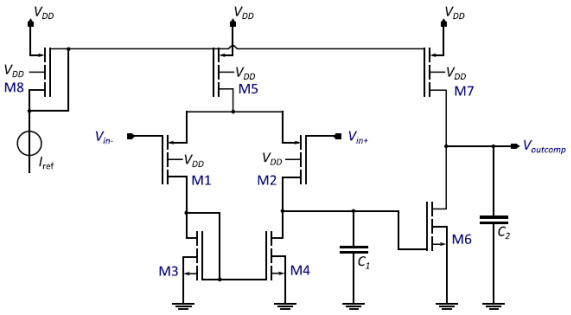

Figure 6: Proposed two-stage comparator.

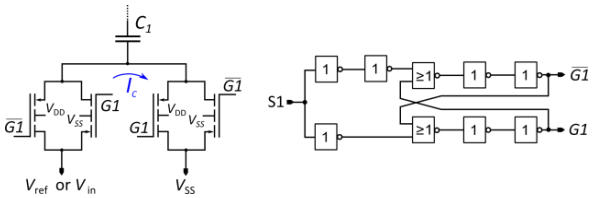

Figure 7: SPDT switch is built with two transmission gates and a non-overlapping signal generator controlled by the input S1.

the conventional binary weighted capacitor array. An attenuation capacitor C_{atten} connects the two Sub-DACs. Figure 5 displays the implemented DAC. The odd number of the attenuation capacitor can be determined by [6]

$$C_{atten} = \frac{Sum\ of\ all\ MSB\ capacitors}{Sum\ of\ all\ LSB\ capacitors}$$

The comparator is one of the critical circuits in the ADC, because its behavior affects the performance of the whole ADC. Main characteristics of the comparator are the propagation delay, power consumption and input resolution. In order to achieve a simple structure and low power consumption a two-stage comparator in an open-loop configuration and PMOS-input pairs are used (Figure 6). It has been proven with simulations that applied stress does not change the decision along the required stress range. The propagation delay of the comparator reaches 200 ns with a bias current of I_{ref} = 5 µA.

2) SPDT Switch

The SPDT (Single Pole Double through) switch S_1 through S_8 and S_{in} are built with two *transmission gates switches* with a non-overlapping signal generator controlled by input S (shown in figure 7). These types are used to reduce the voltage drop along the whole input range and the charge injection. The non-overlapping signal generator avoids the cross current I_C between the two transmission gates.

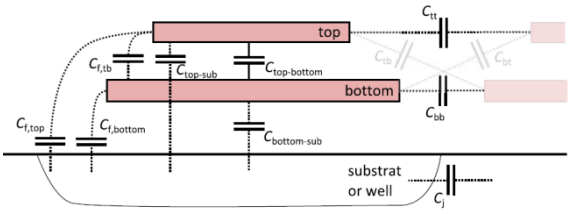

Figure 8: Schematic of a Gate Forest™ unit capacitor with all important parasitics.

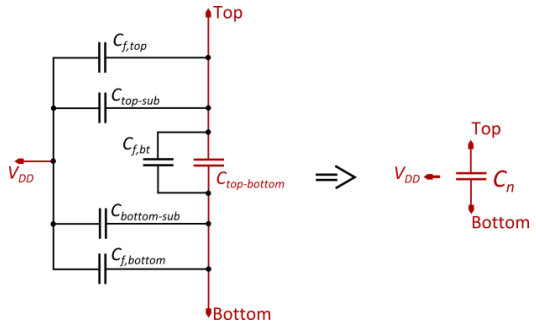

Figure 9: Non-ideal equivalent circuit of one capacitor unit cell.

3) SAR + Control Unit

The successive approximation register is built up with conventional shift registers as described in [7]. An additional control unit guarantees a correct cycle through all conversion steps.

IV. Non-ideal Capacitor Macro model

In order to simulate a realistic behavior of the unit capacitor, a non-ideal capacitor macro model of one unit cell is presented in figure 8. Besides the main capacitance between the top and bottom plate $C_{top\text{-}bottom}$, it includes all parasitic capacitances like the fringe $C_{f\text{-}tb}$ and substrate capacitances $C_{bottom\text{-}sub}$.

Based on the structure of one unit cell, a non-ideal equivalent circuit can be generated. For simplification, the parasitics C_{tt} and C_{bb}, between neighboring unit cells, are neglected. The result is a three terminal device for one unit cell either connected to the ground potential or to V_{DD} depending on the substrate or well connection (shown in Figure 9).

V. Simulation Results

The simulations are carried out under a stress range of −200 MPa to 200 MPa.

A. Differential Non-Linearity

The simulated differential non-linearity (DNL has a maximum of 0.4 LSB and a minimum of -0.4 LSB and is therefore below +/-0.5 LSB, which is shown in Figure 10. This good result confirms the 8-bit resolution of the ADC. It also predicts that there are no missing codes.

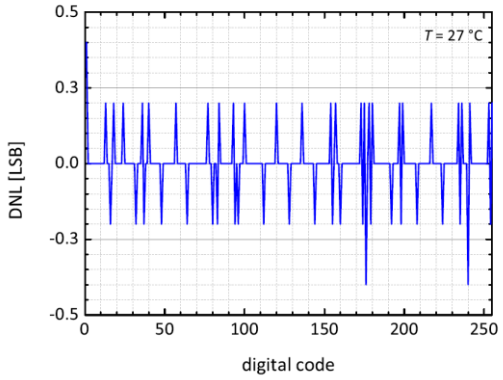

Figure 10: Simulated differential non-linearity of the ADC.

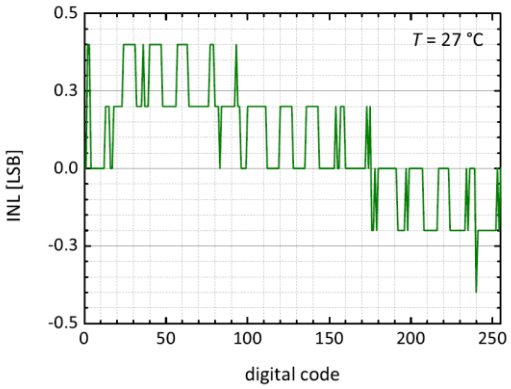

Figure 11: Simulated integral non-linearity of the ADC.

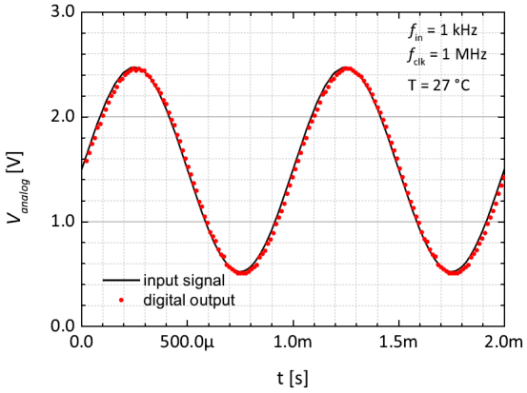

Figure 12: Sampled 1 kHz sinus wave.

B. Integral Non-Linearity

Figure 11 displays the integral non-linearity (INL). Again, the result is with a maximum of 0.4 LSB and a minimum of -0.35 LSB in a good range. Furthermore, the offset is about +0.24 LSB between the digital codes 20_{dec} to 90_{dec} and -0.24 LSB between the codes 70_{dec} to 255_{dec}.

Table 1: Summary of Performance

Simulated Parameter	Value
Technology	0.5-µm (GFQ)
Power consumption	615 µW
Supply voltage	5 V
Stress range	−200 MPa ≤ σ ≤ 200 MPa
Sampling rate	100 kSPS
Input full-scale	0 V – 5 V
Max. INL/DNL	+/- 0.4 LSB
SFDR	56 dB
Chip area	460 µm × 860 µm

C. Sampled Sinus-Wave

A transient simulation of a sampled 1 kHz sinus wave is depicted in Figure 12. It confirms the correct conversion sequence of the ADC.

VI. Gate Forest Layout

The analog block including the Sub-DAC, the comparator and all switches S_1 until S_8 are shown in the Gate Forest™ layout in Figure 13.

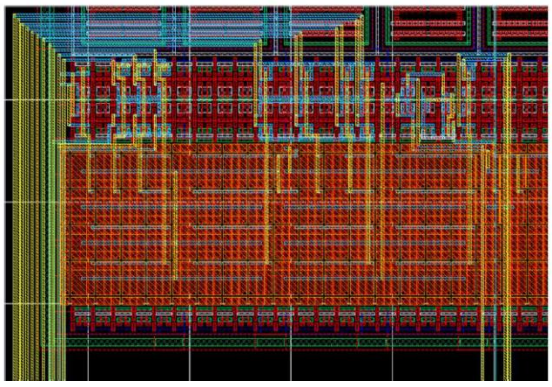

Figure 13: Gate Forest™ layout of the Sub-DAC and all switches.

VII. Conclusion

A low power successive approximation ADC for ultra-thin chips is designed using the 0.5-µm CMOS Gate ForestTM process. Thanks to the stress insensitive capacitive DAC the conversion results are unchanged under applied external stress. Table 1 summarizes the achieved ADC properties.

Acknowledgement

The author would like to thank Y. Mahsereci, C. Scherjon and H. Richter for their support along the way.

References

[1] J. N. Burghartz, W. Appel, C. Harendt, H. Rempp, H. Richter, M. Zimmermann, "Ultra-thin chips and related applications, a new paradigm in silicon technology," *ESSCIRC Proceedings*, pp. 29-36, Sept. 2009.

[2] A. T. Bradley, R. C. Jaeger, J. C. Suhling, K. J. O'Connor, "Piezoresistive characteristics of short-channel MOSFETs on (100) silicon," *IEEE Transactions on Electron Devices*, vol. 48, no. 9, pp. 2009-2015, Sept. 2001.

[3] Y. U. Mahsereci, N. Wacker, H. Richter, J. N. Burghartz, "An ultra-thin CMOS in-plane stress sensor," 9*th Conference on Ph.D. Research in Microelectronics and Electronics* (*PRIME*), pp. 317-320, June 2013.

[4] J .L. McCreary, P. R. Gray, "All-MOS charge redistribution analog-to-digital conversion techniques 1," *IEEE Journal of Solid-State Circuits*, vol. 10, no. 6, pp. 371-379, Dec. 1975.

[5] You-Kuang Chang; Chao-Shiun Wang; Chorng-Kuang Wang, "A 8-bit 500-KS/s low power SAR ADC for bio-medical applications," *Solid-State Circuits Conference*, pp. 228-231, Nov. 2007.

[6] R. J. Baker, "CMOS Circuit Design, Layout, and Simulation," third edition 2010, WILEY, IEEE PRESS.

[7] L. Filipovic, L. MacEachern, "A 10-bit low-power SAR ADC with a tunable series attenuation capacitor," *International Conference on Microelectronics*, 2008. ICM 2008. , vol., no., pp. 399-402, Dec. 2008.

Alexander Frank received the M.Sc. degree in electrical and electronics engineering from the University of Stuttgart, Germany, in 2013. Since 2014 he is with the Institut für Microelektronik Stuttgart (IMS-Chips) and his work is focused on the development of mixed-signal ASICs.

Hochschule Ulm

Eine Ansteuer- und Auswerteeinheit für Wegsensoren nach dem LVDT-Prinzip

Matthias Schnell, Peter Jonski, Gerhard Forster

***Zusammenfassung*—Zur Messung von Wegstrecken existiert eine Vielzahl von Sensoren. Werden besondere Anforderungen an Genauigkeit und Robustheit der Sensoren gestellt, so greift man oft auf Sensoren zurück, die auf dem Prinzip des Differentialtransformators beruhen. Dieser Beitrag befasst sich mit der Entwicklung eines Schaltungsblocks zur Synthese von Ansteuersignalen für Wegsensoren nach dem LVDT-Prinzip (LVDT: Linear Variable Differential Transformer) und zur Auswertung der entsprechenden Ausgangssignale. Die Umsetzung sollte ausschließlich digital erfolgen und als IP zur Realisierung auf einem FPGA bereitgestellt werden. Ausgehend von einer Anforderungsanalyse werden systemtheoretische Überlegungen zum ressourcenschonenden Aufbau und zur Abbildung der Gesamtspezifikation auf das Anforderungsprofil der Teilschaltungen wie PWM, LUT, Tiefpassfilter, Anti-Aliasing-Filter, ADU etc. angestellt. Der IP ist vollständig in MATLAB/Si-mulink modelliert und in VHDL umgesetzt.**

***Schlüsselwörter*—LVDT, VHDL, MATLAB/Simulink, FPGA.**

I. Einleitung

In vielen industriellen und wissenschaftlichen Arbeitsgebieten ist die Messung von Wegstrecken von elementarer Bedeutung. So existiert beispielsweise in den heutigen modernen Luftfahrzeugen eine große Anzahl von Sensor-Aktor-Systemen. In vielen dieser Systeme fallen zurückgelegte Wegstrecken als Messgrößen an, die zur weiteren Verarbeitung benötigt werden. Zur Bestimmung dieser Größen gibt es heute eine Vielzahl an Alternativen.

In der Praxis werden dafür üblicherweise Differentialtransformatoren eingesetzt. Der Vorteil dieser Sensoren liegt in ihrer Robustheit bei gleichzeitiger Präzision sowie der günstigen Integrationsfähigkeit. Man unterscheidet die Sensoren hinsichtlich ihrer Wirkungsweise. So gibt es zum einen die linear bewegten Sensoren, sogenannte Linear Variable Differential Transformers (LVDT), welche eine Wegstrecke messen und zum anderen die rotierend bewegten Rotary Variable Differential Transformers (RVDT), die zur Messung von Winkelgrößen eingesetzt werden.

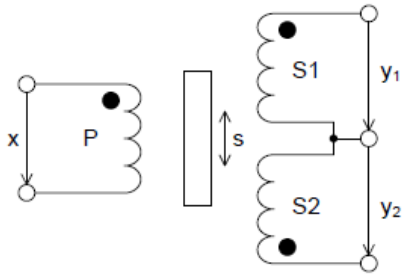

Abbildung 1: Schematischer Aufbau eines LVDT-Sensors.

Beiden Sensoren liegt das Transformatorprinzip zugrunde, bei dem eine an die primärseitige Spule angelegte Spannung zwei sekundärseitig befindliche gegenphasig geschaltete Spulen erregt. Abbildung 1 zeigt die schematische Darstellung eines LVDT-Sensors. Im Ruhezustand des Sensors heben sich die in den Sekundärspulen induzierten Spannungen auf. Zur Bestimmung der Messgröße dient der zwischen den Spulen befindliche Weicheisenkern. Wird dieser verschoben, ändert sich die magnetische Kopplung und damit der symmetrische Aufbau des Transformators und die Messgröße fällt als messbare Spannung auf der Sekundärseite des Transformators an. Aus dieser Spannung kann durch Korrelation mit der Erregerspannung auch eine Richtungsinformation gewonnen werden.

Die algorithmische Auswertung der Sensorsignale kann entweder über eine an den Sensor angebundene Zentraleinheit mittels Software oder durch eine festverdrahtete Logik erfolgen. Der Ansatzpunkt der diesem Beitrag zu Grunde liegenden Arbeit ist die Auswertung mittels FPGA (Field Programmable Gate Array), da diese gegenüber einer softwarebasierten Auswertung Vorteile bietet, insbesondere durch deterministische Ausführungszeiten.

In Abschnitt 2 wird zuerst ein kurzer Einblick in die Funktionsweise von LVDT-Sensoren gegeben. Abschnitt 3 behandelt die Erzeugung des Eingangssignals für den Sensor. Die Darstellung der Signalverarbeitung zur Berechnung der Positionsinformation aus den Sen-

Matthias Schnell, matthias.schnell@newtec.de und Peter Jonski, peter.jonski@newtec.de, NewTec GmbH, Buchenweg 3, 89284 Pfaffenhofen a. d. Roth.

Gerhard Forster, forster@hs-ulm.de, Hochschule Ulm, Prittwitzstraße 10, 89075 Ulm.

sor-Ausgangssignalen erfolgt in Abschnitt 4. Die Ergebnisse der Simulationen werden in Abschnitt 5 behandelt. Abschnitt 6 dient zur Zusammenfassung der Ergebnisse und gibt einen Ausblick auf zukünftige Anwendungen.

II. LVDT-Sensoren

A. Aufbau

Ein LVDT-Sensor besteht im Wesentlichen aus Kern und Hülle, wobei der Kern frei beweglich und im Normalfall auch reibungsfrei innerhalb der Hülle positioniert ist. Der Kern besteht aus einem hochpermeablen Material. Er ist im Allgemeinen mit einem Stab versehen, damit er mechanisch an dem Objekt befestigt werden kann, dessen Bewegung erfasst werden soll. Die Hülle wird ihrerseits unbeweglich fixiert. In der Hülle befinden sich die Spulen, die zur Funktionsweise als Differentialtransformator beitragen. Es gibt drei Spulen, eine Primärspule und zwei Sekundärspulen. Die Spulen sind nach außen hin magnetisch abgeschirmt, um eine Ein- und Auskopplung von elektromagnetischen Störungen zu verhindern.

B. Funktionsweise

Die Primärspule wird mit einer Wechselspannung, dem sogenannten Erregersignal, gespeist. Durch den hochpermeablen Kern wird in den beiden Sekundärspulen jeweils eine Wechselspannung induziert. Diese Spannung ist von der Position des Kerns abhängig. Ist der Kern mittig zwischen S1 und S2 (Nullposition) positioniert, so wird in beiden Spulen dieselbe Spannung induziert. Befindet sich der Kern näher an einer der beiden Spulen, so koppelt an dieser mehr magnetischer Fluss ein und die induzierte Spannung an der Spule steigt, während sie an der anderen Spule um den gleichen Betrag sinkt. Zwischen der Kernposition und der Amplitude der induzierten Spannung in den Sekundärspulen besteht somit ein linearer Zusammenhang. Dieser ist darüber hinaus unabhängig von der Betriebstemperatur sowie der absoluten Amplitude und Frequenz des Erregersignals [1].

Ziel der Arbeit war es, ein System für den Betrieb eines LVDT-Sensors zu entwerfen, das auf einem Chip integrierbar ist. Aufgrund der Funktionsweise des Sensors sind dafür zwei Hauptfunktionalitäten zu realisieren. Zum einen muss das Erregersignal (sinusförmiges Wechselspannungssignal) erzeugt werden und zum anderen muss mittels geeigneter Signalverarbeitung aus den Spannungen an den Sekundärspulen die Kernposition berechnet werden.

Grundsätzlich stehen dafür zwei Methoden zur Verfügung: Entweder wählt man den klassischen Weg und verwendet eine analoge Demodulationsschaltung oder man entscheidet sich, wie in dieser Arbeit geschehen, für eine digitale Signalverarbeitung. Letztere ist heutzutage die bevorzugte Methode, da man von den Vorzügen der digitalen Signalverarbeitung profitiert, vor allem was die Genauigkeit der berechneten Position betrifft. Außerdem lässt sie sich mit einem FPGA umsetzen.

Die Realisierung der zwei „Hauptfunktionalitäten“ Erregersignalerzeugung und Signalverarbeitung der Ausgangssignale soll nun im Folgenden dargestellt werden.

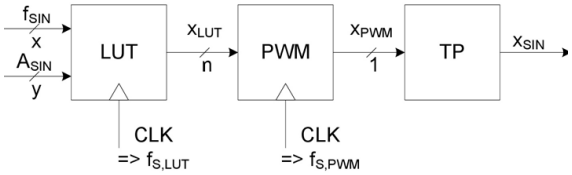

Abbildung 2: Prinzip der Sinussignalerzeugung.

III. Erregersignalerzeugung

An das erzeugte Sinussignal werden folgende Anforderungen gestellt:

- Einstellbare Amplitude A_{SIN} = 1 V … 10 V
- Einstellbare Frequenz f_{SIN} = 1 kHz … 10 kHz
- Klirrfaktor des erzeugten Sinus THD < 3%
- Sinussignal muss gepuffert geliefert werden

Die einstellbare Amplitude und Frequenz sind gewünscht, damit das System für möglichst viele Sensoren eingesetzt werden kann. Da der Betriebsbereich der meisten Sensoren in dem Bereich 1 V … 10 V bzw. 1 kHz … 10 kHz liegt, soll eine Auswahl von Amplitude und Frequenz in diesem Bereich möglich sein. Das erzeugte Sinussignal ist als Erregersignal für einen LVDT-Sensor geeignet, wenn es einen Klirrfaktor von höchstens 3% aufweist [2]. Außerdem muss das Signal gepuffert sein, da der Sensor als Last getrieben werden muss.

In Abbildung 2 ist das gewählte Prinzip der Sinussignalerzeugung dargestellt. Es besteht aus Lookup Table (LUT), Pulsweitenmodulation (PWM) und anschließender Tiefpass-Filterung (TP). An dieser Stelle soll noch angemerkt werden, dass auch andere Möglichkeiten zur Sinussignalerzeugung wie Direct Digital Synthesis (DDS) oder der CORDIC-Algorithmus möglich sind. Da der geforderte Klirrfaktor aber auch mit der gewählten, weniger Ressourcen verbrauchenden, Methode erreicht werden kann, wurde dieses Prinzip umgesetzt. Es wäre ebenfalls möglich gewesen, statt einer PWM einen Parallelbus zur Darstellung des Signalpegels zu verwenden und diesen anschließend geeignet D/A zu wandeln. Allerdings benötigt man für die PWM nur eine einzige Signalleitung, was schließlich zur Wahl der PWM beitrug. Der prinzipielle Ablauf zur Erzeugung des Sinussignals ist wie folgt: Die LUT bein-

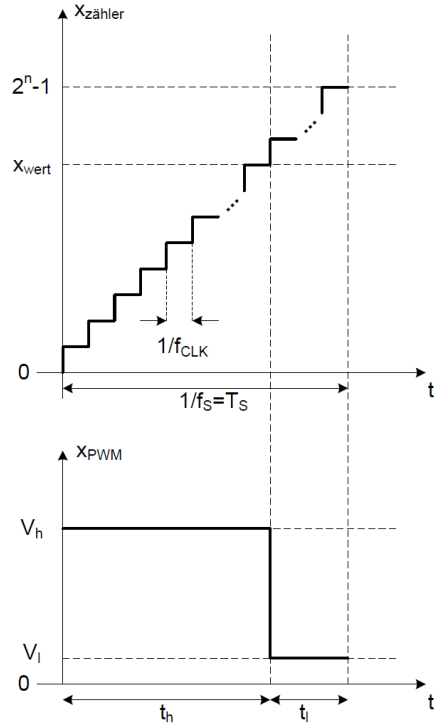

Abbildung 3: Erzeugung des Puls-Pausen-Verhältnisses.

haltet die Abtastwerte eines Sinussignals. Diese Abtastwerte werden durch die PWM in Form eines variablen Puls-Pausen-Verhältnisses (entspricht einem definierten Gleichanteil im Signal) im PWM-Signal umgesetzt. Durch die anschließende TP-Filterung werden hochfrequente Signalanteile ausgefiltert, sodass nur noch der Gleichanteil und somit der darzustellende Signalwert übrig bleibt. Bei entsprechender Auflösung lassen sich auf diese Weise analoge Sinussignale aus den digital hinterlegten Abtastwerten in der LUT erzeugen. PWM kombiniert mit TP-Filterung stellt somit eine Form der D/A-Wandlung dar.

A. Lookup-Tabelle

Aus der Klirrfaktor-Anforderung THD < 3% resultiert die Dimensionierung der LUT mit einer Auflösung von n = 8 bit und m = 100 Abtastwerten pro Periode. Die Wahl dieser Werte erfolgte durch Simulation[1], um sicherzustellen, dass der angestrebte Klirrfaktor unter Minimierung des Ressourcenbedarfs erreicht wird. Die dafür in der LUT benötigte Samplingrate berechnet sich gemäß [3] zu:

$$f_{S,LUT} = m \cdot f_{SIN}$$

Die Samplingrate der LUT bewegt sich damit, je nach Frequenz des zu erzeugenden Sinus, im Bereich $f_{S,LUT}$ = 100 kHz … 1 MHz.

B. Pulsweitenmodulation

Ein PWM-Signal ist ein digitales Signal mit fester Periodendauer aber variierendem Puls-Pausen-Verhältnis (Duty Cycle). Wird das Puls-Pausen-Verhältnis über die Zeit geändert und das PWM-Signal anschließend Tiefpass-gefiltert, so entsteht nach dem Filter ein analoges Signal, da durch das Unterdrücken der hochfrequenten Signalanteile idealerweise nur das durch das Puls-Pausen-Verhältnis dargestellte Gleichsignal übrig bleibt [4]. Durch die Kombination aufeinanderfolgender Abtastwerte eines Signals (bzw. der damit verbundenen Puls-Pausen-Verhältnisse) lassen sich mittels PWM beliebige digitale (abgetastete) Signale in analoge Signale wandeln. Die Erzeugung des Puls-Pausen-Verhältnisses erfolgt dabei mittels eines Zählers, der von „0“ bis „2^{n-1}“ zählt. Erreicht der Zähler den darzustellenden Wert, so ändert sich der Pegel des PWM-Signals von „1“ auf „0“. Dieses Prinzip ist in Abbildung 3 dargestellt. Die dazu nötige Taktfrequenz ist damit 2^n mal so hoch wie die der LUT, da der Zähler innerhalb einer Sampledauer 2^n Zustände durchlaufen muss. Nach [4] gilt damit:

$$f_{S,PWM} = 2^n \cdot f_{S,LUT} = m \cdot 2^n \cdot f_{SIN}$$

Die Samplingrate der PWM bewegt sich damit, je nach Frequenz des zu erzeugenden Sinus, im Bereich $f_{S,PWM}$ = 25,6 MHz … 256 MHz.

Die Frequenz f_{SIN} des Sinussignals wird damit durch die Wahl der Abtastrate realisiert. Zur Realisierung der Amplitude wird eine Reskalierung der Werte aus der LUT vorgenommen. Werden die Logikpegel der PWM mit +10 V und -10 V festgelegt, so entsteht bei Ausnutzen des kompletten Wertebereichs ein Sinus mit einer Amplitude von 10 V. Um nun einen Sinus geringerer Amplitude zu erzeugen, könnte man entweder einen Variable Gain Amplifier (VGA) verwenden, oder man schränkt den Wertebereich der dargestellten Werte ein. Letzteres wurde für die Realisierung gewählt, weil dafür kein zusätzlicher externer VGA benötigt wird. Die Reskalierung nutzt dabei aus, dass sich der darzustellende Abtastwert der LUT zwischen -2^{n-1} und $+2^{n-1}-1$ (symmetrisch um Null) bewegt. Damit kann eine einfache Multiplikation des darzustellenden Werts mit dem Faktor

$$\frac{A_{SIN}}{A_{SIN,\max}} = \frac{A_{SIN}}{10\ \mathrm{V}}$$

vorgenommen werden, um anstatt eines Sinus mit 10 V Amplitude eine Amplitude von A_{SIN} zu erzeugen.

C. Tiefpassfilter

Zur Filterung eines PWM-Signals wird ein Tiefpassfilter verwendet. Die Wahl des Filtertyps und dessen

[1] MATLAB/Simulink

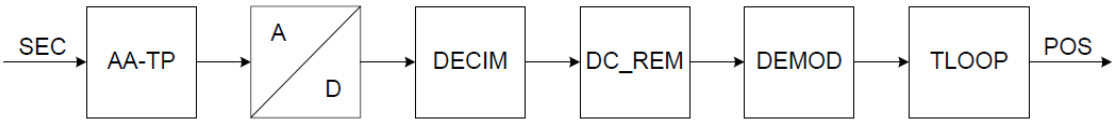

Abbildung 4: Blockschaltbild der LVDT-Signalverarbeitung.

Design haben einen großen Einfluss auf die Performance des gefilterten Signals. Es gibt eine Vielzahl an möglichen Designs für Tiefpassfilter. Die beiden modellierten Alternativen sind ein Butterworth-Filter sowie ein Tschebyscheff Typ II-Filter. Beide liefern eine ähnliche Performance, unterscheiden sich aber in der Art und Weise der Realisierung in Hardware. Ein Butterworth-Filter kann als passives Filter realisiert werden (z.B. als Reihenschaltung von RC-Tiefpassfiltern erster Ordnung) während Tschebyscheff-Filter üblicherweise als aktive Filter realisiert werden. Unabhängig davon, für welche Realisierung man sich entscheidet, bleiben die Spezifikationen des Filters dieselben. Festzulegen sind:

- Erlaubter Ripple im Passbereich
- Eckfrequenz f_{pass} des Passbereichs
- Minimale Dämpfung A_{stop} im Stopbereich
- Eckfrequenz f_{stop} des Stopbereichs

Da die Amplitude des Sinussignals nicht beeinträchtigt werden soll, ist kein Ripple im Durchlassbereich erlaubt. Die Dämpfung im Stopbereich muss so groß sein, dass ein Klirrfaktor von 3% nicht überschritten wird. Eine Dämpfung von

$$A_{stop,\mathrm{Rec}} = 80\ \mathrm{dB}$$

wurde durch Simulation[1] als ausreichend befunden. Die 3dB-Eckfrequenz des Durchlassbereichs wurde so gewählt, dass die höchste vorkommende Sinusfrequenz $f_{SIN,\max}$ das Filter ungehindert passiert:

$$f_{pass,\mathrm{Rec}} = 2 \cdot f_{SIN,\max} = 2 \cdot 10\ \mathrm{kHz} = 20\ \mathrm{kHz}$$

Die Eckfrequenz des Stopbereichs richtet sich nach der geringsten Frequenz im System, die auf jeden Fall ausreichend gedämpft werden muss. In dem vorliegenden System kommen nur die Samplingrate der LUT sowie die Samplingrate der PWM in Frage. Da die Samplingrate der LUT die geringere Frequenz hat, wurde diese als Eckfrequenz festgelegt:

$$f_{stop,\mathrm{Rec}} = f_{S,LUT,\min} = 100\ \mathrm{kHz}$$

Damit ist die Spezifikation des analogen Tiefpassfilters vollständig. Die Realisierung erfolgt je nach gewähltem Filtertyp. Da sich die Arbeit lediglich mit dem digitalen Anteil beschäftigt, wurde keine Realisierung des Filters umgesetzt, sondern lediglich die Spezifikation ausgearbeitet. Das analoge Tiefpassfilter könnte beispielsweise als aktives Filter mittels einer geeigneten OPV-Schaltung realisiert werden, womit gleichzeitig die Bereitstellung eines gepufferten Signals erfüllt wäre.

IV. LVDT-Signalverarbeitung

Das prinzipielle Vorgehen bei der Verarbeitung der LVDT-Signale ist in Abbildung 4 dargestellt. Die beiden vom Sensor ankommenden Signale werden zuerst tiefpassgefiltert (Block AA-TP) um Aliasing bei der anschließenden A/D-Wandlung zu vermeiden. Bei der A/D-Wandung wird mit Überabtastung gearbeitet, um die Auflösung zu erhöhen und die Anforderungen an das Anti-Aliasing-Tiefpassfilter zu entschärfen [5]. Das A/D-gewandelte Signal muss anschließend dezimiert werden (Block DECIM). Nach der Dezimation wird noch der Gleichanteil aus den Sensorsignalen entfernt (Block DC_REM). In einem letzten Schritt werden die Signale demoduliert (Block DEMOD), bevor aus den Signalen mittels eines speziellen Algorithmus (Block TLOOP) die Positionsinformation gewonnen wird. Im Folgenden soll näher auf die Dimensionierung der entsprechenden Systemkomponenten eingegangen werden.

A. Anti-Aliasing-Tiefpassfilter und A/D-Wandler

Entsprechend des Nyquist-Kriteriums muss ein Signal vor der A/D-Wandlung so gefiltert werden, dass es keine Signalanteile oberhalb der halben Abtastrate enthält [5]:

$$f_{Nyq} = 2 \cdot f_{SIG,\max} = 2 \cdot f_{SIN}$$

Das Tiefpassfilter muss also so dimensioniert sein, dass es die Signalanteile bis $f_{SIN,\max}$ (10 kHz) passieren lässt und gleichzeitig alle Signalanteile oberhalb der halben Abtastrate ausreichend dämpft. Für die Dämpfung im Stopbereich wurden dafür

$$A_{stop,AA} = 60\ \mathrm{dB}$$

festgelegt und durch Simulation als ausreichend befunden. Die Eckfrequenz des Passbereichs des Filters wurde so gewählt, dass, wie erwähnt, die höchste vorkommende Sinusfrequenz $f_{SIN,max}$ das Filter ungehindert passiert:

$$f_{pass,AA} = 2 \cdot f_{SIN,\max} = 2 \cdot 10\ \mathrm{kHz} = 20\ \mathrm{kHz}$$

Um die Eckfrequenz für den Stopbereich des Filters zu erhöhen, wird das Signal mit einer wesentlich höheren Abtastrate $f_{S,AD}$ abgetastet, als es nötig wäre. Der Faktor der Überabtastung ist nach [5] folgendermaßen definiert:

$$k_{OS} = \frac{f_{S,AD}}{f_{Nyq}}$$

Bei der Wahl von k_{OS} gilt es, einen Kompromiss zwischen sich ergebender Samplingrate, Gewinn an Auflösung sowie Komplexität des Dezimationsfilters zu finden. Wählt man k_{OS} sehr groß, so werden zwar die Anforderungen an das Anti-Aliasing-Filter entschärft und man erhält einen größeren Gewinn an Auflösung, allerdings kann die dafür nötige hohe Samplingrate im A/D-Wandler nicht (oder nur mit kostenintensiver Hardware) realisiert werden und das Dezimationsfilter wird sehr groß, was mehr Ressourcenbedarf sowie eine erhöhte Gruppenlaufzeit nach sich zieht. Wählt man k_{OS} klein, so kann man mit niedrigen Samplingraten arbeiten und das Dezimationsfilter wird nicht zu groß, allerdings steigen damit die Anforderungen an das Anti-Aliasing-Filter und es ergibt sich ein geringerer Auflösungsgewinn.
Für diese Arbeit wurde ein Faktor von

$$k_{OS} = 128$$

gewählt. Damit berechnet sich die Abtastrate zu:

$$f_{S,AD} = k_{OS} \cdot f_{Nyq} = 256 \cdot f_{SIN}$$

Die Samplingrate bei der A/D-Wandlung bewegt sich damit, je nach Frequenz des Erregersignals (und damit auch der Sensor-Ausgangssignale), im Bereich $f_{S;AD}$ = 256 kHz … 2,56 MHz.

Damit ergibt sich für das Anti-Aliasing-Tiefpassfilter eine Eckfrequenz für den Stopbereich von

$$f_{stop,AA} = \frac{f_{S,AD,\min}}{2} = \frac{256\ \text{kHz}}{2} = 128\ \text{kHz}$$

Zusätzlich zu dem Herabsenken der Anforderungen an das Anti-Aliasing-Filter hat Überabtastung einen weiteren Vorteil: die Erhöhung der Auflösung bei der A/D-Wandlung. Es kann gezeigt werden, dass die Leistung des Quantisierungsrauschens für ein sich stetig änderndes Signal gleichmäßig über alle auftretenden Frequenzen verteilt ist (weißes Rauschen). Bedenkt man, dass die Leistung des Quantisierungsrauschens unabhängig von der gewählten Abtastrate ist, so folgt daraus, dass die Leistungsdichte des Rauschens bei zunehmender Abtastrate sinkt. Durch die anschließende digitale Tiefpassfilterung werden alle Frequenzen oberhalb $f_{SAMPLE}/_2$ unterdrückt, auch das Rauschen. Übrig bleibt damit nur noch das Rauschen unterhalb dieser Frequenz. Damit verbessert sich das SNR bezogen auf eine AD-Wandlung ohne Überabtastung. Nach [5] verbessert sich das SNR (in dB) um

$$SNR_{add} = 10 \cdot \log_{10}(k_{OS})$$

Dies entspricht einem Gewinn von 3dB bei einer Verdopplung der Abtastrate. Der Gewinn an SNR entspricht dabei einem Gewinn an Auflösung von 0,5 bit pro Verdopplung der Abtastrate. Für die A/D-Wandlung wurden A/D-Wandler mit einer Auflösung von 12 bit vorgesehen. Durch die Überabtastung erhöht sich die Auflösung um 3,5 bit. Die maximal verfügbare Auflösung nach dem Dezimationsfilter ist damit 15,5 bit.

B. Dezimationsfilter

Das Dezimationsfilter hat zwei Aufgaben. Zum einen dient es als Tiefpassfilter, um alle Signalanteile oberhalb des relevanten Frequenzbands (DC bis f_{SIGmax}) zu unterdrücken. Zum anderen soll eine Reduzierung der Datenrate (Dezimation) um einen Faktor k_{Dec} vorgenommen werden. Die normierte Grenzfrequenz f_{grenz} des digitalen Filters ist gemäß [6]

$$f_{grenz} = \frac{f_{S,AD}/2}{k_{Dec}} \hat{=} \frac{\pi}{k_{Dec}}$$

Die Dezimation wird dadurch erreicht, dass man nur jeden k_{Dec}-ten Wert aus dem Ausgangssignal des Filters weiter verwendet. Wird ein FIR-Filter verwendet, so hat dies den Vorteil, dass nur jeder k_{Dec}-te Ausgangswert berechnet werden muss, da alle anderen Werte ohnehin verworfen würden. Diesen Umstand kann man bei einem IIR-Filter nicht ausnutzen, da ein Ausgangswert von vorhergehenden Zuständen abhängt und man deshalb alle Werte berechnen muss. Aus diesem Grund wird das Filter meist als FIR-Filter entworfen. Um sich im Falle eines FIR-Filters die Berechnung der unnötigen Ausgangswerte sparen zu können, wird das Filter in einer dafür geeigneten Struktur, der sogenannten polyphasen Struktur, realisiert. Details zur Realisierung eines FIR-Dezimationsfilters in der polyphasen Struktur sind in [6] zu finden.

Zum Entwurf des Filters (bzw. der Filterkoeffizienten) wurde das „Filter Design & Analysis Tool“[2] verwendet. Dazu sind folgende Größen festzulegen:

- Dezimationsfaktor k_{Dec}
- Eckfrequenz f_{pass} des Passbereichs
- Minimale Dämpfung A_{stop} im Stopbereich
- Eckfrequenz f_{stop} des Stopbereichs

Da aufgrund des Vorgehens bei der Demodulation (siehe Abschnitt IV Demodulation) eine Abtastrate von

$$f_{S,Dec} = 4 \cdot f_{SIN}$$

[2] FDA-Tool von MATLAB

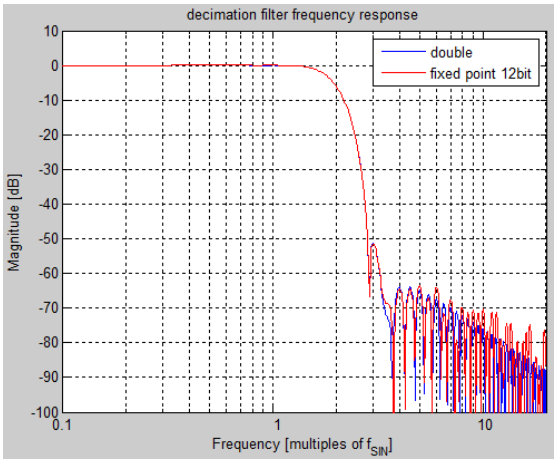

Abbildung 7: Frequenzgang des Dezimationsfilters mit 12bit-Koeffizienten.

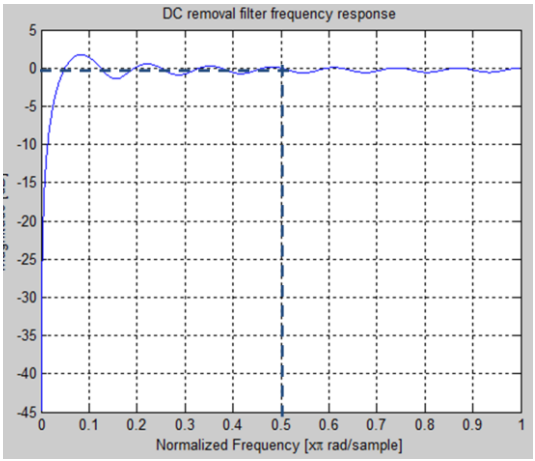

Abbildung 6: Frequenzgang des DC-Removal Filters.

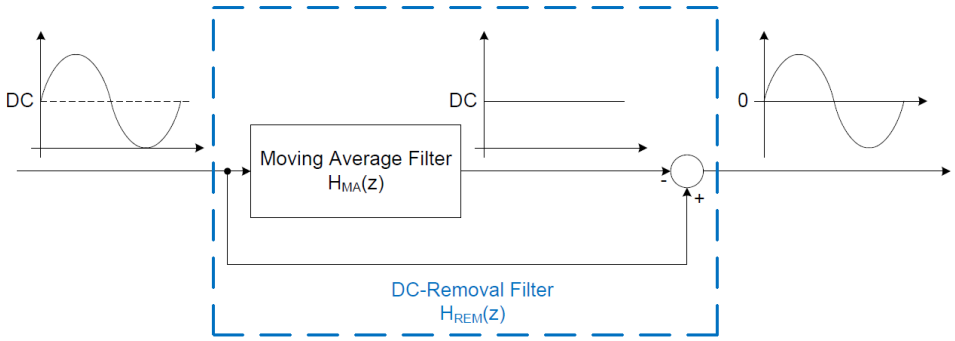

Abbildung 5: Prinzip des DC-Removal-Filters.

benötigt wird, muss das Filter eine Dezimation um den Faktor

$$k_{Dec} = \frac{f_{S,AD}}{f_{S,Dec}} = \frac{256 \cdot f_{SIN}}{4 \cdot f_{SIN}} = 64$$

vornehmen. Auch bei diesem Filter wurde die Eckfrequenz des Passbereichs so gewählt, dass das Ausgangssignal des Sensors das Filter ungehindert passiert:

$$f_{pass,Dec} = 2 \cdot f_{SIN} \mathrel{\hat{=}} \frac{\pi}{64}$$

Für die Dämpfung im Stopbereich wurden

$$A_{stop,Dec} = 50\ \text{dB}$$

festgelegt und durch Simulation[1] als ausreichend befunden. Durch Abtasten der LVDT-Ausgangssignale (der Sinusfrequenz f_{SIN}) mit der dezimierten Abtastrate $f_{S,Dec} = 4 \cdot f_{SIN}$ entsteht im Spektrum die niederfrequenteste Kopie des Trägersignals bei

$$f_{stör} = 4 \cdot f_{SIN} - f_{SIN} = 3 \cdot f_{SIN}$$

Als Eckfrequenz für den Stopbereich des Filters wurde somit diese Frequenz gewählt:

$$f_{stop,Dec} = 3 \cdot f_{SIN}$$

Um diese Vorgaben zu erreichen, wurde ein FIR-Filter vom Grad 511 (512 Filterkoeffizienten) gewählt und die Koeffizienten wurden basierend auf den Anforderungen mit Hilfe von MATLAB berechnet.

Abschließend ist noch die Frage zu klären, mit welcher Anzahl an Bits die Koeffizienten für die Realisierung gespeichert werden sollten. Dazu wurden Simulationen des Filterverhaltens für unterschiedliche Wortlängen durchgeführt. Die Skalierung der Koeffizienten erfolgte dabei entsprechend [7]. Ergebnis war, dass eine Darstellung der Koeffizienten mit 12 bit ausreicht, um die geforderte Funktionalität des Filters unter Minimierung der benötigten Ressourcen sicherzustellen. Der Frequenzgang des so entworfenen Filters ist in Abbildung 7 zu sehen. Man erkennt, dass das Sinussignal mit der Frequenz f_{SIN} das Filter ungehindert passiert, während Signale mit einer Frequenz oberhalb von $3f_{SIN}$ mit der gewünschten Dämpfung von 50 dB gedämpft werden.

C. DC-Removal-Filter

Bevor das A/D-gewandelte Signal in dem Algorithmus zur Positionsberechnung verwendet werden kann, muss noch der Gleichanteil im Signal entfernt werden (DC-Removal). Seitens des Sensorprinzips ist kein Gleichsignalanteil zu erwarten. Allerdings kann beispielsweise im Falle eines Leitungsbruchs in einer der Spulen ein DC-Anteil im Signal entstehen. Deshalb wird ein Filter verwendet, das den sogenannten gleitenden Mittelwert (Moving Average) berechnet und von dem Eingangssignal subtrahiert, um den Wechselsignalanteil zu erhalten. Dieses Prinzip ist in Abbildung 5 dargestellt. Das Filter bildet dazu den Mittelwert der letzten 16 Abtastwerte. Die Wahl zur Mittelung über genau 16 Abtastwerte hat zwei Gründe.

Zum einen soll der Mittelwert eines Sinussignals mit vier Abtastwerten pro Periode berechnet werden. Dazu sollte über ein ganzzahliges Vielfaches einer Periodendauer gemittelt werden, also ein Vielfaches von vier Abtastwerten. Eine weitere nützliche Eigenschaft des so entworfenen Filters wird sichtbar, wenn man den Frequenzgang in Abbildung 6 betrachtet. Der Gleichanteil wird, wie gewünscht, sehr stark gedämpft. Außerdem passiert der Wechselsignalanteil (Frequenz f_{SIN} ≙ normierte Frequenz $\pi/2$) das Filter ungedämpft (0 dB). Dadurch erhält man ein Sinussignal ohne Gleichanteil, dessen Amplitude nicht beeinträchtigt wird. Die Amplitude enthält die Positionsinformation und soll aus diesem Grund nicht beeinträchtigt werden.

D. Demodulation

Das Ziel der Demodulation ist die Bestimmung der Amplituden der Signale aus den beiden Sekundärspulen, da darin die Information über die Position steckt. In dem Algorithmus, der anschließend folgt, werden nur die Amplitudenverläufe benötigt. Darum muss im Zuge der Demodulation aus den Sinussignalen die Information über die Amplitude extrahiert werden.

Um dies zu erreichen, bedient man sich einer speziellen Methode: Zwei aufeinanderfolgende Abtastwerte werden als Real- und Imaginärteil einer komplexen Zahl c interpretiert:

$$c = \mathrm{Re}(c) + j \cdot \mathrm{Im}(c)$$

In trigonometrischer Form ausgedrückt:

$$c = r \cdot \cos(\varphi) + j \cdot r \cdot \sin(\varphi)$$

Dabei ist r die Amplitude und φ die Phase bei der komplexen Darstellung der Schwingung. Der Betrag der komplexen Zahl berechnet sich gemäß

$$|c| = r = \sqrt{\mathrm{Re}(c)^2 + \mathrm{Im}(c)^2}$$

Wendet man dies auf das Signal $x[n]$ an, das die Werte eines abgetasteten Sinussignals enthält, so lässt sich die Amplitude A_{SIN} des Sinus folgendermaßen berechnen:

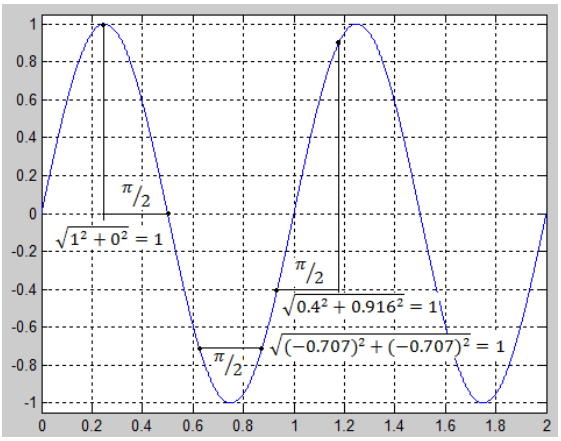

Abbildung 8: Prinzip der Amplitudenbestimmung.

$$A_{SIN} = \sqrt{x[n]^2 + x[n-1]^2}$$

Voraussetzung dafür ist allerdings, dass zwei aufeinanderfolgende Abtastwerte folgende Bedingungen erfüllen:

$$x[n-1] = A_{SIN} \cdot \sin(\varphi)$$
$$x[n] = A_{SIN} \cdot \cos(\varphi)$$

Zwischen Sinus und Cosinus besteht folgende Beziehung

$$\cos(\varphi) = \sin(\varphi + \pi/2)$$

Die Bedingungen sind dementsprechend:

$$x[n-1] = A_{SIN} \cdot \sin(\varphi)$$
$$x[n] = A_{SIN} \cdot \sin(\varphi + \pi/2)$$

Damit ist ersichtlich, dass zwei aufeinanderfolgende Abtastwerte einen Abstand von exakt $\frac{\pi}{2} = 90°$ haben müssen. Abbildung 8 zeigt das Prinzip der Amplitudenberechnung. Abgebildet sind drei Sample-Paare unterschiedlicher Position, welche alle dieselbe korrekte Amplitude liefern. Für die Samplingrate $f_{S,Dec}$ folgt deshalb:

$$f_{S,Dec} = \frac{360°}{90°} \cdot f_{SIN} = 4 \cdot f_{SIN}$$

Die Samplingrate zur Amplitudenbestimmung muss also das Vierfache der Sinusfrequenz betragen. Diese Forderung muss zu jeder Zeit erfüllt sein, um die Funktionalität der Amplitudenbestimmung zu gewährleisten. Diese Forderung hat dementsprechend rückwirkend Auswirkungen auf die A/D-Wandlung mit Überabtastung sowie den Filterentwurf, was in den Abschnitten A bis D bereits berücksichtigt wurde.

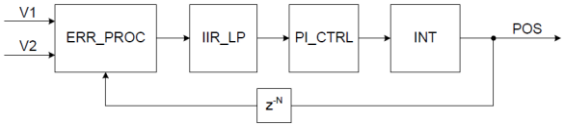

Abbildung 10: Blockschaltbild der Regelschleife.

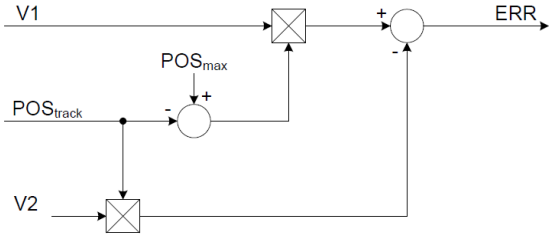

Abbildung 11: Blockschaltbild der Fehlerberechnung.

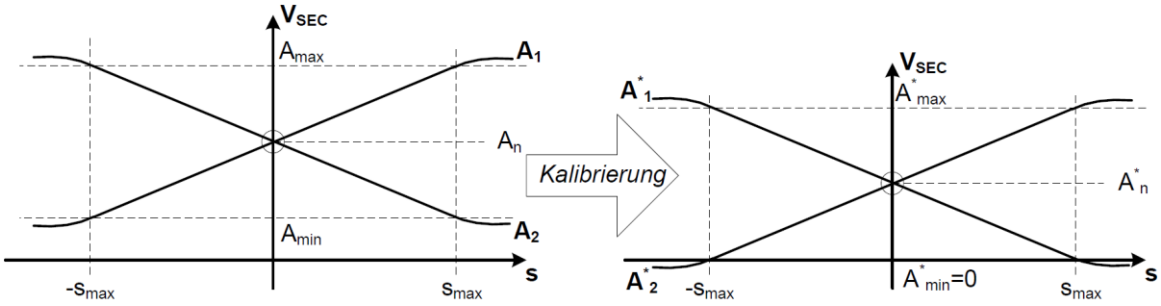

Abbildung 9: Prinzip der Kalibrierung.

E. Algorithmus zur Positionsbestimmung

Zur Extraktion der Positionsinformation aus den Spannungen an den Sekundärspulen des LVDT-Sensors wird eine geschlossene Regelschleife verwendet. Der Entwurf der Regelschleife ist an das in [8] dargestellte Vorgehen angelehnt. Die Regelschleife ist eine Regelschleife vom Typ 2 (zwei Integratoren), welche aus einem PI-Regler (Block PI_CTRL), einem IIR-Tiefpassfilter zur Unterdrückung von hochfrequenten Störungen außerhalb des relevanten Frequenzbands (Block IIR_LP) sowie einem Integrator (Block INT) besteht. Abbildung 10 zeigt das entsprechende Blockschaltbild.

Die Regelschleife funktioniert folgendermaßen: Aus der interpolierten Position und den demodulierten Sekundärspannungen wird ein Fehler berechnet (Block ERR_PROC). Die interpolierte Position kann dabei um die Gruppenlaufzeit des FIR-Dezimationsfilters verzögert werden, um die Laufzeit durch das Filter zu kompensieren. Dies kann in Anwendungen nötig sein, bei denen sich der Sensor sehr schnell bewegt und eine hohe Gruppenlaufzeit existiert. Die Regelschleife sorgt dafür, dass der berechnete Fehler zu Null wird und somit die interpolierte Position identisch mit der realen Position ist.

Voraussetzung dabei ist, dass bei maximaler Auslenkung ($s = s_{max}$) des Sensors immer jeweils eine der beiden Sekundärspannungen eine Amplitude von Null ($A_{min} = 0$) hat. Dies ist bei den meisten Sensoren nicht der Fall, weshalb eine Kalibrierung (siehe Abbildung 9) nötig ist. Dazu wird die Amplitude $A_{\min}$ der Spannungen an den Sekundärspulen bei maximaler Auslenkung gemessen und bei den weiteren Berechnungen von den aktuellen Amplitudenwerten subtrahiert, um einen kalibrierten Verlauf der Sekundärspannungen zu erhalten.

1) Fehlerberechnung

Bei der Fehlerberechnung soll aus den Spannungen an den Sekundärspulen sowie der interpolierten Position ein Fehler berechnet werden, über den die interpolierte Position nachgeregelt werden soll. Abbildung 11 zeigt das Blockdiagramm der verwendeten Methodik.

Zur Berechnung des Fehlers wird die Spannung an der ersten Sekundärspule V_1 mit der inversen Trackingposition ($POS_{max} - POS_{track}$) und die Spannung an der zweiten Sekundärspule V_2 mit der Trackingposition POS_{track} multipliziert. Die beiden Ergebnisse werden voneinander subtrahiert um den Fehler zu berechnen. Die inverse Trackingposition wird dabei berechnet, indem man den Wert der Trackingposition von dem maximal möglichen Wert subtrahiert. Dazu muss man wissen, dass die Position durch eine festgelegte Anzahl an Bit dargestellt wird. Die Position bewegt sich im Bereich [-1;+1]. Bei der gewählten Darstellung mit $n = 16$ bit bewegt sich die Trackingposition im Bereich 0 … 65535, wobei der Wert 0 der Position -1 und der Wert 65535 der Position +1 zugeordnet ist.

2) IIR-Tiefpassfilter

Um eventuelle Störungen oberhalb des relevanten Frequenzbereichs zu eliminieren, wird ein IIR-Filter verwendet. Das Filter ist ein Tschebyscheff Typ II Tiefpassfilter zweiter Ordnung mit der Übertragungsfunktion

$$H_{IIR}(z) = V \cdot \frac{(z - z_{NS1}) \cdot (z - z_{NS2})}{(z - z_{PS1}) \cdot (z - z_{PS2})}$$

Das komplexe Nullstellenpaar

$$z_{NS1,2} = a_{NS} \pm j \cdot b_{NS}$$

wurde mittig in dem aufgrund der Samplingfrequenz $f_{S,AD}$ festgelegten Frequenzbereich $f = 0 \ldots \frac{f_{S,AD}}{2}$ positioniert. Es gilt deshalb für die Frequenz f_{NS}, auf der die Nullstellen des Filters platziert sind:

$$f_{NS} = \frac{f_{S,AD}/2}{2} = \frac{f_{S,AD}}{4} \mathrel{\hat{=}} \frac{\pi}{2}$$

Damit wird das Nullstellenpaar entsprechend bei $\pm\frac{\pi}{2} \mathrel{\hat{=}} \pm j$ platziert, was zu einer starken Dämpfung des Filters im entsprechenden Frequenzbereich führt, wodurch die hochfrequenten Signalanteile unterdrückt werden sollen. Es gilt also:

$$z_{NS1,2} = a_{NS} \pm j \cdot b_{NS} = \pm j$$

und damit:

$$a_{NS} = 0$$
$$b_{NS} = 1$$

Die Position des komplexen Polstellenpaars

$$z_{PS1,2} = a_{PS} \pm j \cdot b_{PS}$$

bestimmt das Dynamikverhalten des Filters und kann je nach Bedarf der Anwendung platziert werden. Setzt man die allgemeinen Werte für die Pol- und Nullstellen ein, so ergibt sich nach Ausmultiplizieren und Zusammenfassen:

$$H_{IIR}(z) = V \cdot \frac{z^2 - 2a_{NS} \cdot z + (a^2{}_{NS} + b^2{}_{NS})}{z^2 - 2a_{PS} \cdot z + (a^2{}_{PS} + b^2{}_{PS})}$$

V ist dabei ein Faktor, den man üblicherweise so wählt, dass im Nennerpolynom eine 1 als Faktor vor dem z^2 steht. Das von den Polstellen bestimmte Dynamikverhalten entspricht dem eines PT2-Glieds, welches ebenfalls zwei Polstellen besitzt. Legt man das Dynamikverhalten eines PT2-Glieds zugrunde, lässt sich dies auf die Polstellen des Filters übertragen und man erhält dasselbe Dynamikverhalten.

Die Übertragungsfunktion $H_{PT2}(s)$ eines kontinuierlichen, schwingungsfähigen, PT2-Glieds lautet nach [9]:

$$H_{PT2}(s) = \frac{1}{\frac{1}{\omega_0{}^2} \cdot s^2 + \frac{2D}{\omega_0} \cdot s + 1}$$

Dabei steht D für die Dämpfung und ω_0 für die Eigenfrequenz des schwingungsfähigen PT2-Glieds.

Nach [10] kann die Konversion von s-Transformation auf z-Transformation erfolgen, indem man s mit $\frac{z-1}{T_S}$ (T_S: Sampledauer) ersetzt (Zero-order-hold-Ansatz bzw. Rechteckintegration):

$$H_{PT2}(z) = \frac{1}{\frac{1}{\omega_0{}^2} \cdot \left(\frac{z-1}{T_S}\right)^2 + \frac{2D}{\omega_0} \cdot \left(\frac{z-1}{T_S}\right) + 1}$$

Durch erneutes Ausmultiplizieren und Zusammenfassen ergibt sich

$$H_{PT2}(z) = \frac{\omega_0{}^2 T_S{}^2}{z^2 + (2D\omega_0 T_S - 2) \cdot z + (\omega_0{}^2 T_S{}^2 - 2D\omega_0 T_S + 1)}$$

Ein Koeffizientenvergleich der beiden Übertragungsfunktionen und Lösen des sich ergebenden Gleichungssystems für a_{PS} und b_{PS} ergibt die Zusammenhänge

$$V = \omega_0{}^2 T_S{}^2$$
$$a_{PS} = 1 - D\omega_0 T_S = 1 - D \cdot \sqrt{V}$$
$$b_{PS} = \omega_0 T_S \cdot \sqrt{1 - D^2} = \sqrt{V} \cdot \sqrt{1 - D^2}$$

Durch diese Vorschrift kann die Platzierung der Polstellen des IIR-Filters auf Basis der Dynamikeigenschaften eines PT2-Glieds vorgenommen werden. Dabei sind die Werte für die Sampledauer T_S, die Eigenfrequenz ω_0 sowie die Dämpfung D des Filters festzulegen. Die Sampledauer ist dabei bereits durch die Samplingrate $f_{S,AD}$ festgelegt:

$$T_S = \frac{1}{f_{S,AD}} = \frac{1}{256 \cdot f_{SIN}}$$

Die Eigenfrequenz der Schwingung wurde doppelt so groß gewählt wie die Frequenz f_{SIN} des Erregersignals bzw. der Sekundärsignale:

$$\omega_0 = 2 \cdot \omega_{SIN} = 4 \cdot \pi \cdot f_{SIN}$$

Für die Dämpfung wurde der in der Regelungstechnik übliche Wert von $\frac{1}{\sqrt{2}}$ gewählt. Dieser bietet einen guten Kompromiss zwischen Einregelzeit und Höhe des Überschwingers.

$$D = \frac{1}{\sqrt{2}}$$

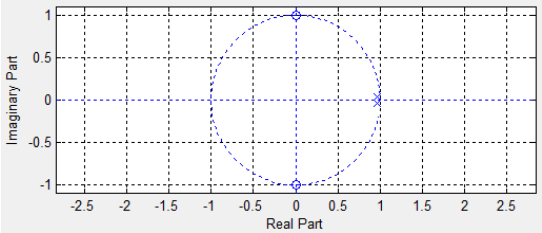

Abbildung 13: Pol-Nullstellendiagramm des IIR-Filters.

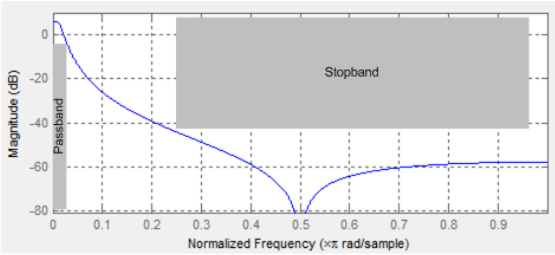

Abbildung 14: Frequenzgang des IIR-Filters.

Setzt man diese Werte ein, so ergibt sich:

$$V = \left(\frac{4\pi f_{SIN}}{256\, f_{SIN}}\right)^2 \approx \left(\frac{1}{20}\right)^2 = \frac{1}{400}$$

$$a_{PS} = 1 - \frac{1}{\sqrt{2}} \cdot \frac{1}{20} \approx 0{,}964645$$

$$b_{PS} = \frac{1}{20} \cdot \sqrt{1 - \left(\frac{1}{\sqrt{2}}\right)^2} \approx 0{,}035355$$

Damit ist die Position des Polstellenpaars festgelegt:

$$z_{PS1,2} = a_{PS} \pm j \cdot b_{PS} = 0{,}964645 \pm j \cdot 0{,}35355$$

Abbildung 13 zeigt das Pol-Nullstellen-Diagramm des so entworfenen Filters. Die markante Lage der Nullstellen bei $\pm j$ ist gut erkennbar. Außerdem liegen die Polstellen innerhalb des Einheitskreises, was die Stabilität des Filters bestätigt. Die Auswirkung der Nullstellen ist auch gut im Frequenzgang des Filters in Abbildung 14 ersichtlich, der Einbruch bei $\frac{\pi}{2}$ ist gut zu erkennen, durch den eine hohe Dämpfung im Stopband erreicht wird. Um das Dynamikverhalten des Filters zu betrachten, kann man die Sprungantwort verwenden. Diese ist in Abbildung 12 dargestellt. Man erkennt den charakteristischen Verlauf eines gut gedämpften PT2-Glieds mit einem leichten Überschwinger. Man erkennt auch, dass das Filter stabil ist, da ein stationärer Zustand erreicht wird.

Auch bei diesem Filter ist wieder die Frage zu beantworten, mit wie vielen Bits die Koeffizienten dargestellt werden und wie die Skalierung erfolgt. Wie bereits beim Dezimationsfilter durchgeführt, wurde auch

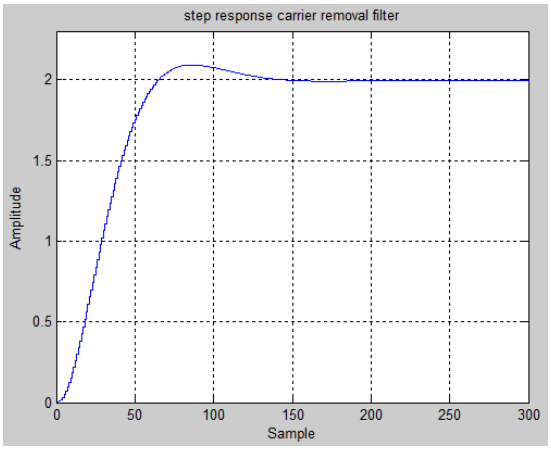

Abbildung 12: Sprungantwort des IIR-Filters.

hierfür das Filterverhalten für verschiedene Wortlängen simuliert. Im Gegensatz zu einem FIR-Filter mit Polstellen im Ursprung, ist die Lage der Polstellen beim IIR-Filter ein kritischer Faktor, da das Dynamikverhalten stark davon beeinflusst wird. Vor allem wenn die Polstellen, wie in diesem Fall, an der Grenze des Stabilitätsbereichs positioniert sind, kann eine Verringerung der Auflösung dazu führen, dass die Polstellen in den instabilen Bereich wandern. Ergebnis war, dass für die Koeffizienten eine Auflösung von 16 bit gewählt werden muss, um eine korrekte Funktion des Filters sicher zu stellen.

3) PI-Regler

Das Signal aus dem IIR-Filter wird an einen PI-Regler übergeben. Das Signal am Ausgang des Reglers entspricht der Bewegungsgeschwindigkeit des Sensors. Der PI-Regler besteht aus einem Proportional- und einem Integralanteil. Der Proportionalanteil entspricht einer Verstärkung des Eingangssignals mit einem Faktor K_P. Der Integralanteil besteht aus einem diskreten Integrator mit Verstärkung K_I.

Nach [8] hat der PI-Regler Einfluss auf die Bandbreite der Regelschleife. Die beiden Parameter, die variabel sind, sind die Verstärkungsfaktoren K_P und K_I des Proportional- bzw. Integralanteils. Aus den in der Quelle angeführten Beispielwerten ergibt sich für eine hochauflösende Anwendung ein Verhältnis

$$\frac{K_I}{K_P} = \frac{0{,}001}{0{,}05} = 0{,}02.$$

Im Hinblick auf die Realisierung in einem FPGA wurde darauf geachtet, dass die Werte für K_I und K_P Zweierpotenzen sind, da solch eine Division durch eine einfache Schiebeoperation realisiert werden kann, ohne Hardware-Ressourcen zu verwenden. Aus diesem Grund wurden folgende Werte gewählt:

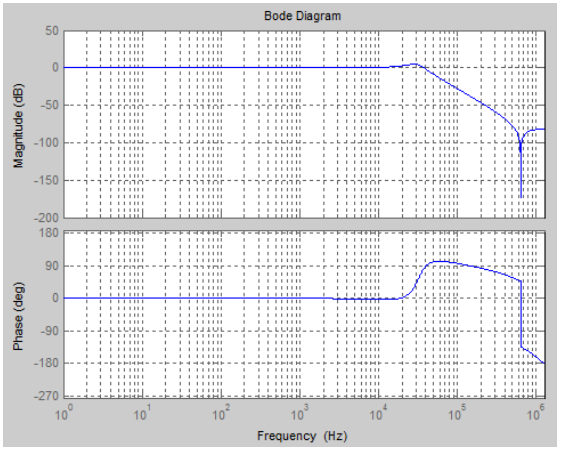

Abbildung 15: Bodediagramm der geschlossenen Regelschleife.

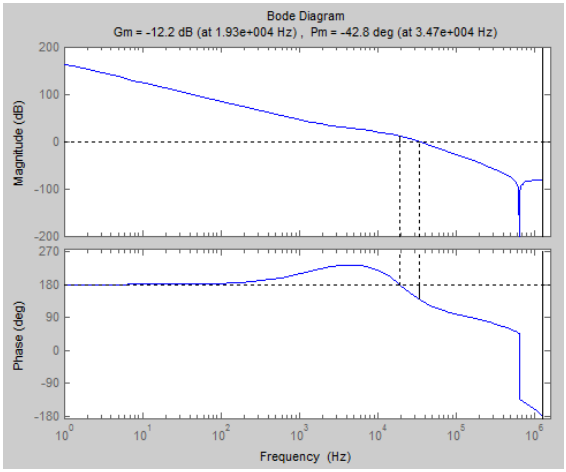

Abbildung 16: Amplituden- und Phasenreserve der offenen Regelschleife im Bodediagramm.

$$K_P = 1$$

$$K_I = \frac{1}{32}$$

Damit ergibt sich ein Verhältnis $\frac{K_I}{K_P} = \frac{1}{32} \approx 0{,}03$.

4) Integrator

Der Integrator dient dazu, das Geschwindigkeitssignal am Ausgang des PI-Reglers in ein Positionssignal umzuwandeln. Dies erfolgt durch eine Integration ohne jegliche Verstärkung.

5) Stabilität der Regelschleife

Um eine Aussage über das Verhalten der geschlossenen Regelschleife machen zu können, wurde deren Verhalten mit MATLAB simuliert. Dazu wird die Übertragungsfunktion jeder beteiligten System-komponente der Regelschleife durch die entsprechende diskrete Übertragungsfunktion definiert. Anschließend wird daraus die Übertragungsfunktion des offenen Regelkreises und anschließend die des geschlossenen Regelkreises mit der Rückkopplung der berechneten Position ermittelt. Für all diese Operationen bietet MATLAB bereits die entsprechenden Funktionen. Abschließend kann man das Bode-Diagramm der geschlossenen Regelschleife berechnen und anzeigen lassen, wie in Abbildung 15 zu sehen.

Aussagekräftiger für die Stabilität sind die Werte für Amplituden- und Phasenreserve des Regelkreises wie in Abbildung 16 dargestellt. Wie sich der Abbildung entnehmen lässt, ergibt sich eine Amplitudenreserve von 12,2 dB und eine Phasenreserve von 42,8°. Ein stabiles System sollte mindestens eine Phasenreserve von 30° aufweisen, normalerweise wählt man 45° um auf der sicheren Seite zu sein. Die ermittelten 42,8° sprechen also für ein stabiles System.

V. Simulationsergebnisse

Um einen ersten Einblick in das entworfene System zu erhalten, wurde ein Modell des Systems in MATLAB/Simulink implementiert. Simulink stellt eine einfache und effiziente Möglichkeit dar, die Funktionalität des Systems abschätzen zu können. Es wurden zwei unabhängige Modelle erstellt, eines für die Erzeugung des Erregersignals und eines für die Signalverarbeitung (Positionsbestimmung). Dabei wurde folgendes beachtet:

- Die Datentypen der Eingangs- und Ausgangssignale eines Blocks müssen entsprechend der eingesetzten Wortlängen festgelegt werden, d.h. in Fixed-Point-Notation, da ansonsten immer mit double-Werten gerechnet wird.
- Bei Verwendung von Fixed-Point-Signalen muss die Art des Rundens vorgegeben werden (floor).
- Bei der Wahl der Wortlänge innerhalb der Simulink-Blöcke (z.B. für die Akkumulatoren in einem Filter) ist darauf zu achten, dass kein Überlauf eintritt. Diese Wahl kann man auch Simulink überlassen, indem man keine Werte vorgibt, was zu empfehlen ist.
- Schaltet man zwei Blöcke mit unterschiedlicher Samplingrate zusammen, so muss dazwischen ein Block „Rate Transition“ eingefügt werden, der für eine korrekte Konversion der Samplingrate sorgt.
- Der Simulator sowie die verwendeten Blöcke sollten auf Sample-basierte Berechnung eingestellt werden, um Zeit-unabhängig simulieren zu können. Dabei ist außerdem zu beachten, dass eine festgelegte Abtastrate im System eingestellt ist, sodass alle Samples im System einen festen Bezug untereinander haben.

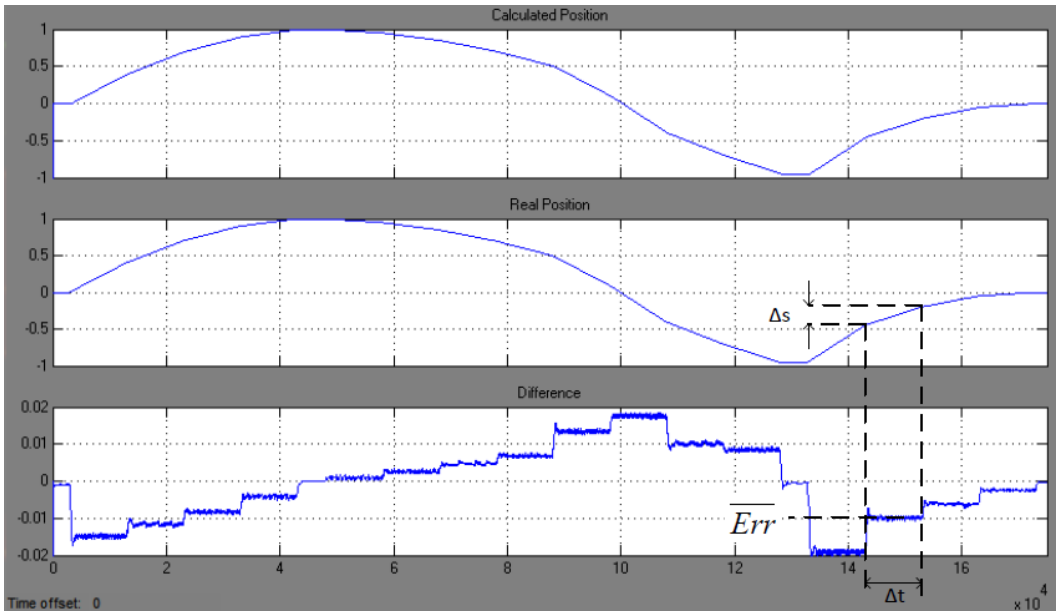

Abbildung 17: Definition der Messgrößen.

Da zur Berechnung der Position eine Regelschleife eingesetzt wird, ist natürlich von Interesse, wie gut die Position bei verschiedenen Auslenkungsgeschwindigkeiten getrackt wird. Dazu wurden verschiedene Geschwindigkeiten simuliert und der entstehende Positionsfehler (Unterschied zwischen realer und berechneter Position) ermittelt. Bezeichnet man das Positionssignal mit $s(t)$, so entspricht die Geschwindigkeit

$$v(t) = \frac{ds(t)}{dt}$$

Da die Steigung abschnittsweise konstant definiert wurde, gilt für $v(t)$:

$$v(t) = \frac{\Delta s}{\Delta t}$$

Eingestellt wurden unterschiedliche Werte von Δs, wobei für Δt der konstante Wert von 10000 Samples gewählt wurde:

$$\Delta t = 10000 \cdot T_S$$

Für jedes $\frac{\Delta s}{\Delta t}$ wurde der mittlere Positionsfehler $\overline{Err}$ abgelesen, wie in Abbildung 17 dargestellt. Trägt man den Fehler über der Geschwindigkeit auf (siehe Abbildung 18), so kann man nähere Aussagen zur Abhängigkeit des Positionsfehlers von der Auslenkungsgeschwindigkeit machen.

In der Abbildung lässt sich ein etwa geradliniger Verlauf erkennen. Legt man den Wert 0.01 als Grenzwert für den Fehler fest (wie vom Kunden gefordert wurde), so lässt sich ablesen, dass eine Auslenkung des Sensors mit einer Geschwindigkeit von bis zu

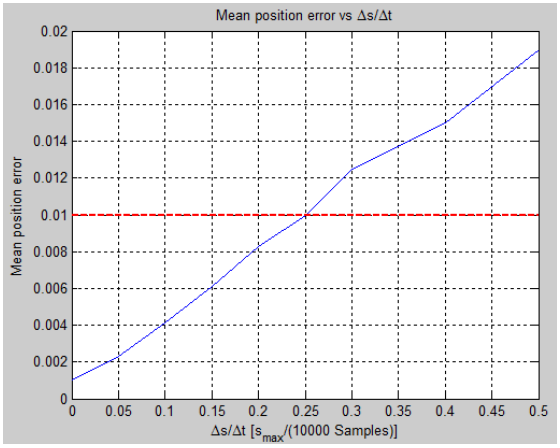

Abbildung 18: Abhängigkeit des Positionsfehlers von der Auslenkungsgeschwindigkeit des Sensors.

$$v_{\max} = \frac{0{,}25 \cdot s_{\max}}{10000 \cdot T_S}$$

mit der geforderten Genauigkeit erfassen lässt. Diese hängt von der maximalen Auslenkung s_{max} des Sensors sowie der Samplingdauer T_S ab. Für die Samplingdauer gilt wiederum:

$$T_S = \frac{1}{4 \cdot k_{Dec} \cdot f_{SIN}}$$

Damit ergibt sich:

$$v_{\max} = \frac{0{,}25 \cdot s_{\max} \cdot 4 \cdot k_{Dec} \cdot f_{SIN}}{10000} = 10^{-4} \cdot s_{\max} \cdot k_{Dec} \cdot f_{SIN}$$

Tabelle 1: Genutzte Ressourcen im Spartan-6 FPGA.

Ressource	Verfügbar	Genutzt	Nutzung
Slice Register	18224	2786	15%
Slice LUT	9112	2363	25%
Occupied Slice	2278	817	35%
MUXCY	4556	1068	23%
Bonded IOB	232	54	23%
RAMB8BWER	64	4	6%
BUFG/ BUFGMUX	16	1	6%
ILOGIC2/ ISERDES2	248	26	10%
OLOGIC2/ OSERDES2	248	2	1%
DSP48A1	32	9	28%

Mit Kenntnis der maximalen Auslenkung des Sensors, des Dezimationsfaktors k_{Dec} und der Frequenz des Erregersignals f_{SIN} kann man die maximal erfassbare Geschwindigkeit näherungsweise abschätzen. Nimmt man beispielsweise einen Sensor mit einer maximalen Auslenkung von 10 cm, einer Erregersignalfrequenz von 5 kHz und, wie in diesem Fall, einer Dezimation mit Faktor 64 an, so berechnet sich v_{max} zu

$$v_{max} = 10^{-4} \cdot 0.1\,\mathrm{m} \cdot 64 \cdot 5\,\mathrm{kHz} = 3.2\,\mathrm{m/s}$$

Dieser Wert hängt von der Dimensionierung der Regelschleife ab und kann bei Bedarf durch entsprechende Änderungen an der Regelschleife auf die Bedürfnisse angepasst werden.

VI. Implementierung

Die Simulationsmodelle boten eine Basis für die anschließende Implementierung des Systems in VHDL. Achtet man dabei explizit auf eine identische Implementierung des VHDL-Codes wie in Simulink vorgenommen, so ergeben sich identische Signalverläufe. Damit wurde sichergestellt, dass sich das System so verhält, wie es in Simulink modelliert und simuliert wurde. Die Implementierung der Systemkomponenten erfolgte dabei mittels Prozessen, in deren Sensitivitätsliste lediglich das Reset-Signal und der Systemtakt enthalten sind. Das stellt sicher, dass ein taktsynchrones System implementiert wird, bei dem jede Aktion lediglich von dem Systemtakt bzw. dem Reset-Signal ausgeht.

Da zum Zeitpunkt der Implementierung die Zielhardware (Xilinx Spartan 6) bereits definiert war, konnten

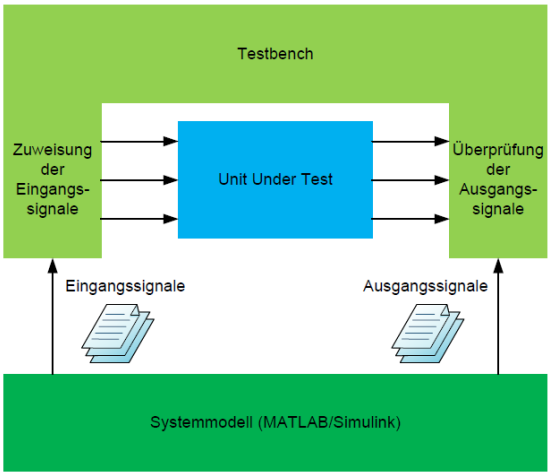

Abbildung 19: Testumgebung.

bestimmte Funktionen wie ROM-Strukturen, Multiplizierer für das FIR-Dezimationsfilter, die Berechnung der Wurzelfunktion sowie konstante Verstärkungsfaktoren mit Hilfe des IP-Core-Generators LogiCore von Xilinx realisiert werden. Dies hat den großen Vorteil, dass die damit erstellten Funktionen bereits während der Generierung für die Zielhardware optimiert werden. Damit kann eine wesentlich bessere Performance erzielt werden, als es bei einer Implementierung „von Hand“ der Fall wäre. Außerdem verkürzt dieses Vorgehen die Zeit, die zur Implementierung des Systems nötig ist.

Im Sinne der Wartbarkeit des Codes wurde für die so erzeugten IP-Blöcke jeweils ein Wrapper-Modul geschrieben, das die Umsetzung der Schnittstellen der generierten Blöcke auf die systeminternen Schnittstellen vornimmt. Somit ist es bei der Änderung der Zielhardware relativ einfach, die neuen IP-Blöcke in das System einzubinden, da lediglich Änderungen an den entsprechenden Wrapper-Modulen nötig sind.

Nach der Implementierung des Systems in VHDL wurde dieses für die Zielhardware, ein Spartan 6 – FPGA der Firma Xilinx, synthetisiert. Eine Übersicht, welche Ressourcen das FPGA zur Verfügung stellt und wie viel davon für das System benötigt wird, ist in Tabelle 1 zu sehen. Von den herkömmlichen Ressourcen wie LUTs, Multiplexer oder DSP-Slices wurden in etwa 25% für das System benötigt. Der Rest der nicht verwendeten Ressourcen kann für weitere beliebige Funktionalitäten verwendet werden.

VII. Test des Systems

Den Abschluss des Projekts bildete die Testphase, in der die Funktionalität des Systems in Bezug auf die Systemanforderungen getestet wurde. Dazu wurden verschiedene Testfälle definiert, mit denen sich die einzelnen Anforderungen abprüfen lassen. Zur Verifikation des Systems mussten alle Testfälle erfolgreich abgeschlossen werden.

Diese Tests wurden zweimal durchgeführt: Einerseits als reine Logiksimulation des implementierten VHDL-Codes und andererseits als Timing-Simulation nach der Synthese des Codes für die Zielhardware. Die Logiksimulation alleine reicht nicht zur Verifikation aus, da dabei nicht das Zeitverhalten auf der realen Hardware simuliert wird. Aus diesem Grund wurde ebenfalls eine Timing-Simulation unter Verwendung der Daten über Verzögerungs- und Laufzeiten im FPGA durchgeführt. Damit ist sichergestellt, dass das System auch in Verbindung mit der Zielhardware die Anforderungen erfüllt.

Bei allen Tests wurde der Umstand ausgenutzt, dass Systemmodelle in MATLAB/Simulink existieren, die sich identisch zu dem in VHDL implementierten System verhalten. Die Testumgebung ist in Abbildung 19 dargestellt.

Der Prüfling (Unit Under Test) wird in eine Testbench eingebunden. Die Testbench erzeugt die Eingangssignale für den Prüfling und nimmt eine Überprüfung der Ausgangssignale vor. Erweitert wird diese Umgebung durch das Systemmodell, das in MATLAB/Simulink entworfen wurde. Mit dem Modell können die Ein- und Ausgangssignale auf einfache Art und Weise erzeugt und in Textdateien abgespeichert werden. Diese Dateien werden anschließend von der Testbench verwendet, um die Eingangssignale zu erzeugen und die Ausgangssignale des Prüflings gegen die Soll-Werte aus dem Modell zu überprüfen.

Damit ist eine effiziente Möglichkeit zum Test des Systems gegeben, da sich die Signale dank des Systemmodells sehr einfach erzeugen lassen und leicht Änderungen (falls sich Prüfvorschriften ändern) vornehmen lassen. Für den Test des Systems wurde eine dafür vorgesehene, von NewTec selbst entwickelte, skriptbasierte Testumgebung verwendet, die automatisiert die Eingangssignale erzeugt und die entsprechende Überprüfung der Ausgangssignale vornimmt.

VIII. Fazit

Es wurde ein digitales System zur Ansteuerung eines LVDT-Sensors und zur Auswertung der Sensorsignale zur Ermittlung der Position umgesetzt. Das System wurde in MATLAB/Simulink modelliert und simuliert. Die Implementierung erfolgte in VHDL. Außerdem erfolgte eine Synthese auf den Xilinx Spartan 6 FPGA als Zielhardware. Das System wurde mittels Logik- und Timingsimulation getestet und die Systemfunktionalität verifiziert. Das System lässt sich in Form eines IP-Cores in Folgeprojekten verwenden, in denen eine Anbindung von LVDT-Sensoren notwendig ist.

Es ist hervorzuheben, dass dieses System aufgrund desselben Wirkungsprinzips ohne weitere Anpassungen auch für RVDT-Sensoren verwendet werden kann. Die berechnete Position im Bereich s = -1 … +1 entspricht dabei nicht einer (Strecken-) Position im Bereich $-s_{max} \ldots +s_{max}$ sondern einer (Winkel-) Position im Bereich $-\varphi_{max} \ldots +\varphi_{max}$.

Außerdem existiert ein Simulationsmodell in MATLAB/Simulink für das Gesamtsystem, dessen Verhalten identisch mit dem realen System ist. Dieses kann zusätzlich zur Simulation des Systemverhaltens verwendet werden.

IX. Ausblick

Basierend auf den Ergebnissen dieser Arbeit könnte in einem Folgeprojekt das vorhandene System um Monitoring-Funktionalitäten erweitert werden, die sich mehrheitlich auf die Kontrolle von Grenzwerten oder die Reaktion auf einzelne Statussignale beschränken und auf die aus Zeitgründen verzichtet wurde. Der Aufwand dafür ist relativ gering einzuschätzen.

Außerdem könnte das System so erweitert werden, dass es nicht nur in Kombination mit LVDT- und RVDT-Sensoren, sondern auch mit den artverwandten Synchro- oder Resolver-Systemen verwendet werden kann. Diese beiden Systeme basieren ebenfalls auf dem Transformatorprinzip, unterscheiden sich aber in der Art und Weise, wie die Positionsinformation in den Sensorsignalen enthalten ist. Aus diesem Grund müssten kleinere Änderungen (z.B. an der Fehlerberechnung) oder Ergänzungen (z.B. falls ein fester Phasenbezug zum Erregersignal nötig ist) an der Signalverarbeitung vorgenommen werden. Durch diese Erweiterung könnte das System in Kombination mit einer Vielzahl an Sensoren betrieben werden.

Danksagung

Die Autoren bedanken sich bei der NewTec GmbH, Pfaffenhofen, für die Ermöglichung der Masterarbeit, auf der diese Veröffentlichung beruht.

Literaturverzeichnis

[1] Macro Sensors, „LVDT Basics“, *Macro Sensors*, Pennsauken, 2003.

[2] J. Szczyrbak, E. D. D. Schmidt, “LVDT Signal Conditioning Techniques”, *LucasVarity*, 1997.

[3] A. Chrysafis, “Motorola Digital Signal Processors – Digital Sine-Wave Synthesis Using the DSP56001/2”, *Motorola*; 1988

[4] M. Mitchell, “Using PWM Timer_B as a DAC”, *Texas Instruments Incorporated*, 2000.

[5] Texas Instruments, “Oversampling Techniques using the TMS320C24x Family”, *Texas Instruments*, 1998.

[6] P. Boaz, “A Course In Digital Signal Processing; 1st Edition”, *John Wiley & Sons*, Hoboken, 1997.

[7] R. Yates, “Practical Considerations in Fixed-Point FIR Filter Implementations, Revision PA5”, *Digital Signal Labs*, 2010.

[8] M. Staebler, “TMS320F240 DSP Solution for Obtaining Resolver Angular Position and Speed”, *Texas Instruments Incorporated*, 2000.

[9] W. Schroer,“ Systemdynamik, Manuskript zur Vorlesung, 3. Auflage“, *Hochschule Ulm*; Ulm, 2012.

[10] Analog Devices, “ADSP-21990: Implementation of PI Controllers”, *Analog Devices Incorporated*, 2001.

Matthias Schnell erhielt den akademischen Grad des B.Eng. in Nachrichtentechnik sowie den Grad des M.Eng. in Systems Engineering and Management (Electrical Engineering) im Jahr 2012 bzw. 2013 von der Hochschule Ulm. Seit 2014 arbeitet er als Systemingenieur im Bereich Hardware-Entwicklung bei der NewTec GmbH System-Entwicklung und Beratung in Pfaffenhofen.

Peter Jonski ist seit 1994 bei der NewTec GmbH System-Entwicklung und Beratung in der Entwicklung elektronischer Schaltungen und programmierbarer Logik beschäftigt. Er erhielt den Grad des Dipl.Ing (FH) im Fachbereich Nachrichtentechnik/Industrieelek-tronik im Jahr 1993 an der Fachhochschule Ulm. Seit dem Jahr 2000 ist er Gruppenleiter eines Entwicklungsteams mit dem Schwerpunkt Systementwicklung.

Gerhard Forster studierte Physik mit dem Schwerpunkt Quantenelektronik an der Universität Heidelberg. Nach seinem Diplom-Abschluss 1977 befasste er sich als Wissenschaftlicher Mitarbeiter am damaligen Forschungsinstitut von AEG-Telefunken (später Daimler Forschungszentrum) mit der Entwicklung und Anwendung neuer Halbleiterprozesse. Zuletzt war er als Teamleiter zuständig für die Entwicklung und den Test anwendungsspezifischer Integrierter Schaltungen aus den Gebieten der Nachrichtentechnik und der Automobilelektronik. Seit 1992 ist er Professor für Elektronik und Mikroelektronische Schaltungen an der Hochschule Ulm. Seine Schwerpunkte liegen auf dem Gebiet des Entwurfs von Mixed-Signal-ASICs. Zwischen 2001 und 2010 hatte er die Leitung der Fakultät Elektrotechnik und Informationstechnik inne. Prof. Forster ist Herausgeber des vorliegenden Tagungsbandes.

Evaluation eines Zynq-7000 SoCs mittels Bildkompression und High-Level Synthese

Francisco de Asís Molina Martel, Thomas Perschke, Frank Kesel, Manuel Gaiser

***Zusammenfassung*—Die Zynq-7000 SoC Plattform der Firma Xilinx verfügt neben einem ARM Dual-Core Cortex-A9 Prozessor über rekonfigurierbare Logik. Damit ist dies eine ideale Plattform für die Nutzung von rekonfigurierbaren Hardware-Beschleunigern. Mit Hilfe eines Demonstrationssystems und einer JPEG-Encoder Implementierung wird untersucht, ob die Zynq-7000 Plattform effizient für die Bildverarbeitung genutzt werden kann. Die Prozessorkerne werden als Asymmetrisches Multiprozessorsystem (AMP) betrieben. Für das HW/SW Co-Designs wird das High-Level Synthesewerkzeug Xilinx Vivado HLS eingesetzt. Anhand des JPEG-Encoders wird untersucht, wie die spezifische Implementierung des Algorithmus Ressourcenverbrauch und Durchsatz beeinflusst.**

***Schlüsselwörter*—Zynq-7000 SoC, Asymmetric Multiprocessing (AMP), High-Level Synthese (HLS), JPEG Encoder.**

I. EINLEITUNG

Field Programmable Gate Arrays (FPGAs) sind durch ihre Flexibilität bezüglich der einsetzbaren Ressourcen eine sehr gut geeignete Plattform für die Echtzeitbildverarbeitung auf Pixelebene [1]. Je höher die Abstraktionsebene eines Verarbeitungsalgorithmus ist, wie sie beispielsweise die Klasse der merkmalsbasierten Algorithmen repräsentiert, desto schwieriger und zeitaufwändiger sind FPGA-Implementierungen zu realisieren. Diese Domäne wird von prozessorbasierten Plattformen in der Regel besser abgedeckt.

Die von den FPGA-Herstellern bereitgestellten Soft-IP Prozessoren wie der Altera Nios-Prozessor oder der Xilinx Microblaze-Prozessor sind ein erster Schritt für den Einsatz von Prozessoren auf einem FPGA. Durch die Implementierung mit den vorhandenen FPGA-Ressourcen können allerdings typischerweise nur Taktraten zwischen 100 MHz und 200 MHz erreicht werden.

Durch die Einführung von neuen ARM-basierten System-on-Chip (SoC) Plattformen mit rekonfigurierbarer Logik, wie die Xilinx Zynq-7000 SoC [2] Familie, bieten sich neue Möglichkeiten für die effiziente Implementierung von Bildverarbeitungsalgorithmen. Durch das gleichzeitige Aufkommen kommerziell erhältlicher High-Level Synthesewerkzeuge wird auch das notwendige HW-SW Co-Design erleichtert.

Im Folgenden wird ein Demonstrationssystem für den Einsatz der Zynq-Plattform für die Bildverarbeitung vorgestellt. Für das System wird ein ZedBoard eingesetzt [3]. Als Bildquelle dient eine angeschlossene USB-Webcam. Für die Untersuchungen wird das System als Asymmetric Multiprocessing (AMP) System ausgelegt, bei dem auf einem Prozessorkern das Betriebssystem Linux eingesetzt wird, während der zweite Prozessorkern ohne Betriebssystem (Bare-metal System) nur eine Bildverarbeitungsapplikation ausführt. Als Testapplikation dient eine als C-Quellcode vorliegende Referenzimplementierung eines JPEG-Encoders. Die rechenaufwändigen Teile des Encoders werden mit Hilfe der High-Level Synthese als IP-Blöcke in der rekonfigurierbaren Logik implementiert.

II. JPEG KOMPRESSION

Im Jahr 1992 wurde von der „Joint Photographics Experts Group“ die Kompressionsmethode „JPEG“ für Bilder standardisiert. Der JPEG Algorithmus filtert den redundanten Teil der ursprünglichen Bildinformation heraus, die in der Regel im RGB Format vorliegt. JPEG verwendet dazu die Ergebnisse von Studien über die menschliche visuelle Wahrnehmung. Diese Studien zeigen, dass Menschen Farbunterschiede nicht so gut erkennen können wie Intensitätsunterschiede. Deshalb wird der RGB Farbraum in den geeigneteren YC_bC_r Farbraum umgewandelt. Die Farbkanäle werden danach in der Regel unterabgetastet. Menschen können hohe räumliche Frequenzen nicht gut wahrnehmen. Deshalb wird das Bild in Blöcke unterteilt und durch Anwenden der zweidimensionalen Diskreten Kosinustransformation (DCT) in den örtlichen Frequenzraum transformiert. Auf diese Weise können die hohen Ortsfrequenzen in der nachfolgenden Quantisierungsstufe einfach herausgefiltert werden. Eine weitere Datenreduzierung findet durch Kodierungsverfahren wie Differential Pulse Code Modulation, Lauflängenkodierung und Huffman-Kodierung statt.

Francisco de Asís Molina Martel, fran.molina.martel@gmail.com, Frank Kesel, frank.kesel@hs-pforzheim.de, Manuel Gaiser, manuel.gaiser@hs-pforzheim.de sind Mitglieder der Hochschule Pforzheim, Tiefenbronner Straße 65, 75175 Pforzheim; Thomas Perschke, thomas.perschke@iosb.fraunhofer.de, ist Mitarbeiter des Fraunhofer-IOSB, Ettlingen.

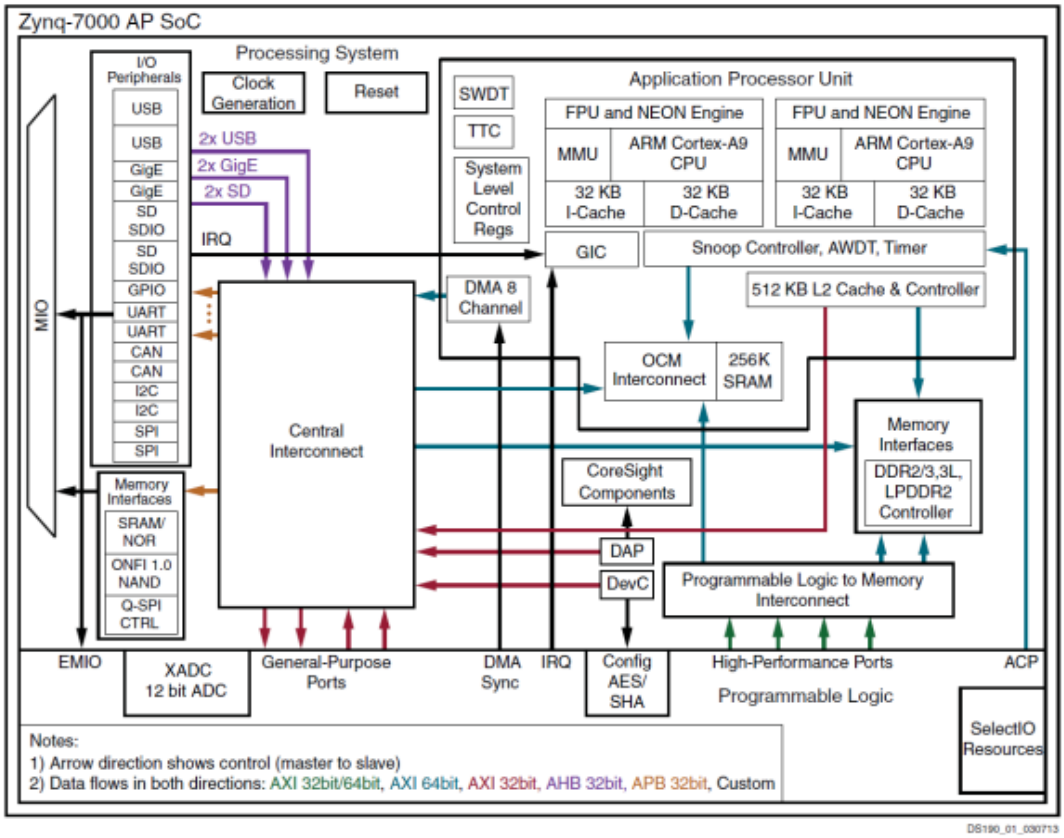

Abbildung 1: Übersicht des Zynq-7000 SoCs [2].

JPEG ist in seiner meistgenutzten Form eine verlustbehaftete Kompression.

Moderne Videokompressionstechniken wie MPEG-1, MPEG-2, MPEG-4, H.263 und H.264 basieren auf denselben Grundlagen wie JPEG. Diese Verfahren sind jedoch komplexer, da zur weiteren Kompression Bewegungsprädiktionen zwischen den Frames genutzt werden.

III. Zynq-7000 SoC

Die Zynq-7000 Plattform (Abbildung 1) verbindet einen Dual-Core ARM Cortex-A9 Prozessorsystem mit rekonfigurierbarer Logik. Das Prozessorsystem besteht zum einen aus den mit bis zu 1 GHz taktbaren Cortex-A9 Kernen mit Fließkommaeinheiten, Neon media-processing engines, separaten L1-Caches und einem gemeinsamen L2-Cache. Als interner Speicher steht ein 256 kB RAM-Speicher, das On-Chip Memory (OCM) zur Verfügung. Zur Anbindung von statischem und dynamischem externem Speicher wird das External Memory Interface bereitgestellt. Für den Transfer zwischen zwei Speichern und zwischen Speicher und Peripherie kann ein 8-kanaliger DMA-Controller eingesetzt werden. Zu den wichtigsten unterstützen I/O-Schnittstellen zählen: 10/100/1000 Ethernet, USB2.0 OTG, CAN 2.0B, SPI, UART und General Purpose I/Os. Die rekonfigurierbare Logik enthält neben Logikblöcken, internen BlockRAM-Speichern, DSP-Blöcken und I/O-Blöcken, je nach Ausführung, Gigabit-Transceiver und PCI Express Endblöcke.

IV. Asymmetric Multiprocessing (AMP)

Der Zynq-7000 SoC wird von mehreren Betriebssystemen unterstützt. Auf der Zynq-Plattform getestete Betriebssysteme sind unter anderem Linux, Android 0 und FreeRTOS [5]. Auf den einzelnen Prozessorkernen können außerdem unterschiedliche Betriebssysteme unabhängig voneinander laufen. Eine solche Konfiguration wird als „Asymmetric Multiprocessing“ (AMP) bezeichnet. In dieser Konfiguration werden im Normalfall relevante Informationen zwischen den einzelnen Prozessorkernen ausgetauscht, um deren Prozesse zu verknüpfen.

Für die vorliegenden Untersuchungen wird ein Linux/Bare-metal System genutzt. Auf dem ersten Prozessorkern (CPU0) wird ein Linaro [6] Ubuntu System eingesetzt. Diese Distribution ist insbesondere für ARM-Prozessoren optimiert. Die Verwendung von Linux kann bei komplexen Systemen, wie einem Bildverarbeitungssystem, die Systementwicklung vereinfachen. Vorhandene Treiber für die Nutzung

externer Peripherien, wie beispielsweise eine USB-Kamera oder eine SD-Speicherkarte, erleichtern den Aufbau des Systems.

Auf dem zweiten Prozessorkern (CPU1) läuft ein Bare-metal System. Bei einem Bare-metal System werden Applikationen ohne den Einsatz eines Betriebssystems ausgeführt. Durch den Verzicht auf ein Betriebssystem können Applikationen prinzipiell schneller ausgeführt werden.

V. High-Level Synthese

Bei der High-Level Synthese wird der zu implementierende Algorithmus mittels einer der Programmiersprachen C, C++ oder SystemC beschrieben. Daraus wird automatisch HDL Code für die physikalische Implementierung generiert.

Da man sich auf einer abstrakteren Ebene als der Register-Transfer-Ebene befindet, muss die Struktur der Hardware nicht beschrieben werden. Die High-Level Synthese automatisiert diese Aufgabe, so dass der Prozess der Hardwareentwicklung beschleunigt wird. Außerdem können mehrere Implementierungslösungen schnell untersucht werden [7]. Die Hardwareimplementierung kann dadurch einfacher bezüglich Performance oder Ressourcenverbrauch optimiert werden.

Vivado HLS ist ein High-Level Synthesewerkzeug der Firma Xilinx. Vivado HLS erzeugt HDL Code aus dem entwickelten C, C++ oder SystemC Code. Die Optimierung der Implementierung wird durch den Einsatz von Direktiven gesteuert. Wichtige Beispiele für Direktiven, die unterschiedliche Formen der Parallelverarbeitung anwenden, sind:

- Loop Pipelining
- Loop Unrolling
- Dataflow

Beim Loop Pipelining wird versucht, die einzelnen Berechnungsschritte innerhalb einer Schleife gleichzeitig auszuführen. Hierbei verarbeiten die einzelnen Stufen Daten, die jeweils zu unterschiedlichen aufeinanderfolgenden Zeitpunkten zur Verarbeitung anliegen. Dies erreicht man durch das Einfügen von Speicherelementen zwischen den einzelnen Verarbeitungseinheiten.

Das Loop Unrolling entrollt Schleifen, d.h. die einzelnen Berechnungsstufen werden in mehrfacher Ausfertigung implementiert. Entspricht deren Anzahl der Anzahl der Schleifeniterationen, bezeichnet man dies als vollständiges Entrollen, im Gegensatz zum teilweisen Entrollen. Die Dataflow Optimierung entspricht einem Pipelining auf der Ebene von Funktionen und hängt dementsprechend von den Datenabhängigkeiten zwischen den einzelnen Funktionen ab.

Die Resultate der Optimierungen werden im Wesentlichen durch die Parameter:

- Latenz
- Initiation Intervall
- Durchsatz
- Ressourcenverbrauch

beschrieben. Die Latenz bezeichnet die Zeitdifferenz zwischen der Eingabe der Daten und der Ausgabe des Resultats, der Durchsatz die Frequenz, mit der neue Ausgabedaten erzeugt werden. Das Initiation Intervall (II) bezeichnet die Zeitdifferenz zwischen zwei aufeinanderfolgenden Ausführungen eines Schleifenblocks. Der Ressourcenverbrauch ist die Anzahl an FPGA-Komponenten, die für die Realisierung der Hardwareimplementierung benötigt werden. Nach der High-Level Synthese kann mit Vivado HLS die gewählte Implementierungsvariante als IP-Block exportiert werden und in einem entsprechenden Werkzeug (Vivado oder EDK XPS) genutzt werden.

VI. Systemaufbau

Für die Evaluierung der Zynq-7000 Plattform wird der in Abbildung 2 dargestellte Systemaufbau genutzt. Die einzelnen Komponenten des Systems sind: ein ZedBoard, eine USB Kamera und eine SD Karte mit einer FAT und einer EXT4 Partition. Die SD Karte wird sowohl für das Booten und die Konfiguration des ZedBoards, als auch als nichtflüchtiger Speicher für Linux genutzt. Auf dem ZedBoard befindet sich ein XC7Z020-CLG484-1 Zynq-7000. Die zwei ARM Cortex-A9 Prozessorkerne werden in einer Linux/Bare-metal AMP Konfiguration genutzt. Beide Prozessoren können über gemeinsame Ressourcen miteinander kommunizieren. In der vorliegenden Konfiguration wird ein gemeinsamer Bereich des OCM für die Kommunikation genutzt. Das Linux-System und das Bare-metal System haben jeweils einen eigenen nichtüberlappenden Speicherbereich im externen DDR3-SDRAM Speicher des Zed-Boards. Das Linux-Betriebssystem auf CPU0 ist wegen seiner vorhandenen Treiber für die Zugriffe auf Peripherien, wie die Kamera oder die EXT4-Partition der SD-Karte, zuständig. Die CPU1 mit der Bare-metal Applikation führt die JPEG Encoder Software aus. Da dieses System nicht durch den Overhead eines Betriebssystems belastet wird, verhält sich die Ausführung des Codes deterministischer. Die kritischen Teile des Algorithmus, die sich gut parallelisieren lassen, werden in der FPGA-Logik realisiert.

Die Bilddaten werden in der Abbildung 2 angezeigten Reihenfolge verarbeitet. Zunächst wird auf dem Linux-System mit Hilfe einer OpenCV-Routine ein Bild mit der Kamera (1) aufgenommen und die Bilddaten werden automatisch im externen Speicher abgelegt (2). Als nächstes wird mit Hilfe der Interprozessorkommunikation die Information über die Kameraauflösung an das Bare-metal System übergeben und nach der Bestätigung des Empfangs der Information wird der Transfer von Bilddaten im RGB-Format gestartet.

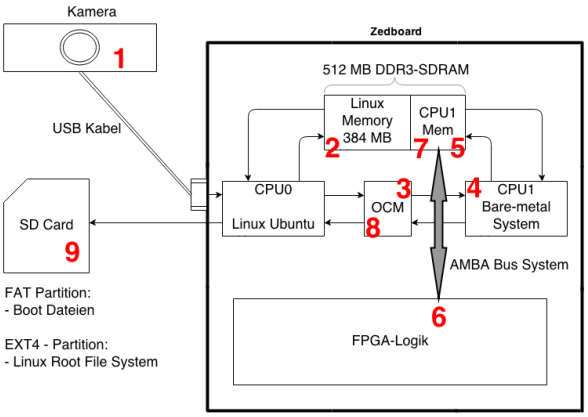

Abbildung 2: Skizze der Komponenten und Reihenfolge der Datenverarbeitung.

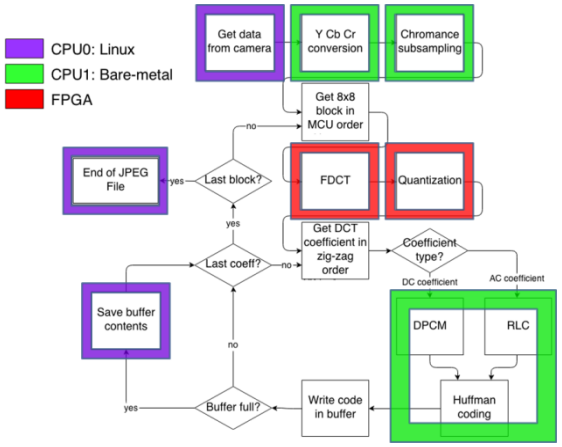

Abbildung 3: JPEG Encoder Partitionierung auf dem Zynq-7000 SoC.

Die RGB-Daten werden über den OCM (3) zum Bare-metal System transferiert. Dort wird direkt der erste Teil des Algorithmus, die YC_bC_r Farbumrechnung, ausgeführt (4). Das transformierte Bild wird im DDR3-SDRAM gespeichert (5) und weiter verarbeitet.

Die kritischen Teile des Algorithmus werden in der Programmierbaren Logik ausgeführt. Für die Datenübertragung zwischen Prozessor und FPGA wird eine High-Performance AMBA Bus Schnittstelle (HPI) mit AXI4-Stream Protokoll verwendet. Die Datenübertragung erfolgt hierbei mittels eines DMA-Transfers direkt zwischen dem DDR3-SDRAM Speicher und der Programmierbaren Logik (6).

Nachdem die Daten in der Programmierbaren Logik verarbeitet wurden, werden diese zurück in den DDR3-SDRAM Speicher transferiert (7). Diese Daten werden danach weiter komprimiert und durch den OCM an das Linux-System transferiert (8). Das encodierte Bild wird zum Schluss in der EXT4-Partition der SD Karte abgelegt (9).

VII. Profiling

Die JPEG Encoder Software wurde als Linuxapplikation profiliert, da die Kameradaten dem Bare-metal System nicht direkt zur Verfügung stehen. Auf diese Weise können die zeitkritischen Teile des Algorithmus ohne großen Kommunikationsoverhead analysiert werden. Die Ergebnisse ergeben sich aus dem Mittelwert von 10 Programmausführungen und können in Tabelle 1 betrachtet werden. Insgesamt dauert die komplette Ausführung etwa 20 Sekunden.

Die fdct Funktion verbraucht etwa 94,2% der gesamten Ausführungszeit des Programms. Diese verarbeitet jeweils einen 8x8 Block und implementiert die FDCT (Forward Discrete Cosinus Transform) und die Quantisierungsstufe. Obwohl die Verarbeitung eines Blocks weniger als eine Millisekunde dauert, macht sich die Zeitdauer durch die große Anzahl von Funktionsaufrufen deutlich bemerkbar.

Tabelle 1: Profilierungsergebnisse.

% der Zeit	Aufrufe	Zeit/Aufruf	Funktion
94,2	21600	0,87 ms	fdct
0,9	71802	0,00 ms	codeLumAC
0,8	1	160 ms	readBMPfile
0,6	1	110 ms	chromaSubs
0,5	921600	<0,01 ms	RGB2C_r
0,5	921600	<0,01 ms	RGB2Y
0,4	21600	<0,01 ms	set_raw8x8
0,4	921600	<0,01 ms	RGB2C_b
0,1	14400	<0,01 ms	codeLumDC
0,0	7200	<0,01 ms	codeChrDC

Daher ist für eine Verbesserung der Gesamtlaufzeit eine Beschleunigung der DCT Funktion notwendig. Die Ausführungszeit aller anderen Funktionen ist im Vergleich sehr gering. Aus diesem Grund können diese zunächst vernachlässigt werden.

VIII. Partitionierung des JPEG Encoders

Die kritischen Teile des Algorithmus werden im FPGA realisiert. Die Partitionierung des Algorithmus wird in Abbildung 3 dargestellt. Im ersten Schritt wird auf CPU0 die Kamera ausgelesen und die Daten werden an CPU1 übertragen. Diese fängt dann direkt mit der Ausführung der Farbumrechnung in den YC_bC_r Farbraum an. Das gesamte Bild wird transformiert und im DDR3-SDRAM gespeichert. Als nächstes findet eine Unterabtastung der C_bC_r Farbwerte statt. Die entstehenden Daten werden wiederum im externen Speicher abgelegt.

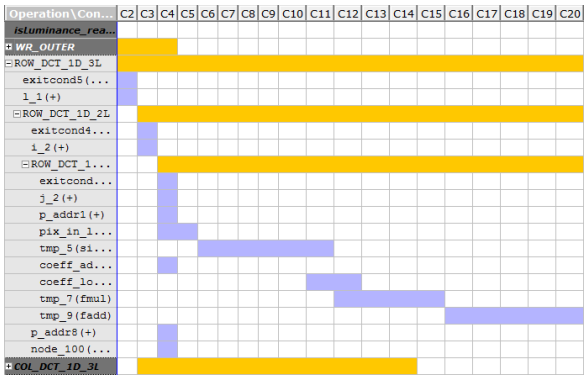

Abbildung 4: Analyse der High-Level Synthese mit Fließkommazahlen.

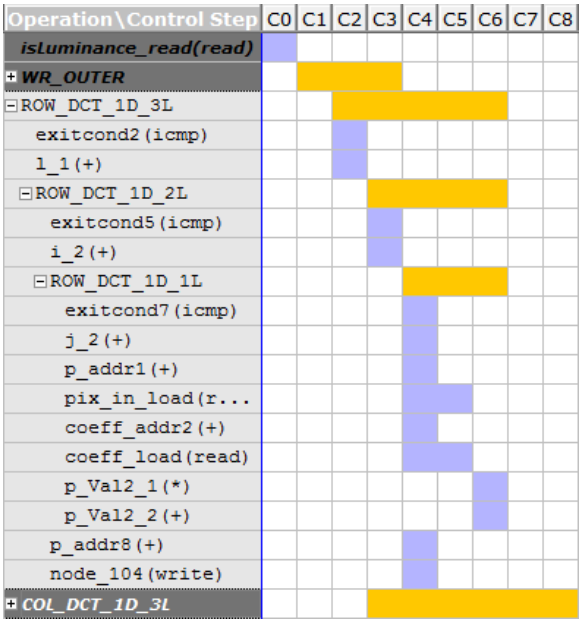

Abbildung 5: Analyse der High-Level Synthese mit Festkommazahlen.

Im nächsten Schritt werden die Bilddaten immer in 8x8 großen Pixelblöcken verarbeitet. CPU1 holt sich die entsprechenden Bilddaten aus dem DDR3-SDRAM Speicher und schreibt sie an eine definierte Stelle, auf die der DMA-Controller Zugriff hat. Der 8x8 große Pixelblock wird dann über eine High-Performance AMBA-Bus Schnittstelle direkt zum FPGA übertragen.

Der Pixelblock wird mit einer Diskreten Kosinustransformation und einer Quantisierung im FPGA verarbeitet. Nachdem der Hardwareblock mit der Datenverarbeitung fertig ist, werden die DCT Koeffizienten an eine vordefinierte Adresse in den SDRAM Speicher geschrieben.

Die DCT Koeffizienten werden im letzten Schritt der Verarbeitung mittels Entropiekodierung kodiert. Die kodierten Daten werden über den OCM an das Linux System gesendet, so dass dieses die Daten als JPEG-Datei auf die EXT4-Partition der SD Karte schreiben kann.

IX. Implementierung

Bei der High-Level Synthese der FDCT und der Quantisierung wird wie folgt vorgegangen. Zuerst wurde eine Testbench in C/C++ geschrieben. Dieses Programm soll die gewünschte Funktionsweise der Hardware durch einen Vergleich mit einer Softwareimplementierung überprüfen. Es ist damit auch möglich, eine Simulation des RTL-Designs zu verifizieren.

Um den Durchsatz zu erhöhen, werden die Fließkommazahlen durch Festkommazahlen ersetzt. Zuerst muss die notwendige Gesamtzahl der Bits und die Zahl der Nachkommabits bestimmt werden. Da in der Quantisierungsstufe auf ganze Zahlen gerundet wird, wird keine hohe Genauigkeit erwartet. Die geplante maximale Genauigkeitstoleranz der Ergebnisse beträgt in diesem Fall 0,1. Aus dem Wertebereich und der Genauigkeitstoleranz wird eine bestimmte Bitbreite gewählt. In diesem Fall werden die Daten mit maximal 16 binären Nachkommastellen berechnet.

Die Latenz, das Initiation Intervall und dementsprechend auch der Durchsatz werden beim Übergang von Fließkommazahlen zu Festkommazahlen stark verbessert. In Abbildung 4 und Abbildung 5 sind zwei durch Vivado HLS erzeugte Diagramme der FDCT Implementierung zu sehen. Die gelben Balken zeigen Schleifen an, die blauen Balken repräsentieren die einzelnen Operationen. Auf der x-Koordinate des Diagrammes sind die Zeitzyklen aufgeführt.

Beide Diagramme entsprechen dem gleichen Hardwaredesign, mit der einzigen Ausnahme, dass in Abbildung 4 die Berechnungen in Fließkommazahlen und in Abbildung 5 in Festkommazahlen erfolgen. Aus Übersichtlichkeitsgründen wird nur die erste dreifach verschachtelte Schleife detailliert angezeigt.

Wenn man beide Diagramme vergleicht, fällt auf, dass durch das Anwenden von Festkommazahlen eine große Anzahl an Zyklen gespart wird. Bei den Fließkommazahlen benötigt eine Multiplikation vier Zyklen und eine Addition fünf Zyklen. Hingegen benötigt die Festkommazahlenberechnung in diesem Fall nur jeweils einen Zyklus für eine Multiplikation und eine Addition. Die Latenz der innersten Schleife beträgt nur noch 3 Zyklen gegenüber 17 Zyklen im Falle der Fließkommazahlen.

Als nächstes wird überprüft, ob die Hardware immer noch das gewünschte Verhalten aufweist und ob ihre Genauigkeit zufriedenstellend ist. Sobald die Verifizierung beendet ist, kann die Durchführung von Optimierungen durch das Einsetzen von Direktiven beginnen.

Für die High-Level Synthese werden zwei unterschiedliche Implementierungen der Diskreten Kosinustransformation analysiert. Basierend auf der Definitionsgleichung der Diskreten Kosinustransformation

$$X[k,l] = \frac{C_k \cdot C_l}{4} \cdot \sum_{n=0}^{7}\sum_{m=0}^{7}(x[n,m] \cdot \cos[\frac{(2n+1)\cdot k \cdot \pi}{16}] \cdot \cos[\frac{(2m+1)\cdot l \cdot \pi}{16}])$$

kann man diese direkt als eine vierfach verschachtelte Schleife implementieren. Aus der obigen Gleichung sieht man, dass, damit jeder DCT Koeffizient ausgerechnet werden kann (zwei Schleifen), eine zweidimensionale gewichtete Summe des 8x8 Pixelblocks gebildet werden muss (weitere zwei Schleifen).

Eine zweidimensionale Diskrete Kosinustransformation kann in zwei eindimensionale Transformationen aufgespalten werden. Die aufgespaltene DCT wird in Matrixform in der folgenden Gleichung dargestellt.

$$DCT = C \cdot X \cdot C^T$$

DCT bezeichnet die 8x8 Matrix mit den DCT Koeffizienten. *X* ist der eingehende 8x8 Datenblock und *C* ist eine 8x8 Matrix mit den entsprechenden Elementen, um aus der Multiplikation mit *X* eine eindimensionale DCT zu bilden. Eine zweidimensionale DCT kann also in eine eindimensionale DCT in horizontaler Richtung und eine eindimensionale DCT in vertikaler Richtung aufgespalten werden. Die Gleichung beinhaltet zwei Matrixmultiplikationen, die sich im Quellcode als zwei dreifach verschachtelte Schleifen ausdrücken lassen. Durch diese Umformung wird ein weiterer Performancegewinn möglich.

X. ERGEBNISSE

Die Ergebnisse der High-Level Synthese sind in Tabelle 2 bis Tabelle 5 dargestellt. Dort werden die Resultate für die beiden vorhandenen Implementierungen gezeigt. In den Tabellen 2 und 4 sind die Ergebnisse der 2D-DCT Implementierung zu sehen. Die restlichen beiden Tabellen 3 und 5 stellen die Ergebnisse der 1D-DCT Implementierung dar. Die Ergebnisse, die maximale Taktfrequenz und der Ressourcenverbrauch, sind keine exakten Ergebnisse nach dem Place & Route Prozess, sondern eine Schätzung des High-Level Synthesewerkzeugs. In der ersten Spalte links stehen die in Vivado HLS angegebenen Direktiven. Die Angabe der Direktiven ist eine Methode, um ein Hardware Design zu optimieren. Die Pixelrate ergibt sich aus der Multiplikation des Durchsatzes, gegeben durch die Taktrate f_{clock} dividiert durch das Initiation Intervall *II*, und der Anzahl an eingehenden bzw. ausgehenden Pixel.

Tabelle 2: Performance Ergebnisse der 2D DCT Implementierung.

Direktive	Pixelrate [MHz]	II [Takte]	Maximale Taktrate [MHz]
Standard	0,25	105388	418
Pipeline 1. Loop	1,51	5011	118
Pipeline 3. Loop	9,64	2776	418
Array Partition	45,60	407	290
Dataflow	54,75	339	290

$$Pixelrate = \frac{64 \cdot f_{clk}}{II}$$

Die maximale Taktrate wird durch den kritischen Pfad bestimmt. Es ist die maximale Frequenz, mit dem die erstellte Hardware getaktet werden kann. Obwohl die maximale Taktrate in manchen Fällen kleiner wird, kann die Pixelrate durch die Angabe von Direktiven gesteigert werden. Der Grund dafür ist, dass das Initiation Intervall des Blocks kleiner geworden ist, und zwar in einem höheren Maß als die Abnahme der Taktrate.

In Tabelle 4 und Tabelle 5 sind die verbrauchten FPGA-Ressourcen zu sehen. Aus Platz- und Energieverbrauchsgründen versucht man üblicherweise den Ressourcenverbrauch gering zu halten. Für die Systemimplementierung ist im vorliegenden Fall jedoch der maximale Durchsatz die einzige Zielgröße. Der verwendete Zynq-7000 SoC XC7Z020-CLG484-1 verfügt über 140 36 KB große Dual-Port BlockRAM Zellen, 220 integrierte DSP-Blöcke, 106400 Flipflops und 53200 Lookup Tabellen. Die 36 KB großen BlockRAM-Blöcke werden in Vivado HLS in zwei 18 KB Single-Port Blöcken aufgeteilt. Dadurch entstehen 280 kleinere Speicherblöcke, die unabhängig voneinander genutzt werden können.

Die Nummerierung der verschachtelten Schleifen beginnt bei der innersten Schleife. Das heißt, dass die Direktive „Pipeline 1. Loop“ sich auf die innerste Schleife bezieht. Beim Pipelining äußerer verschachtelter Schleifen werden automatisch die inneren Schleifen komplett entrollt. Dies bewirkt einen erhöhten Ressourcenverbrauch, wie in Tabelle 4 und Tabelle 5 zu sehen ist. Ein großer Gewinn an Datendurchsatz wird in Tabelle 2 und Tabelle 3 erst nach Einsatz der Direktive „Array Partition“ deutlich. Durch diese Direktive partitioniert Vivado HLS die verwendeten 8x8 Matrizen und verteilt diese Strukturen über unterschiedliche BlockRAMs oder speichert diese im logikbasierten internen Speicher, so dass parallele Zugriffe auf diese Speicher möglich sind.

Tabelle 3: Performance Ergebnisse der 1D DCT Implementierung.

Direktive	Pixelrate [MHz]	II [Takte]	Maximale Taktrate [MHz]
Standard	2,65	5876	243
Pipeline 1. Loop	4,70	1607	118
Pipeline 2. Loop	30,58	607	290
Array Partition	80,70	230	290
Dataflow	218,35	85	290
Pipeline 3. Loop	88,36	176	243
Dataflow	174,74	89	243

Tabelle 4: Ressourcenverbrauch der 2D DCT Implementierung.

Direktive	BRAM	DSP	FF	LUT
Standard	4 (1%)	5 (2%)	395 (0%)	617 (1%)
Pipeline 1. Loop	4 (1%)	5 (2%)	405 (0%)	700 (1%)
Pipeline 3. Loop	11 (3%)	131 (59%)	8322 (7%)	14075 (26%)
Array Partition	2 (0%)	131 (59%)	9226 (8%)	17137 (32%)
Dataflow	2 (0%)	131 (59%)	9679 (9%)	18376 (34%)

In der High-Level Synthese-Literatur wird empfohlen, bei verschachtelten Schleifen zuerst das Pipelining auf die innersten Schleifen und danach auf die äußeren Schleifen anzuwenden, bis die gewünschte Performance erreicht wird. Im Allgemeinen resultiert aus diesem Vorgehen das beste Flächenverbrauch/Performance Verhältnis [8].

Das Ergebnis mit dem höchsten Datendurchsatz hat eine Pixelrate von 218,35 MHz bei einer maximalen Taktrate von 290 MHz. Ohne Berücksichtigung der Datenübertragung zwischen DDR3-SDRAM Speicher und FPGA beträgt das Initiation Intervall der Hardware 293,1 ns. Dies entspricht einem Beschleunigungsfaktor von 3000 gegenüber der ursprünglichen Software Lösung mit einem Initiation Intervall von mindestens 0,87 ms.

Die Datenübertragung zwischen DDR3-SDRAM Speicher und FPGA konnte momentan nur auf einem Bare-metal System untersucht werden, die Implementierung auf dem AMP ist noch im Gange. Daher kann dieser Einfluss noch nicht genau berücksichtigt werden. Die gemessene Latenz der Datenübertragung eines 8x8 großen Pixelblocks vom DDR3-SDRAM Speicher zum FPGA und zurück beträgt im Bare-metal System 17 µs. Nimmt man diese Zeit auch für das AMP System an, so erhält man den Beschleunigungsfaktor 50.

Tabelle 5: Ressourcenverbrauch der 1D DCT Implementierung.

Direktive	BRAM	DSP	FF	LUT
Standard	6 (2%)	3 (1%)	253 (0%)	335 (0%)
Pipeline 1. Loop	6 (2%)	3 (1%)	321 (0%)	544 (1%)
Pipeline 2. Loop	13 (4%)	17 (7%)	938 (0%)	426 (0%)
Array Partition	19 (6%)	17 (7%)	3006 (2%)	1837 (3%)
Dataflow	34 (12%)	17 (7%)	3468 (3%)	4198 (7%)
Pipeline 3. Loop	25 (8%)	132 (60%)	8973 (8%)	5086 (9%)
Dataflow	24 (8%)	140 (63%)	9334 (8%)	7472 (14%)

XI. FAZIT

Für die Evaluierung der Zynq-7000 Plattform wurde der Zynq Dual-ARM A9 Prozessor als Linux/Bare-metal Asymmetric Multiprocessing (AMP) System konfiguriert. Als Testplattform diente ein Zed-Board mit angeschlossener USB-Kamera. Als Testapplikation diente eine Softwareimplementierung eines JPEG-Encoders.

Die JPEG-Encoder-Software wurde durch Profiling näher analysiert. Die Ergebnisse zeigen dabei, dass die Diskrete Kosinustransformation mehr als 90% der gesamten Ausführungszeit verbraucht. Die Hardwareimplementierung der Diskreten Kosinustransformation und der Quantisierung wurden durch Einsatz der High-Level Synthese generiert und mehrere Implementierungsvarianten untersucht. Die maximale erreichte Pixelrate beträgt 218,35 MHz. Ohne Berücksichtigung der DMA-Datentransfers entspricht dies einer Beschleunigung um den Faktor 3000. Mit einer Abschätzung der Transferzeit aus einem Bare-metal System erhält man eine Beschleunigung um den Faktor 50.

LITERATURVERZEICHNIS

[1] M. Gokhale, P. Graham, “Reconfigurable Computing”, Springer 2005.

[2] Xilinx, “Zynq-7000 All Programmable SoC Technical Reference Manual”, UG585 (v1.6), June 28, 2013.

[3] http://www.zedboard.org.

[4] M. Barbareschi, A. Mazzeo, A. Vespoli, "ZedAndroid: Google Android porting on ZedBoard", http://wpage.unina.it/mario.barbareschi/zedroid/index.html, October 29, 2013.

[5] http://www.xilinx.com/products/silicon-devices/soc/zynq-7000/operating-systems/

[6] http://www.linaro.org.

[7] J. P. Elliot, "Understanding behavioural synthesis: a practical guide to high-level design", Springer, 1999.

[8] M. Fingeroff, "High-Level Synthesis", Blue Book, 2010.

Francisco de Asís Molina Martel hat sein Bachelorstudium im Studiengang Energie- und Automatisierungstechnik an der Hochschule Karlsruhe im Oktober 2011 abgeschlossen. Sein Masterstudium in Embedded Systems hat er in Januar 2014 abgeschlossen. In seiner Masterarbeit befasste er sich mit HW-SW-Co-Design auf der Zynq-7000 SoC Plattform.

Thomas Perschke ist promovierter Diplom-Physiker. Zurzeit arbeitet er in der Abteilung Objekterkennung am Fraunhofer-Institut für Optronik, Systemtechnik und Bildauswertung in Ettlingen.

Frank Kesel erhielt den akademischen Grad des Dipl.-Ing in Elektrotechnik im Jahr 1988 von der Universität Karlsruhe (TH) und den Grad des Dr.-Ing. in Elektrotechnik von der Universität Hannover im Jahr 1994. Er ist Professor für integrierte Schaltungstechnik an der Hochschule Pforzheim.

Manuel Gaiser erhielt den akademischen Grad des Dipl.-Ing. (FH) an der Fachhochschule Karlsruhe in Nachrichtentechnik. Er ist Laboringenieur an der Hochschule Pforzheim in den Bereich Embedded Systems und digitale Hardware. Außerdem beschäftigt er sich mit diversen Forschungs- und Entwicklungsprojekten.

Algorithmus und FPGA-Architektur einer modifizierten generalisierten Hough-Transformation für Stereo-Korrespondenz

Frank Schumacher, Thomas Greiner

***Zusammenfassung*— In dieser Arbeit wird ein zweistufiger Ansatz vorgestellt, um die Stereokorrespondenz eines Stereobildpaars zu berechnen. In der ersten Stufe werden mit einer modifizierten generalisierten Hough-Transformation und absoluten Differenzen grobe Disparitäten in Bildzeilensegmenten bestimmt. In der zweiten Stufe werden die groben Disparitäten evaluiert und mit einer Census-Transformation verfeinert. Schließlich wird das Ergebnis mit einem 3x3-Median-Filter geglättet und die Disparitätskarte erstellt. Weiterhin wird in dieser Arbeit eine FPGA-Architektur vorgestellt, in der dieser zweistufige Ansatz umsetzt ist. Die Architektur ist parallel und arbeitet im Raster-Scan–Modus, d.h. pro Taktzyklus wird ein Pixelpaar verarbeitet und der dazugehörige Disparitätswert ermittelt. Dieser Ansatz erreicht Disparitätsqualitäten, die vergleichbar mit publizierten Hardware-Ansätzen aus der Literatur sind, belegt dabei aber deutlich weniger Hardwareressourcen und erreicht einen höheren Datendurchsatz. Synthetisiert, beispielsweise für einen Xilinx Virtex 5 FPGA, erreicht die vorgestellte Architektur für Bilder der Größe 640x480 und einer Census-Fenstergröße von 9x5 eine Pixelfrequenz von 125 MHz, was einer Wiederholrate von 407 Bildern pro Sekunde entspricht. Das Design belegt dabei 9.548 (11,6%) Register, 29.232 (35,6%) Look-Up-Tabellen, 10.347 (50,5%) Slices und 8 (2,8%) Block-RAMs.**

***Schlüsselwörter*—Bildverarbeitung, Hough-Transformation, Stereo, FPGA-Architektur, Echtzeit.**

I. EINLEITUNG

Als Teildisziplin der digitalen Bildverarbeitung beschäftigt sich die Stereokorrespondenz mit der Berechnung der Tiefenkarte aus einem Stereobildpaar. Das Bildpaar wird in der Regel mit einem Stereokamerasystem aufgenommen, welches zu einer Szene zwei Bilder mit einem horizontalen Versatz liefert. Wenn die Epipolarbedingung erfüllt ist, d.h. die beiden Bilder vertikal rektifiziert sind, kann aus dem horizontalen Versatz eines zusammengehörigen Pixelpaars dessen Tiefe in der Szene bestimmt werden. Die horizontalen Verschiebungen d der Pixel des linken und rechten Bilds bilden die Disparitätskarte.

Die Tiefe Z eines Pixels p mit Disparität d kann mit $Z_p = f \cdot b/d_p$ berechnet werden mit f als Fokuslänge der Kameras und b als sog. Baseline, d.h. dem Abstand der beiden Kameras. Um den horizontalen Versatz, Disparität genannt, jedes Pixelpaars zu bestimmen, müssen korrespondierende Pixel in den Bildpaaren gefunden werden. Hierfür wird üblicherweise zu jedem Pixel aus dem linken Bild aus einer Menge von Pixeln aus dem rechten Bild ein Ähnlichkeitsmaß berechnet und das Paar mit der größten Ähnlichkeit als Pixelpaar definiert. Die Qualität von Algorithmen zur Stereo-Korrespondenz wird quantitativ durch den Vergleich der berechneten Disparitätskarte mit einer Groundtruth von Referenzdatensätzen ermittelt, wie beispielsweise dem bekannten Middlebury Datensatz [1]. Weitere wichtige Parameter von Implementierungen sind Durchsatz, Latenz und Hardware-Ressourcenbelegung. Die aktuell besten Verfahren bezüglich der Tiefenkartenqualität basieren auf Optimierungsalgorithmen, die zwischen den linken und rechten Pixeln der Bildpaare eine globale Kostenfunktion minimieren, siehe z.B. [2]. Diese Verfahren sind sehr aufwändig und kaum für Echtzeit- oder eingebettete Systeme geeignet, da ihre Anforderungen an Rechenleistung, Speicher und Energiebedarf sehr hoch sind [3].

Alternativ existieren so genannte lokale Verfahren, die die Korrespondenz von Pixelpaaren innerhalb begrenzter Bereiche finden. Diese Verfahren erzielen eine niedrigere Qualität, können aber Echtzeit erreichen. Diese Algorithmen arbeiten mit lokalen Fenstern, in denen benachbarte Pixel mit einer Gewichtung beaufschlagt werden. Die Gewichtungen sind meist abhängig von der Farbdifferenz oder dem örtlichen Abstand der Fensterpixel zum Referenzpixel in der Mitte des Fensters. Innerhalb des Fensters werden Kosten aufsummiert, die ein Ähnlichkeitsmaß definieren, mit dem das aktuelle linke Pixel mit den rechten Pixeln im Disparitätssuchbereich D verglichen werden. Betrachtet man übliche Stereobilder, dann liegt zu einem linken Pixel $p_L(y,x)$ das passende rechte Pixel im Bereich

Frank Schumacher, frank.schumacher@hs-pforzheim.de und Thomas Greiner, thomas.greiner@hs-pforzheim.de sind Mitglieder der Hochschule Pforzheim, Tiefenbronner Str. 65, 75175 Pforzheim.

$p_R(y, x - D \dots x)$ wobei D die maximale Disparität bezeichnet und in der Regel eine feste Größe der vorliegenden Szene ist. Dies bedeutet, dass für jedes linke Pixel D Berechnungen durchgeführt werden müssen, um das am besten passende rechte Pixel zu finden. Wenn Fenster der Größe $w = m \times n$ eingesetzt werden, sind $m \cdot n \cdot D$ Berechnungen pro Pixel notwendig. Für parallele Architekturen bedeutet dies ein hohes Maß an erforderlichen Ressourcen.

Diese Arbeit beschreibt einen neuartigen zweistufigen Algorithmus für eine FPGA-Architektur, der die Stereokorrespondenz im Raster-Scan-Verfahren effizient und mit hohem Datendurchsatz berechnet. Hierfür werden im ersten Schritt die Bildzeilen in Segmente aufgeteilt, für welche dominierende Disparitäten mit einer modifizierten generalisierten Hough-Transformation berechnet werden. Im zweiten Schritt werden die durch die dominierenden Disparitäten definierten Pixel auf Ähnlichkeit mit einer Census-Transformation getestet. Die dadurch verfeinerten Disparitäten werden schließlich mittels Median-Filter geglättet. Der Ansatz erzeugt Disparitätskarten mit vergleichbarer Qualität anderer publizierter Architekturen, welche allerdings deutlich niedrigere Bildwiederholraten erzielen und dabei mehr Hardware-Ressourcen belegen.

II. Zweistufiger Stereo-Algorithmus

Die meisten lokalen Stereo-Korrespondenz-Algorithmen berechnen die Übereinstimmungskosten eines linken Pixels zum Pixelbereich im rechten Bild mit relativ großen Fenstergrößen, wie z.B. [4]. Deshalb müssen für jedes linke Bildpixel $m \cdot n \cdot D$ Operationen, insgesamt also $h_I \cdot w_I \cdot m \cdot n \cdot D$ und $h_I \cdot w_I \cdot D$ Vergleiche durchgeführt werden mit $h_I \times w_I$ als Bildgröße. Abhängig vom tatsächlich gewählten Ansatz kann eine Operation dabei beispielsweise aus der Summe der absoluten Differenzen oder einer Bildfilterung bestehen. Diese Berechnungskomplexität wurde in dem beschriebenen Ansatz mittels eines zweistufigen Algorithmus reduziert. In der ersten Stufe werden dominierende Disparitäten innerhalb von Bildzeilen berechnet. Die Zeilen werden in Segmente der Länge D aufgeteilt und für jedes Pixel in einem Segment die Übereinstimmungskosten mittels absoluter Differenzen (AD) der Pixelintensitäten berechnet. Der minimale AD-Wert wird als beste Übereinstimmung gewählt und sein Segment-Index bestimmt. Die Indizes eines Segments werden in einem Hough-Transformation-basierten Voting-Schema genutzt, um die am besten passende Pixelverschiebung in diesem Segment zu akkumulieren. Das Voting und die Akkumulation finden in einem 1D-Hough-Raum, genannt $houghline$ statt, siehe [5] für Details.

Wir nennen dieses Vorgehen In-Line-Voting, bei welchem die Epipolarbedingung rektifizierter Stereobilder genutzt wird. Mit der Annahme, dass einige Pixel in einem Segment dieselben Verschiebungen zueinander aufweisen, bildet sich ein Maximum in dem Hough-Raum. Wenn alle Indizes eines Segments gevotet und akkumuliert sind, zeigt das Maximum die dominierende Disparität dieses Segments an:

$$d_{dom} = D - argmax(houghline(1 \dots D))$$

Dadurch ergibt sich für jede Bildzeile der Länge w_I eine Menge von $\lceil \frac{w_I}{D} \rceil$ dominierenden Disparitäten. Mit diesen grob bestimmten Disparitäten kann bereits eine Disparitätskarte erstellt werden, deren Qualitätsergebnisse jedoch noch nicht vergleichbar sind. Um vergleichbare Qualitätsergebnisse zu erzielen, werden die Disparitäten in der zweiten Stufe des Algorithmus verfeinert.

In der zweiten Stufe werden die groben Disparitätswerte zunächst expandiert, δ Pixel nach rechts und δ Pixel nach links. Wenn sich das aktuelle Segment nicht am rechten oder linken Bildrand befindet, werden außerdem die dominierenden Disparitäten der beiden Nachbarsegmente aufgenommen mit derselben Expansion. Dadurch wird für jedes Segment eine Menge von $(2\delta + 1) \cdot 3$ Disparitäten auf Gültigkeit geprüft. Für die Segmente an den Bildrändern wird jeweils nur ein Nachbarsegment berücksichtigt.

Mit den expandierten Disparitäten werden aus dem rechten Bild Pixelkandidaten ausgewählt, die genauer auf Übereinstimmung mit linken Pixeln getestet werden. Hierfür wird aus den Disparitäten ein Indexvektor ind bestimmt, der k Indizes enthält, die auf Pixel im rechten Bild referenzieren und die mit dem aktuell verarbeiteten linken Pixel verglichen werden. Der Vergleich erfolgt mit der Census-Transformation (CT) [6] und die Übereinstimmungskosten werden mittels Hamming-Abstand (engl.: Hamming-Distance HD) bestimmt. Dies ist eine gängige Methode in der Stereo-Korrespondenz. Für einen Census-Wert im linken Bild wird der Hamming-Abstand zu einer Menge rechter Pixel bestimmt und im Hamming-Abstandsvektor hdv gespeichert:

$$hdv(i) = HD\left[CT\left(p_L(y,x)\right), CT\left(p_R\left(y, x - ind(i)\right)\right)\right] \; mit \; i = 0 \dots k-1$$

Das Minimum des Hamming-Vektors zeigt die beste Übereinstimmung des linken Pixels $p_L(y,x)$ und einem Pixel im rechten Pixelbereich $p_R(y, x - D \dots x)$ an und damit die beste Disparität. Aus diesen wird die verfeinerte Disparitätskarte erstellt. Schließlich wird ein 3x3 Median-Filter genutzt, um Ausreißer zu eliminieren und dadurch die Disparitäten weiter zu verfeinern.

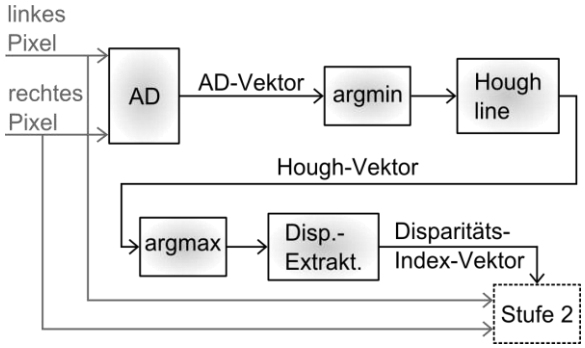

Abbildung 1: Stufe 1 des vorgestellten Algorithmus.

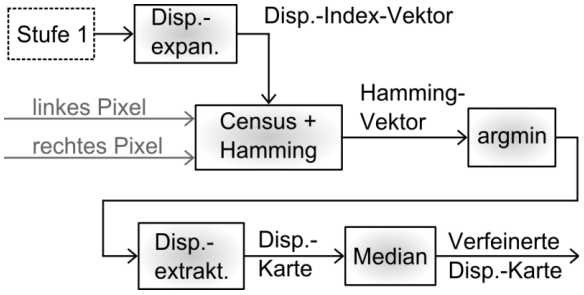

Abbildung 2: Stufe 2 des vorgestellten Algorithmus.

III. Hardware-Architektur

Betrachtet man Details tatsächlicher Stereo-Korrespondenz-Implementierungen, so ist ersichtlich, dass CPU- oder GPU-Umsetzungen nicht echtzeitfähig oder aufgrund hohen Energiebedarfs nicht für eingebettete Systeme geeignet sind [3].

In dieser Arbeit ist eine FPGA-Architektur beschrieben, die eine gute Balance zwischen Disparitätskarten-Qualität, Datendurchsatz und Hardware-Ressourcenbelegung bietet. FPGAs sind auch sehr gut für die Raster-Scan-Verarbeitung geeignet. Dies ist ein wichtiges Kriterium für eingebettete Bildverarbeitungssysteme, da die meisten bildgebenden Chips ein Bildpixel pro Takt liefern. Bildverarbeitungssysteme, die ein Eingangspixel pro Taktzyklus verarbeiten können, erreichen hohe Verarbeitungsgeschwindigkeiten und benötigen sehr geringe Zwischenspeichergrößen, da nur wenige Bildzeilen und keine gesamten Bilder gepuffert werden müssen.

Viele Bildverarbeitungsroutinen, beispielsweise 2D-Filter, lassen sich einfach in Raster-Scan umsetzen: Einige Bildzeilen der Länge w_I werden in Zeilenpuffern zwischengespeichert und die meisten Operationen können mittels relativ kleiner, paralleler Schaltungen umgesetzt werden [7]. Für Verarbeitungsfenstergrößen von $w = m \times n$ müssen w Operationen parallel durchgeführt und $(m-1)$ Bildzeilen zwischengespeichert werden. Solche Schaltungen liefern nach $(m-1) \cdot w_I$ initialen Takten ein Ausgangspixel pro Taktzyklus, wenn sie korrekt gepipelinet sind.

Wie zuvor erwähnt, müssen bei der Stereo-Korrespondenz in jedem Taktzyklus ein linkes Bildpixel und D rechte Bildpixel verarbeitet und verglichen werden. Deshalb benötigen herkömmliche fensterbasierte lokale Ansätze $m \cdot n \cdot D$ parallele Operationen in jedem Taktzyklus, was eine große Menge Hardwareressourcen belegt. Aus diesem Grund ist die vorgeschlagene Hardware-Architektur ebenfalls zweistufig angelegt.

A. Stufe 1: Hough-Linien-Voting und dominierende Disparitäten

Die erste Stufe, dargestellt in Abbildung 1, besteht aus Modulen zur Berechnung der absoluten Differenzen (AD) aus den Eingangspixeln, dem Hough-Linien-Voting und der Bestimmung der dominierenden Disparitäten. Das AD-Modul puffert D rechte Eingangspixel in einer Schieberegisterpipeline. Sobald ein neues Pixelpaar eingetaktet ist, werden mit einer aus kombinatorischer Logik bestehenden Schaltung D AD-Werte parallel berechnet.

Der resultierende AD-Vektor wird in ein argmin-Modul gegeben, welches den Index des kleinsten Werts bestimmt. Es wurden zwei Versionen des argmin-Moduls erstellt: Eine parallele Version mit kombinatorischem Komparator-Baum und D Komparatoren und eine zweite Version mit Zwischenregisterpipelines, welche durch den kürzeren kritischen Pfad eine höhere Taktfrequenz erreicht, aber mehr Logikressourcen belegt und eine höhere Latenz aufweist. In der gesamten FPGA-Architektur sind drei argmin- und argmax- Module eingesetzt. Die unterschiedlichen Umsetzungen können genutzt werden, um die Geschwindigkeit des Systems zu erhöhen oder weniger Logikressourcen zu belegen.

Im Hough-Linien-Modul wird mit den Minimum-Indizes der AD-Vektoren in einem Hough-Akkumulator gevotet. Der Akkumulator ist als weitere Schieberegisterpipeline umgesetzt, die D Voting-Werte beinhaltet. Sind alle Voting-Werte eines Segments gepuffert, zeigt der Maximum-Index die dominierende Disparität des aktuellen Segments an. Der Index dieses Maximums wird im letzten Modul der Stufe 1 extrahiert und schließlich in Stufe 2 gegeben.

B. Stufe 2: Präzisierung der Disparitäten

Die Module der Stufe 2 sind in Abbildung 2 illustriert. Das erste Modul expandiert die Disparitäten. In diesem werden $(2\delta + 1) \cdot 3$ Disparitätswerte zu einem Vektor extrahiert und neben dem linken und rechten Bildpixelstrom in das Modul zur Census-Transformation gegeben. Die Census-Transformation ist eine

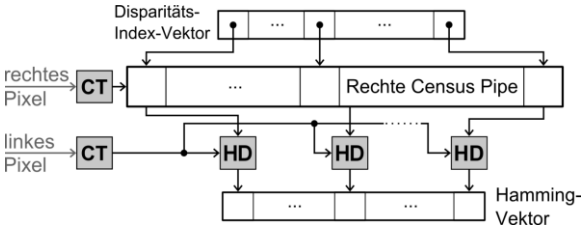

Abbildung 4: Census-Transformation und Hamming-Distanz mit rechter Census-Werte-Pipeline.

Nachbarschaftsoperation und es wurden verschiedene Fenstergrößen hinsichtlich Hardware-Komplexität und Disparitätsqualität evaluiert. Aufgrund der Raster-Scan-Verarbeitung müssen Bildzeilen gepuffert werden, bevor der Nachbarschaftsoperation genügend Daten zur Verfügung stehen und sie durchgeführt werden können. Dafür wurde das interne Block-RAM des FPGAs als FIFO-Puffer genutzt sowie kurze Eingangsregisterpipelines, um die Daten zu synchronisieren. Dadurch können Zeilenpuffer und parallele Nachbarschaftsoperationen effizient umgesetzt werden [8]. Es wurden die Fenstergrößen $5 \times 5, 7 \times 7, 9 \times 9, 11 \times 11, 9 \times 5$ evaluiert, siehe Abschnitt IV.

Das Modul zur Census-Transformation ist in Abbildung 4 dargestellt. Es puffert die Census-transformierten Werte des rechten Bildes in einer Pipeline mit Länge D. Das Modul erhält neben dem linken und rechten Bildpixel auch den Vektor mit den Indizes der expandierten Disparitäten. Diese Indizes verweisen auf Werte in der Census-Pipeline, die Kandidaten für eine genauere Prüfung mittels Hamming-Abstand sind. In jedem Taktzyklus wird der Hamming-Abstand berechnet, implementiert als bitweise XOR-Funktion und Aufsummierung der verbleibenden Einsen, zwischen dem aktuellen linken Pixel-Census-Wert und den rechten Pixel-Census-Kandidaten. Das Ergebnis wird in einer Hamming-Vektor-Pipeline gespeichert und das Minimum zeigt die beste Disparität für das aktuelle linke Pixel an.

C. Nachbearbeitung

Mit der besten Disparität für jedes Pixel wird die Disparitätskarte erstellt. Um Ausreißer in der Karte zu entfernen, wird sie mit einem 3x3 Median-Filter gefiltert. Da in der Literatur zahlreiche effiziente Hardware-Umsetzungen von Median-Filtern existieren, wird in dieser Arbeit nicht näher darauf eingegangen. In [9] wird beispielsweise eine 3x3 Median-Filter-Umsetzung für einen Virtex 5 FPGA vorgestellt, um ein HDTV-Signal (Bildbreite 1920 Pixel) zu verarbeiten. Sie erreicht eine Frequenz von 460 MHz bei sehr geringer Ressourcenbelegung.

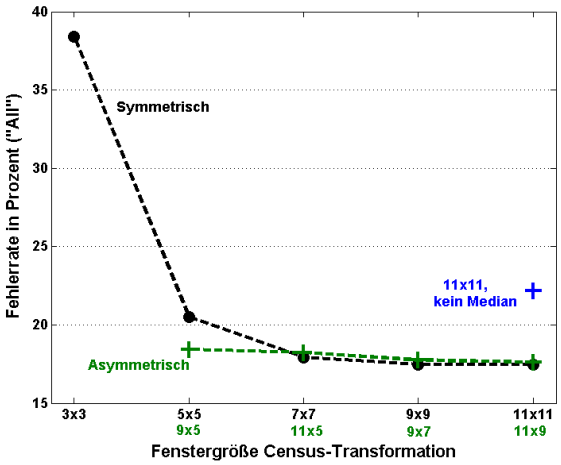

Abbildung 3: Disparitätsqualität für Datensatz *Cones* in Abhängigkeit der Census-Fenstergröße.

IV. Ergebnisse und Diskussion

Die Qualität des vorgestellten Algorithmus wurde mit den Middlebury Stereo-Datensätzen *Cones* und *Teddy* ermittelt [1]. In Abbildung 3 sind die Ergebnisse für verschiedene Census-Fenstergrößen grafisch dargestellt und in Tabelle 1 aufgelistet. Die Disparitätskarten-Qualität ist für verschiedene Datensätze nach [1] des vorgeschlagenen Ansatzes und publizierte Ansätze aus der Literatur angegeben. Die Fehlerraten sind in Prozent aufgelistet für alle Pixel („all“) und Pixel in nicht überdeckten Bereichen (Non Occluded, „Nocc“). Die Census-Transformation-Fenstergröße für den vorgeschlagenen Ansatz ist in der Spalte „Fenster“ zu finden. Die Fenstergrößen, sofern verfügbar, für publizierte Ansätze aus der Literatur sind ebenfalls in dieser Spalte angegeben. Wie erwartet, erreichen größere Fenster eine höhere Qualität, allerdings kann eine Sättigung bei den Ergebnissen ab 7x7 und 9x5 beobachtet werden.

In der Gegenüberstellung der erzielten Ergebnisse mit Ergebnissen anderer Ansätze, die deutlich mehr Ressourcen belegen, z.B. [4], erreicht der vorgeschlagene Ansatz vergleichbare Disparitäts-Qualitäten. Eine weitere Erkenntnis ist, dass asymmetrische Fenstergrößen nur wenig schlechtere Qualität liefern. Das 9x5 Fenster ist lediglich 1-2% schlechter als das 9x9 Fenster. Aus Tabelle 2 und Tabelle 3 ist ersichtlich, dass die Fenstergröße eine wichtige Rolle bei der Ressourcenbelegung und erforderlichen Taktfrequenz spielt. Ein Grund dafür ist die geringere Anzahl benötigter Bildzeilenpuffer. Da die Qualitätseinbußen nur sehr gering sind, stellt diese Variante eine gute Balance zwischen Qualität und Ressourcenbelegung dar.

Tabelle 1: Qualitätsergebnisse.

Ansatz	Daten-satz	Fenster	All	Nocc
Vorg.	Cones	5x5	20,52	10,86
Vorg.	Cones	9x9	17,48	7,36
Vorg.	Cones	9x5	18,44	8,4
[4]	Cones	9x9	15,9	4,63
[10]	Cones	11x11	17,58	7,34
[11]	Cones	13x13	25,9	17,1
[12]	Cones	n.a	26,8	n.a.
Vorg.	Teddy	5x5	29,23	21,19
Vorg.	Teddy	9x9	22,88	15
Vorg.	Teddy	9x5	24,34	15,7
[4]	Teddy	9x9	17,6	6,55
[10]	Teddy	11x11	21,5	12,5
[11]	Teddy	13x13	28,1	21,5
[12]	Teddy	n.a	36,8	n.a.

Tabelle 2 und Tabelle 3 zeigen die Ergebnisse der Hardware-Synthese mit Xilinx Synthesis Technology (XST) für einen Xilinx Virtex 5 XC5VFX130T FPGA. Verschiedene Konfigurationen hinsichtlich Bildgrößen, Census-Fenstergrößen und Versionen der argmin/argmax-Funktionen wurden realisiert. Die verschiedenen Varianten der argmin/argmax-Funktionen sind gekennzeichnet mit „Baum“ für Umsetzungen mit kombinatorischen Komparatorbäumen und „Reg.“ für Umsetzungen mit Zwischenschieberegisterpipelines. Tabelle 2 stellt die maximal erreichbare Takt-Frequenz der Architekturen in MHz, die resultierende Bildwiederholrate in Bildern pro Sekunde (engl.: Frames per Second, Fps) und Geschwindigkeit in Millionen Disparitätsevaluationen pro Sekunde (engl.: Million Disparity Evaluations per Second MDE/s) nach [3] dar. In Tabelle 3 sind die Ressourcenbelegungen angegeben als FPGA-Slices und benötigte Block-RAMs. Die Prozentzahlen in den Klammern geben die relativen Chipflächenbelegungen für die jeweils genutzten FPGA-Derivate an.

Vergleiche mit anderen Ansätzen zeigen, dass die vorgestellte Architektur deutlich weniger Hardware-Ressourcen belegt und trotzdem einen höheren Datendurchsatz erreicht. Beispielsweise benötigt das 9x9-Design von [4] 86% der Slices des aktuell größten Virtex6 FPGAs. Speziell die asymmetrischen Census-Fenster demonstrieren eine gute Balance zwischen Disparitätskartenqualität, Datendurchsatz und Ressourcenbelegung.

Tabelle 2: XST Synthese-Ergebnisse: Erreichte Taktfrequenz in MHz, Bilder pro Sekunde in Fps und Millionen Disparitätsevaluationen pro Sekunde.

Ansatz	Bild-größe	Fen-ster	Fre-quenz	Fps	MDE/ s
Reg.	450 x 375	5x5	176,6	1046	11479
Reg.	450 x 375	9x9	45,6	270	2964
Reg.	450x 375	9x5	126,1	747	8196
Baum	450 x 375	5x5	140,7	833	9145
Baum	450 x 375	9x9	45,6	270	2964
Baum	640 x 480	9x5	125,1	407	8131
Baum	1280x 1024	9x5	119,2	90	7748
[4]	640 x 480	9x9	n.a.	45	829
[10]	640 x 480	11x11	93	230	4521
[11]	640 x 480	13x13	155	30	589
[12]	640 x 480	n.a.	50,5	57	4505

V. FAZIT

In dieser Arbeit wird ein Algorithmus und eine Architektur für die Berechnung der Echtzeit-Stereo-Korrespondenz vorgestellt. Um die inhärente Komplexität dieser Aufgabe zu reduzieren, wird ein zweistufiger Ansatz vorgeschlagen. Dadurch werden vergleichbare Qualitätsergebnisse erreicht, aber erst kürzlich publizierte Architekturen hinsichtlich Datendurchsatz und Ressourcenbelegung übertroffen. Um den Ansatz weiter zu verbessern, kann eine adaptiv gewichtete Kostenfunktion in der Korrespondenzberechnung genutzt werden, wie in anderen Veröffentlichungen vorgeschlagen wird.

DANKSAGUNG

Diese Arbeit entstand im Zentrum für Angewandte Forschung (ZAFH) MERSES (Modellgestützte Entwurfs- und Realisierungsmuster für Signalverarbeitende Eingebettete Systeme). Es wird gefördert durch die Europäische Union, Europäischer Fonds für regionale Entwicklung (EFRE) und das Land Baden-Württemberg, Ministerium für Wissenschaft, Forschung und Kunst.

Tabelle 3: XST Synthese-Ergebnisse: Ressourcenbelegung der FPGA-Slices und Block-Ram-Blöcke.

Ansatz	Bild-größe	Fenster	Slices	RAM
Reg.	450 x 375	5x5	7804 (38%)	8 (2,8%)
Reg.	450 x 375	9x9	14897 (73%)	16 (5,4%)
Reg.	450x 375	9x5	10234 (50%)	8 (2,8%)
Baum	450 x 375	5x5	7976 (39%)	8 (2,8%)
Baum	450 x 375	9x9	14729 (72%)	16 (5,4%)
Baum	640 x 480	9x5	10347 (50%)	8 (2,8%)
Baum	1280x1 024	9x5	10266 (50%)	8 (2,8%)
[4]	640 x 480	9x9	102439 (86%)	32 (3,7%)
[10]	640 x 480	11x11	47156 (53%)	258 (77%)
[12]	640 x 480	n.a.	35715 (84%)	99 (26%)

LITERATURVERZEICHNIS

[1] D. Scharstein and R. Szeliski, "High-accuracy stereo depth maps using structured light," *IEEE Conf. on Computer Vision and Pattern Recognition* (CVPR 2003), vol. 1, pp. 195–202, 2003.

[2] C. Shi et. al., "High-accuracy stereo matching based on adaptive ground control points," Submitted to *IEEE Trans. on Image Processing*, 2012.

[3] B. Tippetts, D. J. Lee, K. Lillywhite, and J. Archibald, "Review of stereo vision algorithms and their suitability for resource-limited systems," *Journal of Real-Time Image Processing*, 2013.

[4] S. Perri, P. Corsonello, and G. Cocorullo, "Adaptive Census transform: a novel hardware-oriented stereovision algorithm," *Computer Vision and Image Understanding*, vol. 117, no. 1, pp. 29–41, 2013.

[5] F. Schumacher and T. Greiner, "Extension and FPGA architecture of the generalized Hough transform for real-time stereo correspondence," *Conf. on Design and Architectures for Signal and Image Processing* (DASIP 2013), 2013.

[6] R. Zabih and J. Woodfill, "Non-parametric local transforms for computing visual correspondence," in *Computer Vision ECCV* '94, ser. Lecture Notes in Computer Science, J.-O. Eklundh, Ed. Springer Berlin Heidelberg, 1994, vol. 801, pp. 151–158.

[7] D. G. Bailey, Design for embedded image processing on FPGAs. New York NY: Wiley, 2011.

[8] M. Holzer et. al., "Optimized hardware architecture of a smart camera with novel cyclic image line storage structures for morphological raster scan image processing," *IEEE Int'l. Conf. on Emerging Signal Processing Applications* (ESPA), pp. 83–86, 2012.

[9] G. Szedo, "Two-dimensional rank order filter," Xilinx Application Note XAPP953, 2006.

[10] S. Jin et.al., "FPGA design and implementation of a real-time stereo vision system," *IEEE Trans. on Circuits and Systems for Video Technology*, vol. 20, no. 1, pp. 15–26, 2010.

[11] C. Ttofis and T. Theocharides, "Towards accurate hardware stereo correspondence: a real-time FPGA implementation of a segmentation-based adaptive support weight algorithm," *Design, Automation Test in Europe Conference Exhibition* (DATE), 2012.

[12] M. Tomasi et. al., "Real-time architecture for a robust multi-scale stereo engine on FPGA," *IEEE Trans. on VLSI Systems*, vol. 20, no. 12, pp. 2208–2219, 2012.

Frank Schumacher studierte an der Hochschule Pforzheim Elektrotechnik und erhielt 2006 den Abschluss Diplom-Ingenieur. In 2008 folgte ein Masterabschluss in Information Systems. Aktuell forscht er im Projekt Merses an der Hochschule Pforzheim am modellgestützten Entwurf und der Realisierung signalverarbeitender eingebetteter Systeme. Seine Forschungsinteressen sind Hardwareentwurf, signalverarbeitende Systeme und Bildverarbeitung.

Prof. Dr. Thomas Greiner ist Sprecher des Zentrums für Angewandte Forschung MERSES und einer der Sprecher des in Zusammenarbeit mit der Universität Tübingen angebotenen Promotionskollegs „Entwurf und Architektur Eingebetteter Systeme", Hauptarbeitsgebiete: System- und Softwaremodellierung, Entwurf und Architektur signalverarbeitender eingebetteter Systeme.

HOCHSCHULE KONSTANZ
TECHNIK, WIRTSCHAFT UND GESTALTUNG

Entwicklung eines FPGA-basierten KNX-Controllers

Torge Bonert, Gregor Burmberger

***Zusammenfassung*—Der KNX ist ein Feldbus im Bereich der Gebäudeautomation. Um KNX-Komponenten mit dem Bus zu verbinden, gibt es auf dem Markt zahlreiche Produkte, die alle einen Nachteil haben: Sie arbeiten meist nur auf dem Physical Layer und geben Aufgaben für die Kommunikationssteuerung an den Host ab. In diesem Paper wird der Ansatz einer Implementierung eines KNX-Controllers auf einem FPGA vorgestellt, um die Nachteile der erhältlichen Komponenten zu umgehen und einen Ansatzpunkt für die Entwicklung eines eigenständigen Controllers zu haben.**

Schlüsselwörter—KNX, KNX-Controller, FPGA.

I. Einleitung

Im Bereich der Gebäudeautomation können durch intelligente Gebäudesysteme Eigenschaften wie Energieeffizienz, Komfort, Sicherheit und Nutzbarkeit verbessert werden. Dazu können in bestehende Gebäude bzw. Neubauten Sensoren wie z.B. Verbrauchszähler oder Bewegungsmelder und Aktoren wie Relais oder Antriebe für Motoren installiert und miteinander vernetzt werden. Für die Vernetzung von Sensoren und Aktoren ist der KNX de-facto der Standard.

Beim Einsatz von KNX-Systemen im Feld zeigt sich, dass die erhältlichen Komponenten zur Anbindung der Systeme an den KNX im Vergleich zu anderen Bussystemen wenig leisten. So übernehmen diese nur die physikalische Anbindung. Für die Steuerung der Kommunikation muss ein zusätzlicher Mikrocontroller mit entsprechender Kommunikationssoftware bereitgestellt werden. Hier zeigt sich, dass die erhältlichen Produkte die meisten Aufgaben zur Kommunikationssteuerung an den angeschlossenen Host (Mikrocontroller oder PC) abgeben und selbst wenig dazu beitragen.

Eine Komponente zur Anbindung an einen Bus soll aber so wenig Aufgaben wie möglich an den Host abgeben und die Steuerung der Kommunikation selbstständig übernehmen, um den Host nicht zusätzlich zu belasten. Dieser soll lediglich die ihm zugeteilten Aufgaben ausführen und Daten verarbeiten bzw. weitergeben.

Im Rahmen einer Projektarbeit an der HTWG Konstanz wird ein Ansatz für die Entwicklung eines FPGA-basierten KNX-Controllers implementiert, der die Nachteile der erhältlichen Komponenten umgeht und als Grundlage für weitere Entwicklungen dient. Das Projekt baut auf [1] auf.

Im Folgenden werden die Ergebnisse der Arbeit präsentiert. Dazu wird zunächst auf den KNX sowie den TP-UART, ein Interface zur Anbindung von Systemen an den KNX, eingegangen. Anschließend folgt die Spezifikation des Controllers bevor auf die Implementierung eingegangen wird. Abschließend folgt eine Zusammenfassung der Arbeit.

II. KNX

Der KNX ist ein Bussystem im Bereich der Gebäudeautomation, der von der KNX-Association im Standard EN 50090 bzw. ISO/IEC 14543-3 spezifiziert ist. Der KNX ist dezentral aufgebaut. Die Intelligenz ist dabei auf alle angeschlossenen Komponenten verteilt und es können über 10000 Komponenten in einem System enthalten sein. Als Übertragungsmedien können Twisted Pair, Power Line, Funk oder Ethernet eingesetzt werden. Die in diesem Paper gezeigten Verfahren und Implementierungen beziehen sich auf die Übertragung mittels Twisted Pair.

A. Physical Layer

Die Datenübertragung erfolgt byteweise mit 9600 Baud und ist asynchron. Der Idle-Pegel ist logisch 1 und liegt bei einer Spannung von 21 bis 32 V_{DC}. Um eine logische 0 zu senden, muss der Sender den Spannungspegel für 35 µs auf eine Spannung von 3 bis 9 V_{DC} bringen. Eine Drossel in der Spannungsversorgung des KNX sorgt dabei für das Einbrechen der Spannung. Nach 35 µs erfolgt ein von der Drossel verursachter Überschwinger. Es fließt bei der Übertragung einer logischen 0 ein der Versorgungsgleichspannung überlagerter Wechselstrom, während beim Senden einer logischen 1 kein Strom fließt, siehe Abbildung 1.

Die Übertragung der Daten beginnt mit einem Startbit. Danach folgen acht Datenbits, ein Paritybit sowie

T. Bonert, torge.bonert@htwg-konstanz.de, ist Student an der HTWG Konstanz, G. Burmberger, gregor.burmberger@htwg-konstanz.de, ist Mitglied der HTWG Konstanz, Brauneggerstr. 55, 78462 Konstanz.

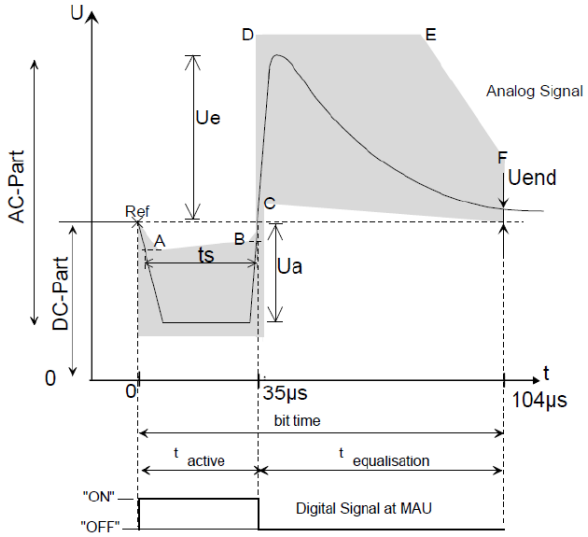

Abbildung 1: "0" - Bit Frame [2].

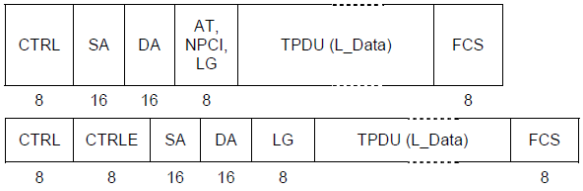

Abbildung 2: L-Data-Standard-Frame (oben) und L-Data-Extended-Frame (unten) [2].

ein Stoppbit. Die Übertragung eines weiteren Datenbytes erfolgt nach zwei Bitzeiten Leerlauf auf dem Bus.

B. Data Link Layer

Im KNX-Standard sind vier verschiedene Datenframes des Data Link Layers spezifiziert. Zur Übertragung von Daten existieren der L-Data-Standard- und der L-Data-Extended-Frame (Abbildung 2). Diese beiden unterscheiden sich in der Anzahl der übertragbaren Nutzdaten. Während der L-Data-Standard-Frame 14 Bytes an Nutzdaten übertragen kann, können mit dem L-Data-Extended-Frame bis zu 255 Byte Nutzdaten übertragen werden. Um den Empfang der beiden Frametypen zu quittieren bzw. abzulehnen, ist der Acknowledgement-Frame spezifiziert, der aus einem Byte aufgebaut ist und verschiedene Statusmeldungen codiert.

Darüber hinaus ist im KNX-Standard die sog. Poll-State Kommunikation mit dem Poll-Data-Request-Frame spezifiziert. Mit diesem Frame können von mehreren Komponenten Informationen abgefragt werden. Die adressierten Komponenten antworten daraufhin mit jeweils einem Byte. Im Vergleich zu dem in Abbildung 2 gezeigten L-Data-Standard-Frame folgt beim Poll-Data-Request-Frame nach der Zieladresse (DA in Abbildung 2) die Anzahl der erwarteten Daten und eine Frame Check Sequence (FCS in Abbildung 2).

C. Hardware Interface / TP-UART

Zur Anbindung von Komponenten an den KNX gibt es auf dem Markt zahlreiche Produkte, die alle meist nur auf dem Physical Layer arbeiten. Die dabei am häufigsten eingesetzte Komponente ist der TP-UART der Firma Siemens. Dieser stellt die Verbindung zwischen einem Host und dem KNX her. Als Host kann dabei ein Mikrocontroller in einem KNX-System oder ein PC in einer Laborumgebung eingesetzt werden.

Der TP-UART dient zur Anbindung an den KNX und zur Decodierung bzw. Codierung der Bits. Er stellt über einen UART die Verbindung zwischen KNX und Host her und enthält das TP-UART Host-Protokoll, das für die Schnittstelle zum Host entwickelt ist und das meiste des KNX Protokollstacks bis zum Data Link Layer verarbeitet. Das TP-UART Host-Protokoll weist zwei zeitkritische Bedingungen auf, die für einen Mikrocontroller eher unproblematisch sind, eine PC-Anwendung aber vor Probleme stellt.

Eine zeitkritische Bedingung bezieht sich auf die Endeerkennung eines Frames. Dieses wird nicht vom TP-UART erkannt, sondern muss vom Host erkannt werden. Hierfür muss die Bitzeit nach dem Empfang eines Frames ausgewertet werden. Die Auswertung von Zeitintervallen stellt für Mikrocontroller kein Problem dar, da mit einem Timer eine genaue Zeitauswertung erfolgen kann. Wird als Host hingegen ein PC eingesetzt, bietet dieser nur den Zugriff auf den Datenstrom, ohne dabei jegliche Zeitinformation zu erhalten. Eine Auswertung des Zeitintervalls ist somit nicht möglich.

Beim Empfang eines Frames sendet der TP-UART die Bytes ohne Pufferung direkt an den Host. Werden die Daten von einem Mikrocontroller gepuffert, so geht auch hier jegliche Zeitinformation verloren und ein Mikrocontroller kann das Ende eines Frames nicht mehr erkennen [3].

Die zweite kritische Zeitbedingung betrifft das Senden der ACK-Information vom Host an den TP-UART. Der Host muss diese Information innerhalb einer gewissen Zeit nach Empfang der Zieladresse senden. Dabei muss er die Adresse selbst auswerten, da der TP-UART keinen Empfangsfilter für eingehende Frames hat. Die ACK-Information wird vom TP-UART benötigt, um den Empfang zu quittieren bzw. abzulehnen. Der TP-UART sendet die ACK-Information nur dann auf dem Bus weiter, wenn diese innerhalb einer bestimmten Zeit empfangen wurde [3]. Während diese Zeit im Datenblatt des TP-UART festgelegt ist, sagt das Datenblatt nichts darüber aus, was passiert, wenn die Zeit nicht eingehalten wird. Dadurch kann es passieren, dass falsche Quittierungen vom TP-UART gesendet werden [1]. Eine genauere

Untersuchung dieses zeitkritischen Problems erfolgte in [4]. Neben den zeitkritischen Bedingungen wird bei der Verwendung des TP-UARTS der Host teils unnötig belastet. So werden die vom TP-UART auf dem Bus gesendeten Daten gleichzeitig an den Host gesendet. Außerdem muss der Host die Berechnung der FCS übernehmen.

Zusammenfassend zeigt sich, dass der TP-UART die komplette physikalische Anbindung und teilweise Dienste von höheren Schichten übernimmt. Er bildet den Data Link Layer nicht komplett ab, weshalb diese Dienste vom Host mit übernommen werden müssen [1].

III. Spezifikation

Im Rahmen dieses Projekts wird ein Ansatz zur Implementierung eines KNX-Controllers auf einem FPGA gezeigt. Dabei ist zu berücksichtigen, dass der Controller die Steuerung der Kommunikation übernimmt, keine zeitkritischen Aufgaben an den Host abgibt und diesen so wenig wie möglich belastet. Dabei wird zwischen dem Senden von Daten auf dem Bus und dem Empfangen von Daten vom Bus unterschieden.

A. Senden

Beim Senden von Daten auf dem Bus soll der Host die zu sendenden Daten in den Controller laden, das Senden übernimmt anschließend der Controller. Dazu soll der Host dem Controller zuerst mitteilen, dass es sich um Daten handelt, die der Controller auf dem Bus senden soll. Dies erfolgt durch ein vorangestelltes Byte vor dem eigentlichen Frame (Abbildung 3). Das Byte dient als Kennung und ermöglicht die Unterscheidung zwischen L-Data-Standard-, L-Data-Extended- und Poll-Data-Request-Frame (Data in Abbildung 3) und Acknowledgement-Frame (ACK, NAK, BUSY in Abbildung 3). Nach dem Byte folgt dann der eigentliche Frame. Vor dem Senden der Daten auf dem Bus soll der Controller die FCS berechnen und an den Frame anhängen.

Das in Abbildung 3 spezifizierte Byte soll zusätzlich als Quittierung der vom Host empfangenen Daten dienen, welches der Host nach dem Erhalt von Daten an den Controller senden soll. Der Controller soll dieses auswerten und die entsprechende Meldung auf dem Bus senden.

Beim Senden ist zu beachten, dass das Timing der KNX-Spezifikation eingehalten wird. So erfolgt die Übertragung eines jeden weiteren Bytes nach einer Pause von zwei Bitzeiten. Je nach Priorität des Frames beginnt das Senden eines neuen Frames nach unterschiedlichen Bitzeiten Leerlauf. Bei hoher Priorität beginnt das Senden nach 50 Bitzeiten, bei niedriger nach 53 Bitzeiten.

Bei einer Nichtbestätigung oder einem Timeout können die Daten nochmals gesendet werden. Die

Octet 0								
label								
7	6	5	4	3	2	1	0	
0	0	0	0	1	0	1	1	Data
1	1	0	0	1	1	0	0	ACK
0	0	0	0	1	1	0	0	NAK
1	1	0	0	0	0	0	0	BUSY

Abbildung 3: Label Byte.

Octet 0								
state								
7	6	5	4	3	2	1	0	
1	0	0	0	0	0	0	1	Data
1	0	1	0	1	0	1	0	ACK
0	0	1	1	1	1	0	0	NAK
1	1	0	0	0	0	1	1	BUSY

Abbildung 4: State Byte.

Häufigkeit des nochmaligen Sendens kann frei spezifiziert werden. Wird ein Frame nochmals gesendet, so muss im Control Feld ein entsprechendes Bit gesetzt werden. Erfolgt beim x-ten Mal Senden eine Nichtbestätigung bzw. ein Timeout, so soll der Controller den entsprechenden Status dem Host mitteilen.

B. Empfangen

Beim Empfangen soll der Controller zwischen den unterschiedlichen Frametypen unterscheiden. Während beim Acknowledgement-Frame keine Adressauswertung erfolgen soll, soll der Controller beim L-Data-Standard-, L-Data-Extended- und Poll-Data-Request-Frame die Empfangsadresse auswerten. Entspricht diese der Adresse des Hosts, sind die Daten für diesen und der Controller soll die Daten weiter verarbeiten, ansonsten soll er sie verwerfen.

Der Controller soll während des Empfangens die FCS berechnen und diese mit der übertragenen FCS vergleichen. Stimmen diese überein, soll der Controller die Daten weiter verarbeiten, ansonsten soll er den Acknowledgement-Frame senden und die Daten verwerfen. Sind die Daten korrekt empfangen, soll der Controller den Host darüber informieren, dass neue Daten zur Verfügung stehen. Dazu soll der Controller ein Byte generieren (Abbildung 4) und an den Host als Ankündigung neuer Daten senden. Danach soll der Controller das Senden der Daten ohne die FCS beginnen.

Beim Empfang eines Acknowledgement-Frames soll der Controller diesen auswerten und je nach Status das Byte nach Abbildung 4 an den Host senden, um diesen über den Sendestatus zu informieren. NAK und BUSY soll der Controller erst nach der festgelegten Anzahl an Wiederholungsversuchen senden. Beim Empfang der Bytes als Antwort auf den Poll-Data-Request-Frame soll der Controller diese Anhand ihrer Response-Slot-Number auswerten.

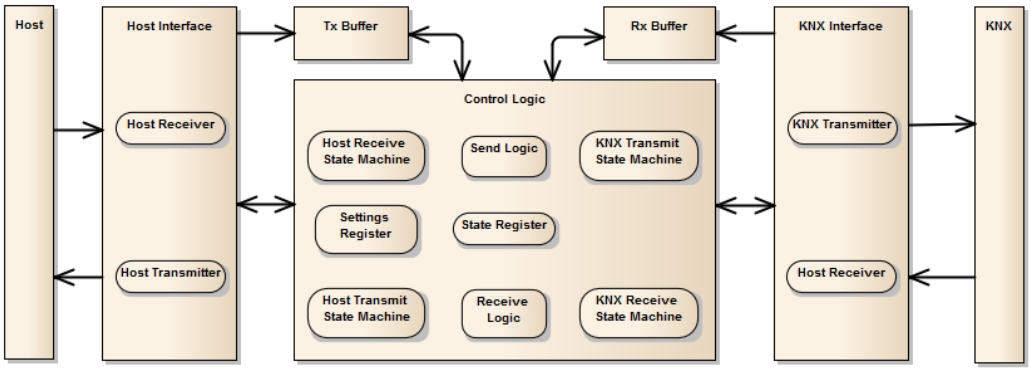

Abbildung 5: Architektur KNX-Controller.

C. Empfangs- bzw. Sendepuffer

Zum Speichern der zu sendenden bzw. der empfangenen Daten sollen ein Sende- und ein Empfangspuffer dienen. Beim Senden soll der Sendespeicher die Daten vom Host zuerst zwischenspeichern, bevor der Controller diese auf den Bus sendet. Erst wenn alle Daten komplett vom Host empfangen sind, soll der Controller mit dem Senden beginnen. Beim Empfang soll der Empfangsspeicher die Daten speichern, bevor er die Daten an den Host weiterleitet. Erst wenn der komplette Frame korrekt empfangen ist, soll der Controller die Daten weiterleiten.

Die Daten sollen in den beiden Speichern so lange zur Verfügung stehen, bis diese von neuen Daten überschrieben werden. Beim Sendespeicher soll dies erst erfolgen, wenn der Host das ACK-, NAK- oder BUSY-Byte nach Abbildung 4 empfangen hat, beim Empfangsspeicher erst, wenn der Controller das ACK, NAK- oder BUSY-Byte nach Abbildung 3 empfangen hat.

D. Schnittstellen

Für die Anbindung des Controllers an den Host und KNX soll dieser jeweils eine separate Schnittstelle haben. Bei der Schnittstelle zum KNX muss dessen Timing beachtet werden. Beide Schnittstellen sollen einen Sender und Empfänger haben.

IV. Implementierung

Basierend auf den spezifizierten Anforderungen baut die Architektur des Controllers auf. Sie lehnt sich an [1] an und ist in Abbildung 5 zu sehen. Für die Implementierung wird ein Cyclone II der Firma Altera eingesetzt.

A. KNX Interface

Das *KNX Interface* ist die Schnittstelle zum KNX. Es enthält ein Modul für die Übertragung von Daten auf dem Bus (*KNX Transmitter*) und ein Modul für den Empfang von Daten (*KNX Receiver*). Die Übertragung auf dem KNX erfolgt mit 9600 Baud und ähnelt dem UART. Aus diesem Grund basieren die beiden Module auf UART-Schnittstellen von [5].

Die beiden Module sind an den KNX angepasst. Beim Senden werden Start-, Parity- und Stoppbit an die Daten angefügt. Das Paritybit wird im Modul berechnet. Damit die Daten mit 9600 Baud gesendet werden, wird der Systemtakt intern geteilt. Zusätzlich wird nach dem Senden des Stoppbits der Leerlauf von zwei Bitzeiten eingefügt. Die zu sendenden Daten werden byteweise in einem Register abgelegt und bitweise ausgelesen.

Das Empfangsmodul empfängt die Daten als Bits und speichert diese in einem Register ab. Sobald ein Byte empfangen ist, wird dieses ausgegeben und das Register kann neu beschrieben werden. Das Empfangsmodul wertet zusätzlich die Leerlaufzeit auf dem KNX mit internen Zählern aus, die für das Senden der Frames mit unterschiedlichen Prioritäten und der Auswertung des Acknowledgement-Frames benötigt werden.

B. Host Interface

Das *Host Interface* ist die Schnittstelle zum Host. Über dieses werden die Daten vom Host empfangen und für den Controller aufbereitet, et vice versa. Es enthält ein Modul für das Empfangen (*Host Receiver*) und ein Modul für das Senden (*Host Transmitter*) von Daten. Dieses Modul basiert ebenfalls auf den UART-Schnittstellen von [5] und arbeitet wie das KNX Interface, nur ohne dessen Anpassungen an den KNX.

C. TX Buffer

Der *TX Buffer* ist der Sendespeicher des Controllers. In ihm werden die Daten vom Host gespeichert, bevor sie auf den Bus gesendet werden. Der Speicher ist als Block RAM ausgelegt. Die Größe des RAMs ist auf die Größe des L-Data-Extended-Frames ausgelegt, da dieser der maximalen Größe eines Frames entspricht.

Beim Speichern der Daten werden diese byteweise abgelegt. Erst wenn der Frame komplett vom Controller empfangen ist, werden die Daten aus dem Speicher ausgelesen. Dazu wird die Speicherstelle solange

inkrementiert, bis das Ende der gespeicherten Daten erreicht ist. Die Steuerung des Speichers übernimmt die *Control Logic*. Die Daten bleiben solange erhalten, bis neue Daten vom Host empfangen werden.

D. RX Buffer

Der *RX Buffer* ist der Empfangsspeicher des Controllers. In ihm werden die vom Bus empfangenen Daten zuerst zwischengespeichert, bevor sie weitergeleitet werden. Wie beim *TX Buffer* handelt es sich um einen Block RAM. Er weist die gleiche Größe auf und arbeitet identisch. Der Steuerung übernimmt ebenfalls die Control Logic.

E. Control Logic

Die Control Logic ist das zentrale Element des Controllers. Das Modul enthält die Logik des Controllers. Für die Steuerung des Controllers sind vier Zustandsmaschinen implementiert. Somit ist es möglich, Daten vom Host an den Controller zu senden und gleichzeitig Daten vom KNX zu empfangen, und umgekehrt.

Die *Host Receive State Machine* (Abbildung 6) steuert den Empfang der vom Host übertragenen Daten. Dabei wertet die State Machine das vom Host gesendete Byte (Abbildung 3) aus und signalisiert im Folgezustand (*START_DATA_TRANSMISSION*) den Start der Übertragung. Es wird gewartet, bis alle Daten vom Host empfangen sind, bevor in den Ausgangszustand zurückgekehrt wird (IDLE). Mit dem Start der Übertragung werden die Daten gleichzeitig in den Empfangsspeicher geschrieben.

Sind die Daten empfangen, folgt das Senden der Daten auf dem KNX. Dies übernimmt die *KNX Transmit State Machine* (Abbildung 7). Dabei werden die im KNX Interface implementierten Zähler vor dem Senden ausgewertet und je nach Priorität des zu sendenden Frames in den jeweiligen Zustand (*HIGH_PRIO* bzw. *LOW_PRIO*) gewechselt. Nach der Auswertung beginnt das eigentliche Senden der Daten.

Dazu wird zuerst das Senden eines Bytes signalisiert (*SEND_DATA*), danach das entsprechende Byte aus dem Sendespeicher geholt (*GET_BYTE*) und gewartet (*WAIT_FOR_SENDING*) bis das Byte vom *KNX Interface* verarbeitet wird. Dann wird wiederum signalisiert, dass ein Byte folgt … usw.. Erst wenn der Frame komplett aus dem Speicher geholt ist, ist dieser übertragen und es wird in den Zustand *FRAME_COMPLETE* gewechselt. In diesem Zustand ist der gesamte Frame auf dem Bus gesendet. Da jetzt die Reaktion bzw. der Acknowledgement-Frame der adressierten Teilnehmer ausgewertet werden muss und der Controller daraufhin entsprechend reagieren soll, sind in der Zustandsmaschine die dafür entsprechenden Zustände angelegt.

Im Zustand *NAK* und *BUSY* wird das Senden des Frames wie spezifiziert wiederholt. Da hier unterschiedliche Leerlaufzeiten zu berücksichtigen sind, wird in unterschiedliche Zustände gewechselt. Wird

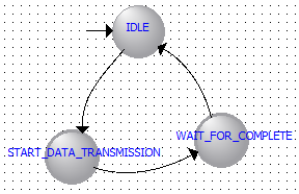

Abbildung 6: Host Receive State Machine.

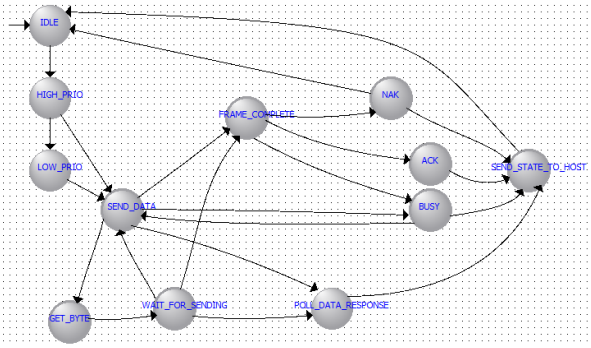

Abbildung 7: KNX Transmit State Machine.

auch beim x-ten Mal keine Bestätigung erhalten, wird der Sendestatus an den Host gesendet (*SEND_STATE_TO_HOST*). Beim Erhalt einer Bestätigung wird sofort in diesen Zustand gewechselt (*ACK*). Handelt es sich bei den zu sendenden Daten um einen Poll-Data-Request-Frame, wird nach dem Senden in den Zustand *POLL_DATA_RESPONSE* gewechselt. In diesem Zustand soll solange verblieben werden, bis alle angeforderten Informationen empfangen sind.

Die *KNX Receive State Machine* (Abbildung 8) steuert den Empfang der auf dem KNX empfangenen Daten. Sobald Daten zur Verfügung stehen, wird der Controller darüber informiert (*START_TRANSMISSION*) und die Daten werden zur Auswertung weitergeleitet. Gleichzeitig werden die Daten in den Empfangsspeicher geschrieben. Auch hier wird gewartet, bis alle Daten empfangen sind (*WAIT_FOR_COMPLETE*), bevor zurück in den Ausgangszustand (*IDLE*) gewechselt wird.

Das Senden der Daten an den Host steuert die *Host Transmit State Machine* (Abbildung 9). Die State Machine unterscheidet zwischen dem Senden des Sendestatus (*SEND_TRANSMISSION_STATE*) und dem Senden von Daten (*SEND_DATA*). Das Senden funktioniert wie bei der Bus Transmit State Machine, in dem zuerst das Senden eines Bytes signalisiert wird (*SEND_DATA*), danach das Byte aus dem Speicher geholt (*GET_BYTE*) und gewartet wird, bis es gesendet ist (*WAIT_FOR_SENDING*). Dies erfolgt solange, bis alle Daten gesendet sind. Danach wechselt die State Machine in den Ausgangszustand (*IDLE*).

Die Zustände der vier State Machines hängen teilweise voneinander ab, weshalb sie untereinander verbunden sein müssen. Es hat sich gezeigt, dass hierfür

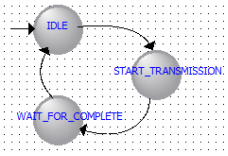

Abbildung 8: KNX Receive State Machine.

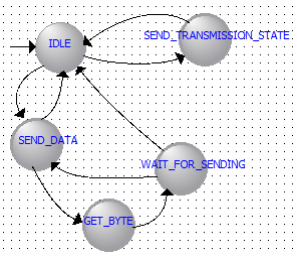

Abbildung 9: Host Transmit State Machine.

Tabelle 1: Ressourcenbedarf.

Modul	LEs	Register	Memory Byte
Host Interface	123	86	
KNX Interface	251	171	
TX Buffer	16	27	321
RX Buffer	16	26	321
Control Logic	576	238	
Benötigte HW	1034 (6%)	548 (3%)	642 (2%)
Cyclone II	18 752	18 752	239 616

der Ansatz eines zusätzlichen Registers vorteilhafter ist, als die State Machines direkt miteinander zu verbinden. Mit dem zusätzlichen Register sind die voneinander abhängigen Zustände in einem gemeinsamen Register implementiert. Das hierfür verantwortliche Modul ist das *State Register*. In diesem werden Zustände der State Machines ausgewertet und entsprechende Ausgänge gesetzt.

Um Daten auf dem Bus zu senden, im Sendespeicher abzulegen und zu verarbeiten, ist neben der dafür verantwortlichen State Machine die *Send Logic* implementiert. Dieses Modul übernimmt die Aufgaben, die von den State Machines nicht übernommen werden können. Mit dem Modul werden die vom Host gesendeten Daten ausgewertet, gleichzeitig in den Sendespeicher abgelegt und nach dem Empfang aller gesendet. Es wertet das Control-Feld aus und unterscheidet dabei zwischen den unterschiedlichen Frametypen. Daneben erkennt das Modul das Ende der gesendeten Daten, indem die Bytes der gesendeten Daten mitgezählt werden. Das Ende wird signalisiert, damit das Senden beginnen kann. Dabei werden die Bytes nacheinander aus dem Sendespeicher geholt, in dem die jeweilige Speicherstelle bis zum Ende der Daten inkrementiert wird. Beim Senden wird die FCS mit einer XOR-Verknüpfung berechnet und an das Ende des Frames angehängt.

Um Daten vom Bus zu empfangen, im Empfangsspeicher abzulegen und zu verarbeiten ist neben der dafür verantwortlichen State Machine die *Receive Logic* implementiert. Da der Empfang von Daten vom Bus dem Empfang von Daten vom Host ähnelt, ähnelt die Logik dieses Moduls der der Send Logic. Das Modul wertet das Control-Feld aus, um zwischen den Frametypen zu unterscheiden. Zusätzlich wertet das Modul die Empfangsadresse aus und vergleicht diese mit der im *Settings Register* abgelegten. Stimmen diese überein, wird die Verarbeitung fortgesetzt, ansonsten nicht. Zusätzlich wird während des Empfangens die FCS mit einer XOR-Verknüpfung berechnet und mit der gesendeten FCS verglichen. Stimmen diese überein, wird die Verarbeitung fortgesetzt, ansonsten nicht. Das Ende eines Frames wird erkannt, indem die empfangenen Bytes mitgezählt werden. Sind die Daten korrekt empfangen, werden diese durch Inkrementieren der Speicherstelle aus dem Empfangsspeicher geholt und an den Host gesendet.

Um Einstellungen am Controller vorzunehmen bzw. zu speichern, ist dafür eigens das Settings Register implementiert. Über dieses sollen Einstellungen wie z.B. die Adresse des Hosts oder die Anzahl an Sendewiederholungen vorgenommen werden.

V. Zusammenfassung

Im hier gezeigten Ansatz war es das Ziel, die Nachteile der erhältlichen Komponenten, insbesondere des TP-UART, zu umgehen und eine Basis für weitere Entwicklungen zu schaffen. Mit der Spezifikation des Controllers wurde die Basis für diese Implementierung geschaffen. Zusätzlich soll sie als Ansatz für weitere Entwicklungen dienen. Im implementierten Controller sind noch nicht alle Punkte der Spezifikation umgesetzt. Trotzdem kann gezeigt werden, dass der Host von den zeitkritischen Aufgaben befreit werden kann und diese komplett vom Controller übernommen werden können. Die Nachteile der bisher erhältlichen Komponenten zur Anbindung der KNX Systeme an den KNX sind somit umgangen.

Der Ressourcenbedarf in Tabelle 1 zeigt, dass für weitere Entwicklungen der Einsatz eines FPGAs in Betracht gezogen werden sollte, da die internen RAM-Blöcke ideal als Speicher genutzt werden können. Zusätzlich zeigt sie, dass ein FPGA mit geringeren Ressourcen eingesetzt werden kann.

LITERATURVERZEICHNIS

[1] P. Roth: „Entwicklung eines KNX-Controllers", *HTWG Konstanz*, 2012.

[2] KNX Association: „03_02_02 Communication Medium TP 1 v1.1 AS", *KNX Association*, 2008.

[3] A. Fernbach: „Enhanced TP-UART interface board", *Automation Systems Group – Institute of Computer Aided Automation Vienna University of Technology*, 2009.

[4] G. Neugschwandtner, A. Fernbach: „Design of an enhanced TP-UART based KNX PC interface", *Automation Systems Group – Institute of Computer Aided Automation Vienna University of Technology*, 2008.

[5] http://opencores.org, Datum des Zugriffs 01.10.2013.

Torge Bonert erhielt den akademischen Grad des B. Eng. in Elektro- und Informationstechnik im Jahr 2012 von der Hochschule Pforzheim. Er studiert im 3. Semester des Masterstudiengangs Elektrische Systeme an der HTWG Konstanz, den er voraussichtlich im März 2014 abschließen wird.

Gregor Burmberger ist Professor für Eingebettete Systeme und Prozessorarchitekturen an der HTWG Konstanz. Vor seiner Berufung war er als Experte u.a. auf dem Gebiet der Bussysteme für namhafte Automobilhersteller tätig. Mit seiner Erfahrung in Eingebetteten Systemen, FPGAs und SOPCs unterstützt er Bachelor- und Masterarbeiten im Bereich Digitale Systeme. Gregor Burmberger hat an der TU München Elektrotechnik studiert und im Bereich Realzeitsysteme promoviert.

HOCHSCHULE KONSTANZ
TECHNIK, WIRTSCHAFT UND GESTALTUNG

Synthese eines ΔΣ-AD-Umsetzers auf einem Low-Power-FPGA

Martin Miller, Andreas Wilhelm, Irenäus Schoppa

***Zusammenfassung*—Das hier beschriebene und auf einem Low-Power-FPGA aus der IGLOOnano-Familie realisierte System ist ein Digital-Analog-Umsetzer, der nach dem Prinzip der Delta-Sigma-Modulation arbeitet und eine Auflösung von 10 Bit mit einer Bandbreite von 2,7 kHz bei einer Abtastfrequenz von 1,8432 MHz erreicht. Die dafür notwendige digitale Signalverarbeitung erfolgt ausschließlich im FPGA und besteht aus einem Dezimierungsfilter auf der Grundlage der SINC-Funktion fünfter Ordnung sowie aus einem Kompensationsfilter mit vierzehn Koeffizienten. Das System wurde mit Hilfe der Histogramm-Methode statistisch untersucht und daraus der differentielle und der integrale Nichtlinearitätsfehler ermittelt.**

***Schlüsselwörter*—Low-Power-FPGA, IGLOOnano, Analog-Digital-Umsetzer, Delta-Sigma-Modulation, Dezimierungsfilter, Kompensationsfilter.**

I. Einleitung

Die Umsetzung analoger Signale in digitale Werte spielt in der digitalen Messwert- und Signalverarbeitung moderner eingebetteter Systeme eine äußerst wichtige Rolle. Solche Umsetzung kann ein Entwickler heute mit einem diskreten AD-Umsetzer oder mit einem, der bereits in einem Mikrocontroller oder auch in einem FPGA fest integriert ist, vornehmen. Es gibt mehrere Verfahren zur Umsetzung analoger Signale in digitale Werte. Ein interessantes Verfahren, das sich besonders gut für eine digitale Realisierung eignet, ist die Umsetzung analoger Signale in digitale Werte nach dem Delta-Sigma-Prinzip.

Die nach diesem Prinzip arbeitenden Analog-Digital-Umsetzer (ΔΣ-AD-Umsetzer) sind schon seit vielen Jahren bekannt und in zahlreichen Literaturquellen, z.B. [1]-[4] detailliert beschrieben. Häufig kommen ΔΣ-AD-Umsetzer als diskrete Bausteine zur Anwendung, insbesondere in Audio-Anwendungen. In Systemen mit geringer Bandbreite (≤ 10 kHz) und reduzierter Auflösung (≤ 12 Bit) lassen sich ΔΣ-AD-Umsetzer oft kostensparrend mit Mikrocontrollern realisieren, wie in [5]-[8] berichtet wird.

M. Miller, mamiller@htwg-konstanz.de, und A. Wilhelm, anwilhel @htwg-konstanz.de, sind Studenten an der HTWG Konstanz, I. Schoppa, ischoppa@htwg-konstanz.de, ist Mitglied der HTWG Konstanz, Brauneggerstr. 55, 78462 Konstanz.

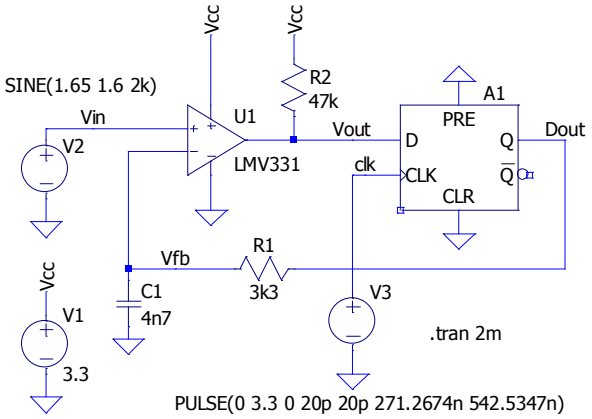

Abbildung 1: LTSpice-Modell der analogen Eingangsstufe mit einem synchronen D-Flipflop.

Durch den Einsatz programmierbarer Logikbausteine eröffnen sich dem Anwender neue Möglichkeiten, ΔΣ-AD-Umsetzer auch mit höheren Bandbreiten (> 10 kHz) und größeren Auflösungen (> 12 Bit) zu verwirklichen. Insbesondere in Verbindung mit FPGAs können ΔΣ-AD-Umsetzer effizient realisiert werden, wie in [9]-[13] beschrieben ist. Die gesamte komplexe, digitale Signalverarbeitung wird in einem FPGA implementiert. In der analogen Eingangsstufe, außerhalb des FPGAs, bleiben einige Widerstände und Kondensatoren sowie ein einfacher Komparator übrig.

Die hier beschriebene Realisierung ist für den Einsatz in einem HART-Modem vorgesehen, weshalb der ΔΣ-AD-Umsetzer folgende Parameter erreichen soll: Abtastfrequenz des Eingangssignals 1,8432 MHz, Auflösung 10 Bit, Bandbreite des Eingangssignals DC bis 3,6 kHz.

II. AD-Umsetzung nach Delta-Sigma-Prinzip

Abbildung 1 zeigt ein Modell der analogen Eingangsstufe des ΔΣ-AD-Umsetzers. Das Model wurde mit LTSpice erstellt und enthält neben dem SPICE-Modell des Low-Voltage-Komparators LMV331 auch ein einfaches, passives, auf dem Widerstand R_1 und dem Kondensator C_1 basierendes Tiefpassfilter in der

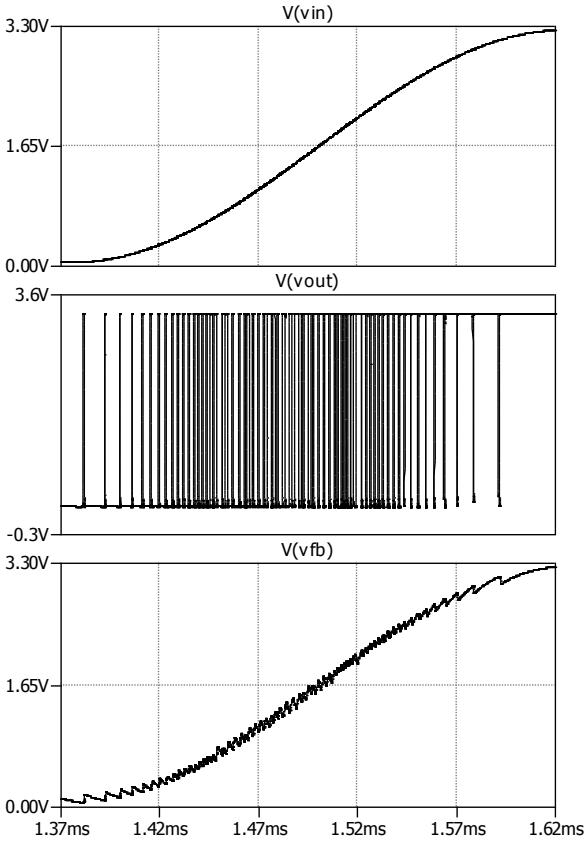

Abbildung 2: Signalverläufe an drei charakteristischen Punkten in der Schaltung aus Abbildung 1.

Signalrückführung, drei Spannungsquellen: V_1 als Generator der Versorgungsspannung +3,3 V, V_2 als Generator des Sinussignals und V_3 als Taktgenerator mit 1,8432 MHz, sowie das SPICE-Modell eines synchronen D-Flipflops, das den Übergang zwischen dem analogen und dem digitalen Bereich bildet. In Abbildung 2 sind drei Ausschnitte aus der analogen LTSpice-Simulation der Schaltung aus Abbildung 1 dargestellt. Sie zeigen Signalverläufe an drei charakteristischen Punkten in dieser Schaltung. Im oberen Diagramm ist der Verlauf der Eingangsspannung V_{in} vom minimalen bis zum maximalen Amplitudenwert zu sehen. Als Eingangsspannung dient eine Sinusspannung mit einer Amplitude von 1,6 V, einer Frequenz von 2,0 kHz und einem DC-Offset von 1,65 V. Im mittleren Diagramm ist der Verlauf der pulsdichte-modulieren Ausgangsspannung V_{out} am Ausgang des Komparators LMV331 dargestellt. Die Pulsdichte verändert sich hier von einer geringen Dichte bei niedriger Eingangsspannung bis zu hoher Dichte bei hoher Eingangsspannung. Im unteren Diagramm sieht man den Verlauf der Spannung V_{fb} in der Rückkopplung nach der Filterung durch das passive Tiefpassfilter aus R_1 und C_1. Man sieht hier auch, wie die Spannung am Kondensator C_1 auf- und entladen wird, und wie sie dem Verlauf der Eingangsspannung V_{in} folgt.

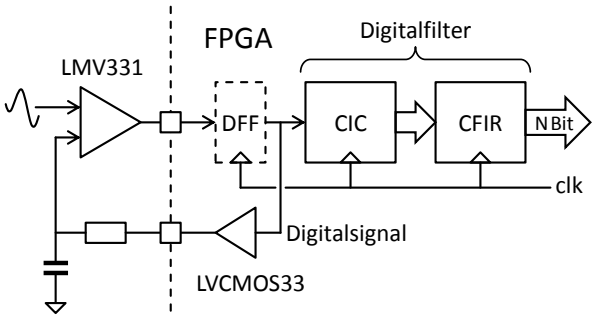

Abbildung 3: Blockschaltbild des ΔΣ-DA-Umsetzers mit einem Digitalfilter in zweistufiger Realisierung im FPGA.

Die Pulsdichte-modulierte Ausgangsspannung V_{out} wird in einem synchronen D-Flipflop erfasst und steht als digitaler 1-Bit-Datenstrom D_{out} für weitere Verarbeitungsschritte zur Verfügung. In Abbildung 3 ist die Blockstruktur des ΔΣ-AD-Umsetzers mit dem zweistufigen digitalen Filter im FPGA schematisch dargestellt. Die digitale Signalverarbeitung im FPGA erfolgt in zwei Filterblöcken: CIC- und CFIR-Filter.

A. CIC-Dezimierungsfilter

Ein CIC-Dezimierungsfilter (Cascaded-Integrator-Comb-Filter), das im weiteren Text nur als CIC-Filter bezeichnet wird, ist in der Literatur z.B. [14]-[17] auch unter dem Namen Hogenauer Filter bekannt. Das CIC-Filter hat eine Tiefpasscharakteristik und erfüllt zwei Aufgaben: 1) den seriellen 1-Bit-Datenstrom in einen parallelen N-Bit-Datenstrom umzusetzen und 2) die hohe Datenrate am Eingang auf eine niedrigere Datenrate am Ausgang zu reduzieren. Beide Aufgaben kann ein CIC-Filter besonders ressourcenschonend bewerkstelligen.

Die Struktur eines CIC-Filters, die als Blockschaltbild in Abbildung 4 zu sehen ist, besteht i.d.R. aus mehreren Integrationsstufen, einem Reduktionsglied und mehreren Differentiationsstufen. Diese Struktur zeichnet sich dadurch aus, dass sie ganz ohne Multiplikationen und ohne Koeffizienten-Speicher realisiert wird. Ihre Implementierung basiert ausschließlich auf einfachen arithmetischen Operationen wie Additionen und Subtraktionen. Beispiele von VHDL-Implementierungen eines dreistufigen CIC-Filters findet man z.B. in [16] oder [18].

Ein CIC-Filter zeichnet sich durch drei charakteristische Parameter aus: Anzahl der Stufen N, Dezimierungsrate R und Anzahl von Verzögerungen in einer Differentiationsstufe M. Diese Parameter lassen sich relativ leicht mit der Filter-Toolbox von Matlab berechnen, und zwar in Abhängigkeit von der vorgegebenen Abtastfrequenz, der geforderten Bandbreite und der erwarteten Auflösung, wie es bspw. In [17] und [19] beschrieben ist. Somit ergeben sich für das CIC-Filter die Werte $N = 5$, $R = 128$ und $M = 1$. Mit Matlab lässt sich auch die Registerbreite in den einzelnen Stufen ermitteln. In dem hier berechneten CIC-Filter

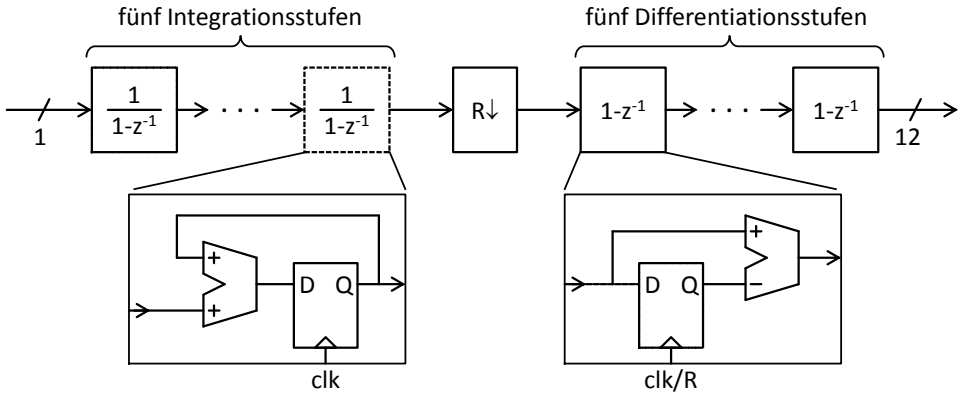

Abbildung 4: Struktur des CIC-Filters bestehend aus fünf Integrationsstufen und fünf Differentiationsstufen.

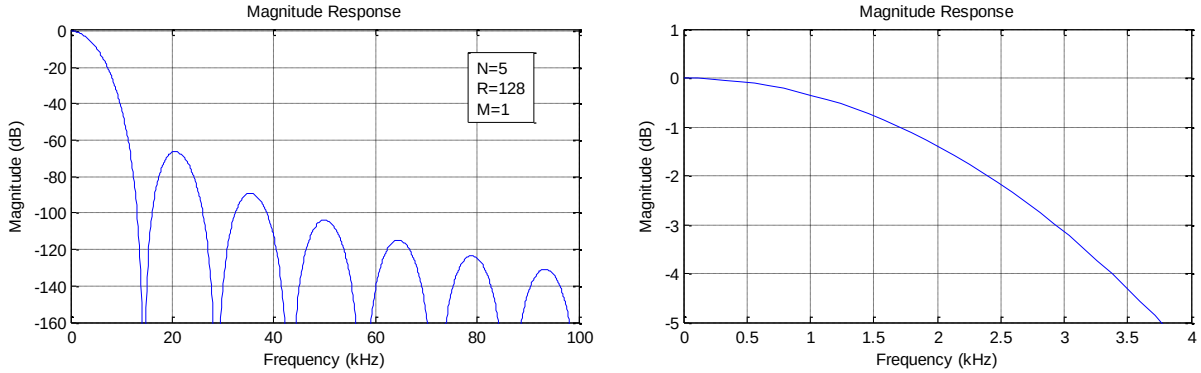

Abbildung 5: Betragsfrequenzgang des fünfstufigen CIC-Filters sowie ein Ausschnitt daraus im relevanten Nutzbereich.

ergeben sich dann die Registerbreiten {36, 36, 32, 26, 21, 28, 17, 16, 15, 15}.

Neben Werkzeugen zur Berechnung eines CIC-Filters stellt Matlab auch weitere Toolboxen und Werkzeuge zur Verfügung, u.a. solche, mit denen Filtercharakteristiken graphisch dargestellt werden können. Im linken Diagramm der Abbildung 5 ist der Betragsfrequenzgang des fünfstufigen CIC-Filters in einem Frequenzbereich von bis zu 100 kHz dargestellt. Man sieht dort, dass das CIC-Filter eine Dämpfung von 60 dB bereits bei der Frequenz von 21,6 kHz erreicht, was für die geforderte Auflösung notwendig ist. Im rechten Diagramm ist ein Ausschnitt aus demselben Betragsfrequenzgang zu sehen, allerdings in dem relevanten Frequenzbereich bis 4 kHz. Hier kann man erkennen, dass die Grenzfrequenz des CIC-Filters knapp unterhalb von 3 kHz liegt.

B. CFIR-Filter

Da das CIC-Filter eine starke Dämpfung im Durchlassbereich aufweist, ist es notwendig, diese Charakteristik zu korrigieren. Die Korrektur erfolgt mit einem CFIR-Filter (Compensation FIR Filter). Mit Matlab ist es möglich, Koeffizienten des CFIR-Filters passend zu dem eingesetzten CIC-Filter berechnen zu lassen. Nach der Dezimierung mit dem CIC-Filter stehen exakt 128 Takte zur Verfügung, die eine Zeitgrenze für die digitale Signalverarbeitung darstellen. Nun

Tabelle 1: Koeffizienten c0 bis c13 des CFIR-Filters

c0	0,013671875	c7	0,4619140625
c1	0,03515625	c8	0,166015625
c2	0,005859375	c9	-0,080078125
c3	-0,0859375	c10	-0,0859375
c4	-0,080078125	c11	0,005859375
c5	0,166015625	c12	0,03515625
c6	0,4619140625	c13	0,013671875

besteht die Herausforderung darin, ein CFIR-Filter mit möglichst minimaler Anzahl an Koeffizienten zu finden, so dass alle Berechnungen innerhalb einer Zeitspanne von 128 Takten abgeschlossen sind. Da Low-Power-FPGAs aus der IGLOOnano-Familie von Microsemi [20], [21] leider keine Hardware-Unterstützung für die digitale Signalverarbeitung in Form von speziellen DSP-Kernen oder in Hardware realisierten Multiplikationseinheiten beinhalten, müssen Multiplikationen mit sequentiellen Multiplizierern durchgeführt werden. Für das CFIR-Filter wurde die Anzahl der in der Synthese zur Verfügung stehenden sequentiellen Multiplizierer auf zwei beschränkt. Unter diesen Beschränkungen konnten vierzehn CFIR-Koeffizienten mit Hilfe von Matlab gefunden werden. In Tabelle 1 sind diese Koeffizienten zusammengefasst.

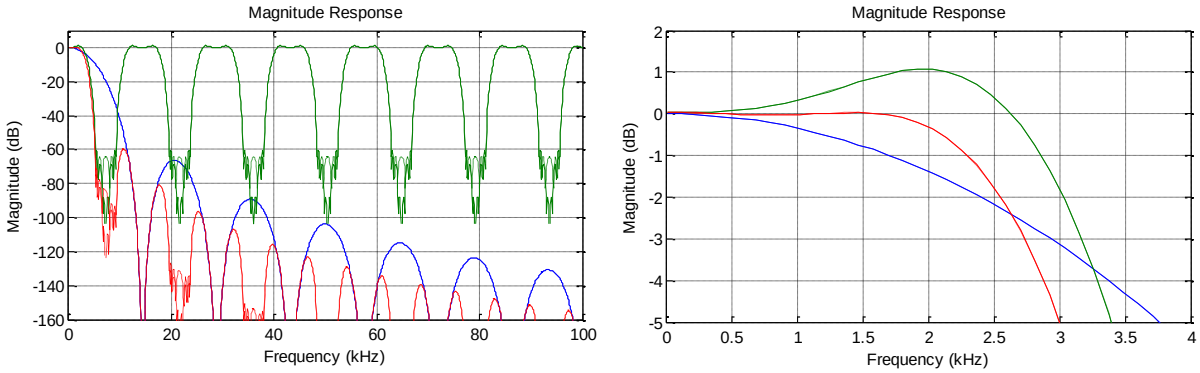

Abbildung 6: Betragsfrequenzgang des Digitalfilters in zweistufiger Realisierung mit dem CIC-/CFIR-Filters sowie ein Ausschnitt daraus im relevanten Nutzbereich.

Im linken Diagramm der Abbildung 6 sieht man an der roten Kurve den Betragsfrequenzgang des gesamten Digitalfilters, bestehend aus dem CIC- und dem CFIR-Filter. Die blaue Kurve stellt den Betragsfrequenzgang des CIC-Filters und die grüne Kurve den Betragsfrequenzgang des CFIR-Filters dar. Im rechten Diagramm der Abbildung 6 ist ein Ausschnitt aus demselben Betragsfrequenzgang zu sehen, allerdings in dem relevanten Frequenzbereich bis 4 kHz. Anhand der roten Kurve sieht man hier eine signifikante Verbesserung im Durchlassbereich, und zwar bis zu einer Frequenz von ca. 1,8 kHz. Allerdings hat sich die Grenzfrequenz des gesamten Filters etwas verschlechtert und liegt nun bei ca. 2,7 kHz.

III. Histogramm-Methode

Die Histogramm-Methode ist eine gängige Technik, vor allem statische Parameter eines AD- oder auch DA-Umsetzers mit statistischen Methoden zu ermitteln. Diese Methode ist in zahlreichen Literaturquellen umfassend beschrieben, weshalb hier auf eine detailliertere Beschreibung verzichtet und auf weiterführende Literatur z.B. [22] oder [23] verwiesen wird.

Die theoretische Häufigkeit $h(n)_T$ aller erfassten Werte wird mit der Formel

$$h(n)_T = \frac{M_T}{2^N - 2}$$

berechnet, wobei M_T die Summe aller Werte und N die Auflösung des AD-Umsetzers sind. Aus den ermittelten Häufigkeiten der einzelnen Werte $h(n)_A$ lassen sich der differentielle und der integrale Nichtlinearitätsfehler (*DNL* und *INL*) mit folgenden Formeln ausrechen:

$$DNL(n) = \frac{h(n)_A}{h(n)_T} - 1$$

$$INL(n) = \sum_{i=0}^{n} DNL(i)$$

Die Histogramm-Methode kam sowohl bei der Auswertung der Resultate aus der VHDL-Simulation als auch bei der Analyse von Resultaten aus Messungen an der realen Schaltung zum Einsatz.

IV. Evaluierungsumgebung

Die Entwicklungs- und Experimentierumgebung umfasste mehrere Softwarewerkzeuge, zwei Entwicklungsplatinen, einen Signalgenerator, ein Oszilloskop und einen Labor-PC.

Der Digitalteil des ΔΣ-AD-Umsetzers wurde in der Hardwarebeschreibungssprache VHDL modular, hierarchisch und synthesegerecht beschrieben und mit dem VHDL-Simulator ModelSim von Mentor Graphics umfassend simuliert. In der Simulationsumgebung wurde das Verhalten der analogen Schaltung aus Abbildung 1 nachgebildet, so dass das VHDL-Modell des ΔΣ-AD-Umsetzers mit der Histogramm-Methode evaluiert werden konnte. Aus der so gewonnenen statistischen Häufigkeitsverteilung der einzelnen Messwerte wurden die beiden Nichtlinearitätsfehler *DNL* und *INL* berechnet. Die Resultate sind in Abbildung 7 und in Abbildung 8 graphisch dargestellt. Wie man leicht erkennen kann, liegen beide Nichtlinearitätsfehler in einem äußerst engen Intervall von nur ± 0,2 LSB.

Basierend auf diesen vielversprechenden Resultaten wurde aus der VHDL-Beschreibung mit der integrierten Entwicklungsumgebung Libero IDE 9.1 von Microsemi eine fertige Schaltung für den Low-Power-FPGA vollständig synthetisiert. Als Basishardware diente das Entwicklungsboard IGLOO-nano Starter Kit von Microsemi mit einem AGLN250V2 [24]. Dieses Entwicklungsboard verfügt über mehrere Stiftleisten, über die man eine Zusatzplatine mit analogen Komponenten gemäß der Abbildung 1 auf dem Entwicklungsboard aufsetzen kann. Das Entwicklungsboard war über eine USB-Schnittstelle mit dem Labor-PC verbunden, auf dem Messwerte kontinuierlich erfasst und anschließend mit Excel ausgewertet wurden. Als Signalquelle für Messungen diente der Arbiträrsignal-/Funktionsgenerator AWG3390 von Keithley, der ein analoges Signal in Form einer symmetri-

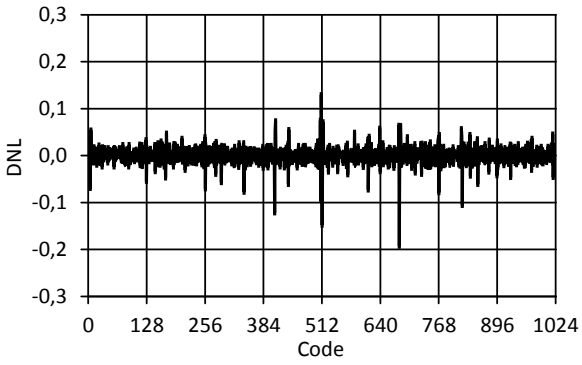

Abbildung 7: Der differentielle Nichtlinearitätsfehler DNL ermittelt während der VHDL-Simulation.

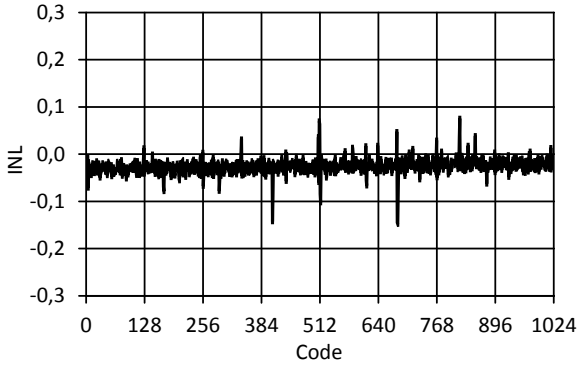

Abbildung 8: Der integrale Nichtlinearitätsfehler INL ermittelt während der VHDL-Simulation.

schen Rampe mit einer Amplitude von 1,7 V und einem DC-Offset von 1,65 V generiert.

Ein Messvorgang umfasste über eine Million einzelne Messwerte, aus denen die Häufigkeitsverteilung bestimmt wurde. Sie ist in Abbildung 9 graphisch dargestellt. Basierend auf der Häufigkeitsverteilung lassen sich z.B. mit Excel auch die beiden Nichtlinearitätsfehler *DNL* und *INL* berechnen. Der differenzielle Nichtlinearitätsfehler *DNL*, der in Abbildung 10 zu sehen ist, liegt in einem engen Intervall von ±0,6 LSB und erreicht somit ein gutes Ergebnis. Der integrale Nichtlinearitätsfehler *INL*, der in der Abbildung 11 zu sehen ist, weist eine signifikante Abweichung von den in der Simulation ermittelten Daten auf. Die Ursache dazu muss noch ermittelt werden, sie liegt möglicherweise in Nichtlinearitäten im Signalpfad.

V. Ausblick

Die Entwicklung des hier vorgestellten Systems ist noch nicht vollständig abgeschlossen. Durch neue, aus Messungen und Analyse gewonnene Erkenntnisse lassen sich einige, verbesserungswürdige Punkte angeben:

- Einsatz eines schnelleren Komparators,
- Erhöhung der Abtastfrequenz,
- Einsatz einer präziseren Referenzspannungsquelle im IO-Buffer des Low-Power-FPGAs,

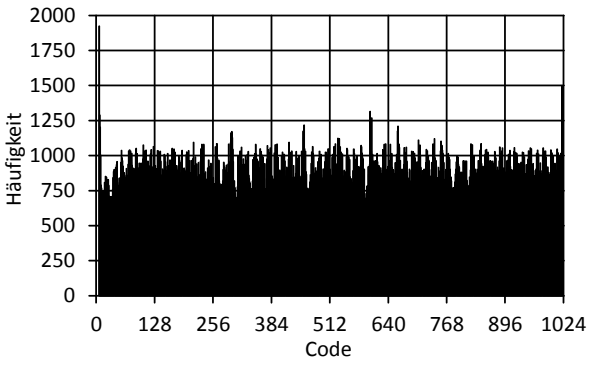

Abbildung 9: Häufigkeitsverteilung der einzelnen Messwerte.

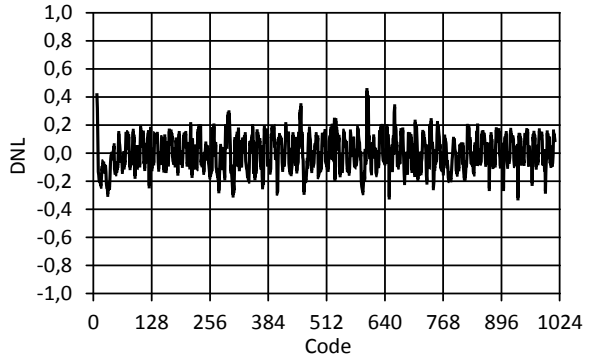

Abbildung 10: Der differentielle Nichtlinearitätsfehler DNL ermittelt anhand der Häufigkeitsverteilung aus Abbildung 9.

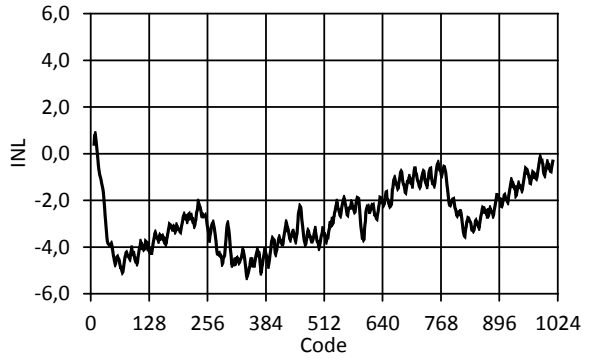

Abbildung 11: Der integrale Nichtlinearitätsfehler INL ermittelt anhand der Häufigkeitsverteilung aus Abbildung 9.

- Genauere Nachbildung der analogen Eingangsstufe im Rahmen der VHDL-Simulation,
- Reduzierung der verbrauchten Ressourcen im FPGA durch die digitale Signalverarbeitung,
- Optimierung der Koeffizienten im CFIR-Filter.

LITERATURVERZEICHNIS

[1] W. Kester: „ADC Architectures III, Sigma-Delta ADC Basics“, Tutorial MT-022, Rev. A, *Analog Devices, Inc*., 2009.

[2] W. Kester: „ADC Architectures IV: Sigma-Delta ADC Advanced Concepts and Applications“, Tutorial MT-023, Rev. A, *Analog Devices, Inc*., 2009.

[3] S. Park: „Principles of Sigma-Delta Modulation for Analog-to-Digital Converters“, Applikationsbericht APR8/D Rev.1, *Motorola, Inc*., 1993.

[4] R. Schreier: „Understanding Delta-Sigma Data Converters“, *Wiley & Sons*, 2004.

[5] De Barros Soldera, J.D., Espindola, M., Olmos, A.: „Implementing a 10-Bit Sigma-Delta Analog-to-Digital Converter Using the HC9S08Rx MCU Family Analog Comparator“, Applikationsbericht AN2688, Rev. 0.1, *Freescale Semiconductor*, 2005.

[6] D. Peter, B. Baker, D. Butler, H. Darmawaskita: „Make a Delta-Sigma Converter Using a Microcontroller’s Analog Comparator Module“, Applikationsbericht AN700, *Microchip Technology, Inc*., 1998.

[7] STMicroelectronics: „Implementation of SIGMA-DELTA ADC with ST7FLITE05/09“, Applikationsbericht AN1827, *STMicroelectronics*, 2004.

[8] R. Wender, D. Ihme: „16-bit Sigma Delta ADC Design“, Applikationsbericht TSA002, *Triad Semiconductor*, 2007.

[9] Lattice: „Delta-Sigma ADC“, Reference Design RD1063, *Lattice Semiconductor, Corp*., 2009.

[10] Lattice: „Leveraging FPGA and CPLD Digital Logic to Implement Analog to Digital Converters“, White Paper, *Lattice Semiconductor, Corp*., 2010.

[11] Lattice: „Simple Delta-Sigma ADC“, Reference Design RD1066, *Lattice Semiconductor, Corp*., 2010.

[12] J. Logue: „Virtex Analog to Digital Converter“, Applikationsbericht XAPP155, *Xilinx, Inc*., 1999.

[13] Microsemi: „Stellamar Introduces All Digital, Fully Synthesizable Analog-to-Digital Converters“, Applikationsbericht, *Microsemi, Corp*., 2011.

[14] Altera: „Understanding CIC Compensation Filters“, Applikationsbericht AN455, *Altera Corp*., 2007.

[15] J. Kennedy: „Cascaded Integrator Comb (CIC), Product Specification“, *Xilinx, Inc*., 2002.

[16] U. Meyer-Baese: „Digital Signal Processing with Field Programmable Gate Arrays“, *Springer Verlag*, 2007.

[17] L. Milic: „Multirate Filtering for Digital Signal Processing“, *Idea Group Reference*, 2009.

[18] M. Oljaca, T. Hendrick: „Combining the ADS1202 with an FPGA Digital Filter for Current Measurement in Motor Control Applications“, Applikationsbericht SBAA094, *Texas Instruments, Inc*., 2003.

[19] R. A. Losada: „Digital Filters with MATLAB“, *MathWorks, Inc*., 2008.

[20] Actel: „IGLOO nano Low Power Flash FPGAs with Flash Freeze Technology“, Datenblatt, *Actel Corp*., 2010.

[21] Actel: „IGLOO nano Low-Power Flash FPGAs Handbook“, Datenbuch, *Actel Corp*., 2010.

[22] IEEE: „IEEE Standard for Terminology and Test Methods for Analog-to-Digital Converters“, IEEE Std 1241™-2010, *Institute of Electrical and Electronics Engineers, Inc*., 2011.

[23] W. Kester, D. Sheingold: „Testing Data Converters, in Analog-Digital Conversion“, *Analog Devices, Inc*., 2004.

[24] Actel: „IGLOO nano Starter Kit“, User’s Guide, *Actel Corp*., 2010.

Martin Miller studiert im 6. Semester Angewandte Informatik an der HTWG Konstanz und beschäftigt sich seit knapp einem Jahr mit Themen im Bereich Embedded Systems. Im Rahmen eines Projektes implementierte er einen Delta-Sigma-AD-Wandler auf einem Low-Power-FPGA.

Andreas Wilhelm studiert im 6. Semester Angewandte Informatik an der HTWG Konstanz und beschäftigt sich seit knapp einem Jahr mit Themen im Bereich Embedded Systems. Im Rahmen eines Projektes implementierte er einen Delta-Sigma-AD-Wandler auf einem Low-Power-FPGA.

Irenäus Schoppa studierte Informatik an der Technischen Universität Berlin und erhielt dort im Jahre 1993 den akademischen Grad Dipl.-Informatiker. Im Jahre 1998 promovierte er dort zum Dr.-Ing.. Seit dem Jahr 2008 ist er Professor für Hardware-Software Codesign an der HTWG Konstanz.

Thermische Optimierung des Prozess-Scheduling für Multicore-Prozessoren

Daniel Jäckle, Axel Sikora

***Zusammenfassung*—Die zunehmende Anzahl von Transistoren mit immer kleineren Strukturgrößen führt zu einer zunehmenden Leistungsaufnahme in modernen Prozessoren. Das gilt insbesondere für High-End Prozessoren, die mit einer hohen Taktfrequenz betrieben werden. Die aufgenommene Leistung wird in Wärme umgewandelt, die in einer Temperaturerhöhung der Prozessoren resultiert. Hohe Betriebstemperaturen verursachen u.a. eine verringerte Rechenleistung, eine kürzere Lebensdauer des Prozessors und höhere Leckströme. Aus diesen Gründen wird aktives, dynamisches thermisches Management immer wichtiger. Dieser Beitrag stellt eine Erweiterung zu dem Standard-Linux-Scheduler in der Kernel-Version 3.0 für eingebettete Systeme vor: einen PID-Regler, der unter Angabe einer Solltemperatur eine dynamische Frequenz- und Spannungsskalierung durchführt. Die Experimente auf dem Freescale i.MX6 Quadcore-Prozessor zeigen, dass der PID-Regler die Betriebstemperatur des Prozessors an die Solltemperatur regeln kann. Er ist die Grundlage für eine in Zukunft zu entwickelnde prädiktive Regelung.**

***Schlüsselwörter*—Dynamisches thermisches Management, PID-Regler, Multicore, Linux-Scheduler.**

I. Einleitung

Dynamisches thermisches Management (DTM) wird eingesetzt, um den Prozessor vor Schaden durch hohe Betriebstemperaturen zu bewahren. Einerseits gilt es dabei, die durchschnittliche Temperatur gering zu halten und andererseits Spitzenwerte zu vermeiden - unter der Berücksichtigung einer maximalen Rechenleistung und eines kleinen zusätzlichen Rechenaufwands. Es ist damit ein wichtiger Bestandteil in modernen Prozessoren. In den letzten Jahren wurden verschiedene Hardware-basierte und Software-basierte Verfahren entwickelt. Software-basierte Verfahren verwenden Funktionen und Eigenschaften der Software. Insbesondere werden dafür Merkmale des Betriebssystems verwendet und mit Algorithmen zur Einbeziehung thermischer Eigenschaften erweitert, wie beispielsweise thermisches Thread- und Prozess-Scheduling [1]-[4] oder Prozessmigrationstechniken [5]-[7]. Thermisches Prozess-Scheduling beeinflusst die vom Standard-Scheduler vorgegebene Ausführungsreihenfolge der Prozesse, indem es thermische Eigenschaften der Prozesse in das Scheduling-Verfahren einbezieht. Bei Prozessmigrationstechniken werden die thermischen Eigenschaften verwendet, um die Erwärmung auf den verschiedenen Kernen in einem Multicore-Prozessor auszugleichen. Hardware-basierte Verfahren greifen auf Ressourcen bzw. Regelparameter des Prozessors zurück, wie beispielsweise dynamische Frequenz- und Spannungsskalierung (DVFS) [8]-[11], Clock Gating [12] oder Fetch Toggling [13]. DVFS führt eine Skalierung der Taktfrequenz und der Spannungsversorgung einzelner oder mehrerer Kerne in einem Multicore-Prozessor durch, während Clock Gating das Taktsignal für einzelne Komponenten des Prozessors aktiviert bzw. deaktiviert [12]. Die Verfahren im so genannten dynamischen thermischen Management können in regelungstechnische und nicht regelungstechnische Ansätze unterteilt werden [13]. Neben einer Übersicht von verschiedenen Techniken wird in [13] beschrieben, dass regelungstechnische Ansätze in DTM-Techniken

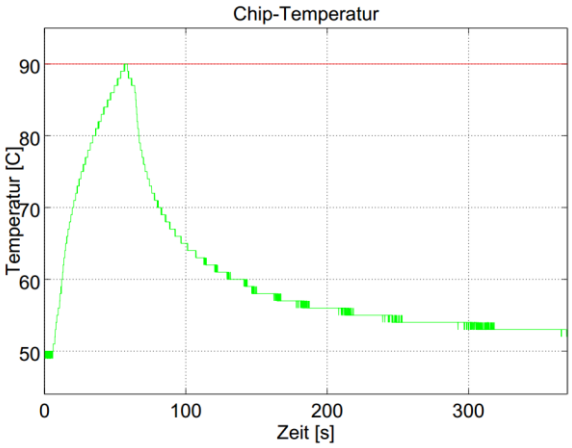

Abbildung 1: Temperaturentwicklung auf dem i.MX6 Quadcore-Prozessor ohne Kühlkörper bei einer Umgebungstemperatur von 21 °C. Im Zeitintervall von 0 s bis 60 s werden vier rechenintensive Prozesse ausgeführt.

Daniel Jäckle, daniel.jaeckle@hs-offenburg.de, Axel Sikora, axel.sikora@hs-offenburg.de, Badstraße 24, 77652 Offenburg.

erfolgreich eingesetzt werden können. Darüber hinaus stellen die Autoren von [13] mehrere PID-Regler auf Basis von Fetch Toggling vor. Basierend auf den Ergebnissen von [13] wurde nun ein PID-Regler unter Verwendung von DVFS als Stellglied entwickelt. Dafür wurde der Standard-Linux-Scheduler in der Kernel-Version 3.0 so erweitert, dass bei jedem Prozesswechsel eine Abtastung erfolgt. In [9] wird ebenfalls eine DTM-Technik vorgestellt, die bei jedem Prozesswechsel in Linux der Kernel-Version 2.6.29.1 ausgeführt wird. Allerdings wird in [9] kein PID-Regler eingesetzt, sondern ein prädiktives Verfahren, das auf Basis von thermischer Prozessklassifizierung DVFS durchführt. Die Ergebnisse von [9] zeigen jedoch, dass es möglich ist, den Scheduler mit einem DVFS-basierten DTM zu erweitern.

II. Problematik

Die grundlegende Problematik der Wärmeentwicklung von Multicore-Prozessoren wird in einem Beispiel in Abbildung 1 verdeutlicht. Die Abbildung zeigt ein Diagramm mit der Temperaturentwicklung des i.MX6 Quadcore-Prozessors beim Ausführen einer Instanz eines rechenintensiven Programms auf jedem Kern. Die Temperatur erreicht nach ca. 60 s die maximale Betriebstemperatur von 90 °C. Für die Temperaturmessungen wurde der interne Temperatursensor des Prozessors verwendet. Die Temperaturentwicklung steht im direkten Zusammenhang mit der aufgenommenen elektrischen Leistung des Prozessors und ist damit abhängig von der prozessorspezifischen Kapazität, der Taktfrequenz und der Versorgungsspannung. Die prozessorspezifische Kapazität wird durch die Summe der Kapazitäten der internen Logikgatter des Prozessors definiert. Diese Kapazitäten werden durch Schaltvorgänge aufgeladen und entladen. Die folgende Gleichung definiert die dynamische, aufgenommene Leistung mit der Abhängigkeit von der Kapazität *C*, der Taktfrequenz *f* und der Spannung *U* [1]:

$$P_{dynamisch} = CfU^2 \quad (1)$$

Daraus resultiert die Bausteintemperatur, die abhängig von der Umgebungstemperatur und der thermischen Leitfähigkeit zur Umgebung ist. Diese Leitfähigkeit bestimmt, wie viel Wärme des Prozessors an die Umgebung abgegeben werden kann, und ist abhängig von verschiedenen Parametern, wie z.B. Größe und Aufbau des Kühlkörpers und Konvektion. Ein DTM-Verfahren muss diese Faktoren und Parameter, die verantwortlich für die Erwärmung des Prozessors sind, berücksichtigen und in einem thermischen Modell beschreiben. In vielen Fällen ist dies nicht analytisch möglich. Oft sind Parameter bzw. Messgrößen nicht bekannt (z. B. die prozessorspezifische Kapazität) oder müssen aufwändig gemessen werden (z.B. die Umgebungstemperatur). Soll ein DTM-Verfahren auf verschiedenen Plattformen eingesetzt werden, ist

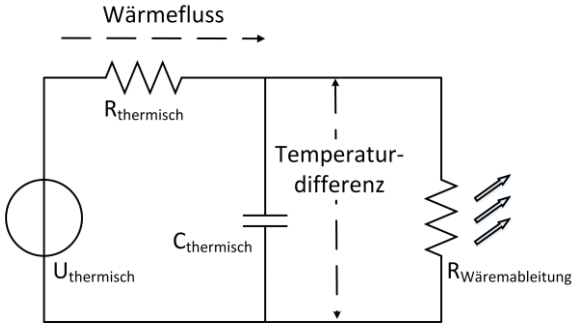

Abbildung 2: Modellierung der Erwärmung des i.MX6 Quadcore-Prozessors mit einem thermischen RC-Glied [2].

weiterführend zu beachten, dass die Parameter bei verschiedenen Prozessoren abweichen und je nach Einsatzgebiet unterschiedliche Umgebungsparameter vorherrschen. Eine Temperaturkurve wie in Abbildung 1 kann bei anderen Prozessoren oder Platinen eine andere Charakteristik aufweisen, da die Temperaturabgabe an die Umgebung von verschiedenen Faktoren wie passive Kühlkomponenten, Printed Circuit Board (PCB) Design und dem Gehäuse abhängt [1].

Nicht alle Prozessoren unterstützen interne Temperatursensoren oder haben Regelungsmechanismen für eine Frequenzskalierung. Beispielsweise können Multicore-Prozessoren keinen, einen oder mehrere interne Temperatursensoren haben oder können die Taktfrequenz einzeln für jeden Kern (lokal) oder zusammen für alle Kerne steuern (global).

Zusätzlich muss der Zielkonflikt zwischen der möglichst guten Nutzung der Rechenleistung und einem möglichst geringen zusätzlichen Rechenaufwand eines Verfahrens zur Temperaturregelung betrachtet werden. Zusammenfassend ist DTM ein komplexes Problem, das von verschiedenen Faktoren abhängig ist. In den nächsten Kapiteln beschreiben wir, wie wir diese Problematik in einem Grundaufbau mit einem PID-Regler lösen.

III. Thermisches Modell

Viele bekannte Arbeiten verwenden thermische Modelle, um die Betriebstemperaturen von Prozessoren zu berechnen. In [2] wird ein Verfahren vorgestellt, das die Temperaturerwärmung in Analogie zu einem thermischen RC-Glied modelliert. Dadurch können die Temperaturen für verschiedene Komponenten eines Prozessors durch Leistungsmessung berechnet werden, beispielsweise für einen Kern eines Multicore-Prozessors [2]. Das Verfahren aus [2] vergleicht den Wärmefluss mit einem Strom, der durch einen thermischen Widerstand fließt und eine Temperaturdifferenz hervorruft, die mit einer Spannung korrespondiert. Abbildung 2 zeigt die Modellierung eines Kerns mit einem thermischen RC-Glied. In einem Multicore-Prozessor kann der Wärmefluss und die resultierende Temperaturdifferenz für jeden Kern einzeln wie in Abbildung 2 beschrieben werden. Des

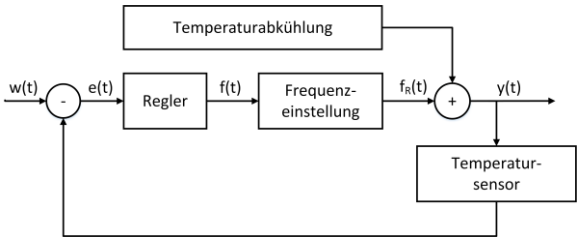

Abbildung 3: PID-Regler: Regelkreis.

Weiteren definiert [2] die Berechnung der prozessorspezifischen, thermischen Konstanten *R* und *C*. Der i.MX6 Quadcore-Prozessor hat nur einen internen, digitalen Temperatursensor. Dadurch kann die Temperatur des gesamten Prozessors ausgelesen werden. Um den zusätzlichen Rechenaufwand gering zu halten, verwenden wir für die Implementierung der Temperaturberechnung kein komplexes thermisches Modell, sondern direkt die Messwerte des integrierten Temperatursensors. Diese Technik hat zum einen den Nachteil, dass man nicht für jeden Kern die zugehörige Betriebstemperatur messen kann. Zum anderen hat der Temperatursensor nur eine Auflösung von 1 °C. Der große Vorteil liegt allerdings darin, dass keine weiteren Berechnungen zur Bestimmung der Temperatur durchgeführt werden müssen. Weiterführend ist dieses Modell nur abhängig von einem Temperatursensor. Es müssen keine spezifischen Parameter des Prozessors bekannt sein. Dadurch ist es einfach, das Verfahren auf andere Prozessoren zu übertragen.

IV. Verfügbare Regelungsmechanismen

Der i.MX6 Quadcore-Prozessor unterstützt einen Sicherheitsmechanismus, der den Prozessor vor Überhitzung schützt. Wenn die maximale Betriebstemperatur von 90 °C erreicht ist, wird automatisch die nächste, niedrigere Taktfrequenz der Kerne eingestellt. Die Messkurve in Abbildung 1 zeigt, dass die Prozessortemperatur nach dem Erreichen der maximalen Betriebstemperatur wieder fällt. Der i.MX6 skaliert automatisch die Taktfrequenzen für alle Kerne herunter. Neben globalem DVFS mit den drei Taktfrequenzen 996 MHz, 792 MHz und 396 MHz unterstützt der i.MX6 Quadcore-Prozessor noch Clock und Power Gating, GPU Clock Management und die Aktivierung bzw. Deaktivierung von internen Spannungsreglern [1]. Diese Arbeit konzentriert sich auf DVFS.

V. PID-Regler

A. Der Regelkreis

Abbildung 3 zeigt den Regelkreis des PID-Reglers. Der i.MX6 Quadcore-Prozessor unterstützt globales DVFS und verfügt über einen integrierten Temperatursensor. Somit wird der Regelkreis für die Regelung der globalen Temperatur des i.MX6 Prozessors verwendet. Der Temperatursensor stellt das Messglied dar. Die Regelabweichung ist eine Temperaturdifferenz, die als Basis für die Berechnung der nächsten Taktfrequenz verwendet wird. Im Regler wird die Regelabweichung in eine Frequenzabweichung $e(t)$ umgerechnet. Der Regler berechnet mit der Frequenzabweichung die Stellgröße $f(t)$:

$$f(t) = K_P e(t) + \sum_{i=0}^{t} K_I T_I e(i) + \frac{K_D}{T_D}(e(t) - e(t-1)) \qquad (2)$$

Der Anteil des numerischen Integrals wird mit einer minimalen bzw. maximalen Grenze limitiert, so dass er keine unverhältnismäßigen Werte annehmen kann. Die Stellgröße $f(t)$ kann kontinuierliche Werte annehmen. Da der i.MX6 Quadcore-Prozessor nur drei Taktfrequenzen unterstützt, wird die Frequenz $f(t)$ in eine Taktfrequenz $f_R(t)$ umgerechnet, die der i.MX6 Prozessor unterstützt. Die zur Taktfrequenz zugehörige Betriebsspannung wird ebenfalls gesetzt.

$$f_R(t) = \begin{cases} 996\ \text{MHz} \mid f(t) >= 996\ \text{MHz} \\ 792\ \text{MHz} \mid 996\ \text{MHz} > f(t) > 396\ \text{MHz} \\ 396\ \text{MHz} \mid f(t) <= 396\ \text{MHz} \end{cases} \qquad (3)$$

Die Abgabe von Wärme an die Umgebungstemperatur wird im Regelkreis als Störgröße modelliert. Die Abgabe ist abhängig von der aktuellen Umgebungstemperatur, welche nur mit einem weiteren, externen Temperatursensor gemessen werden kann.

B. Erweiterung des Schedulingalgorithmus

Der Regelkreis wird direkt in den Schedulingalgorithmus von Linux implementiert. Der Ausführungszeitpunkt für den Regelkreis ist ein Prozesswechsel. In Linux führt jeder Kern den Scheduler mit einem zugehörigen Set von Prozessen einzeln aus. Wann genau ein Prozesswechsel stattfindet, hängt von dem aktuellen Zustand des Systems bzw. von den auf den Kernen ausführbaren Prozessen ab. Die Linux-Kernel-Version 3.0 verwendet einen modularen Scheduler, der Prozesse in Klassen einteilt [14] [15]. Konventionelle Prozesse werden von dem Completely Fair Scheduler (CFS) behandelt und Echtzeitprozesse vom Real-Time Scheduler. In Abhängigkeit von dieser Klassifizierung berechnet bzw. legt der jeweilige Scheduler fest, wie lange ein Prozess maximal auf einem Kern ausgeführt wird, bis er wieder unterbrochen wird [15]. Ebenfalls wird damit definiert, welche anderen Prozesse, die auf die Ausführung warten, den aktuell ausgeführten Prozess unterbrechen dürfen. So kann beispielsweise die Ausführung eines konventionellen Prozesses durch einen Real-Time-Prozess unterbrochen werden, noch bevor die maximale Ausführungszeit abgelaufen ist. Die minimale Ausführungszeit (minimale Granularität) ist für Prozesse der CFS-Klasse auf eine Millisekunde festgelegt und wird zur Laufzeit dynamisch berechnet [15]. Prozesse der Real-Time-Klasse werden entweder als Round-Robin oder First-In, First-Out deklariert. Round-Robin-Prozesse haben eine maximale Standardausführungszeit von 100 ms [15], während

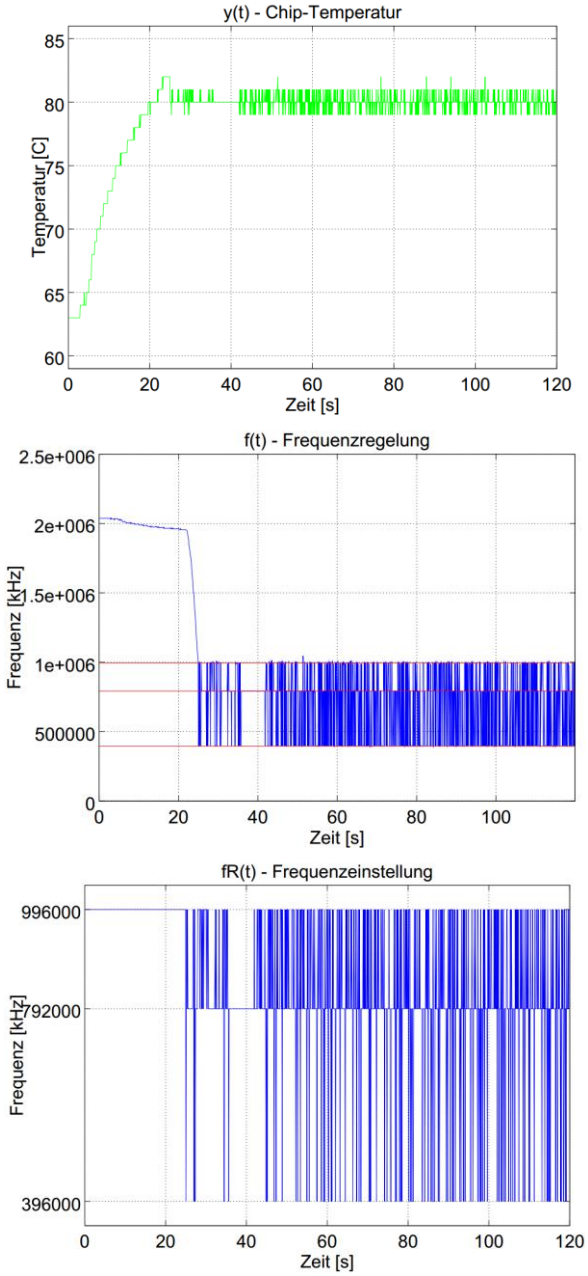

Abbildung 4: PID-Regler nach Gleichung (2) mit K_P = 0,4, K_I = 0,1, K_D = 0,2 und $T_I = T_D = 1$ bei einer Umgebungstemperatur von 21°C.

einem First-In, First-Out-Prozess keine maximale Ausführungszeit zugeordnet ist – ein solcher Prozess wird so lange ausgeführt, bis er von einem Prozess der gleichen Klasse mit einer höheren Priorität unterbrochen wird, oder selbstständig seine Ausführung beendet [15]. Der Abtastzeitpunkt ist dadurch nicht auf ein konstantes Intervall festgelegt, sondern kann ebenso wie ein Prozesswechsel an unterschiedlichen Zeitpunkten auftreten.

VI. Experimente

Alle Experimente wurden mit gleichem Rechenaufwand auf dem i.MX6 Prozessor mit den Parametern aus Tabelle 1 durchgeführt. Das Messintervall in Tabelle 1 beschreibt das Intervall, bei dem die Taktfrequenz und die Betriebstemperatur für die Diagramme gemessen wurden. Mit dem PID-Regler aus Gleichung (3) und den Parametern K_P=0,4, K_I=0,1 und K_D=0,2 konnten wir die besten Ergebnisse erzielen, die in Abbildung 4 dargestellt sind. Zu sehen ist die stabile Regelung auf 80 °C, wobei das Quantisierungsrauschen auf Grund der Auflösung des Temperatursensors von 1 °C deutlich erkennbar ist. Der PID-Regler ist in der Lage, mit einer geeigneten Wahl der Parameter K_P, K_I, und K_D für maximale Aufheizszenarien und typische Abkühlungscharakteristiken die eingestelle Betriebstemperatur mit einem Fehler von max. 1 K einzuhalten. Die Größe dieses Fehlers ist im gegebenen Fall vom Quantisierungsfehler des Temperatursensors abhängig.

Tabelle 1: Parameter der Experimente

Umgebungstemperatur	21 °C
Solltemperatur	80 °C
Programminstanzen	4
Sättigungswerte des Integrals	+/- 100% des Sollwerts
Messintervall	100 ms
T_I, T_D	1

VII. Fazit

Diese Arbeit stellt einen PID-Regler vor, der durch eine thermische Optimierung des Prozess-Scheduling in Linux Anwendung in der Regelung der Betriebstemperatur von Multicore-Prozessoren findet. Der PID-Regler ist der Grundstein für weitere Entwicklungen. Im nächsten Schritt soll der PID-Regler durch eine Prädiktion erweitert werden, die auf thermischen Charakteristiken von Prozessen basiert und sich das thermische Modell aus [2] zu Nutze machen soll. Darauf aufbauend sollen Prozessklassen bezüglich der Taktfrequenz bevorzugt, weitere Regelungsgrößen mit einbezogen und die thermische Wechselwirkung von Prozessorkernen untereinander modelliert werden. Schließlich soll das thermische Management mit Energiemanagement verbunden werden.

Literaturverzeichnis

[1] Freescale Smiconductor, „i.MX 6 Series Thermal Management Guidelines,“ *Application Note* AN4579.

[2] K. Skadron, M. R. Stan, W. Huang, S. Velusamy, K. Sankaranarayanan, D. Tarjan, „Temperature-Aware Microarchitecture,“ *International Symposium on Computer Architecture*, San Diego, Juni 2003.

[3] Y. Inchoon, L. C. Chun, E. J. Kim, „Predictive dynamic thermal management for multicore systems," *Design Automation Conference*, Anaheim, Juni 2008.

[4] J. Yang, Z. Xiuyi, C. Marek, Z. Youtao, J. Lingling, „Dynamic Thermal Management through Task Scheduling," *IEEE International Symposium on Performance Analysis of Systems and Software*, Austin, Juni 2008.

[5] M. D. Powell, M. Gomaa, T. N. Vijaykumar, „Heat-and-run: leveraging smt and cmp to manage power density through the operating system," *Proceedings of the 11th International Conference on Architectural Support for Programming Languages and Operating Systems*, Boston, Oktober 2004.

[6] A. K. Coskun, T. S. Rosing, C. K. Gross, „Utilizing Predictors for Efficient Thermal Management in Multiprocessor SoCs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Dezember 2009.

[7] J. Choi, C. Y. Cher, F. Hubertus, H. F. Hamann, A. J. Weger, P. Bose, „Thermal-aware task scheduling at the system software level," *ACM/IEEE International Symposium on Low Power Electronics and Design*, Portland, August 2007.

[8] J. Donald, M. Martonosi, „Techniques for multicore thermal management: Classification and new exploration," *International Symposium on Computer Architecture*, Washington, Juni 2006.

[9] L. Shu, X. Li, „Temperature-aware energy minimization technique through dynamic voltage frequency scaling for embedded systems," *International Conference on Education Technology and Computer*, Shanghai, Juni 2010.

[10] Y. Liu, H. Yang, R. P. Dick, W. Hui, L. Shang, „Thermal vs Energy Optimization for DVFS-Enabled Processors in Embedded Systems," *International Symposium on Quality Electronic Design*, San Jose, März 2007.

[11] J. S. Lee, K. Skadron, S. W. Chung, „Predictive Temperature-Aware DVFS," *IEEE Transactions on Computers*, Januar 2010.

[12] W. Liao, L. He, K. M. Lepak, „Temperature and supply Voltage aware performance and power modeling at microarchitecture level," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Juni 2005.

[13] K. Skadron, T. F. Abdelzaher, M. R. Stan, „Control-Theoretic Techniques and Thermal-RC Modeling for Accurate and Localized Dynamic Thermal Management," *International Symposium on High-Performance Computer Architecture*, Boston, Januar 2002.

[14] D. P. Bovet, M. Cesati, „Understanding the Linux Kernel, 3rd Edition," O'Reilly, November 2005.

[15] R. Love, „Linux Kernel Development, 3rd Edition," Addison-Wesley Professional, 2010.

Daniel Jäckle erhielt den Bachelor of Engineering im Jahr 2010 von der Dualen Hochschule Baden-Württemberg in Lörrach. Derzeit studiert er an der Albert-Ludwigs-Universität in Freiburg im Masterstudiengang Angewandte Informatik und schreibt derzeit seine Masterarbeit im Labor Embedded Systeme und Kommunikationselektronik an der Fachhochschule in Offenburg.

Dr.-Ing. Axel Sikora studierte an der RWTH Aachen Allgemeine Elektrotechnik (Dipl.-Ing.) und das Zusatzstudium zum Wirtschaftsingenieur (Dipl. Wirt.-Ing.). Er promovierte am Fraunhofer Institut für Mikroelektronische Schaltungen und Systeme in Duisburg über ein Thema der digitalen Schaltungsentwicklung auf SOI-Technologien. Anschließend arbeitete er in verschiedenen Positionen in der Telekommunikations- und Halbleiterindustrie. In 2000 wurde er als Professor an die Berufsakademie Lörrach (heute Duale Hochschule Baden-Württemberg Lörrach) berufen, wo er ab 2001 den Studiengang Informationstechnik aufbaute und leitete. 2002 gründete er das Steinbeis-Transferzentrum für Embedded Design und Networking. 2011 wurde er auf die Professur für Embedded Systeme und Kommunikationselektronik an der Hochschule Offenburg berufen und beschäftigt sich dort in Lehre und Forschung mit der Konzeption, der Implementierung und der Verifikation von sicheren und zuverlässigen drahtlosen und drahtgebundenen Kommunikationsprotokollen.

Dr. Sikora ist Autor, Koautor, Herausgeber und Mitherausgeber von zahlreichen Fachbüchern und Veröffentlichungen. Er arbeitet in den Programmkomitees verschiedener Workshops und Konferenzen mit, u.a. ist er Mitglied im Steering Board der Embedded World Conference.

Choosing the Right Processor Core - An Evaluation Technique

André Zeps, Peter Schulz

***Abstract*—The Multi-Core-Systems-on-Chip research group of University of Applied Sciences and Arts in Dortmund is evaluating processor architectures for homogeneous and heterogeneous Multi- and Many-Core architectures. Applied research in this field is performed in cooperation with the University of Bielefeld by adopting their Network on Chip (NoC) mechanisms and processor core designs. Topics in this research are for example: „Multi-Core to peripherals bridging", „high-bandwidth connection mechanisms between NoC and video output" etc. Because of the heterogeneous approach there is a degree of freedom to choose the best ensemble of different processor cores for a certain task. The work presented here mainly deals with techniques how to enable the evaluation of different processor cores in an application specific System on Chip (SoC) environment. The solution presented here is based on the idea to „swap" different processors at a certain position in the SoC design. A "CPU-wrapper" has been designed which abstracts the oddities of different CPUs. The method has been implemented for the Configurable Ressource Efficient VLIW Architecture (CoreVA) VLIW-processor designed by the University of Bielefeld [1], [2] and the „Plasma - most MIPS I(TM) opcodes"-project hosted on OpenCores [3] which will be called PlasmaMIPS in this document. The swap-mechanism is implemented by means of using a parameter in the hardware description language source. The feasibility of this approach has been proven by simulation and implementation as an FPGA-based prototype.**

***Index Terms*—Verilog, VHDL, ASIC, FPGA, SoC.**

I. Introduction

During the development of system on chip solutions it is sometimes desired to select a specific CPU core fitting the needs for target applications. It can be assumed that the current planned project is not the first developed SoC and experiences using a different core exists in the team. This paper shall give an idea about designing a system which is capable of switching a CPU core while synthesizing the design.

A. Zeps, andre.zeps@fh-dortmund.de is a research assistant at Dortmund University of Applied Sciences and Arts, Sonnenstraße 96, 44139 Dortmund, Germany. P. Schulz, peter.schulz@fh-dortmund.de is a professor at Dortmund University of Applied Sciences and Arts, Sonnenstraße 96, 44139 Dortmund, Germany.

II. Research Background and Motivation

The results presented in this paper are an outcome of the research cooperation between Dortmund University of Applied Sciences and Arts and Bielefeld University. The Dortmund partner is the so-called Chip-Lab. Chip-Lab has been founded to provide support in hardware-related topics for PIMES. PIMES means „Process Improvement for Mechatronic and Embedded Systems". PIMES is an interdisciplinary resarch group. PIMES members are seven professors from two different departments of Dortmund University of Applied Sciences and Arts. Bielefeld partner is the „Cognitronics and Sensor Systems" (K&S) research group of CITEC. CITEC means „Center of Excellence Cognitive Interaction Technology". Both partners have overlapping research interests. K&S provides a strong RISC-architecture (CoreVA) and networkon-chip (NoC) components. Figure 1 shows a general view of a Many-Core system based on a NoC. PIMES works on a tool chain for Multi-Core Software Development. Both partners are Europractice-members and support each other in the microelectronics field. There is an opportunity for Dortmund graduates to receive the doctorate degree from Bielefeld University.

The division of labour between K&S and Chip-Lab is as follows: K&S works on a road-map of further increasing the number of cores per chip [4]. Chip-Lab specializes in NoC-to-peripheral bridging, debug-via-NoC mechanisms and other application related tasks. In common both partners evaluate possible applications for Many-Core Systems, esp. in the field of robotics. It is also intended to evaluate different processor architectures as possible PEs (processing elements). As a first step in this cooperation Chip-Lab has to gain knowledge about the K&S CoreVA architecture [2]. The first approach for that is to incorporate the CoreVA into the Chip-Lab design flow and to develop an FPGA-based application.

The Master's thesis "Design and Implementation of a System-on-a-Programmable-Chip for HDMI-Graphic-Applications" in 2013 lead to the start of this approach [5]. It's goal was to evaluate the CoreVA soft-core designed by the University of Bielefeld and

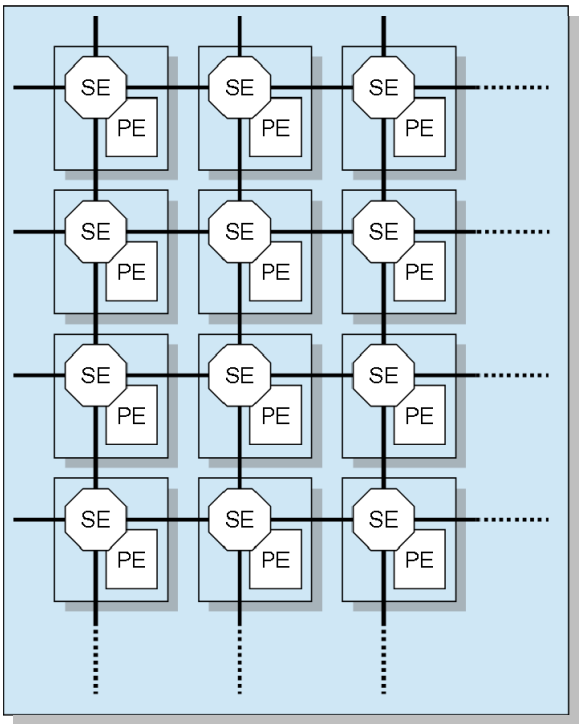

Figure 1: Many-Core System based on Network-on-Chip (NoC) Architecture. Processing elements (PE) are attached to switched eleents (SE). SE's are connected via links to nearest neighbour SE's.

it's software development kit while at the same time evaluating possibilities of transmitting HDMI signals using OSERDES [6] transmitters of Xilinx FPGAs [5]. As the CoreVA wasn't known yet at Chip-Lab there was an uncertainty about the reliability of it's VHDL code and development kit. As the thesis should still lead to success for the HDMI part the idea of a switchable CPU core using a parameter was born. This was further pushed as another soft core namely the PlasmaMIPS was successfully used until that point. This lead to the decision of designing a CPU switching function around the CoreVA and PlasmaMIPS.

III. Design Considerations

The CoreVA and the PlasmaMIPS have a different bus interface and such making the interchangeability more difficult. A CPU wrapper has to be designed to abstract these differences into a common interface. The tables 1 and 2 show the bus interfaces of both CPUs.

A. PlasmaMIPS

The documentation of the PlasmaMIPS processor doesn't offer much information about it's interface. For usage it had to be extracted from the top level module of the soft-core and the reference implementation delivered with the processor. The PlasmaMIPS offers

Table 1: Bus interface of the PlasmaMIPS based on reverse engineering.

Name	Function
clk	CPU clock
reset in	high active asynchronous reset
address next [31:2]	address for next cycle
byte_we_next [3:0]	byte wide write enable for next cycle
byte_we [3:0]	byte wide write enable for next cycle
data_w [31:0]	data input for current and next write cycle (2 cycles long)
data_r [31:0]	data output for read cycle
mem_pause	stall for the current cycle

a different address bus compared to other cores as the address and byte write enable signals are doubled. For read access the PlasmaMIPS gives an address on address_next which means that the answer has to be available in the next cycle. In that cycle the previous address_next is the current address which makes it easy to set the data input multiplexer for the correct data source according to the address bits. On other architectures the glue logic might need an additional register to phase delay the address one cycle to accomplish the same task.

Having this phase delay for read access is important for quite a few bus slaves as not all of them are capable of giving direct data result in the same cycle. One popular example is the Block-RAM available in most Xilinx FPGAs [7]. The write access is similar to other cores as it is two cycles long and addresses a memory location using address next and address in the next cycle which gives possiblity to use both or one of them.

B. CoreVA

As the documentation of the CoreVA also doesn't offer any information about the bus interface it has to be reverse engineered as well by using the sample reference design and it's simulation. The CoreVA being a VLIW core is capable of having multiple execution units but here we only use one to make the design simpler. The table 2 represents the bus interface in this special case. Also different to the PlasmaMIPS is the utilisation of an Harvard architecture with separated buses for instructions and data which adds to the complexity of an uniform model.

Table 2: Bus interface of the CoreVA based on reverse engineering.

Name	Function
clk	CPU clock
reset_n	low-active asynchronous reset
imem_d [63:0]	data input for instruction fetched in last cycle
imem_a [31:0]	instruction address. addressing sized in words of 64 bit
imem_csb	if active no instruction shall be fetched
dmem_dout [31:0]	data input one cycle after read access
dmem_din [31:0]	data output for write access in the current cycle
dmem_a [31:0]	data address. adressing sized in words of 32 bit
dmem_WE	write enabled
dmem_WE [31:0]	32 bit wide write enable (still only bytes are addressed)
dmem_RE	read enable
cpu_stall_in	stalls the CPU in the current state
trace_pc_out [31:0]	fetch address of the currenty executed opcode
trace_opcode_out [31:0]	currently executed opcode

Data access cycles and instruction fetch cycles work similar to the address_next signal of the PlasmaMIPS as the answer for read cycles is needed in the next cycle. An unusual approach is the 32 bit wide write enable signal which could be assumed to address even single bits which is not the case as only words with the minimum of one byte are adressed leaving 28 of the 32 bits obsolete (mem_access_control.vhd in line 127). Other standards like the Xilinx Block RAM, the Xilinx Memory Controller Block (MCB), the PlasmaMIPS, or the Wishbone bus also only address a minimum of 1 byte and feature a 4 bit wide write enable too [7]-[9].

IV. Implementation

The design is implemented using the Digilent Atlys board [10] which offers a Spartan-6 FPGA big enough to carry 32 bit soft cores while at the same time having enough space left for other high logic block count peripherals. Apart from that it offers direct connection to an HDMI plug without any hard core HDMI transmitting hardware which is important for the second part of the mentioned thesis. Having the PlasmaMIPS as the currently more known processor it is decided to keep the base bus architecture of the SoC around it.

For memory access the Xilinx MCB a hard core element in bigger Spartan-6 devices is used. It abstracts the complex DDR2 interface of the onboard RAM to a less complex burst oriented interface which is capable of handling 1 to 6 bus masters in round robin fashion [8]. With this in mind the CPU wrapper for both CPUs are implemented according to figure 2. Both CPUs shall have access to an external bus interface which is as mentioned above PlasmaMIPS style while having access to DDR2 memory using a maximum of 2 MCB slots. For bootup a BootROM is also included which needs to be located inside the CPU wrapper as the havard architecture of the CoreVA made an external solution impossible. Therefore a dual ported ROM is needed here. For access to the DDR2 memory multiple caches are designed to suit the special needs of every core abstracting its memory interface to the MCB. This is needed as both CPUs have different stalling mechanisms rendering a single cache solution very difficult.

The switching of the CPU is reduced to a simple verilog define which instances one of the CPU wrapper modules. So switching the CPU could be done by altering only one line. For software development a system on a chip peripheral library was designed which could be compiled by the gcc for mips and the proprietary compiler for the CoreVA having the same abstraction not only in hardware but also in software. This way even the software applications need only a minimal effort to keep them compilable for both platforms.

V. Conclusion

The presented technique lead to success as the previously unknown CoreVA CPU could be established in our flow. In case of problems with the current system design a quick switch to the other core can be performed. With this approach the source of malfunction is easier to locate as it's now possible to find out whether the error is inside the CPU core or the system surrounding it.

Still one flaw of this approach might be the direct usage of the MCB as this architecture makes it difficult to use other FPGA types or even other memory types like SRAM.

VI. Prospects and Future Research

Most SoC designers will expect something more open minded for the abstraction. This could be for example the well known wishbone [9] bus some open source cores have. Also having multiple bus interfaces (3 for the CoreVA version) makes the usage of an average wishbone more difficult. One approach which

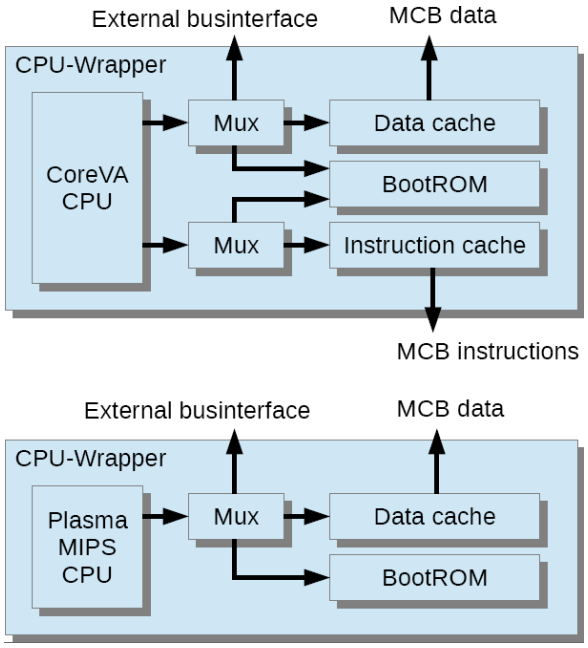

Figure 2: Concept of a CPU wrapper design.

is inspired by the ORPSoC port for the Atlys is to reduce the 3 bus interfaces to only one before feeding into the external memory interface [11]. The existing caches shall still be present but act as an L1 cache for the cores and are still suited for their special stalling mechanisms. But external to that a Wishbone bus or something similar connects to the peripherals and to an L2 cache which then finally connects to the MCB. By redesigning this L2 cache the complexity of the memory change is reduced to only one part compared to every CPU wrapper yet developed.

The results presented here can be used to design future NoC-based Many-Core frameworks for „wrapped CPUs". This enables easy swapping of PEs in future Many-Core systems for comparative studies or for heterogeneous approaches.

ACKNOWLEDGEMENT

At this point the authors want to give thanks to the employees of the CITEC in Bielefeld for providing the CoreVA processor.

REFERENCES

[1] Thorsten Jungeblut,"CoreVA - Ressourceneffizienter VLIW-Prozessor“, http://www.ks.cit-ec.uni-bielefeld.de/de/projekte/coreva-vliw-prozessor.html (last checked 16.01.14).

[2] Thorsten Jungeblut, Dissertation „Entwurfsraumexploration ressourceneffizienter VLIWProzessoren“, CITEC, 2001.

[3] "Plasma - most MIPS I(TM) opcodes”,http://opencores.org/project,plasma (last checked 16.01.14).

[4] Thorsten Jungeblut, Johannes Ax, Mario Porrmann, Ulrich Rückert "A TCMS-based architecture for GALS NoCs,” *IEEE International Symposium on Circuits and Systems* (ISCAS), 2012.

[5] André Zeps, Master’s thesis "Design und Implementierung eines System-on-a-Programmable-Chip für HDMI-Grafik-applikationen“, University of Applied Sciences in Dortmund, 2013.

[6] "Xilinx UG381: Spartan-6 FPGA SelectIO Resources (v1.5 Feb 2,1013)”.

[7] Xilinx UG383: Spartan-6 FPGA Block RAM Resources”, 2011.

[8] "Xilinx UG388: Spartan-6 FPGA Memory Controller”, 2010.

[9] SoC Interconnection: Wishbone – Specification", http://cdn.opencores.org/downloads/wbspec_b4.pdf.

[10] Digilent - AtlysTM Spartan-6 FPGA Development Board, http://www.digilentinc.com/ Products/Detail.cfm?NavPath=2,400, 836&Prod=ATLYS (last checked 16.01.2014.

[11] Orpsoc port to terasic DE0 board, https://github.com/Lampus/orpsoc, 2012.

André Zeps received his M. Sc. degree in Computer Science at the Dortmund University of Applied Sciences and Arts in 2013. He specialized on Embedded Software Development and FPGA design using Verilog. He is working at the Dortmund University of Applied Sciences and Arts as a research assistant for ASIC design since 2012.

Peter Schulz received his diploma in Electrical Engineering from Dortmund University and his doctor's degree in Electrical Engineering from Helmut Schmidt University Hamburg. During his industrial career he worked at Hyperstone electronics in Constance, Thales Naval in Kiel and Rheinmetall Defence Electronics in Bremen. In 2004 he has been appointed to a professorship at Dortmund University of Applied Sciences and Arts. He is teaching Measurement Science, Microprocessing and Digital Systems. Peter Schulz is member of the PIMES researcher group and head of the Chip-Lab laboratory.

MULTI PROJEKT CHIP GRUPPE

Hochschule Aalen
Prof. Dr. Bürkle, (07361) 576-2103
heinz-peter.buerkle@htw-aalen.de

Hochschule Albstadt-Sigmaringen
Prof. Dr. Rieger, (07431) 579-124
rieger@hs-albsig.de

Hochschule Esslingen
Prof. Dr. Lindermeir, (0711) 397-4221
walter.lindermeir@hs-esslingen.de

Hochschule Furtwangen
Prof. Dr. Rülling, (07723) 920-2503
rue@hs-furtwangen.de

Hochschule Heilbronn
Prof. Dr. Gessler, (07940) 1306-184
gessler@hs-heilbronn.de

Hochschule Karlsruhe
Prof. Dr. Koblitz, (0721) 925-2238
rudolf.koblitz@hs-karlsruhe.de

Hochschule Konstanz
Prof. Dr. Schick, (07531) 206-657
cschick@htwg-konstanz.de

Hochschule Mannheim
Prof. Dr. Giehl, (0621) 292-6860
j.giehl@hs-mannheim.de

Hochschule Offenburg
Prof. Dr. Sikora, (0781) 205-416
axel.sikora@hs-offenburg.de

Hochschule Pforzheim
Prof. Dr. Kesel, (07231) 28-6567
frank.kesel@hs-pforzheim.de

Hochschule Ravensburg-Weingarten
Prof. Dr. Siggelkow, (0751) 501-9633
siggelkow@hs-weingarten.de

Hochschule Reutlingen
Prof. Dr. Kreutzer, (07121) 271-7059
hans.kreutzer@hochschule-reutlingen.de

Hochschule Ulm
Prof. Dipl.-Phys. Forster, (0731) 50-28338
forster@hs-ulm.de

www.mpc.belwue.de