MPC

MULTI PROJEKT CHIP GRUPPE
BADEN-WÜRTTEMBERG

Herausgeber: Hochschule Ulm **Ausgabe:** 42 **ISSN** 1862-7102 **Workshop:** Karlsruhe Juli 2009

Cooperating Organisation
Solid-State Circuit Society Chapter
IEEE German Section

**Tagungsband zum Workshop der Multiprojekt-Chip-Gruppe Baden-Württemberg,
Karlsruhe, 10. Juli 2009**

Inhaltsverzeichnis

Diesen Workshopband und alle bisherigen Bände finden Sie im Internet unter:
http://www.mpc.belwue.de

Neues von der DATE 2009

Harald Töpfer

Hochschule Esslingen, Fakultät Mechatronik und Elektrotechnik

Robert Bosch Str. 1, 73037 Göppingen,

harald.toepfer@hs-essslingen.de

Vom 10- 24 April fand in Nizza im Tagunszentrum „Acropolis“ die diesjährige Tagung DATE’09 statt. Die Abkürzung „DATE“ steht für „Design, Automation & Test in Europe“. Schwerpunkte der Tagung waren aktuelle Entwicklungen der Chiparchitektur sowie die dazugehörigen Entwicklungs- und Test-Tools.
Die Tagung bestand aus ca. 120 Vorträgen, 11 Tutorials, der University Booth und einer Präsentation von EDA- und Software-Firmen.

Die Zahl der Tagungsteilnehmer hatte sich trotz der Krise kaum verändert (ca. 1400), nur seitens der Sponsor-Firmen war eine gewisse Zurückhaltung spürbar.

Im folgenden Beitrag möchte ich einige Dinge wiedergeben, die mir besonders interessant erschienen. Die Auswahl ist natürlich subjektiv.

1. Technologie-Trend

Die Entwicklung der CMOS-Technologie war in den letzten Jahren durch eine starke Verkleinerung der Strukturen gekennzeichnet.

Die Entwicklung folgte weitgehend dem bekannten Moore’s Law. Danach verdoppelte sich die Zahl der Transistoren etwa alle 2 Jahre.

Die Chiphersteller sind gegenwärtig dabei die Technologie auf 45nm umzustellen. Das nächste Bild [5] zeigt die voraussichtliche Technologieentwicklung bis 2015.

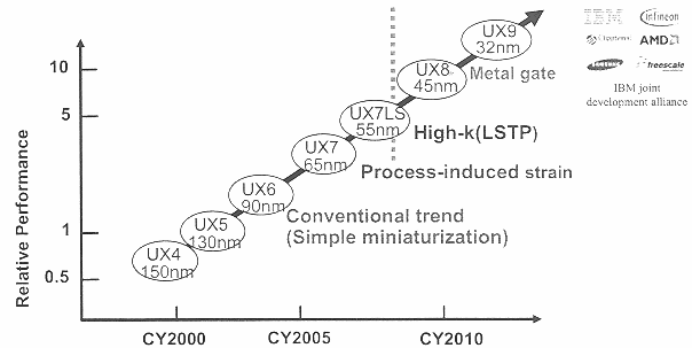


Bild 1: Technologie-Entwicklung 2000- 2015

Danach ist auch in den nächsten Jahren ist mit einer weiteren Verkleinerung der Strukturen zu rechen. Allerdings steigt durch diese kleinen Strukturen der Investitionsaufwand für die neuen Fab’s stark an. Die Chipfertigung konzentriert sich in Zukunft wahrscheinlich auf wenige große Fab’s, wie z.B. TSMC, vorwiegend in Asien.

Es ist fraglich, ob europäische Unternehmen in Zukunft noch den großen Aufwand für die neuen Fertigungsanlagen stemmen können.

Neben diesen betriebswirtschaftlichen Aspekt, sind mit der weiteren Strukturverkleinerung auch massive technische Probleme verbunden.

Die Strukturverkleinerung stößt inzwischen an die physikalischen Grenzen. Die Gate-Oxiddicke der MOS-Transistoren beträgt nur noch etwa 1,5 nm, das sind 5 Atomlagen. Dadurch entstehen Leckströme sowohl durch das Gateoxid (Tunelstöme), als auch zwischen Drain und Source. D.h. das Gateoxid ist kein Isolator mehr und der Transistor schaltet den Drainstrom nicht mehr vollständig ab.

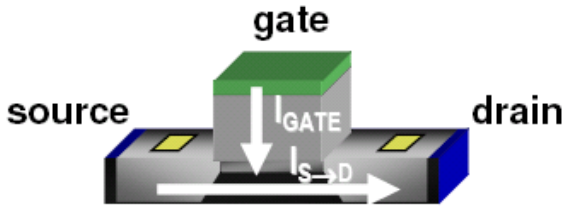
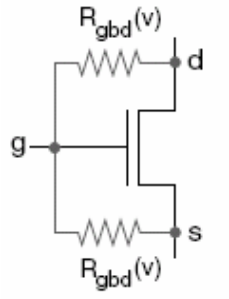


Bild 2: Leckströme am MOS-Transistor [2]

Die Leckströme treten verstärkt ab 65nm auf. Die Beherrschung dieser Leckströme wird zu einem Hauptproblem der zukünftigen Technologie-Entwicklung.

Wegen der Gate-Leckströme muss das Gateoxid als Widerstand R_{gbd} angesehen werden.



$$R_{gbd}(V) = \frac{V}{K \cdot V^{p}}$$

Bild 3: Spannungsabhängiger Widerstand Gate-Oxid R_{gbd}

Das nächste Bild zeigt die Spannungsabhängigkeit von R_{gbd} [1]. Der Widerstand nimmt also stark mit der Versorgungsspannung ab.

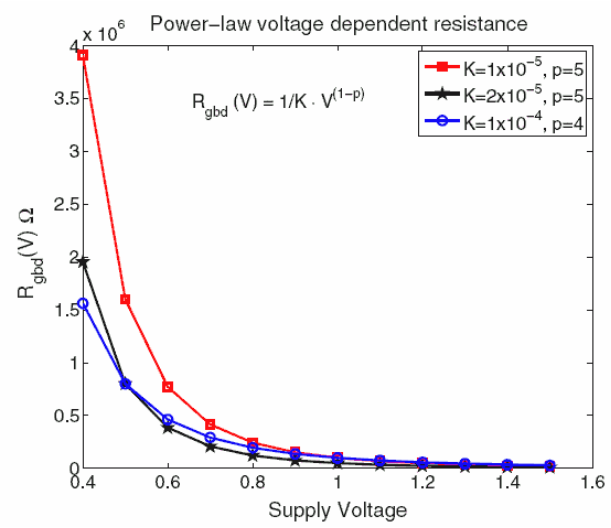


Bild 4: Spannungsabhängigkeit von R_{gbd}

Um den Gate-Leckstrom zu reduzieren, wäre es günstig die Betriebsspannung VCC zu verkleinern. Das ist nur bedingt möglich. Die Verzögerungszeit t_D einer CMOS-Schaltung ist näherungsweise proportional zum Kehrwert von VCC:

$$t_D \sim C \cdot tox / (VCC - VT)$$

Deshalb führt eine Verkleinerung von VCC zu einer Vergrößerung der Verzögerungszeiten.

In der Vergangenheit wurde als Kompensation die Schwellenspannung VT verkleinert. Auch dies ist in Zukunft kaum noch möglich, da bei weiterer Verkleinerung der VT die Source-Drain-Leckströme stark zunehmen. Deshalb wird voraussichtlich die Betriebsspannung VCC nicht mehr wesentlich unter 1V reduziert werden. Die Wahl der Betriebsspannung ist also eine Gratwanderung zwischen Leckstrom und Geschwindigkeit.

Interessant ist auch die Temperaturabhängigkeit der beiden Leckströme. Das nächste Bild [2] zeigt das Temperaturverhalten des Gate- und des Source-Drain-Leckstromes. Bei niedrigen Temperaturen dominiert der Gate-Leckstrom, bei hohen Temperaturen dagegen der Source-Drain-Leckstrom.

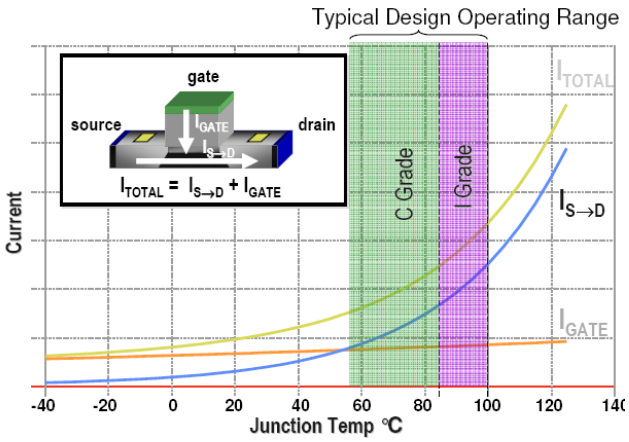


Bild 5: Temperaturabhängigkeit der Ströme I_{GATE} und I_{S-D} [2]

Die Leckströme engen die erlaubte Streuung der Fertigungsparameter ein und sind deshalb auch eine große Gefahr für die Ausbeute (Yield).

Um die Leckstromproblematik in den Griff zu bekommen werden heute 2 Vorkehrungen getroffen:

- Gate-Oxid mit hoher Dielektrizitätskonstante
- Verwendung von bis 3 verschiedene Oxid-Dicken und VT's auf dem Chip

Ab Strukturgrößen von 55nm werden in das Gate-Dielektrikum sogenannte High K-Schichten eingebracht. Dadurch kann die Gate-Oxid-Dicke wieder vergrößert werden und der Gate-Leckstrom sinkt.

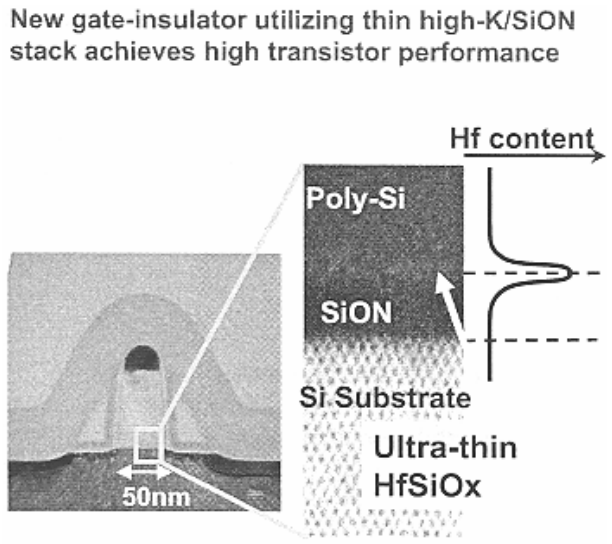


Bild 6: Transistor mit High-K-Gate-Dielektrikum [5]

Durch die obengenanten Maßnahmen können die Chipstrukturen auch in den nächsten Jahren weiter verkleinert werden, bei beherrschbaren Leckströmen.

Ein Beispiel zeigt das nächste Bild. Darin ist der statische Energieverbrauch von FPGA's der Fa. XILINX bei verschiedenen Strukturgrößen und Betriebsspannungen aufgetragen. Es zeigt sich, dass die 45nm-Technologie sogar einen geringeren statischen Energieverbrauch (durch Leckströme) als vorangegangene Technologien hat. Dazu trägt auch die Reduktion der Betriebsspannung von 1,2V auf 1V oder 0,9V bei.

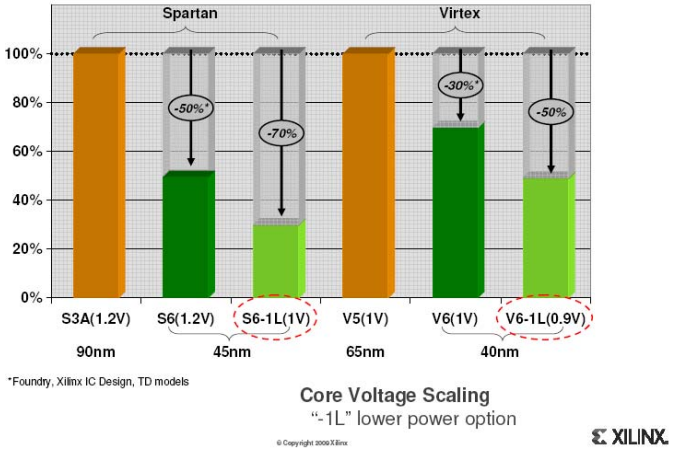


Bild 7: Statischer Energieverbrauch von Spartan- und Virtex-FPGA bei Skalierung [2]

Von einzelnen Referenten wurde hervorgehoben, dass wegen der angesprochenen technologischen Probleme, in Zukunft die Verzögerungszeiten von CMOS-Chips nicht mehr so stark wie in der Vergangenheit reduziert werden können [3].

2. PPGA: Power Optimized Design

Im Rahmen des Tutoriums: "Power Optimized Design for Modern FPGAs" referierte J. Nogera von der Firma Xilinx über stromsparende Designs.

Neben der bereits im vorigen Abschnitt angesprochen Verlustleistung durch Leckströme spielt bei FPGAs die dynamische Verlustleistung eine große Rolle. Die dyn. Verlustleistung ist $P \approx CV^2f$, d.h. sie ist proportional zu den internen Schaltkapazitäten C.

Die dynamische Verlustleistung wird bei FPGAs vor allem durch die Kapazitäten von Clock Tree, von den Verdrahtungsleitungen und der Switch Matrix bestimmt. Das nächste Bild zeigt die Aufteilung des dyn. Leistungsverbrauchs für ein typisches Design.

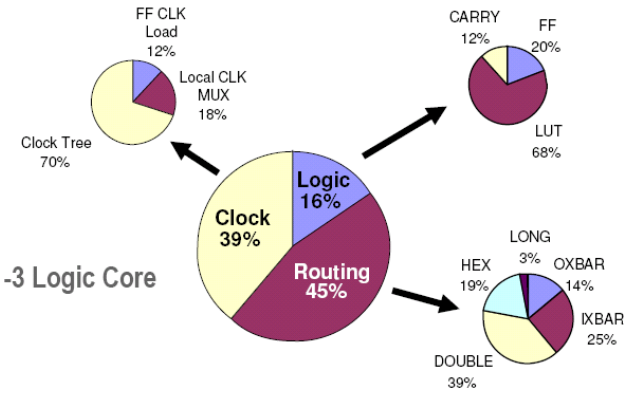


Bild 8: Dynamischer Energieverbrauch aufgeschlüsselt [2], (90nm Spartan-3- FPGA, 100MHz, 12,5% switching activity)

Nach Bild 8 gibt es am ehesten Einsparmöglichkeiten beim Routing und beim Clock-Tree. Entsprechend wird vorgeschlagen das Design in CLOCK-Regionen zu sortieren und zeitweise nicht benötigte Clocks durch enablebare Clock-Buffer abzuschalten oder mit niedrigerer Frequenz zu takten. Auch Speicherblöcke, die nicht ständig benötigt werden sollten in den Ruhephasen disabled werden.

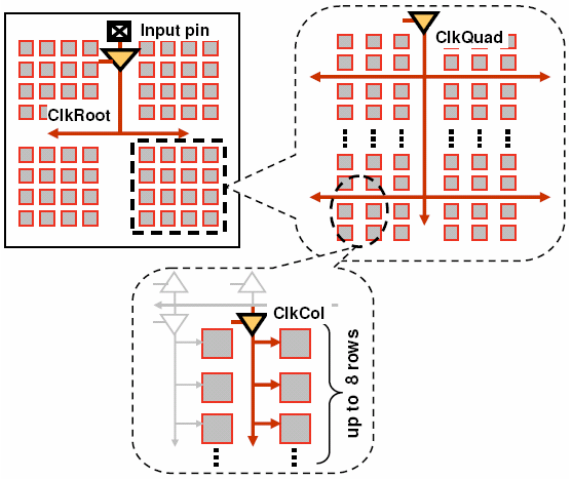


Bild 9: Abschaltbare Clock-Regionen [2]

Glitche, die in einem Logiknetzwerk durch unterschiedliche Laufzeiten entstehen, verursachen zusätzliche Schaltaktivitäten und damit eine Erhöhung des dynamischen Stromverbrauches. Sie sollten deshalb unterdrückt werden. Zur Vermeidung von Glitchen können Latche in die Logik eingebaut werden (Piplining). Das nächste Bild zeigt am Beipiel einer einfachen Logik, wie durch den Einbau von 2 Latchen Glitche verhindert werden.

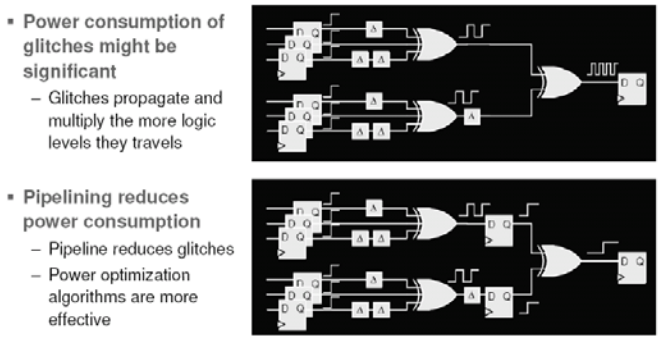


Bild 10: Unterdrückung von Glitchen durch den Einbau von Latchen [2]

Weithin wird in [2] und [7] zur Reduzierung des Dynamischen Stromverbrauches vorgeschlagen:

- Vorzugsweise Verwendung von „Hard-IPs“. Diese haben einen wesentlich geringeren dyn. Stromverbrauch.
- Timing Constraints bei der Synthese realistisch wählen, nicht zu stark einschränken. Synthese für “Area“ optimieren.
- Minimale Slew Rate und Drive-Strength der Pin-Treiber auswählen.
- Power optimierte Synthese und Placement durch optimales Placement kann der dyn. Stromverbrauch ganz erheblich reduziert werden.
- Einsatz spezieller Tools: XPE, XPA (Xilinx Power Estimator / Analyser)

3. Network on Chip (NoC)

Das Thema “Network on Chip” (NoC) oder “System on Chip” (SoC) nahm auf der Tagung einen breiten Raum ein.

Die Firmen NXP und ST präsentierten aktuelle Muli-Core-Chips für Multimedia-Applikationen [11]. Diese Chips bestehen aus mehreren verschiedenen Prozessoren und DSPs, auf die die Verarbeitungs-Algorithmen gemappt werden.

Einen ähnlichen Trend gibt es bei Handy-Chips. Auch hier setzen sich Multi-Core Architekturen immer mehr durch [12]. Es wird erwartet, dass Handy-Chip in wenigen Jahren bis zu 10 Cores enthalten.

Multi-Core Architekturen sind deshalb so erfolgreich, weil man mit mehreren Cores eine höhere

Rechenleistung erreicht als mit einem hochgetakteten Core und das bei geringerem Stromverbrauch. Allerdings erfordern Multi-Core-Architekturen auch eine feinkörnige Aufteilung der Applikation auf die verschiedenen Cores, was aber bei Mobile Phones kein Problem ist.

Das nächste Bild zeigt einen H.264(MPEG 4)-Decoder-Chip der Fa. Toshiba. Der Chip besteht aus 8 Cores (32 Bit), jeweils mit einem L1-Cache. Die Cores sind über einen gemeinsamen L2-Cache verbunden. Der H.264-Algorithmus wird in einzelne Threads zerhackt und parallel auf den verschiedenen Cores ausgeführt.

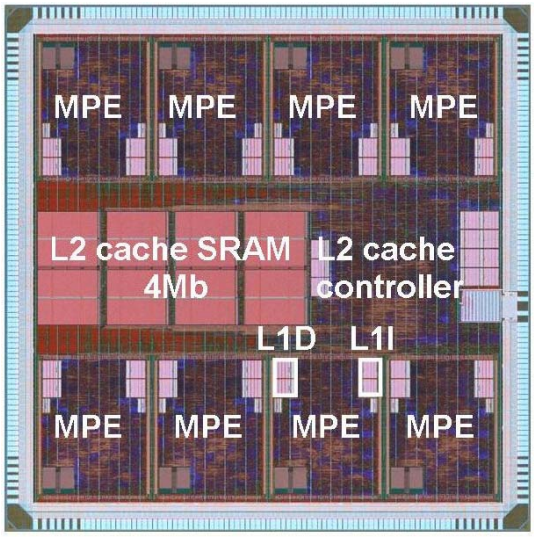


Bild 11: H.264-Decoder-Chip (Toshiba) mit 8 Cores [6]

Neben gegenwärtig produzierten Multi-Core-Chips wurden im Rahmen des Tutoriums „New Developments and Trends in Networks on Chip“ auch einige zukunftsweisende Forschungsprokte zu diesem Thema präsentiert [4], [5], [8].

Kundu (intel) präsentierte einen Research Chip, der aus 80 Cores besteht (Bild 12) [4]. Jeder Core hat eine Abmessung von 1,5 x 2 mm, der Gesamtchip 13 x 21 mm. Jeder Core kann mit 0 - 5 GHz getaktet werden, die Betriebsspannung wird für jeden Core entsprechend der Frequenz auf 0,7 – 1,2 V eingestellt. Bei thermischer Überlastung (Betriebstemperatur 110°C) erfolgt ein „Core Hopping“.

Der Chip eignet sich natürlich nur für Anwendungen, die sich in viele kleine Task zerteilen lassen, wie Recognition, Mining, Synthesis. Wichtig ein schnelles Scheduling der vielen Tasks. Der Scheduler ist deshalb teilweise in Hardware realisiert.

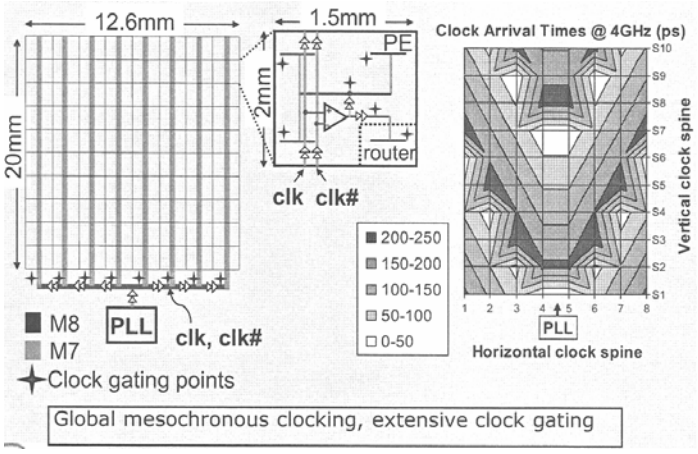


Bild 12: Research-Chip (intel) mit 80 Cores [4], rechtes Bild: Verteilung Clock-Skew über den Chip

Ein großes Problem bei Multi-Core-Archtekturen ist die Verbindung zwischen den Cores. Bei dem intel-Chip wird kein kein Bus, sondern maschenförmiges 2D-Netzwerk mit jeweils 39 Datenleitungen benutzt (Bild 13). An den Kreuzungspunkten des Netzwerks sitzen Router, die die einzelnen Zweige des Netzwerks verbinden. Jeder Core ist ebenfalls über einen Router mittels eines 5-stufigen FIFOs an das Netzwerk angebunden.

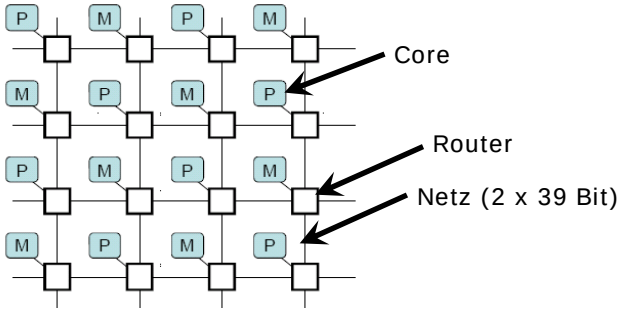


Bild 13: Maschenförmiges Netzwerk für Multi-Core-Chip

Ein weiteres Problem ist die Taktversorgung der vielen Cores. Wegen der großen Chipfläche kann ein synchroner für alle Cores nicht mehr gewärleistet werden. Der intel-Research-Chip hat ein Mesochronous-Clock, d.h. alle Cores bekommen die gleiche Clock-Frequenz, aber mit Phasenunterschieden.

Häufig haben Multi-Core-Chips eine **„Global Asynchron Local Synchron“ (GALS)** Clock-Architektur (Bild 14) [5] [13]. Die Prozessoren (Cores) haben jeweils einen lokalen synchronen Clock (Clock Domain), das Netzwerk selbst ist asynchron. Durch zwischengeschaltete FIFOs wird das Signal zwischen den Clock Domains transferiert.

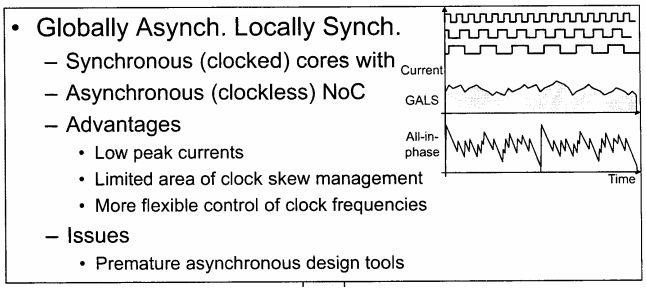


Bild 14: GALS-Clock-Schema [5]

4. Test und Simulation

Der Chiptest und die dazu notwendige Software waren ein Schwerpunkt der Tagung.

Es wurde ein Wireless IC-Test vorgestellt. Die Fa. Scanimetrix bietet eine kommerziell erhältliche Lösung für das kontaktlose Testen von Chips an, d.h. die sonst beim Chiptest benutzen Kontaktierungsnadeln fallen weg [9].

Die Lösung basiert wahrscheinlich auf einer induktiven Kopplung Tester – DUT. Es wird ein Wireless Test AccesssPort (WitAP) benutzt (Bild 15). Allerdings ist bei dem Testverfahren die Energieübertragung zum Chip schwierig (zusätzliche Nadel). Es bleibt abzuwarten, ob sich das Verfahren durchsetzt.

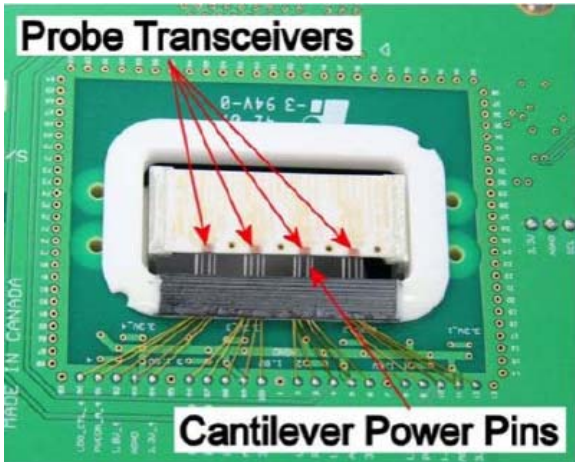


Bild 15: Kontaktloser Chip-Prober der Fa. Scanimetrixs [9]

Oetjens [10] stellte ein Konvertierungsprogramm **VHDL – SystemC** vor. Damit kann aus der VHDL-Beschreibung eines vorhandenen Chips ein Simulationsmodell in SystemC generiert werden. Der erzeugte SystemC-Code soll gut lesbar sein. Das Mapping der Datentypen von VHDL nach SystemC ist schwierig.

5. Literatur

[1] V. Chandra, R. Aitken: “Impact of Voltage Scaling on Nanoscale SRAM Reliability”, Date 2009, S. 387

[2] Noguera: „Power-aware Design Techniques for Xilinx FPGAs“ Tutorial, Date 2009

[3] S. Fujita: “Nano-electronics Challenge - Chip Designers Meet Real Nano-Electronics in 2010s?” , Date 2009, S. 431

[4] P. Kundu (intel): “On Chip Interconnects for Tra-scale Processors”, Tutorial D1, Date 2009

[5] M. Lajolo (NEC): “Toward NoC adaption at NEC”, Tutorial D1, Date 2009

[6] Kodaka et al: „Design and Implementation of Scalable, Transparent Threads for Multi-Core Media Proceccor“, Date 2009, S. 1035-1039

[7] Hübner: „Back-end CAD tool for FPGA power optimization” Tutorial, Date 2009

[8] Locatelli (ST): „Spidergon STNoC IPU: the next generation of NoC” Tutorial, Date 2009

[9] Sellathamby et al (Scanimetrixs): „A Complete Solution for Wireless IC Testing“ , Date 2009, S. 676-681

[10] Oetjens et al: „An Automated Flow for Integrating Hardware IP into the Automotive Systems Engineering Process“, Date 2009, S. 1196-1201

[11] Kollig et al (NXP): „Heterogeneous Multi-Core Platform for Consumer Multimedia Applications”, Date 2009, S. 1254-1259

[12] van Berkel (ST-NXP): „Multi-Core for Mobile Phones”, Date 2009, S. 1260-1265

[13] P. Vivet: “Effizient NoC Design for MPSoC based on GALS Architecture and Fine Grain DVFS”, Tutorial, Date 2009

Prototyp eines digitalen EEGs auf Basis eines FPGAs für die Neurofeedbacksignalverarbeitung

Dipl.-Ing. (FH) Felix M. Klein

HTW Aalen, EDA Zentrum, Anton Huber Strasse 25, 73430 Aalen

Tel. 07361 / 576-4247, Fax 07361 / 576-444249

Neurofeedback und Brain-Computer-Interfaces sind beides Verfahren die ihren Nutzen aus der elektrischen Aktivität des menschlichen Gehirns gewinnen. Wesentlich für beide Verfahren ist die Echtzeitverarbeitung der abgeleiteten EEG[1]-Signale, die mit ihren charakteristischen Merkmalen die elektrische Gehirnaktivität beschreiben. Eine schnelle Signalverarbeitung ist daher die Voraussetzung, damit zeitnah auf Änderungen in den EEG-Signalen reagiert werden kann. Bisherige digitale EEG-Systeme setzen im Allgemeinen eine computergestützte Signalverarbeitung voraus, was den mobilen Einsatz eines solchen Systems weitestgehend verhindert. Vorteile sind daher insbesondere in der Umsetzung der digitalen Signalverarbeitung auf Hardwareebene eines FPGA[2]s zu sehen. Die parallele Verarbeitung der anfallenden Daten und die flexible Konfiguration eines derartigen Systems, bei gleichzeitiger Möglichkeit für einen mobilen Einsatz, bietet für die beiden zuvor genannten Verfahren ein erhebliches Entwicklungspotential.

Ziel dieser Arbeit ist daher die Realisierung eines digitalen EEGs auf Basis eines FPGAs in Form eines Prototyps, der die notwendigen Voraussetzungen für eine schnelle Neurofeedbacksignalverarbeitung auf Hardwareebene bereitstellt und als Grundlage für weitere Untersuchungen in diesem Bereich verwendet werden kann.

1. Motivation

Die Erfassung der elektrischen Gehirnaktivität, mit Hilfe der Elektroenzephalographie, gehört in der medizinischen Diagnostik zu den Standardverfahren, um krankhafte Veränderungen oder Störungen der Hirnaktivität eruieren zu können. Die von der Hirnaktivität hervorgerufenen und bei der Elektroenzephalographie registrierten Potentialschwankungen erlauben Rückschlüsse über den Ablauf der Informationsverarbeitung im Gehirn und den Bewusstseinszustand. Damit verbunden sind charakteristische Merkmale in den abgeleiteten EEG-Signalen, die sowohl willkürlich auftreten als auch willentlich beeinflusst werden können.

Durch die algorithmische Evaluierung der abgeleiteten EEG-Signale und durch die Möglichkeiten der digitalen Signalverarbeitung eröffnen sich neuartige und innovative Anwendungsbereiche, in denen die elektrische Gehirnaktivität erfolgreich und produktiv eingesetzt werden kann. Die computergestützte Verarbeitung der digitalisierten EEG-Signale ermöglicht sowohl die einfache Extrahierung relevanter Signalparameter, als auch eine komplexe Mustererkennung und Merkmalsextraktion auf einer höheren Betrachtungsebene. Bereiche, in denen diese Verfahren ihre Anwendung finden, sind das Neurofeedback und das Brain-Computer-Interface, auch als Gehirn-Computer-Schnittstelle bezeichnet.

Beim Neurofeedback und BCI[3] steht die Echtzeitverarbeitung der EEG-Signale im Vordergrund, damit zeitnah auf Änderungen der elektrischen Gehirnaktivität reagiert und ein entsprechendes Feedback oder Steuersignal generiert werden kann. Der Einsatz eines digitalen EEG-Systems ist für die komplexen algorithmischen Berechnungen eine notwendige Voraussetzung. Rein analoge EEG-Systeme sind aufgrund der fehlenden digitalen Datenaufzeichnung gänzlich ungeeignet. Zudem verhindern deren Abmessungen einen mobilen Einsatz des Geräts. Im Gegensatz dazu ist die Integrationsfähigkeit digitaler EEG-Systeme um ein Vielfaches höher. Weiterhin besteht ein hoher Anspruch an die Qualität der abgeleiteten EEG-Signale. Überlagerte Störungen verfälschen oder verhindern eine fehlerfreie Analyse der Signale. Probleme treten zudem durch die Signalcharakteristika bei der Ableitung und Verarbeitung der EEG-Signale auf. Die bei einer nichtinvasiven Ablei-

[1] EEG - Elektroenzephalographie, Elektroenzephalogramm

[2] FPGA - Field Programmable Gate Array

[3] BCI - Brain-Computer-Interface

tung an der Schädeloberfläche messbaren Potentialdifferenzen weisen sehr geringe Spannungsamplituden auf und erfordern daher eine hohe Verstärkung. Bereits kleinste Störungen wirken sich bei einer EEG-Registrierung, wegen der hohen Verstärkung, massiv auf die Signalqualität aus.

Die Vorteile eines digitalen EEG-Systems auf Basis eines FPGAs sind vor allem in der direkten Umsetzung der digitalen Signalverarbeitung auf Hardwareebene zu sehen. Hieraus resultiert eine parallele, effiziente und zeitgleiche Verarbeitung aller abgeleiteten EEG-Signale. Unterschiedlich komplexe Strategien bei der Signalaufbereitung und der Signalanalyse sind durch die Konfiguration des FPGAs einfach umsetzbar und an die jeweiligen Bedürfnisse anpassbar. Abhängig von den Ressourcen und der Leistungsfähigkeit des eingesetzten FPGAs sind somit Systeme unterschiedlichster Art möglich, die auch ohne Computer für die Signalverarbeitung, die vollständige, kompakte, konfigurierbare und auf einen Anwendungszweck abgestimmte Gesamtfunktionalität für ein einzelnes Gerät bereitstellen können. In Verbindung mit den Weiterentwicklungen bei der Integration der notwendigen analogen Hardwarekomponenten für die EEG-Registrierung auf Chipebene, deren Energieversorgung und drahtlosen Anbindung, ergibt sich ein hohes Entwicklungspotential in den Bereichen des Neurofeedbacks und des BCI.

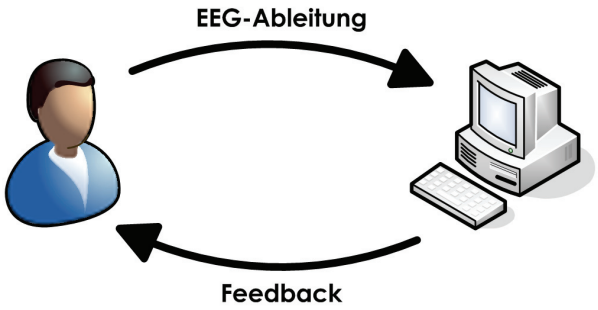


Abbildung 1: Grundprinzip des Neurofeedbacks

2. Grundlagen

2.1. Bio- / Neurofeedback und BCI

Viele der physiologischen Vorgänge im menschlichen Körper werden nicht bewusst wahrgenommen, da eine Wahrnehmung dieser Prozesse mit den vorhandenen menschlichen Sinnesorganen nicht direkt möglich ist. Das Biofeedback setzt daher technische Zusatzgeräte ein, die eine wahrnehmbare Rückmeldung erlauben. Hauptsächlich dienen hierzu visuelle und akustische Reize, die nahezu in Echtzeit generiert werden. Auf diese Weise kann das Ziel des Biofeedbacks erreicht werden, durch technische Maßnahmen die ansonsten unbewussten Vorgänge derart darzustellen, dass diese bewusst wahrgenommen werden können. Abbildung 1 verdeutlicht dieses generelle Prinzip.

Während beim Biofeedback die unterschiedlichsten von Körperfunktionen generierten Signale zur Anwendung kommen, beschränkt sich das Neurofeedback rein auf die elektrische Aktivität des Gehirns. Mit der Rückkopplung von visuellen und akustischen Reizen, welche die momentane elektrische Hirnaktivität repräsentieren und aus dieser abgeleitet sind, wird eine Selbstregulierung der elektrischen Hirnaktivitäten ermöglicht. So können aufgrund des dargebotenen Feedbacks einzelne charakteristische Merkmale des EEGs beeinflusst werden, mit dem Ziel, die Hirnaktivität in Hinsicht auf das Denken, der Aufmerksamkeit oder des Wohlbefindens gezielt zu verbessern. Zum Einsatz kommt dabei das Prinzip der positiven Konditionierung[4]. Mittels einer Belohnung wird dabei der Anreiz gegeben, die gewünschte Hirnaktivität zu erreichen und zu erhalten.

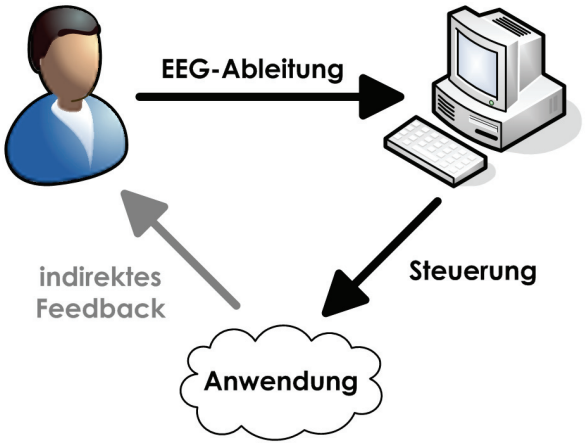


Abbildung 2: Grundprinzip des BCI

Im Gegensatz zum Bio- und Neurofeedback erfolgt bei einem BCI, wie aus Abbildung 2 hervorgeht, keine direkte Rückkopplung der abgeleiteten EEG-Signale. Die algorithmisch ausgewerteten Rohsignale des EEGs dienen beim BCI als Steuersignale für ein externes System, welches dem Nutzer indirekt eine Rückmeldung erteilt. Grundlage sind erkennbare und definierte Muster in der Hirnaktivität, die bei bestimmten Ereignissen oder Verhaltensweisen auf-

[4] Erlernen von Reiz-Reaktions-Mustern

treten und jederzeit kontrolliert reproduziert werden können.

Ein wesentlicher Anwendungsbereich des BCI ist der Einsatz als unterstützendes Hilfsmittel bei neurologischen Störungen und Verletzungen. Patienten mit erheblichen Beeinträchtigungen des neurologischen Systems bietet das BCI die Möglichkeit, Teile dieser Beeinträchtigungen zu kompensieren. An ALS[5] erkrankten Patienten, die aufgrund der Zerstörung des peripheren[6] Nervensystems in ihrer Bewegungs- und Verständigungsmöglichkeit erheblich eingeschränkt sind, ermöglicht das BCI beispielsweise die Steuerung einer Computereingabeschnittstelle, um sich ihrer Umwelt mitzuteilen.

Grundlegende Methode, die sowohl für das Neurofeedback als auch das BCI eingesetzt wird, ist die Elektroenzephalographie.

2.2. Elektroenzephalographie

Eine weit verbreitete Methode zur Messung, Aufzeichnung und Analyse der elektrischen Hirnaktivität ist die Elektroenzephalographie. Erfasst werden mit dieser Methode die spontanen oder evozierten[7] Potentialschwankungen, die bei der Informationsverarbeitung der Nervenzellen in der Großhirnrinde entstehen. Die feststellbaren Potentialschwankungen beruhen dabei auf den synchronisierten Aktivitäten ganzer Nervenzellverbänden.

Typischerweise erfolgt eine nichtinvasive Ableitung der EEG-Signale mit Hilfe von Metallelektroden, welche an den entsprechenden Ableitpunkten auf der Schädeloberfläche angebracht werden. An der Kontaktstelle zwischen dem physiologischen Körpergewebe und dem Elektrodenmaterial erfolgt der Übergang von einer körperseitigen Ionenleitung auf eine Elektronenleitung in der Elektrode. Das elektrische Verhalten der Kontaktstelle stellt für den Signaltransport insgesamt einen komplexen Übergangswiderstand dar. Diese Elektrodenimpedanz wirkt sich in einem hohen Maße auf die erreichbare Qualität der Signalableitung aus und ist massiv vom verwendeten Elektrodenmaterial abhängig.

Für die Ableitung, Verarbeitung und Ausgabe von EEG-Signalen können prinzipiell zwei unterschiedliche Systemarten unterschieden werden. Bei einem analogen System werden die abgeleiteten EEG-Signale ausschließlich analog verarbeitet und ausgegeben. Die Ausgabe des Elektroenzephalogramms, die grafische Darstellung der einzelnen Signalverläufe, basiert hierbei auf einem mechanischen Schreibsystem, bei dem der Verlauf der Potentialschwankungen entweder mit Tinte oder mit einem Thermokamm auf Papier übertragen wird. Die zeitliche Auflösung des abgeleiteten EEGs kann bei diesen Verfahren über die Papiergeschwindigkeit variiert werden.

Bei einem digitalen System erfolgt hingegen eine Wandlung der abgeleiteten EEG-Signale in eine digitale Repräsentationsform. Die digitale Verarbeitung und Ausgabe der Signale erlaubt eine flexible und beliebig oft wiederholbare Auswertung des EEGs ohne zusätzliche Verluste bei der Signalqualität. Die Archivierung ist auf einem beliebigen Datenträger über einen langen Zeitraum hinweg möglich. Ein Monitor dient zur Ausgabe des EEGs, wobei je nach Auflösung des verwendeten Geräts und eingestellter Zeitbasis, die Auflösung des abgeleiteten EEGs verändert werden kann. Gegenüber analogen Systemen bieten digitale Systeme weitere Zusatzmöglichkeiten, wie etwa eine direkte Analyse des Frequenzspektrums.

In der diagnostischen Anwendung können neuronale Funktionen oder mit ihnen verbundene Störungen und Ausfälle anhand des EEGs festgestellt werden. Dies ist möglich, da neurologische Zustände und Aktivitäten über charakteristische Merkmale in den Signalverläufen des EEGs identifizierbar sind. Grundlage für das Neurofeedback und BCI bilden daher die individuell extrahierten und aufbereiteten Merkmale aus den relevanten Signalverläufen, was eine digitale Signalverarbeitung voraussetzt.

Gebräuchliche Merkmale für die Beschreibung von EEG-Signalen sind unter anderem:

- Frequenz
- Amplitude
- Signalverlauf

Beim Neurofeedback werden bevorzugt die in einem EEG-Signal enthaltenen Frequenzanteile für die Generierung des Feedbacks herangezogen. In den enthaltenen Frequenzen spiegeln sich die rhythmischen Aktivitäten des Kortex[8] wider und liefern einen groben Anhaltspunkt über die Aktivierung des Bewusstseins. Insgesamt erstreckt sich der klinisch relevante Frequenzbereich bis etwa 70 Hz.

Die zu erwartenden Amplitudenwerte bei einer nichtinvasiven Ableitung der EEG-Signale an der Schädeloberfläche liegen in einem Bereich von etwa 1 - 100 µV. Invasiv, bei einer Ableitung direkt an der Kortexoberfläche, sind diese Amplitudenwerte ungefähr

[5] ALS - Amyotrophe Lateralsklerose

[6] in den äußeren Bereichen des Körpers befindlich

[7] bewirken, erzeugen; syn. hervorrufen

[8] äußere Schicht des Großhirns

um den Faktor 10 größer. Die Aussagekraft der absoluten Amplitudenwerte ist nur begrenzt verwertbar, da identische Bedingungen zwischen unterschiedlichen Elektrodenmontagen kaum erreicht werden können. Relative Amplitudenänderungen während einer einzelnen Ableitung bieten jedoch eine gewisse Interpretationsmöglichkeit.

Der Signalverlauf ist vorrangig bei der klinischen Befundung von Relevanz. Die Auswertung des EEGs erfolgt anhand von auftretenden Mustern und deren Beziehungen zueinander. Aufgrund der Komplexität erfolgt eine derartige Auswertung zumeist manuell und setzt einen gewissen Grad an Erfahrung voraus oder erfordert einen hohen algorithmischen Aufwand für die Mustererkennung, wie beispielsweise beim BCI.

3. Hardwareentwurf

3.1. Einführung

Der Prototyp des digitalen Elektroenzephalographiesystems wurde in Form einer Erweiterungsplatine für den kombinierten Einsatz mit einer FPGA-Entwicklungsplatine DE2 der Firma Altera entwickelt und realisiert. Ein handelsüblicher Computer ist in der Konzeption für die Steuerung und die sekundäre Überwachung des Gesamtsystems vorgesehen, jedoch nicht für eine digitale Signalverarbeitung. Eine eigens realisierte Anwendungssoftware stellt die hierfür notwendigen Funktionen bereit und ermöglicht zusätzlich die Speicherung der verarbeiteten EEG-Daten auf einem Datenträger des Computers. Die Verbindung für die Kommunikation und den Datentransfer zwischen dem Computer und dem Prototyp stellt eine einzelne USB-Schnittstelle her.

Der gesamte Hardwareentwurf erfolgte unter dem Aspekt der elektrischen Betriebssicherheit, da es sich formalrechtlich bei der realisierten Erweiterungsplatine, respektive dem Gesamtsystem, um ein aktives Medizinprodukt handelt. Entsprechende Anhaltspunkte für die Realisierung ergeben sich aus den Richtlinien und geltenden Gesetzen für medizinische Produkte. Implizit umfasst dies auch die damit verbundenen, gültigen Normen.

Maßgeblich für den Europäischen Raum ist die veröffentlichte EG[9]-Richtlinie 93/42/EWG für Medizinprodukte, sofern es sich nicht um aktive implantierbare Medizinprodukte oder Medizinprodukte für In-vitro-Diagnostika[10] handelt. Für Deutschland ist diese EG-Richtlinie in ein nationales Gesetz, dem Medizinproduktegesetz, umgesetzt worden. Nach §1 MPG[11] ist der Zweck des Medizinproduktegesetzes, den Verkehr mit Medizinprodukten zu regeln und dadurch für die Sicherheit, Eignung und Leistung der Medizinprodukte sowie die Gesundheit und den erforderlichen Schutz der Patienten, Anwender und Dritter zu sorgen. Neben dem Medizinproduktegesetz existieren in Deutschland weitere rechtsverbindliche Verordnungen, die im Falle eines Medizinproduktes zur Anwendung kommen und zur Ergänzung des MPG dienen.

In Bezug auf die mit Medizinprodukten verbundenen Normen existiert auf internationaler Ebene die Normenreihe IEC[12] 60601. Sie stellt den Ausgangspunkt für die entsprechenden Normen auf regionaler und nationaler Ebene dar. Für Deutschland ist dies die Normenreihe DIN[13] EN 60601, auf die im MPG verwiesen wird. Weist ein Medizinprodukt Eigenschaften auf, die in den Anwendungsbereich weiterer EG-Richtlinien fallen, sind bei der Entwicklung, der Herstellung und des Betriebs eines Medizinproduktes auch jene harmonisierten Normen von Relevanz, auf die in diesen EG-Richtlinien verwiesen wird.

3.2. Entwurfsmodell

Das dem Hardwareentwurf zugrundeliegende Entwurfsmodell des Gesamtsystems zeigt Abbildung 3. Der Computer (Modul PC) erfordert im Gegensatz zum Prototyp des digitalen Elektroenzephalographiesystems (Modul Entwurfssystem) keinerlei Entwicklungsaufwand.

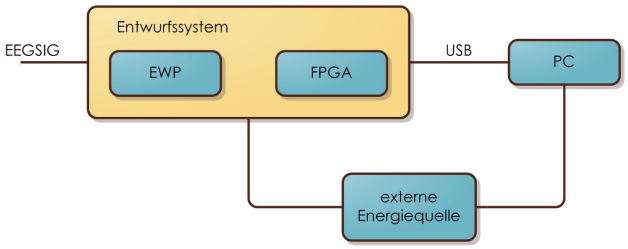


Abbildung 3: Entwurfsmodell des Gesamtsystems

[9] EG - Europäische Gemeinschaft

[10] Befundung von Vorgängen außerhalb des lebenden Organismus

[11] MPG - Medizinproduktegesetz

[12] IEC - International Electrotechnical Commission; dt. Internationale Elektrotechnische Kommission

[13] DIN - Deutsches Institut für Normung

Der Prototyp des digitalen Elektroenzephalographiesystems besteht aus der Erweiterungsplatine (Modul EWP) und der FPGA-Entwicklungsplatine DE2 (Modul FPGA). Die Verbindung der Elektroden mit dem Prototyp ist durch die Schnittstelle EEGSIG repräsentiert, sowie die Verbindung des Prototyps mit dem Computer durch die Schnittstelle USB. Beide Teilsysteme sind für die Versorgung mit einer externen Energiequelle verbunden.

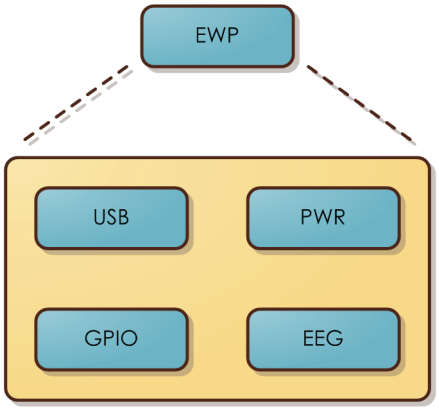


Abbildung 4: Entwurfsmodell der Erweiterungsplatine

Die Konkretisierung der Erweiterungsplatine (Modul EWP) geht aus Abbildung 4 hervor. Die Verbindungen zwischen den Modulen der Erweiterungsplatine sind nicht dargestellt. Verantwortlich für die Spannungsversorgung der Erweiterungsplatine und der FPGA-Entwicklungsplatine zeichnet sich das Modul PWR aus. Das Modul GPIO bündelt zentral sämtliche Signale von und zur FPGA-Entwicklungsplatine. Die notwendige Schnittstellenfunktionalität für die Kommunikation und den Datentransfer mit dem Computer stellt das Modul USB bereit. Durch das Modul EEG erfolgt die analoge Ableitung und Aufbereitung der EEG-Signale, sowie deren anschließende Digitalisierung.

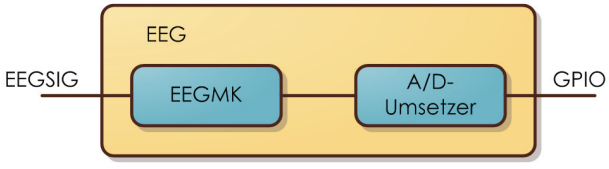


Abbildung 5: Entwurfsmodell des Moduls EEG

Den internen Aufbau des Moduls EEG stellt Abbildung 5 dar. Über die Schnittstelle EEGSIG werden die extern abgeleiteten EEG-Signale direkt der analogen EEG-Messkette (Modul EEGMK) zugeführt und im Anschluss durch das Modul A/D-Umsetzer digitalisiert. Die Übergabe der digitalisierten Werte an das FPGA erfolgt mit Hilfe der Schnittstelle GPIO.

4. FPGA-Design

4.1. Systemaufbau

Das FPGA der FPGA-Enwicklungsplatine DE2 sorgt für die Verbindung aller am Gesamtsystem beteiligten Komponenten und stellt alle notwendigen Funktionalitäten für den Betrieb der Erweiterungsplatine bereit. Zu den Funktionalitäten, die durch das FPGA realisiert werden, zählen:

1. Kommunikation mit dem A/D-Umsetzer für die Konfiguration des Betriebsmodus, der Initiierung einer Abtastung und die Übernahme der resultierenden Abtastwerte.
2. Steuerung, Kontrolle und Fehlerüberwachung der möglichen Betriebszustände des Gesamtsystems. Optische Mitteilung von relevanten Betriebsparametern und aufgetretenen Fehlern über die Statusanzeige mit Hilfe von Leuchtdioden auf der FPGA-Entwicklungsplatine. Auswertung von Statussignalen, die als Betriebsparameter in die Steuerung der Betriebszustände eingehen.
3. Kommunikation mit dem USB-Schnittstellenbaustein für den Empfang von Steuerbefehlen, sowie für den Versand von Status- oder Fehlermitteilungen und der verarbeiteten Abtastwerte über die USB-Schnittstelle an den PC.

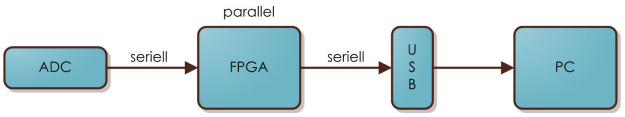


Abbildung 6: Genereller Datenfluss

Der generelle Datenfluss bei der Erfassung von EEG-Signalen entspricht der schematischen Darstellung in Abbildung 6. Die digitalisierten EEG-Signale werden durch das FPGA seriell vom A/D-Umsetzer (ADC) entgegengenommen und parallel, einer im FPGA integrierbaren, digitalen Signalverarbeitungskette bereitgestellt. Nach der Verarbeitung der digitalen EEG-Signale erfolgt der serielle Transfer dieser Daten vom FPGA, über den USB-Schnittstellenbaustein (USB) und die USB-Schnittstelle, an den Computer (PC).

Wie in Abbildung 7 dargestellt, werden bei der Datenübertragung zwischen FPGA und PC, die beiden voneinander unabhängigen Endpunkte A und B des USB-Schnittstellenbausteins FT2232 eingesetzt. Endpunkt A dient sowohl für die Übertragung von Steuerbefehlen an das FPGA, als auch der vom FPGA generierten Status- und Fehlermeldungen an den PC. Der Endpunkt B hingegen, wird ausschließlich für die Übertragung der verarbeiteten EEG-Signale in Richtung des PCs verwendet. Der Empfang von Daten über den Endpunkt B ist nicht vorgesehen.

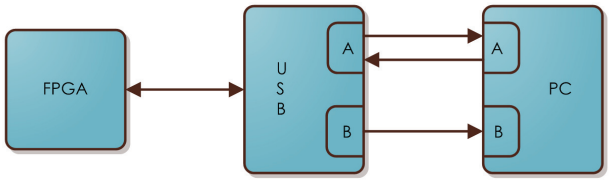


Abbildung 7: Verwendung der beiden USB-Endpunkte

Die Stellung des FPGAs zur gesamten beteiligten Peripherie, inklusive der jeweiligen Signalleitungen, ist zusammenfassend in Abbildung 8 aufgezeigt. Die Verbindung der FPGA-Entwicklungsplatine und der Erweiterungsplatine erfolgt ausschließlich über die frei verwendbaren Signalanschlüsse des FPGAs an der Schnittstelle GPIO. Sämtliche Signalleitungen für die Anbindung der externen Peripherie der Erweiterungsplatine an das FPGA, sowie für die Zustandsmeldung der Versorgungsspannungen des Anwendungsteils und der USB-Schnittstelle auf der Erweiterungsplatine, sind in der Schnittstelle GPIO gebündelt. Als weitere externe Peripherie werden insgesamt fünf Leuchtdioden für eine optische Statusanzeige, sowie ein Taster für die Generierung eines Resetsignals, aus den direkt von der FPGA-Entwicklungsplatine bereitgestellten Ressourcen verwendet.

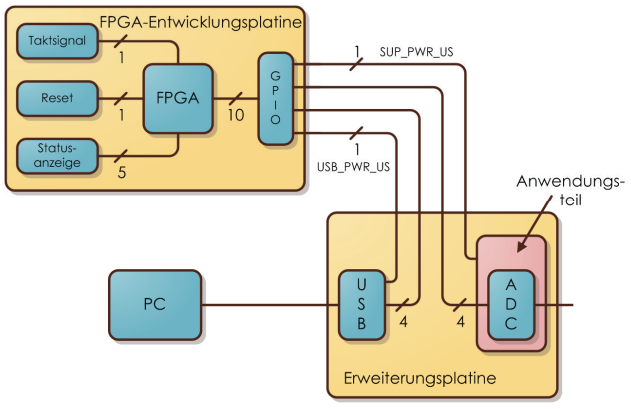


Abbildung 8: Peripherieanbindung an das FPGA

Die Kommunikation mit dem A/D-Umsetzer (ADC) erfolgt nach dem Master-Slave-Prinzip über dessen SPI-Schnittstelle. Dabei fungiert das FPGA als Master, der A/DUmsetzer als Slave. Insgesamt sind vier Signalleitungen für die Anbindung des A/D-Umsetzers an das FPGA erforderlich. Mit dem USB-Schnittstellenbaustein (USB) erfolgt die Kommunikation über dessen „Fast Opto-Isolated Serial Interface“. Diese synchrone Schnittstelle benötigt lediglich vier Signalleitungen, was eine einfache galvanische Entkopplung dieser Schnittstelle ermöglicht. Die serielle Datenübertragung ist durch ein properitäres Protokoll realisiert, welches die Unterscheidungsmöglichkeit zwischen den beiden eigenständigen Endpunkten A und B berücksichtigt. Das FPGA stellt bei diesem properitären Protokoll in beide Richtungen, die steuernde Instanz der Datenübertragung dar.

4.2. Hauptdesign

Der interne Aufbau des Hauptdesigns wird durch das Blockdiagramm in Abbildung 9 wiedergegeben. Insgesamt besteht das Hauptdesign auf dieser Hierarchieebene aus sechs einzelnen Untermodulen, die die jeweils spezifischen Teilfunktionen zur Verfügung stellen. Sämtliche Komponenten des Hauptdesigns befinden sich innerhalb der Taktdomäne des Systemtakts. Zusätzliche Taktfreigabesignale werden, sofern erforderlich, direkt in den jeweiligen Komponenten generiert.

Modul USB

Das Modul USB stellt die steuernde Instanz für die Kommunikation und die Datenübertragung, unabhängig von der Richtung, mit dem USB-Schnittstellenbaustein FT2232 dar. Hierzu steht das Modul USB über vier der Schnittstellensignale des Hauptdesigns in Verbindung mit dem externen USB-Schnittstellenbaustein.

Über das interne Signal USB_TXready signalisiert das Modul USB die generelle Bereitschaft für die Übernahme von Sendedaten. Solange die Übernahmebereitschaft aufrechterhalten bleibt, können dem Modul USB, synchron zum Systemtakt, zum Versand vorgesehene Daten byteweise übergeben werden. Für den Endpunkt A erfolgt dies über die Signale USB_TXAData und USB_writeTXA vom Modul EWP_control aus, für den Endpunkt B über die Signale USB_TXBData und USB_writeTXB vom Modul ADC_Scrambler aus. Das Modul USB regelt den Versand von Daten über die beiden Endpunkte A und B selbstständig. Übernommene Sendedaten werden nach Möglichkeit umgehend an den USB-Schnittstellenbaustein übermittelt. Dem Datenversand über den Endpunkt B kommt, gegenüber dem Datenversand über den Endpunkt A, eine höhere Priorität zu,

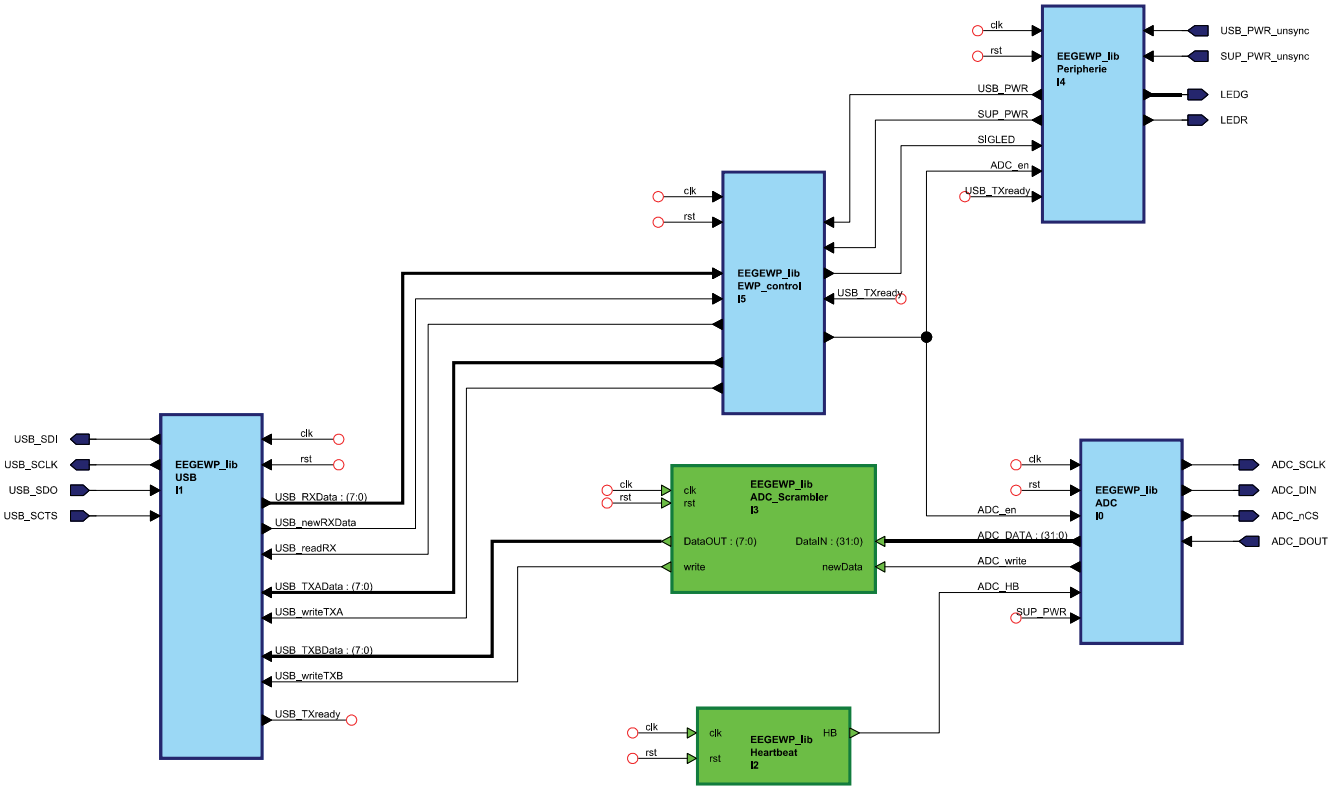


Abbildung 9: Interner Aufbau des Hauptdesigns

da hier bei einer laufenden EEG-Registrierung ein größeres Datenaufkommen auftritt. Die Integrität der Sendedaten ist bei bestehender Übernahmebereitschaft gewährleistet.

Das Modul USB ist ausschließlich für den Empfang von Daten über den Endpunkt A vorgesehen. Über den Endpunkt B empfangene Daten werden stillschweigend verworfen. Stehen neue Daten zur Abholung bereit, wird dies vom Modul USB über das Signal USB_newRXData solange signalisiert, bis alle Empfangsdaten von einem anderen Modul, in diesem Fall durch das Modul EWP_control, übernommen wurden. Die Datenübernahme erfolgt für die Daten des Endpunkts A über die Signale USB_RXDaten und USB_readRX.

Modul ADC

Das Modul ADC regelt die gesamte Kommunikation mit dem A/D-Umsetzer über dessen SPI-Schnittstelle. Hierzu steht das Modul ADC über vier der Schnittstellensignale des Hauptdesigns in Verbindung mit dem externen A/D-Umsetzer. Weiterhin ist das Modul ADC für die Konfiguration des Betriebsmodus, für die Initiierung einer Abtastung beider EEG-Kanäle, sowie für die Übernahme und Bereitstellung der resultierenden Abtastwerte verantwortlich.

Über das vom Modul Peripherie stammende Signal SUP_PWR wird der momentane Versorgungsspannungszustand des Anwendungsteils dem Modul ADC übergeben. Die Initialisierung des A/D-Umsetzers erfolgt in Abhängigkeit von Änderungen dieses Signals autonom. Die Initiierung einer Abtastung ist von den beiden internen Signalen ADC_en und ADC_HB abhängig.

Die generelle Freigabe zur Initiierung einer Abtastung wird durch das Modul EWP_control, die Kontrollinstanz der EEG-Registrierung, über das Signal ADC_en erteilt. Der Abtastzeitpunkt wird periodisch durch das Modul Heatbeat über das Signal ADC_HB festgelegt. Die Initiierung einer Abtastung zu einem bestimmten Abtastzeitpunkt wird kontinuierlich wiederholt, solange die Freigabe erteilt ist. Nach dem Empfang der Abtastwerte beider EEG-Kanäle werden diese Daten über das Signal ADC_DATA, in paralleler Form, den nachfolgenden Modulen bereitgestellt. Die Übernahme der neuen gültigen Abtastwerte wird mit Hilfe des Signals ADC_write angeregt. Das nachfolgende Modul muss in der Lage sein, diese Daten unverzüglich zu übernehmen. Ein ausdrücklicher Handshake erfolgt nicht. Bei der Integration einer zusätzlichen digitalen Signalverarbeitungskette in das FPGA-Design, erfolgt die Ankopplung der Verarbeitungskette über diese beiden Signale.

Modul EWP_control

Das Modul EWP_control stellt alle Funktionalitäten für die Steuerung, Kontrolle und Fehlerüberwachung der möglichen Betriebszustände des Gesamtsystems zur Verfügung. Es regelt anhand der kontinuierlichen Auswertung der internen Statussignale USB_PWR, SUP_PWR und USB_TXready, die von den Modulen Peripherie und USB bereitgestellt werden, die Freigabe einer EEG-Registrierung mit dem Signal ADC_en oder unterbricht eine laufende EEG-Registrierung sobald ein aufgetretener Fehler erkannt wurde. Die Art des aufgetretenen Fehlers entscheidet über das weitere Vorgehen das erforderlich ist, um eine erneute Freigabe der EEG-Registrierung zu ermöglichen. Der Beginn einer EEG-Registrierung kann ausschließlich über die Anwendungssoftware auf dem PC veranlasst werden.

Der Empfang von Daten, wobei es sich ausschließlich um Steuerbefehle handelt, erfolgt über die internen Signale USB_RXData, USB_newRXData und USB_readRX. Stehen neue Empfangsdaten am Modul USB bereit, werden diese übernommen und verarbeitet. Als Reaktion auf einen Steuerbefehl oder bei einem speziellen Fehlerereignis, ist der Versand von Status- und Fehlermitteilungen über die beiden Signale USB_TXAData und USB_writeTXA möglich. Zu Testzwecken besteht zudem die Möglichkeit, den Zustand einer Leuchtdiode der Statusanzeige über Steuerbefehle aus der Anwendungssoftware heraus zu verändern. Der jeweilige Zustand der Leuchtdiode wird dem Modul Peripherie über das Signal SIGLED vorgegeben.

Modul Peripherie

Das Modul Peripherie dient zur Anbindung aller verbleibenden Hardwarekomponenten. Hierzu zählen die Statusanzeige mit den Leuchtdioden und die beiden Statussignale für die Versorgungsspannungszustände der USB-Schnittstelle und des Anwendungsteils. Die externe Verbindung zu den beteiligten Hardwarekomponenten besteht über die Schnittstellensignale USB_PWR_unsync, SUP_PWR_unsync, LEDG und LEDR des Hauptdesigns. Die beiden Statussignale USB_PWR_unsync und SUP_PWR_unsync werden auf den Systemtakt einsynchronisiert und gegebenenfalls entprellt. Die aufbereiteten Eingangssignale werden den anderen Modulen über die entsprechenden internen Signale USB_PWR und SUP_PWR übergeben. Die Steuerung der Statusanzeige wird anhand der Statussignale USB_PWR, SUP_PWR, ADC_en, SIGLED und USB_TXready vorgenommen.

Modul ADC_Scrambler

Das Modul ADC_Scrambler sorgt für die Aufteilung der parallel vorliegenden Abtastwerte eines Abtastzeitpunktes auf einzelne Datenpakete, die unmittelbar über die USB-Schnittstelle an den PC übertragen werden können. Die parallelen Abtastwerte beider EEG-Kanäle werden, sobald neue gültige Werte vorliegen und dies über das Signal ADC_write angezeigt wird, unverzüglich über das Signal ADC_DATA vom Modul ADC entgegengenommen. Entsprechend den Übertragungseigenschaften des Moduls USB, werden die parallelen Daten auf vier einzelne Datenpakete aufgeteilt und byteweise, direkt aufeinander folgend, über die Signale USB_TXBData und USB_writeTXB dem Modul USB, für den Versand über den Endpunkt B zum PC, übergeben. Die Reihenfolge der Datenpakete ist auf die Speicherorganisation des PCs angepasst, damit die Werte der beiden EEG-Kanäle ohne eine weitere softwareseitige Manipulation verwendet werden können.

Modul Heartbeat

Das Modul Heartbeat generiert mit der geforderten Abtastfrequenz von 4 kHz einen periodischen Impuls, der innerhalb des Gesamtsystems den eindeutigen Abtastzeitpunkt festlegt. Über das Signal ADC_HB wird dadurch das Modul ADC veranlasst eine Abtastung der beiden EEG-Kanäle zu initiieren.

5. Literatur

[1] Demos, John N.: Getting Started with Neurofeedback. New York, London: W. W. Norton & Company, 2005

[2] Zschocke, Stephan: Klinische Elektroenzephalographie. 2. Auflage. Heidelberg: Springer Medizin Verlag, 2002

[3] Cooper, R.; Osselton, J. W.; Shaw, J.C.: Elektroenzephalographie. 3. Auflage. Stuttgart: Gustav Fischer Verlag, 1984

[4] Eichmeier, Josef: Medizinische Elektronik - Eine Einführung. 3. Auflage. Berlin, Heidelberg: Springer-Verlag, 1997

[5] Meyer-Waarden, Karsten: Bioelektrische Signale und ihre Ableitverfahren. Stuttgart, New York: Schattauer, 1985

[6] Prutchi, David; Norris, Michael: Design and Development of Medical Electronic Instrumentation - A Practical Perspective of the Design, Construction, and Test of Medical Devices. New Jersey: John Wiley & Sons, Inc., 2005

Open Source-/ Freeware IC Design Flow am Beispiel einer DPLL

[1]Mario Menges, [2]Alexander Schmidt, [1]Prof. Dr.–Ing. W. Bonath

[1]Fachhochschule Gießen-Friedberg, Wiesenstraße 14, 35390 Gießen

[2]Fraunhofer-Institut für Mikroelektronische Schaltungen und Systeme (IMS), Finkenstraße 61, 47057 Duisburg

E-Mail: [1]mariomenges@arcor.de [2]alexander.schmidt@ims.fraunhofer.de

Dieser Beitrag befasst sich mit der Auswahl und dem Einsatz von Open Source und Freeware Tools für IC-Design. Anhand einer rudimentären digitalen PLL und eines Ringoszillators wird exemplarisch der komplette Entwicklungsprozess vom elektrischen Entwurf, der Schaltungssimulation bis hin zum Erzeugen des Layouts demonstriert. Eingesetzt werden dabei die Programme XCircuit zum Zeichnen der Schaltpläne und dem Generieren der Netzliste, DUSpice respektive LTSpice / Switcher CAD (beide basierend auf Berkeley Spice 3f5) zur Simulation sowie LASI zum Erstellen des Layouts. Als Technologie wird ein 0.35 µm CMOS Prozess verwendet.

Grundlage des vorliegenden Beitrages ist die Diplomarbeit von Mario Menges an der FH-Gießen-Friedberg. Die Implementierung eines Design Rule Checks (DRC) wurde von Alexander Schmidt im Rahmen einer Studienarbeit realisiert. Herr Schmidt ist heute Doktorand am Fraunhofer IMS in Duisburg.

Hintergrund und Ziel des Projektes ist der Verzicht auf teure, sehr einarbeitungszeitintensive kommerzielle Software in Hochschulpraktika und das Etablieren alternativer Software, die auch privat von Studenten auf deren PC verwendet werden kann.

1. Einleitung

Der vorliegende Beitrag basiert auf einer Diplom- und einer Studienarbeit und stellt einige deren Themen dar. In dem vorliegenden Beitrag wurde eine komplette Entwicklungsumgebung für integrierte CMOS Schaltungen zusammengestellt, die ausschließlich aus Open Source- / Freeware besteht. Mit Hilfe der ausgewählten Tools soll exemplarisch deren Praxistauglichkeit, u.a. für den Einsatz im Rahmen von Hochschulpraktika, am Beispiel einer DPLL überprüft werden. Die Programme werden in diesem Beitrag kurz vorgestellt und deren Vor- und Nachteile aufgezeigt. Primär werden in diesem Beitrag die Entwurfswerkzeuge betrachtet. Das Schaltungsdesign der DPLL hat hier einen exemplarischen Charakter.

Weiterhin wird die DRC-Datei für den Design Rule Check des verwendeten Layout Tools LASI und den eingesetzten 0,35 µm-Prozess entwickelt und anhand eines Beispiels kurz vorgestellt. Auf Grund der Komplexität und der großen Anzahl der prozessspezifischen Design Rules [2] konnte dieser Check nur in Teilen fertiggestellt werden.

2. Theoretische Grundlagen

2.1. Verwendeter Prozess

Im Rahmen des Beitrages und der zugrundeliegenden Diplomarbeit wurde der Alcatel 0,35 µm CMOS C035-D Prozess verwendet. Bei diesem handelt es sich um einen Prozess mit fünf Metalllagen. Er ermöglicht eine Dichte von 15.000 Gates / mm^2, die Versorgungsspannung beträgt 3,3 V. Die zugehörigen Model Cards lagen für Mentors ELDO Simulator vor und wurden für die Verwendung von DUSpice / LTSpice angepasst.

Die Schaltungen wurden mittels BSIM3-Modellen simuliert.

2.2. PLL

Phasenregelkreise (PLLs) sind aus vielen Anwendungen in der Kommunikationstechnik nicht mehr wegzudenken. Zum Einsatz kommen sie u.a. zur Taktrückgewinnung (Clock recovery) in

Übertragungskanälen, zur Frequenzsynthese (z.B. in Synthesizern) und zur Trennung von Signalen von überlagertem Rauschen. Auch in Radios zur Amplituden- und Frequenzdemodulation (AM / FM) und Fernsehgeräten zur Wiedergewinnung der Farbinformation spielen PLL eine wichtige Rolle. Unterschieden wird zwischen analogen und digitalen PLLs, wobei sich die digitalen weiter in DPLLs (Digital PLLs) und ADPLLs (All Digital PLLs) klassifizieren lassen. Dieser Beitrag befasst sich ausschließlich mit dem Entwurf einer DPLL.

Der Aufbau und die Bestandteile der hier eingesetzten DPLL sind in Abbildung 1 zu sehen.

Das Eingangssignal „data“ wird mit dem vom VCO erzeugtem Ausgangssignal hinsichtlich der Phasenlage vom Phasendetektor verglichen. Weicht die Phase ab, entsteht eine entsprechende zur Phasenabweichung proportionale Steuerspannung am Phasendetektor, welche über den Loop Filter dem VCO zugeführt wird, und diesen ansteuert. Über die Steuerung wird der VCO dazu veranlasst, entweder schneller oder langsamer zu schwingen, und zwar solange, bis das Ein- und Ausgangssignal phasen- und frequenzstarr miteinander verkoppelt ist. Vom ausgerasteten Zustand spricht man, wenn keine phasen-/frequenzstarre Kopplung zwischen den Signalen vorliegt.

Als Phasendetektor kommt ein XOR-Gatter zum Einsatz, bei dem VCO handelt es sich um eine stromgesteuerte Variante eines Ringoszillators.

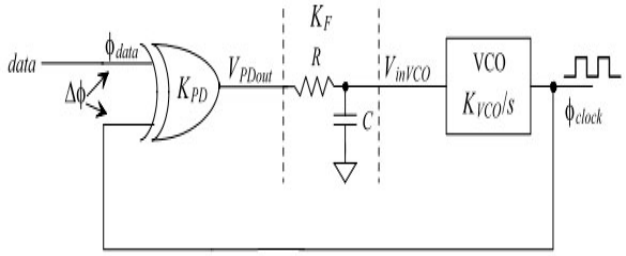


Bild 1: Prinzipschaltbild DPLL [1]

3. Verwendete EDA-Tools

3.1. Auswahl und Auswahlkriterien

Ein Bestandteil und Startpunkt der Arbeit war es, eine komplette Design-Umgebung für die Durchführung der Schaltungsentwicklung auszuwählen. Als Kriterium galt dabei, dass es sich um Programme aus dem Open-Source oder Freeware Bereich handeln muss. So soll gewährleistet werden, dass die Programme bei einem späteren Einsatz für hochschulinterne Laborpraktika für die Studenten frei verfügbar sind und somit auch außerhalb der Labore eingesetzt werden können. Weiterhin wurde als Auswahlkriterium definiert, dass möglichst umfangreiche Dokumentationen der Programme zur Verfügung stehen und entsprechender Support bereit steht. Bei diesen alternativen und kostenlosen Programmen existiert der Support jedoch leider in der Regel nur in Form von Nutzerforen im Internet, und weniger seitens der Programmierer. Weiterhin wurde bei der Selektion Wert auf Benutzerfreundlichkeit und Funktionsumfang gelegt. Die Einarbeitungszeit sollte im Vergleich zum kommerziellen und für die Praktika häufig überdimensionierten Tools gering sein. Ebenfalls von Bedeutung ist, dass regelmäßige Produktupdates seitens des Entwicklers gegeben sind. Ausschlusskriterium war die Verfügbarkeit für Betriebssysteme. Die Verfügbarkeit für Microsoft Windows sollte zwingend gegeben sein, Versionen für Linux oder Mac sind optional.

Nach einschlägiger Recherche fiel die Wahl auf die Programme XCircuit (Schaltplaneditor), DUSpice und LTSpice (Simulator) sowie LASI (Layouteditor).

3.2. XCircuit

Bei XCircuit handelt es sich um Open Source, es existiert eine Windows sowie eine Linux Version. Das Programm ist ressourcenschonend (nur wenige MB groß), leicht zu bedienen und bietet eine Fülle von Funktionen. Eine ausführliche Dokumentation ist in Form einer im Programm integrierten Hilfe sowie als Tutorial gegeben.

Im Tool sind bereits Bauteilbibliotheken implementiert, wobei es sich allerdings lediglich um die Symbole der Bauteile (Logik-Gatter wie NAND, XOR etc.) handelt, die mit keinerlei Funktion belegt sind. Jedoch besteht die Möglichkeit für den Benutzer, seinen Schaltungen diese Symbole zuzuweisen und somit mit einer Funktion zu belegen. Alternativ können auch eigene Symbole gezeichnet werden, mit einer Schaltung (Schematic) verknüpft und in einer User-Bibliothek zur Wiederverwendung abgelegt werden. Alle Elemente wie Widerstände, Kapazitäten, MOSFETs usw. können parametrisiert werden. Eine Beschriftung der Bauteile kann sowohl automatisch (durch fortlaufende Nummern), als auch manuell erfolgen. Der fertige Schaltplan kann anschließend als Spice-Netzliste (Textfile im ASCII-Format) exportiert werden, wobei sowohl Bauteilbezeichner als auch die Parameter mit exportiert werden. Wurden Schematics mit Symbolen

verknüpft (d.h. eigene Bauteile in der Bibliothek abgelegt), so werden diese als Subcircuits exportiert.

3.3. DUSpice, LTSpice / SwitcherCAD III

Beide Programme basieren auf dem an der Universität von Berkeley entwickelten Spice, welches mittlerweile in der Version 3f5 vorliegt. Bei Spice 3f5 handelt es sich um ein Simulator auf Kommandozeilenbasis. Der Vorteil von DUSpice gegenüber Spice 3f5 ist lediglich die Tatsache, dass eine rudimentäre grafische Bedienoberfläche zum Plotten der Messungen implementiert wurde, darüber hinaus sind keine weiteren Funktion mehr ergänzt worden. Eine nützliche Funktion ist mit dem Export des Plots als Excel-Sheet gegeben.

Beiden Tools ist gemeinsam, dass sie kostenlos verfügbar sind, und auch für kommerzielle Zwecke genutzt werden dürfen. Weiterhin sind beide schnell und problemlos zu installieren.

LTSpice / SwitcherCAD III der Firma Linear Technologie besitzt gegenüber DUSpice, abgesehen von der komplett grafischen und bedienerfreundlichen Benutzeroberfläche, einige Vorteile. Es verfügt über eine große Bibliothek von Bauteilen, die über Linear Technologie zu beziehen sind. Für den Entwurf von integrierten Schaltkreisen spielt dies jedoch keine Rolle. Weiterhin bietet das Tool einen erweiterten Funktionsumfang gegenüber Spice. So können beispielsweise Parameter mittels Sweep-Befehl variiert werden und die zugehörigen Plots gemeinsam in einem Diagramm dargestellt werden. So ist es zum Beispiel möglich, den Einfluss von Temperaturschwankungen auf Schaltungen zu simulieren und darzustellen. Seine wahren Stärken kann LTSpice/Switcher Cad jedoch im Postprocessing, d.h. dem Auswerten der Simulationsdaten, zeigen. So stehen Cursor zur Verfügung, mit denen man zu vermessende Bereiche des Plots anfahren kann. Das Tool gibt dann die x- und y-Parameter und die Differenzen *dx* und *dy* aus, um beispielsweise Periodendauer und Frequenz des vermessenen Signals zu bestimmen (wird automatisch vom Programm errechnet). Auch Flankensteilheiten etc. können so schnell und exakt bestimmt werden.

Des Weiteren können den einzelnen Kurven verschiedene Farben zugewiesen werden und die Skalierungen der x- und y-Achsen zwischen linear und logarithmisch umgeschaltet werden. Auch kann das Tool nach Anwahl eines Spannungs- oder Stromverlaufs über der Zeit automatisch dessen Mittelwert und RMS-Wert ermitteln sowie mittels FFT (Fast Fourier Transformation) das Frequenzspektrum ausgeben. Die Netzliste zeigt LTSpice dank Syntax-Highlighting besonders übersichtlich an. Befehle, Bauteile, Parameter und Kommentare werden also entsprechend der Konfiguration des Benutzers in verschiedenen Farben angezeigt. Auch der komplette Verzicht auf den separaten Schaltplaneditor ist möglich, LTSpice verfügt über einen eigenen integrierten Editor [4].

3.4. Lasi

LASI ist ein leistungsfähiges Freeware-Tool für Windows-Systeme. Trotz der geringen Dateigröße von wenigen MB bietet es ähnliche Möglichkeiten, die kosten- und einarbeitungszeitintensive kommerzielle Programme wie Mentor Graphics IC-Station oder Virtuoso der Firma Cadence dieser Art auch bieten. Zu nennen sind beispielsweise Optionen wie LVS-Check (Layout Versus Schematic), DRC, automatische Extraktion von parasitären Widerständen und Kapazitäten und Autorouting (automatisches Verbinden von Zellen im Layout auf Basis der zugehörigen Schaltpläne).

LASI beinhaltet eine umfangreiche und hervorragende Hilfefunktion, im Internet findet man nützliche Tutorials und ein Forum für weitere Hilfestellungen. Produktupdates finden regelmäßig ca. alle 2 Monate statt.

Für das LASI Tool gibt es leider seitens der Halbleiterhersteller keine fertigen Design-Kits, die u.a. vorgefertigte Standard-Zell-Bibliotheken und das File für den DRC enthalten. Daher ergibt sich die Notwendigkeit, sowohl die Zellen als auch den DRC eigenständig zu entwerfen. Dies war unter anderem ein weiteres Ziel der Diplom-/Studienarbeit, auf der dieser Beitrag basiert. Nach relativ kurzer Einarbeitungszeit ist es möglich, das benötigte DRC-File für den jeweiligen Prozess zu programmieren bzw. die Beispiel-Files im LASI Home-Verzeichnis entsprechend an die prozessspezifischen Design-Rules [2] anzupassen. Die Programmierung erfolgt dabei mittels relativ rudimentären, der Maschinensprache Assembler ähnlichen Anweisungen. Detaillierte Informationen dazu findet man in der programminternen Hilfe.

Die gezeichneten Zellen werden von LASI im TLC-Format abgespeichert, einem XML (Extensible Markup Language) basierten Format. Um die Zellen mit anderen Programmen wie Mentor Graphics IC-Station öffnen und bearbeiten zu können, müssen diese als GDSII-File (Graphic Data System II) exportiert werden. Gleiches gilt auch für das Tape-out, d.h. für die Übermittlung der Daten zum

Maskenhersteller. Ein Export in das CIF-Format ist ebenfalls möglich [4].

4. Entwurfsablauf

4.1. Elektrischer Entwurf / Simulation

Um den Entwicklungsaufwand einfach zu halten, wurde eine DPLL-Schaltung, bestehend aus XOR-Phasendetektor, passivem Loop Filter und stromgesteuertem VCO ausgewählt. Als Anwendung wurde kein spezielles Einsatzgebiet festgelegt, denkbar wäre jedoch eine Taktrückgewinnung für ein RZ-codiertes Signal. Die Mittenfrequenz der Schaltung wurde auf 100 MHz festgelegt. Entsprechende Testeingangssignale wurden in SPICE mittels der PWL-Funktion (Piece Wise Linear) erzeugt. Da dieses für die benötigten Signale (Rechteckspannung mit Frequenzsprung) manuell nur äußerst umständlich und zeitaufwändig möglich war, wurde eigens ein rudimentäres C-Tool programmiert. Dieses erzeugt nach Eingabe zweier Frequenzen, Amplitude und einer Zeitdauer, nach der das erzeugte Rechtecksignal der 1. Frequenz auf die 2. Frequenz wechselt, die gewünschten Parameter für die PWL-Spannungsquelle. Alle Komponenten wie XOR, stromgespeiste Inverterstufe etc. wurden als separate Subcircuits ausgeführt. So kann die Anzahl der Inverterstufen des VCO beliebig und flexibel angepasst und leicht parametrisiert werden.

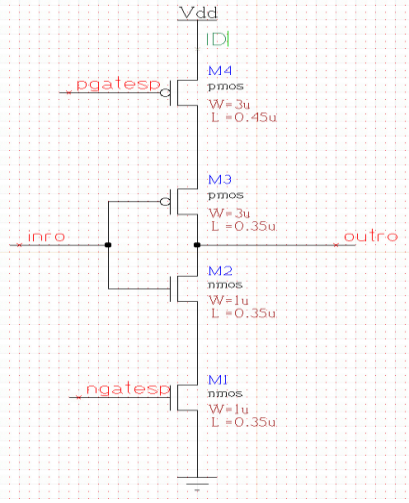


Bild 2: Stromgesteuerte Inverterstufe

Abbildung 2 zeigt eine in Xcircuit gezeichnete Inverterstufe. Die Gates des oberen und unteren MOSFET sind mit den jeweiligen Gates der MOSFETs M3 und M1 der untenstehenden Eingangsschaltung verbunden. Diese Transistoren dienen als Stromquelle, um den Inverter in der Mitte zu speisen. Eingestellt wird der Strom mittels der Stromspiegel in der Eingangsschaltung, d.h. über die *W/L* – Verhältnisse der MOSFETs M4 und M3 respektive über den Widerstand R_{range}.

Erhöhen des Stromes bewirkt eine höhere Oszillationsfrequenz.

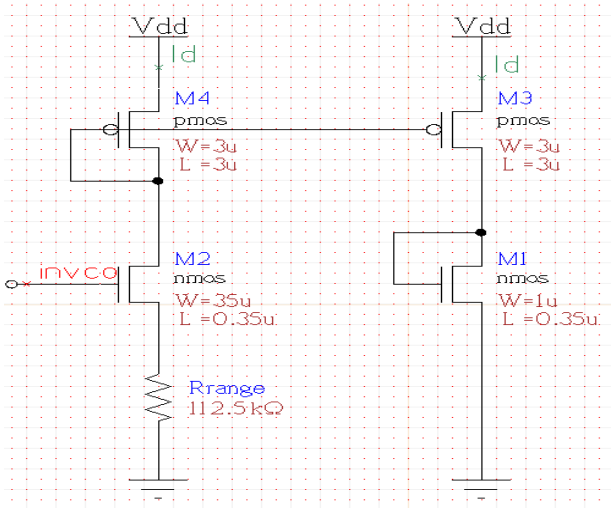


Bild 3: Eingangsschaltung des VCO

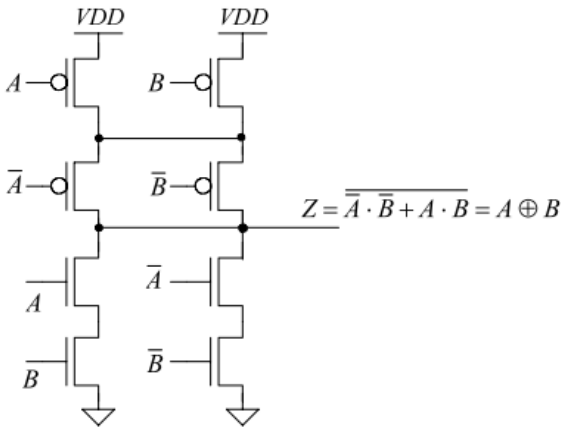


Bild 4: XOR-Phasendetektor [1]

Um die gewünschte Mittenfrequenz zu erreichen, wurden 7 Inverterstufen in Serie geschaltet, die jeweils mit rund 15 µA gespeist werden.

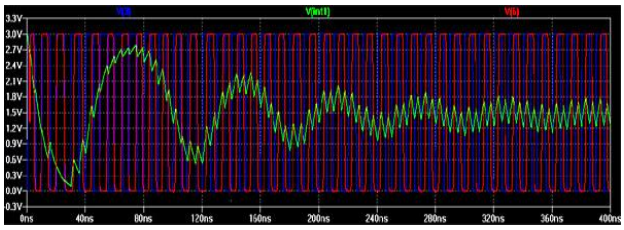


Bild 5: Simulation DPLL: Steuerspannung

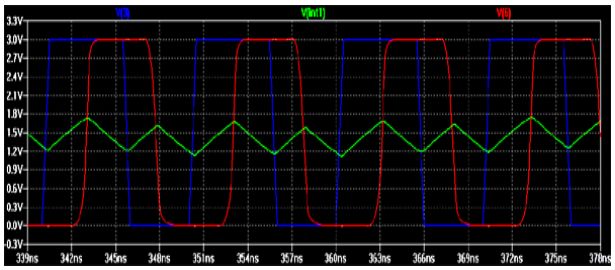


Bild 6: DPLL: PLL eingerastet

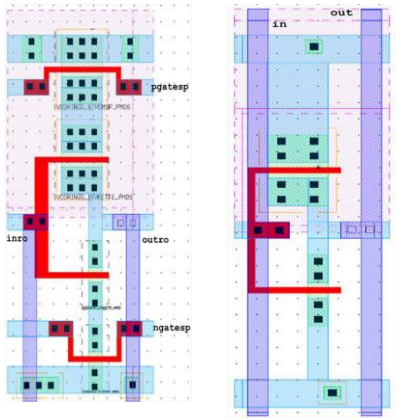


Bild 9: VCO-Inverterstufe und Inverter

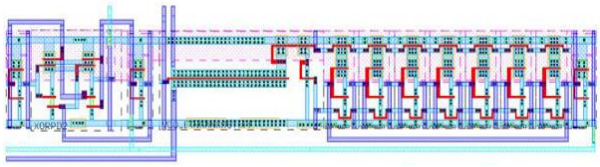


Bild 10: Gesamtlayout DPLL

4.2. Layout, Resimulation, Redesign

Die einzelnen Komponenten wurden mit den Geometrieparametern aus dem elektr. Entwurf, wie auf nachfolgenden Bildern zu sehen, mit LASI gelayoutet.

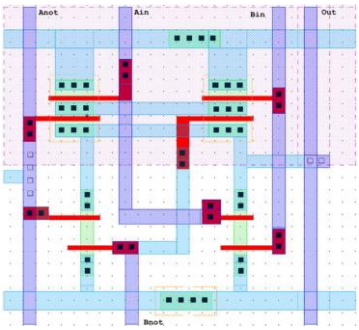


Bild 7: XOR-Phasendetektor

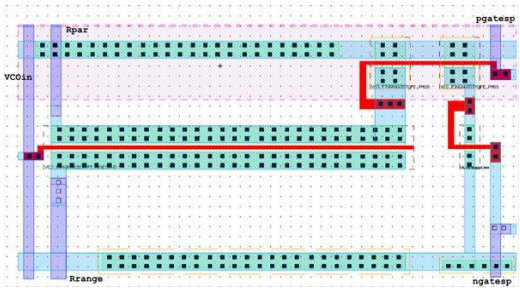


Bild 8: VCO Eingangsstufe

Jedes Layout einer Komponente entspricht einem Subcircuit in der Spice-Netzliste. Auf Grund von Problemen mit der automatischen Extraktion parasitärer Widerstände und Kapazitäten, die bis dato ungeklärt sind, wurden die Parasitäten hilfsweise per Handkalkulation ermittelt.

4.3. Design Rule Check

Zur Überprüfung von Design Rules verwendet LASI PCX - Grafiken, welche während des DRC erstellt und ausgewertet werden. So werden die Vektordaten der unterschiedlichen Layer auf ein Pixelfeld (Bitmap) umgesetzt (gemappt), wo sich dann durch logische Funktionen wie AND, OR und XOR die Pixel der verschiedenen Layer untereinander verknüpfen lassen. Auf diese Weise ist es möglich, einen Bereich des Layers x zu wählen, der innerhalb oder außerhalb des Layers y liegt. Die durch diese Maskierung selektierten Layer lassen sich dann auf Breite oder Abstand hin vermessen. Tritt eine Verletzung der Design Rules auf, ist es möglich, den fehlerhaften Bereich farblich gekennzeichnet in einer PCX - Datei abzulegen und anschließend zu korrigieren. Zur Selektion und Auswertung der einzelnen Layer verwendet LASI einen der Maschinensprache Assembler sehr ähnlichen Code. Auch die Verwendung von Registern, einem kleinen Stack zum Ablegen von Masken sowie die PUSH und POP Funktionen erinnern an die Maschinensprache. Eine recht ausführliche Erläuterung der genauen

Funktionsweise sowie des Befehlssatzes kann in der LASI Hilfe eingesehen werden. Zur Implementierung einer einzelnen Design‑Regel sind die folgenden Schritte notwendig:

1. „mappen“ aller in die Regel einbezogenen Layer. Hier werden alle zu prüfenden Layer sowie die für die Maskierung notwendigen Layer mit dem Befehl *MAP,layer_number* in eine Bitmap gewandelt.
2. Auswählen der Layer direkt oder mit Hilfe von Masken. Es muss darauf geachtet werden, dass immer eindeutig selektiert wird, um Fehler zu vermeiden.
3. Prüfen der Breite oder Distanz. Dabei lassen sich alle notwendigen Messungen durch die Prüfung auf Breite oder Distanz abbilden.
4. Darstellen der Fehler, sowie das Einblenden von zusätzlichen Layern, welche die optische Lokalisierung des Fehlers erleichtern sollen. Alle zusätzlichen Layer müssen jedoch zu Beginn gemappt werden.

Zur Überprüfung aller Regeln müssen diese vier Schritte für jede Regel durchgeführt werden. Anhand von zwei kleinen Beispielen soll nun das Auswählen von Layern und das Maskieren erläutert werden.

Auf minimale Breite prüfen:

```
MAP, 2          ;Layer 2 mappen
PUSH, 2         ;Layer 2 > R-Register
NOTR            ;Inverse von Layer 2 > R-Register
EXPR            ;Layer 2 um Distanz ausweiten
DSP,2,G,2       ;Layer 2 in grün anzeigen
DSPR,W,1        ;Fehler in weiss anzeigen
```

Minimale Distanz zwischen zwei Layern:

```
MAP, 1          ;Layer 1 mappen
MAP, 2          ;Layer 2 mappen
PUSH, 1         ;Layer 1 > R-Register
PUSH, 2         ;Layer 2 > R und Layer 1 > S
EXPRLRS         ;Layer 1 und 2 um Distanz ausweiten
DSP,1,G,2       ;Layer 1 in grün anzeigen
DSP,2,R,2       ;Layer 2 in rot anzeigen
DSPR,W,1        ;Fehler in weiss anzeigen
```

Die Funktionen der Befehle EXPR und EXPRLRS spielen hier eine entscheidende Rolle für das Verständnis der Abstands- und Distanzprüfung. Für eine weiterführende Erläuterung wird an dieser Stelle auf die LASI Dokumentation verwiesen.

Die Anwendung des DRC empfiehlt sich nach jedem Layoutschritt, um nachträgliches Redesign größerer Bereiche zu vermeiden. Um die Rechenzeit der Checks zu optimieren, kann die Resolution der Checks angepasst werden [3].

5. Fazit / Ausblick

Es hat sich gezeigt, dass die ausgewählten und eingesetzten Tools aus dem Open-Source und Freeware Bereich für einen Einsatz im Hochschulbereich sehr gut geeignet sind. Als Alternativen für kommerzielle und teure Programme, die darüber hinaus auf Grund ihrer immensen Komplexität enormen Zeitaufwand zur Einarbeitung erfordern, eignen sie sich hervorragend. Der größte Vorteil den sie bieten ist jedoch, dass sie nicht an den Einsatz auf hochschulinternen Rechnern auf Grund der Lizenzen gebunden sind. So können die Tools auf jedem gängigem PC eingesetzt werden. Eine entsprechende Nutzung der Programme außerhalb der Hochschulpraktika ist somit gegeben, was zur effizienteren Vor- und Nachbereitung auf die Inhalte von Praktikum und Vorlesung beiträgt.

Die verwendeten kostenlosen Programme verfügen ebenfalls über einen enormen Funktionsumfang, der den Ansprüchen der Praktika genügt und teilweise darüber hinausgeht. Der Einarbeitungszeitaufwand für die praktikarelevanten Kenntnisse ist jedoch im Verhältnis zu kommerziellen Tools als gering einzustufen. Das Layout-Tool LASI konnte bereits erfolgreich im Mikroelektronik-Praktikum der Fachhochschule Gießen-Friedberg eingesetzt werden.

Bisher war es noch nicht möglich, den LVS-Check und die automatische Extraktion der parasitären Widerstände und Kapazitäten einzusetzen. Auch der DRC müsste weiter vervollständigt werden. Hier bieten sich zwei Ansatzpunkte für weitere Arbeiten an.

6. Literaturverzeichnis

[1] Baker, R.: CMOS – Circuit Design, Layout and simulation, Wiley-Interscience 2005

[2] Alcatel Microelectronics: C035M-D Design-Rule Manual (DS 13330), 1999

[3] Schmidt, A.: Integration eines Design Rule Checks (DRC) in die Freeware Layout CAD ‐ Software LASI, FH Gießen-Friedberg, 2009

[4] Menges, M.: Entwicklung und Integration eines Ringoszillators und einer DPLL in 0.35µm CMOS-Technologie im Open Source-/Freeware IC-Design Flow, FH Gießen-Friedberg, 2009

Der UMC 0.18 Design Flow am Beispiel eines PDA-Prozessor ICs

M.Sc. Daniel Bau

Prof. Dr.-Ing. Dirk Jansen

Hochschule Offenburg, Badstraße 24

0781/ 205 365, asicadmin@fh-offenburg.de

Im ASIC Design Center der Hochschule Offenburg wird ein Design Kit für die UMC 0.18µm Faraday Technologie aufbereitet. Dabei werden alle benötigten Dateien, welche für einen zunächst rein digitalen Chipentwurf unter Verwendung der Synopsys, Cadence und Mentor Tools benötigt werden, für den UMC 0.18µm Prozess zusammengestellt.

1. Beispielschaltung

Als digitale Schaltung wird ein bereits an der Hochschule Offenburg entworfenes PDA-Prozessor-Design verwendet. Die Beispielschaltung wurde in vorangegangener Arbeit in einem FPGA erfolgreich emuliert. Das Herzstück des PDA-Systems ist der SIRIUS Softcore Prozessor, welcher als VHDL-Code vorliegt und in jede beliebige Zieltechnologie portiert werden kann. Zur Kommunikation mit der Außenwelt sind weitere Peripherien wie SPI-Controller, Timer, Audio-Unit und Interrupt-Controller integriert. Bild 1 zeigt das Blockschaltbild der im ASIC integrierten Schaltungen und seiner extern angeschlossenen Komponenten. Der Systemtakt wird über einen internen vom UMC Design Kit bereitgestellten PLL generiert. Für den Übergang des vorhandenen FPGA-Designs in die UMC-Technologie musste im wesentlichen der Arbeitsspeicher ausgetauscht werden.

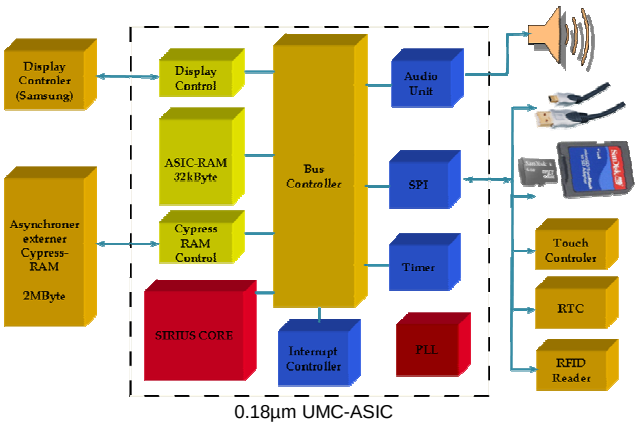


Bild 1: Blockschaltbild des PDA-Prozessor-ICs

Als Speicher wird ein 32kByte Single Port RAM eingesetzt. Die dafür benötigte Daten werden unter Angabe der gewünschten RAM-Konfiguration von Europractice generiert.

2. UMC 0.18µm Design Kit

Der von Europractice angebotene UMC 0.18µm Design Kit wurde von der Firma Faraday entworfen. Die für einen Chipentwurf benötigte Informationen wie Technologie-Daten für Synopsys, Vital-Codes für die Simulation, LEF-Beschreibung der Zellen, Rule-Files für die Verifikation der gerouteten Zelle mit Calibre und Prozess-Dateien für das Routen mit IC-Station sind im Design Kit enthalten. Gegebenenfalls sind die LEF-Files für ein einwandfreies Routen mit Encounter anzupassen. Die GDSII-Beschreibung der Zellen sind im Design Kit aufgrund von Copyrights nicht enthalten. Eine Teilbeschreibung der Zellen, wobei die Anschlußpins und die verwendten Metal-Lagen beschrieben sind, liegt im LEF-Format vor. Um das Portieren des Layouts von Encounter zu IC-Station zu ermöglichen, wurden ausgehend von den LEF-Files die GDSII Beschreibungen erstellt und entsprechende Layer-Mapping Files angefertigt.

2.1. Technolgievergleich

Die zuletzt an der Hochschule Offenburg in zahlreichen erfolgreichen Chipdesigns eingesetzte 0.35µm Technologie der Firma AMIS soll hier als Vergleich herangezogen werden. Der UMC 0.18µm Technologie steht eine zusätzliche Metallschicht zur Verfügung, welche jedoch eine Mindestbreite von 20µm aufweisen muss und gegebenenfalls für Power-Routen benutzt werden kann. Der Unterschied der minimalen Leiterbahnbreite der beiden Technologien liegt bei ca. 0,4µm, was ein kompakteres Routen zulässt. Tabelle 1 stellt den Flächenvergleich von gängigen Zellen gegenüber. Zusätzlich bietet der UMC Design Kit dem Entwickler verifizierte IP Cores an. Ein konfigurierbarer PLL, DLL, differentieller Empfänger, USB Transceiver, sowie spezielle IO-Zellen sind für Hochschulen ohne Aufpreis verfügbar.

Tab.1: Flächenvergleich der Technologien

Zelle	UMC 0.18 [µm]	AMIS 0.35 [µm]	Faktor
Memory	8kWord: 563x1391	8kByte: 1366x1960	6,8
2 Input AND	2,48x5,04	6x12	5,76
2 Input MUX	4,34x5,04	10,5x12	5,76
2 Input NOR	1,86x5,04	4,5x12	5,76
2 Input OR	2,48x5.04	6x12	5,76
D Flip-Flop	10,54x5,04	21x12	4,74
IO Input Buffer	62,6x140,1	144x183	3

3. Entwurfsablauf

Der in Abbildung 3 dargestellte Design Flow zeigt welche Programme zum Einsatz kommen und welche Dateien für die einzelnen Programme benötigt werden. Für das eingesetzte Mikrocontrollersystem SIRIUS existiert ein SIRIUS-Design-Kit, welcher verschiedene Softwaretools zur Verfügung stellt. Unter anderem eine Entwicklungsumgebung, welche es ermöglicht einen C-Code in ein Hex-File zu compilieren und es mit in die VHDL-Simulation miteinzubinden.

3.1 Synthese auf Zieltechnologie

Für die Synthese mit Design Vision von Synopsys werden die Technologiebeschreibung der Logikgatter verwendet. Die im UMC Design Kit enthaltene Dateien sind sowohl als ASCI Datei im ".lib"-Format als auch in compilierter Form im synopsysspezifischen "*.db*-Format vorhanden. Die daraus resultierende Netzliste wird im Verilog-Format für das nachfolgende Layout Programm Encounter herausgeschrieben und als generische VHDL-Netzliste, welche sich lediglich in der Namensgebung der Standardzellen unterscheidet. Ausgehend von den UMC-Standardzellen sind unter Ausschluß von speziellen Zellen die Grundfunktionen der Digitaltechnik wie AND, OR, XOR, NOR, MUX, INV und D-FlipFlop nachprogrammiert. Mithilfe dieser Grundzellen und der generischen Netzliste ist eine Umsetzung in jede beliebige Zieltechnolgie realisierbar. Der ASIC-Entwickler hat dadurch die Möglichkeit, die digitale Zellen seiner Schaltung mit geringsten strukturellen Abweichungen auf einem FPGA zu verifizieren.

3.2 Von der Netzliste zum Chiplayout

Die Verilog-Netzliste, welche man aus der Synthese erhält, kann direkt zum Routen über das CAE-Tool Encounter der Firma Cadence eingelesen werden. Das Tool ist im platzieren der in der Netzliste verankerten Zellen, routen, erneute Taktsynthese, Clock-Tree-Analyse und layouten komfortabel zu bedienen. Durch einen ausgereiften Platzier- und Routingalgorithmus können dicht geroutete Design erzielt werden, um möglichst keine teuere Siliziumfläche zu verschwenden. Nach dem Platzieren, der Taktsynthese und dem Beenden des Routens lässt man sich die um Buffer- und Inverterzellen erweiterte timing-optimierte Netzliste herausschreiben. Mit dem zusätzlich generierten SDF-File (Standard Delay Format) lässt sich unter realen Verzögerungszeiten der digitale Schaltungspart des Designs erneut simulieren. Für analoge Schaltungs-entwicklung ist das Layout-Tool Encounter nicht geeignet.
Das Programm arbeitet mit sogenannten LEF-Files (LEF=Layout Exchange Format) welche die geometrische Lage von Pins, die Dimension und die verschiedenen Layer einer Zelle beschreibt. Die in der Netzliste beschriebenen Standardzellen, IO-Zellen und der Speicher sind vom Hersteller über die Beschreibung im LEF-Format gegeben. Abbildung 2 zeigt den Floorplan des PDA-Prozessor-ICs. Die um den Chip platzierten IO-Zellen werden über ein IO-File in ihrer Orientierung und Position definiert. Das vorliegende Design ist ein pad-limited-placement, was auf 75 verschaltete Pins zurückzuführen ist. Mehr als die hälfte der externen Anschlüsse sind für die Ansteuerung eines 2Mbyte großen Speichers reserviert. Um die Standardzellen und den Speicher sollte immer ein separater Power-Ring geführt werden, um eine optimale Sapnnungsversorgung zu gewährleisten. Nach erfolgreichem layouten wird das entstandene Design in das GDSII-Format mithilfe eines Layer-Mapping-Files exportiert und dann in die IC-Station eingelesen.

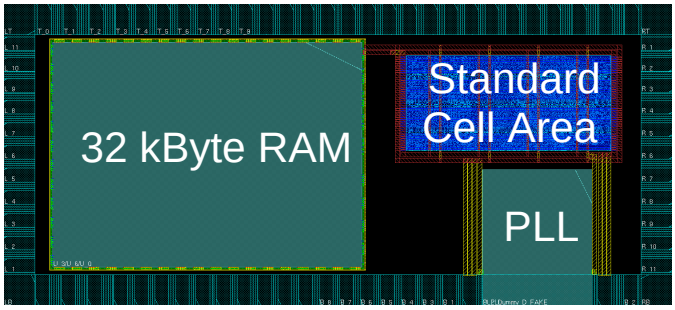


Bild 2: Floorplan mit dem Layout-Tool Encounter

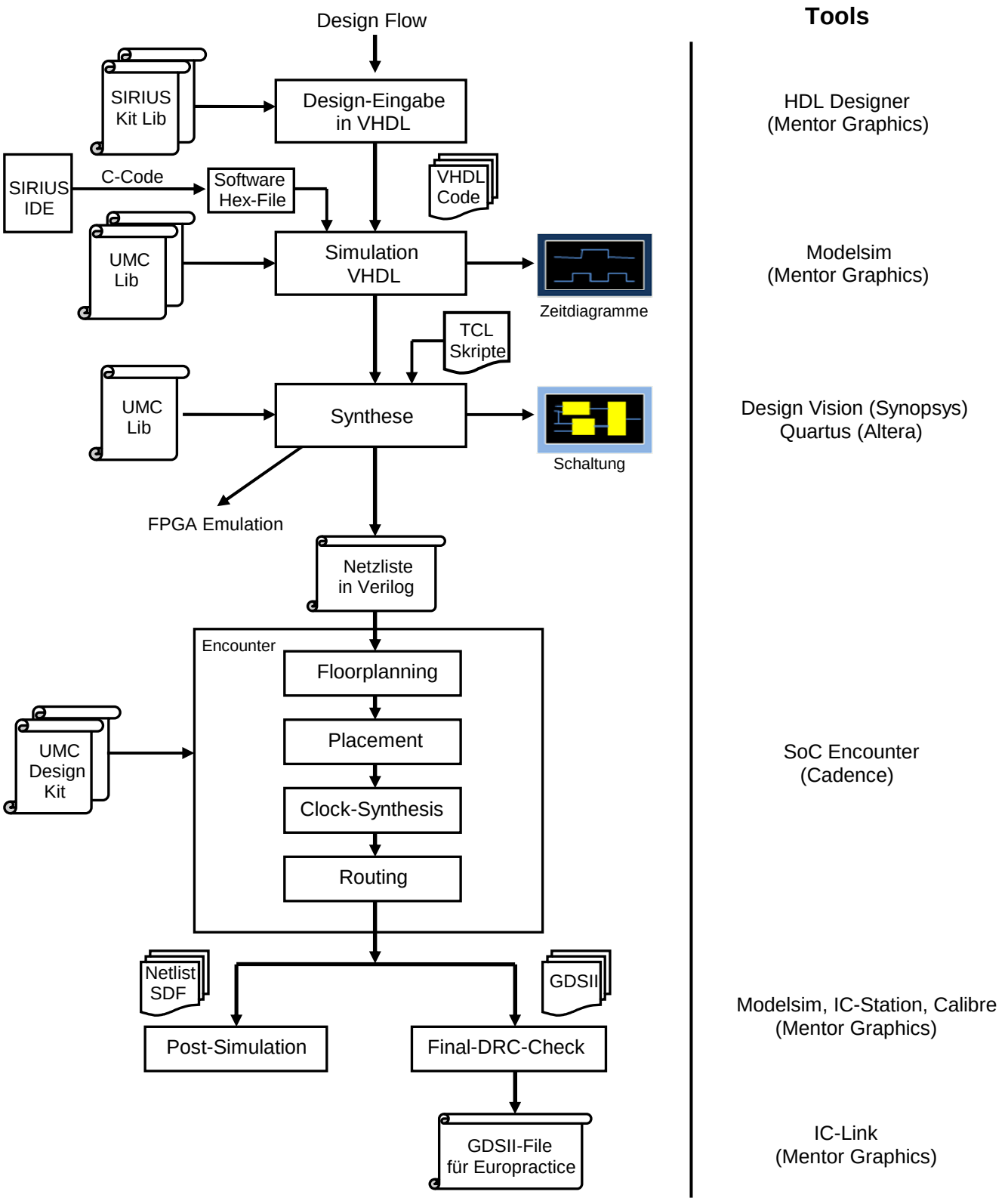


Bild 3: UMC 0.18µm Design Flow

Die IO-Zellbeschreibung beinhaltet nicht die äußeren Pads worauf gebondet wird. Daher werden die Pads in der IC-Station den IO-Zellen nachträglich angehängt. Bei den Pads muss ein Pitch von mindestens 90µm eingehalten werden.
Zuletzt erfolgt dann der abschließende Design Rule Check, welcher das entstandene ASIC-Layout auf die vom Hersteller Faraday geforderten geometrischen Bestimmungen überprüft. Ein ERC-Check wird von Europractice übernommen, um fatale Fehler wie z.B. einen Kurzschluß auszuschließen. Über das Konvertierungsprogramm IC-Link werden die in der IC-Station zuletzt bearbeiteten Maskendaten ins GDSII-Format gewandelt und abschließend dem Hersteller zur Produktion freigegeben. Abbildung 4 zeigt das entwickelte PDA-Prozessor IC Design.

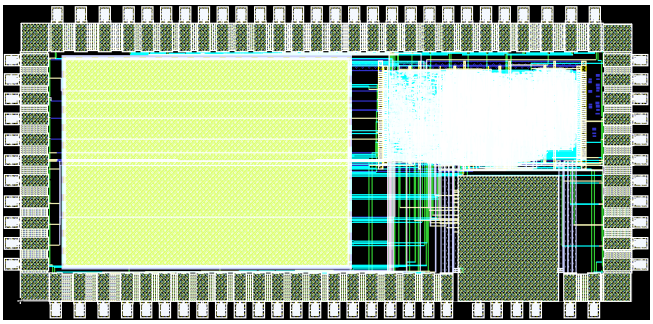


Bild 4: PDA-Prozessor IC Design

4. Ergebnisse und Ausblick

Um beim Anbieter Europractice in das günstige Mini@sic-Programm zu kommen muss eine gewisse Chipfläche erreicht werden. In der UMC 0.18µm Technologie wird ein Standard Block von 5x5mm in 9 Sub-Blöcke aufgeteilt. Aufgrund der hohen Pinanzahl benötigte das PDA-Prozzesor Layout 2 Sub-Blöcke was einer Fläche von 3240 x 1525µm entspricht. Insgesammt wurden 14000 Standard-Zellen verbaut, ein 32kByte großer Speicher eingesetzt und ein von Faraday angebotener PLL verwendet. Das mit Bedacht auf die hohe Pinanzahl von Europractice kleinste angebotene CQFP80-Package weißt eine Dimension von 14 x 14mm auf. Momentan wird die Möglichkeit geprüft in einem Package zwei ICs miteinander zu verschalten. Im vorliegenden UMC Design würde sich der externe Speicher mit über 45Pins dafür anbieten.
Mithilfe des an der Hochschule Offenburg verifizierten UMC 0.18µm Design Flow lassen sich digitale Design entwerfen. Alle durchgeführten Arbeiten werden durch den Entwurf des PDA-Prozessor ICs verifiziert, welcher sich momentan bei Europractice in der Fertigung befindet. Eine geeignete Testplatine zur Inbetriebnahme des ASICs befindet sich momentan in der Entwicklung. Der nächste Meilenstein im UMC 0.18µm Design Flow ist die Entwicklung einer analogen Zelle. Dazu soll in Kürze ein Oszilator entworfen werden. Der aufbereitete UMC Design Kit wird für alle MPC-Mitglieder freigegeben, die eine NDA für UMC bei Europractice unterzeichnet haben.

5. Referenzen

[1] Bau D., Durrenberger M., Jansen D. : SIRIUS Mikrocontrollersystem Design Kit, Technischer Bericht, HS-Offenburg, 2007

[2] Kreker A.: Entwicklung eines PDA-Demonstrators mit OLED Display, Touchscreen und grafischer Bedienoberfläche mit dem im FPGA implementierten Softcore SIRIUS, August 2008

[3] UMC: Faraday ASIC Cell Library FSA0A_C 0.18µm Standard Cell, August 2004

[4] UMC: Faraday 0.18µm Standard Cell Library FSA0A_C Layout/P&R Guideline, December 2004

[5] UMC: 0.18µm/GII Process Memory Generator, March 2006

[6] UMC: Phase-Locked Loop Data Sheet, June 2008

„Entwurf eines High-Speed Multiplexers/Demultiplexers für einen Mischer in 0,35µm Technologie“

Christoph Rahnke, Jürgen Giehl, Bernd Vettermann
Hochschule Mannheim, Paul-Wittsack-Straße 10, 68163 Mannheim
Fakultät für Informationstechnik - Institut für Entwurf integrierter Schaltkreise
christoph.rahnke@freenet.de

Die vorliegende Arbeit entstand im Auftrag des Institutes für digitale Signalverarbeitung der Hochschule Mannheim. Dieses beschäftigt sich mit der Entwicklung von Sendern und Empfängern für den Bereich des Digitalradios. Forschung wird sowohl im Gebiet DAB (Digital Audio Broadcast), ehemals Lang-, Mittel- und Kurzwellenbereich, als auch DRM (Digital Radio Mondiale), ehemals UKW-Bereich, betrieben. Für die Modulation bzw. Demodulation werden hierfür sehr schnelle Mischer benötigt, die in der geforderten Geschwindigkeit und Ausführung derzeit nicht am Markt verfügbar sind. Deshalb kam die Idee diesen Mischer im Institut für Entwurf integrierter Schaltkreise an der Hochschule Mannheim zu entwerfen.
Anmerkung: Die Arbeit ist derzeit noch nicht komplett abgeschlossen, so dass nur ein Teil vorgestellt werden kann.

1 Einleitung

1.1 Einführung

Das bestehende System erreicht eine Schaltgeschwindigkeit von 20ns bzw. 50MHz. In Zukunft sollen Frequenzen bis 120MHz, dies würde dann auch den kompletten FM-Bereich abdecken, mit einem Mischer geschaltet werden. Wegen der gewählten Modulationsart (IQ-Modulation) ist dafür eine Schaltgeschwindigkeit von 2ns für den Mischer notwendig.

Da es bereits Sender und Empfänger gibt, die in vorangegangenen Arbeiten entstanden sind, musste das bestehende System berücksichtigt und seine Signale für die Simulation nachgebildet werden.

1.2 IQ-De-/Modulation

Für das Herauf- und Heruntermischen eines Signals auf die Trägerfrequenz wird die I(nphase)/Q(uadratur)-De-/Modulation eingesetzt. Bei dieser Modulationsart können zwei Signale auf einer Trägerfrequenz moduliert werden, womit sich die Ausnutzung der zur Verfügung stehenden Bandbreite verdoppelt. Abb. 1 zeigt das Prinzip.

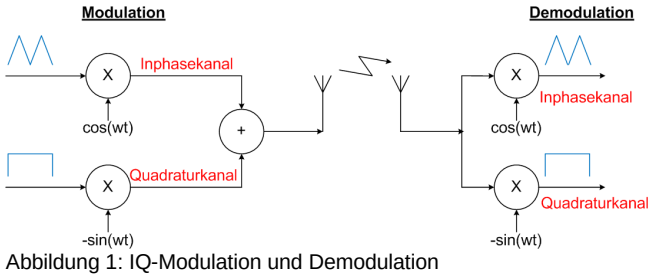


Abbildung 1: IQ-Modulation und Demodulation

Die Multiplikation des Signals wird mit zwei Schaltern realisiert. Diese werden mit Rechtecken angesteuert, die aus cos- und -sin-Signalen entstehen. Jeweils ein Schalter ist dabei für den Inphasekanal, der andere für den Quadraturkanal vorgesehen. Dieses Prinzip wird auch als Double Balanced Polyphasenabtastung bezeichnet. Die Schalter werden als De-/Multiplexer mit zwei Kanälen ausgeführt, die beide gleichzeitig durch vier Schalterstellungen getaktet werden. Somit ergeben sich vier Signale die jeweils 90° zueinander phasenverschoben sind (Abbildung 2).

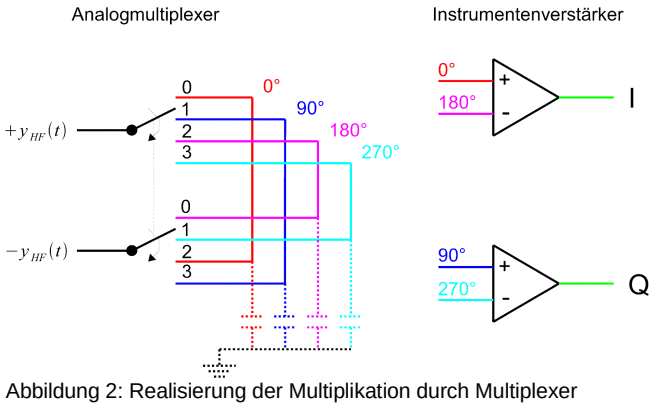


Abbildung 2: Realisierung der Multiplikation durch Multiplexer

In der Abbildung 2 ist die Multiplikation dargestellt. Die Signale $+Y_{HF}$ und $-Y_{HF}$ entsprechen dabei den Signalen im Inphase- bzw. Quadraturkanal. Die Taktung der Schalterstellungen erfolgt, wie schon erwähnt, synchron durch Rechtecke. Die entstandenen Signale (0°, 90°, 180°, 270°) werden anschließend, entsprechend der Abbildung 2 auf Verstärker geführt. Durch das gleichzeitige Takten sind pro Takt immer zwei Signale an einem Verstärker verfügbar. Der andere Verstärker hat während dieses Takts keine Eingangssignale. Durch die Verstärker werden die Signale quadriert und ergeben zwei Signale 'I' und 'Q' die in Abbildung 3 dargestellt sind.

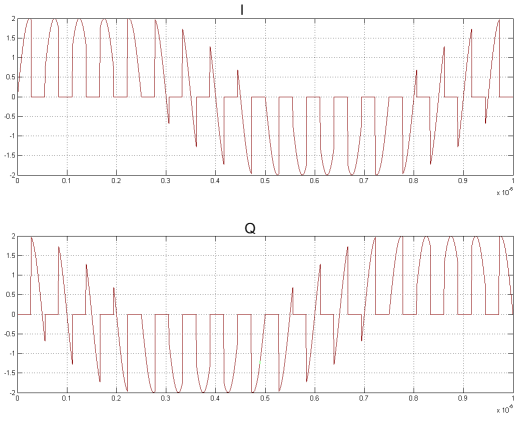


Abbildung 3: Signalverlauf I- und Q-Kanal

In Abbildung 3 ist der Signalverlauf nach der Demodulation eines empfangenen Signals dargestellt. Zum Erzeugen der cos- und -sin-Signales wurde eine DDS (Direct Digital Synthesis) mit 9Mhz betrieben Das empfangene Signal hat eine Frequenz von 10MHz. An der Periodendauer der Einhüllenden lässt sich nun die Frequenz ablesen, auf die heruntergemischt wurde, hier also 1MHz. Dies ist die Frequenz des Signals, welches bei der Modulation am Eingang anlag.

1.3 Aufgabenstellung

Wie schon erwähnt, soll der entwickelte IC-Baustein im Empfänger und im Sender des Digitalradios eingesetzt werden, das bedeutet er muss sowohl als Multiplexer wie auch als Demultiplexer betrieben werden können. Zusätzlich muss der Multiplexer aus zwei Kanälen bestehen, um die IQ-Modulation durchführen zu können. Die Ansteuerung der Multiplexer, die bisher mit einem externen Baustein (Differential Line-Receiver) durchgeführt wurde, sollen als Komparatoren mit in den Chip integriert werden. Weiterhin ist für das Umschalten ein geringer und definierter Durchgangswiderstand R_{ON} von einigen Ohm wichtig und dass das Eingangssignal möglichst gleichmäßig und periodisch als ¼ Periode auf die 4 Ausgänge gelegt wird. Die Spannungsversorgung soll 3,3V betragen.

2. Kompletter Schaltungsaufbau

Am Eingang der Schaltung kommen von einer DDS zwei differentielle Sinussignale, die zueinander um 90° verschoben sind. Komparatoren bilden daraus zwei um 90° verschobene Rechtecksignale, 'I' und 'Q'. Zusätzlich liefern die Komparatoren auch die invertierten Signale 'Iq' und 'Qq'. Mit diesen vier Signalen können Decoder so angesteuert werden, dass ein Decoder nach dem anderen und damit auch das jeweilige Transmissiongate (Schalter), geschaltet wird. Am Ausgang entstehen so Signale die aus jeweils ¼ Periode des empfangenen Signals bestehen.

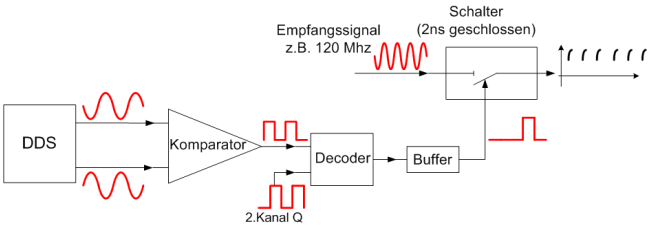


Abbildung 4: Schematische Darstellung der Schaltung

Im Kapitel 4 wird auf die einzelnen Blöcke näher eingegangen werden.

3. Allgemeines

3.1 Zur Simulation

Für die einzelnen Blöcke wurden zuerst Schaltbilder (Schematics) mit dem Mentor IC Studio erstellt und dimensioniert. Für die anschließende Simulation wurden, mit Hilfe der bereits bestehenden Schaltung, die Eingangssignale durch Quellen dargestellt, um so die vorhandene Umgebung nachzubilden. Für die Gatekapazitäten der nachfolgenden Transistoren und die Leitungskapazitäten sind entsprechend große Kapazitäten als Last an den Ausgang der Blöcke angeschlossen worden. Die Simulationen wurden dann mit Änderungen folgender Parameter durchgeführt:

- V_{DD} : Um Schwankungen in der Spannungsversorgung nachzubilden
- Temperatur: Die Schaltung wurde im Bereich zwischen -40°C und 120°C getestet
- worst speed, worst power: Um Prozessschwankungen bei der Herstellung der Transistoren zu simulieren
- Monte-Carlo-Simulation mit Temperatursweep um statistische Schwankungen zwischen den Bauelementen zu berücksichtigen

3.2 Zum Layout

Die Layouts wurden ebenfalls mit dem Mentor IC Studio erstellt. Für die Verdrahtung wurden die Metallisierungsebenen Metal1 (senkrecht) und Metal2 (waagrecht) benutzt. Desweitern sind die Layouts möglichst quadratisch angeordnet und die Pins nach der logischen Anordnung aus dem Schaltbild herausgeführt. Bei einigen Blöcken musste auf Matching der Transistoren geachtet werden, damit Stress-, Prozess und Temperaturgradienten über den Baustein die gleichen Auswirkungen haben. Weiterhin gilt für das Matching, dass der Stromverlauf und die Länge der Zuleitungen für die Transistoren jeweils gleich ausgelegt sind. Bei welchen Transistoren Matching zu beachten ist wird im jeweiligen Block speziell hingewiesen.

4. Einzelne Blöcke

4.1 Komparator

Der Komparator dient dazu, die 2 um 90° verschobenen Sinussignale, die von der DDS kommen, in Rechtecksignale umzuwandeln. Diese Funktion wurde bisher extern durch Differential Line-Receiver durchgeführt. Die beiden Eingangssignale werden dazu auf ein Differenzeingangspaar M3-M4 (Abbildung 5) geführt, welches die Signale mit einander vergleicht. Ist z.B. das 'IN'-Signal geringer als die Referenzspannung, wird der Strom der Stromquelle M2 fast komplett durch R1 geleitet, so dass am Inverter M5-M7 ein Highpegel anliegt. Damit liegt am Ausgang 'Out' ein Lowpegel.

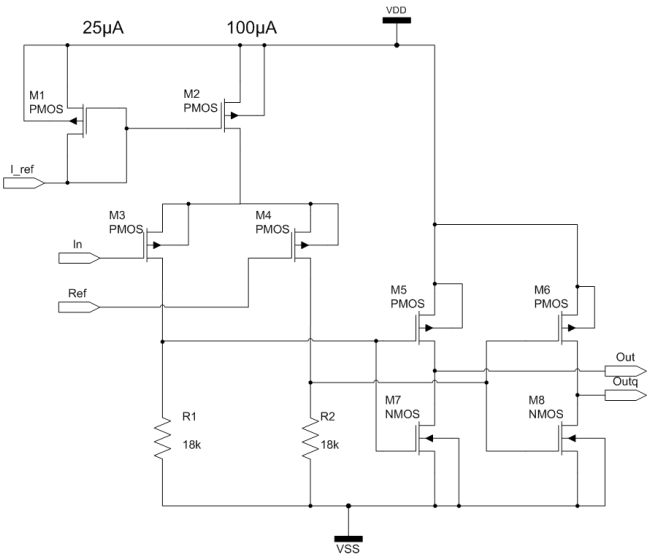


Abbildung 5: Schaltbild Komparator

Beim Layout des Komparators gibt es mehrere Bauteile bei denen Matching zu beachten ist:

- Stromspiegel M1-M2
- Differenzeingangspaar M3-M4
- Widerstände R1-R2

Die Transistoren des Stromspiegels und des Differenzeingangspaars sind gefaltet, soweit wie möglich wieder zusammengefügt und in einer Common-Centroid-Anordnung platziert worden (Abb.6 linke Seite Mitte und linke Seite oben).

Die Widerstände (Abb.6: rechte Hälfte) wurden in jeweils zwei Widerstände mit der halben Länge aufgeteilt, außerdem sind zwei Dummy-Widerstände hinzugefügt worden. Diese sollen Störungen auffangen und dafür sorgen, dass die Widerstände ebenfalls die gleiche Umgebung sehen.

Das Differenzeingangspaar M3-M4 besteht aus relativ kleinen Transistoren, die sich im linken unteren Bereich befinden. Auch diese wurde Common-Centroid angeordnet.

Die restlichen Transistoren sind im freien Bereich zwischen den Widerständen und dem Differenz-eingangspaar platziert.

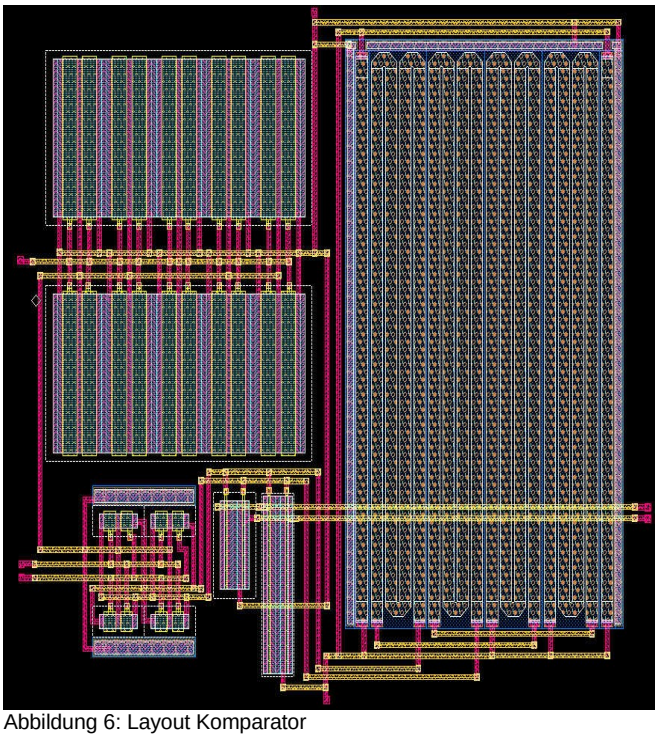


Abbildung 6: Layout Komparator

4.2 Decoder

Der Decoder wählt die Transmissiongates aus, die angesteuert werden sollen. Er besteht aus einem NAND- und einem Inverter-Gatter, die beide aus der AMS Library „Corelib“ entnommen worden sind. Die Auswahl, welcher Decoder schaltet, erfolgt durch die Signale „I“, „Iq“, „Q“ und „Qq“. Pro Kanal werden 4 Decoder benötigt, die alle unterschiedlich angesteuert werden, so dass einer nach dem anderen schaltet. Das Schaltschema ist in Abbildung 2 veranschaulicht.

Zusätzlich zu den erwähnten Simulationen ist außerdem eine Überprüfung des Umschaltkriteriums nach der CMOS-Pegel-Spezifikation erfolgt. Dabei darf der Decoder nicht umschalten wenn $V_{in} < 0,3*V_{DD}$ und muss umschalten wenn $V_{in} > 0,7*V_{DD}$ ist.

4.3 Buffer

Da der Strom, der aus dem Decoder kommt, nicht groß genug ist, um die Gatekapazität des Transmissiongates schnell genug umzuladen und somit zum Schalten zu bringen, muss das Signal verstärkt werden. Dies erfolgt durch eine Kette aus vier Invertern. Die Transistoren der Inverter werden von Stufe zu Stufe in ihren Weiten skaliert (jeweils Faktor 4) um dadurch am Ende der Kette ein schnelles Umladen des Transmissiongates durch einen genügend großen Strom zu ermöglichen. Das Delay welches dabei entsteht spielt keine Rolle, da es für alle Signale gleich ist.

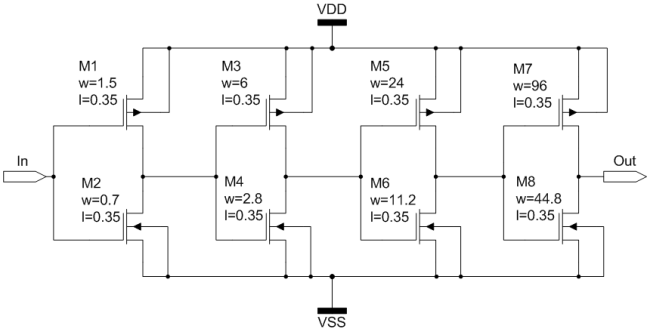


Abbildung 7: Schaltbild Buffer

Beim Layout stellte sich heraus, dass Transistoren (hier M2) mit Minimalabmessungen (l=0,35µm, w=0,4µm) zu Problemen beim Design-Rule-Check (DRC) führen. Durch die knochenartige Struktur des Transistors erscheint eine Fehlermeldung bezüglich des Abstandes der NPLUS-Wanne. Um diesen Fehler zu beheben wurde der Transistor M2 auf die Weite w=0,7µm vergrößert. Dadurch kommt es zu einer rechteckigen Struktur, die keine Fehlermeldungen mehr aufweist. Beim Layout des Buffers war weiter kein Matching zu beachten.

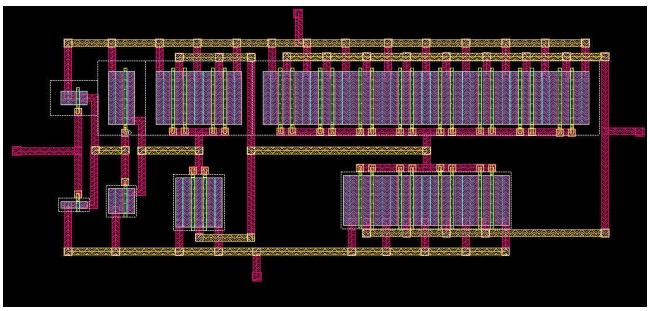


Abbildung 8: Layout Buffer

4.4 Transmissiongate

Das Transmissiongate ist der eigentliche analoge Schalter der das Signal von 'A' nach 'B', z.B. von der Antenne zum Verstärker (DEMUX-Betrieb) durch schaltet. Dabei wird ein PMOS und ein NMOS parallel geschalten und von invertierenden Signalen angesteuert. Die beiden Transistoren wurden dabei gleich groß dimensioniert, um eine Überkopplung über die Gatekapazitäten zu kompensieren. Dies hat allerdings den Nachteil, dass sich für die Transistoren unterschiedliche Widerstandswerte ergeben. Durch

die Parallelschaltung ergibt sich dann wieder ein geringerer Durchgangswiderstand R_{ON}.

Daraus ergibt sich folgendes Schaltbild (Abb.9):

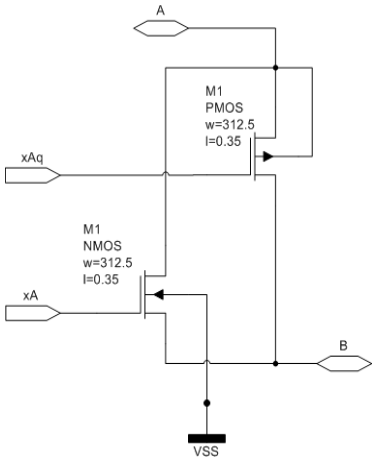


Abbildung 9: Schaltbild Transmissiongate

Das Layout des Transmissiongate ist in Abbildung 10 dargestellt.

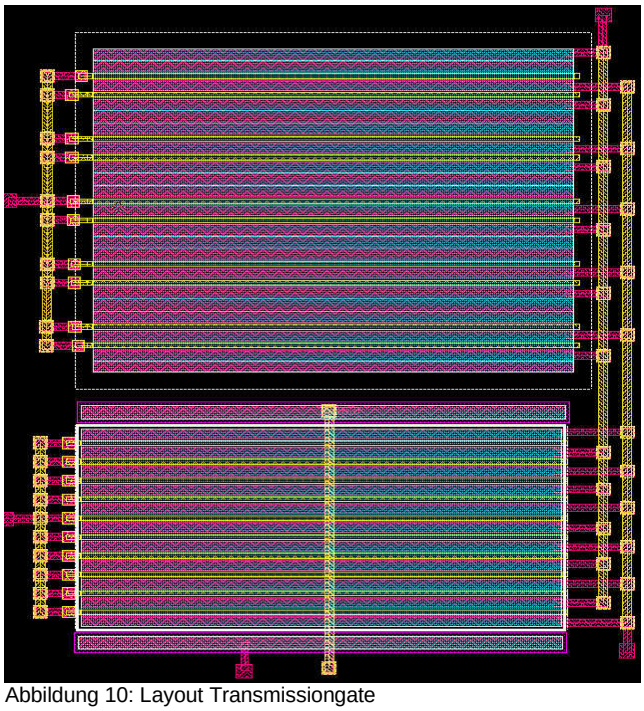


Abbildung 10: Layout Transmissiongate

Der PMOS Transistor M1 ist größer, weil die Bulk-Anschlüsse direkt an den Drainanschluss angefügt wurden. Beim NMOS Transistor M2 ist der Bulkanschluss (=Substratanschluss) oben und unten angefügt.

5. Simulation der kompletten Schaltung

Für die Simulation der kompletten Schaltung wurden entsprechend der Vorgaben zwei differentielle Sinussignale, die um 90° phasenverschoben sind, an den Eingängen angelegt. Der Aufbau (Abbildung 11) besteht weiterhin aus zwei Komparatoren, welche die Rechtecksignale für die IQ-Modulation erzeugen, und 8 Ketten die sich aus jeweils einem Decoder, Buffer und dem Transmissiongate zu zwei Kanälen zusammensetzen. Die Pins 1A,2A und 1B1 bis 2B4 sind als bidirektionale Pins ausgeführt, da diese das zu schaltende Signal leiten. Für den MUX-Betrieb läuft das Signal von 1B1-1B4 nach 1A (Kanal 1) bzw. von 2B1-2B4 nach 2A (Kanal 2). Für den DEMUX-Betrieb ist der Signalverlauf entsprechend umgekehrt.

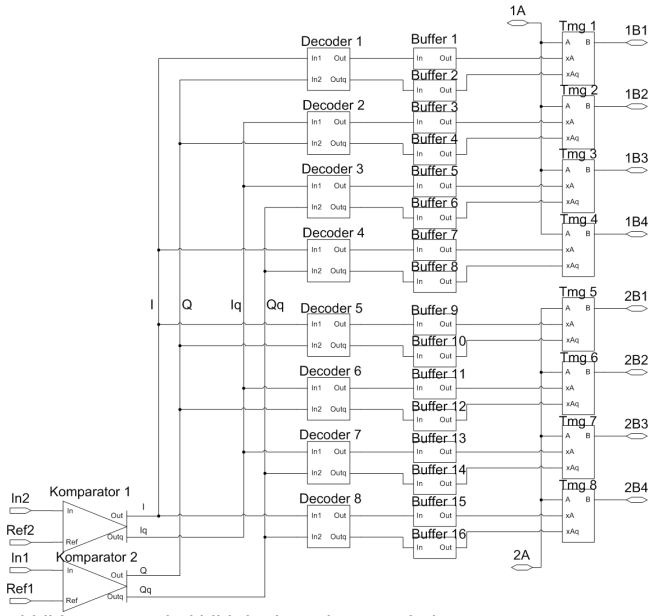


Abbildung 11: Schaltbild der kompletten Schaltung

In Abbildung 12 ist die Ansteuerung der Transmissiongates sowie die Signale 'A' und 'B' zu erkennen. Untereinander angeordnet sind die vier Transmissiongates des Kanal 1 die nacheinander angesteuert werden, wie man im zeitlichen Versatz der Rechtecke erkennen kann. Die Kurve 1 ist das Signal, welches von der Antenne empfangen wird und am Pin 'A' des Transmissiongate anliegt. Die hervorgehobenen Abschnitte sind Teilstücke der Kurve 2 die am Pin 'B' des Transmissiongates herauskommen.

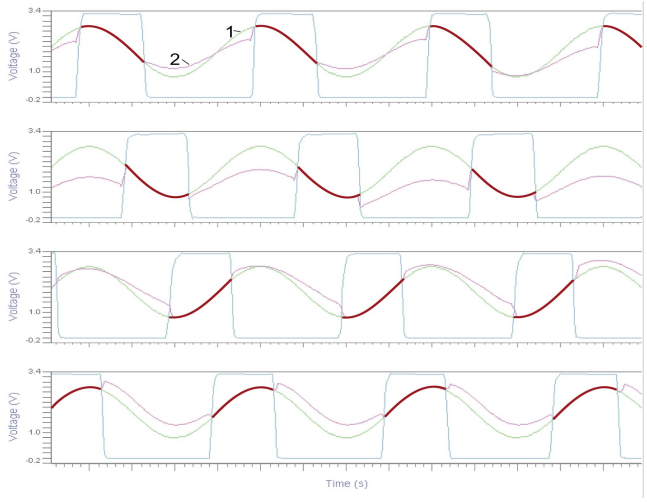


Abbildung 12: Signalverlauf der kompletten Schaltung

Durch die weitere Verdrahtung auf der Platine und das Zusammenführen der Signale aus Kanal 1 und Kanal 2 ist somit eine Demodulation des ursprünglichen Signals möglich. Die Verdrahtung ist nicht in den IC-Baustein aufgenommen worden und soll auch weiterhin extern auf der Platine durchgeführt werden.

6. Ergebnis

Durch die Simulationen konnten für den Baustein folgende Spezifikationen ermittelt werden.

Spannungsversorgung	3,1...3,5	V
Durchgangswiderstand R_{ON}	~10	Ω
Temperaturbereich	-40...120	°C
Frequenz	...125	MHz
Ansteuerung: 2*sin Φ	90	°
DC	1,8	V
Hub	+/-0,6	V

Tabelle 1: Spezifikationen des entstandenen Baustein

Die Funktionen des Multiplexers/Demultiplexers und des Line Receivers konnten in einem Baustein realisiert werden. Dies hat allerdings den Nachteil, dass sich im Vergleich zum Vorgängerbaustein keine Pinkompatibilität herstellen lässt, weil durch die neuen Funktionen zwei Pins hinzugefügt worden sind. Die Vorgabe hinsichtlich der geforderten Geschwindigkeit und des niedrigen Durchgangswiderstand werden ebenfalls erfüllt.

In der nachfolgenden Abbildung 13 ist die Pinbelegung des entworfenen IC Baustein angegeben.

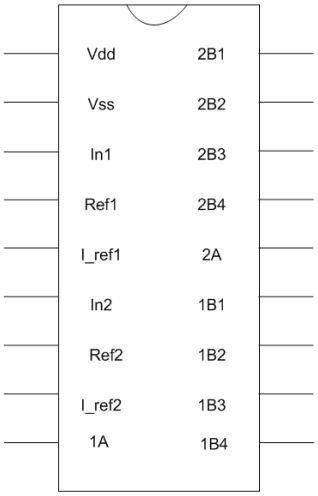


Abbildung 13: IC Baustein

7. Quellenangaben

[1] Giehl, J.: *„Entwurf integrierter Schaltkreise“* Vorlesungsunterlagen, Hochschule Mannheim 2008

[2] Beyer, M.: *„Entwicklung und Optimierung eines direktumsetzenden, abtastenden IQ-Demodulators für das UKW-Band“*

Diplomarbeit, Hochschule Mannheim, 2008

[3] Salewski, C.: *„Aufbau und Erprobung eines Empfängers für Software-Defined Radio bei 40 MHz“*

Diplomarbeit, Hochschule Mannheim, 2008

[4] Brakebusch, T.: *„Aufbau und Erprobung eines Software-Defined Radio Senders bei 40.68 MHz“*

Diplomarbeit, Hochschule Mannheim, 2009

[5] Allen, Holberg: *„CMOS Analog Circuit Design“*

2.Auflage, Oxford University Press, 2002

[6] Laker, Sansen*: „Design of Analog Integrated Circuits and Systems“*

McGraw-Hill International Editions, 1994

Dimensionierung und Entwurfszentrierung analoger Schaltungen mit WiCkeD

Florian Mrugalla & Georg Vallant, Gerhard Forster

Institut für Kommunikationstechnik

Hochschule Ulm, Prittwitzstraße 10, 89075 Ulm

mrugalla@mail.hs-ulm.de vallant@mail.hs-ulm.de forster@hs-ulm.de

Steigende Design-Komplexität und kürzere Produkteinführungszeit (Time To Market) erfordern hohe Design-Produktivität. Im analogen Schaltungsentwurf ist die Produktivität aufgrund fehlender Automatisierung entsprechend gering. Die mit der fortschreitenden Miniaturisierung einhergehende hohe Komplexität und starke Sensitivität auf Fertigungstoleranzen erfordern neue Entwurfsmethoden, die ein optimales DFY (Design for Yield) ermöglichen. Neue, heutzutage angebotene EDA-Werkzeuge verfügen über derartige Methoden.

Vorgestellt wird das von der Firma MunEDA entwickelte Werkzeug WiCkeD®, welches automatische Schaltungsdimensionierung und Entwurfszentrierung unter Berücksichtigung von Prozess- und Mismatchvariationen verspricht. Hierfür nutzt WiCkeD bekannte Schaltungssimulatoren wie Spectre oder ELDO. Anhand unterschiedlich komplexer Anwendungsbeispiele wird die Funktionalität untersucht und bewertet.

1. Motivation

Anfang der 80er Jahre zeichnete sich ab, dass die von Moore postulierte Miniaturisierung (Moore's Law) und die damit verbundene mögliche Design-Komplexität nur durch Produktivitätssteigerung, also erhöhten Einsatz von Designern ausgeschöpft werden kann. Dieser Sachverhalt wurde von der ITRS (International Roadmap for Semiconductors) dargestellt [1]. Die Komplexität wird durch die Anzahl der integrierbaren Transistoren pro Chip und die Produktivität durch die von einem Designer pro Monat entwickelten Transistoren beschrieben. Demnach wächst die Design-Komplexität mehr als doppelt so stark wie die Design-Produktivität – diese Divergenz erzeugt eine Produktivitätslücke. Um das Potential der Technologie auszuschöpfen und diese Lücke zu schließen, wird heute ein verstärkter Einsatz von Ingenieurskräften forciert. Durch die damit verbundene niedrige Wirtschaftlichkeit können sich Mikrochips erst bei höchsten Stückzahlen amortisieren. Besonders dominant wirkt sich die niedrige Produktivität im analogen Schaltungsbereich aus. Eine Vollautomatisierung des Design-Flows war im Analogen bisher nicht möglich; der Entwicklungsprozess hängt hier besonders stark von der Erfahrung des Designers ab. Mit der Miniaturisierung der Strukturen treten vermehrt quantenphysikalische Effekte auf, die eine analytische Fassbarkeit der Transistoren erschweren. Als sehr problematisch erweisen sich in diesem Zusammenhang globale und lokale Fertigungsschwankungen, die zu einer signifikanten Reduktion der Chipausbeute führen können.

Um diese Problematik zu überwinden, müssen neue, alternative Wege beschritten werden. Einen möglichen Lösungsansatz stellt moderne EDA-Software dar, die mit elaborierten Analyse- und Optimierungsfunktionen einen schnellen, automatisierten Designprozess zulässt. Obwohl dieser Wunsch seit langem besteht, ist erst seit kurzem aufgrund hoher Rechenleistung und leistungsfähigen Simulatoren (Spectre, ELDO) derartige Software verfügbar. An der Hochschule Ulm wurde das neue Tool MunEDA WiCkeD® installiert, das eine automatisierte Schaltungsdimensionierung und Entwurfszentrierung analoger Schaltungen unter Berücksichtigung von Prozessvariationen ermöglichen soll.

2. Grundlagen

Die Optimierung im Bereich der Schaltungstechnik befasst sich im Allgemeinen mit nichtlinearen, parametrischen und stochastischen Optimierungsproblemen der Form

$$\min Z(x) \quad \text{mit} \quad x \ni \Re^n$$
$$\text{unter den Nebenbedingungen}$$
$$g(x) \geq 0$$

Allgemein steht *Z(x)* für eine oder mehrere Zielfunktionen, die den Performances entsprechen.

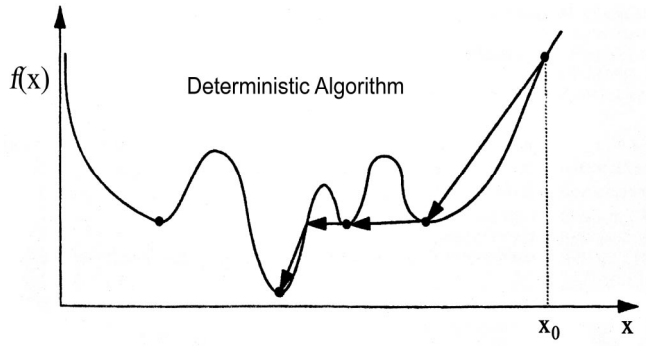


Bild 1: Prinzip der Nominal Optimization

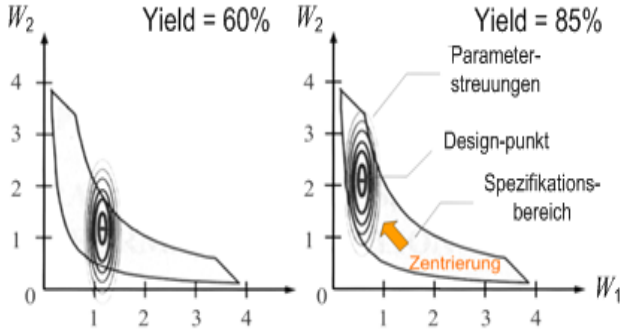


Bild 2: Prinzip der Yield Opimization (Entwurfszentrierung)

x repräsentiert dabei den Vektor der veränderlichen Designvariablen. Die in diesem Fall einzuhaltenden Nebenbedingungen $g(x)$ entsprechen den vorher definierten Constraints. Die Nomenklatur in der Schaltungsoptimierung unterscheidet sich von der klassischen Namensgebung. So gibt es Design-Parameter, Operating-Parameter, Prozess- und Mismatch-Parameter. Designparameter sind alle zu dimensionierenden Bauteilwerte wie Kanallängen L und Kanalbreiten W. Operating-Parameter sind von außen bestimmte Umgebungsparameter wie Temperatur T oder Versorgungsspannung V_{DD}. Die stochastischen Parameter gliedern sich in Prozess- und Mismatch-Parameter: Prozessparameter sind globale Variationen (z. B. Schwankungen der Oxiddicke t_{ox}, Kanallänge L, Kanalbreite W); Mismatch-Parameter sind lokale Variationen eines Bauteilpaares (z. B. Schwellenspannung V_{th} und Ladungsträgerbeweglichkeit μ_0). Zielfunktionen werden als Performances bezeichnet und beschreiben die Schaltungscharakteristik. Performances können z. B. Kleinsignalverstärkung V_0, Offsetspannung U_{off} oder Slew Rate SR sein. Für die Optimierung werden die Spezifikationsgrenzen der Performances durch Ober- und Untergrenzen angegeben. Nebenbedingungen werden als Constraints bezeichnet. Sie stellen Beschränkungen empirischer Natur dar. So ist a priori bekannt, dass sich MOS-Transistoren in starker Inversion befinden und Differenzpaare identisches W und L haben sollen.

Die Komplexität ist in der großen Anzahl veränderlicher Parameter, Zielfunktionen und Nebenbedingungen begründet. Da es sich um ein multidimensionales Optimierungsproblem im Designraum $\Re^n$ handelt, muss durch Variation der Design-Parameter eine ganze Schar von Performances und Constraints in den definierten Grenzen erfüllt werden. Diese Variation der Design-Parameter kann dabei nicht in beliebig großen Bereichen erfolgen, sondern nur in vorher definierten, sinnvollen Wertebereichen (z. B. 0.35 µm < W_1 < 50 µm). Der Designraum $\Re^n$ wird somit durch Parametergrenzen, Spezifikationsgrenzen und Constraints beschnitten. Der so genannte Designpunkt stellt die derzeitige Position innerhalb des Designraumes dar. Es stellt sich prinzipiell die Frage nach einer Gewichtung der Performances. Beim Lösen des Optimierungsproblems wird zunächst von einer Gleichberechtigung aller Performances ausgegangen. Für spezifische Anwendungsfälle und Anforderungen ist oftmals eine Priorisierung der Performances gewünscht (z. B. Bandbreite wichtiger als Verstärkung). Hierzu können die Performances mit einem linearen Faktor zueinander skaliert werden (scaling).

Allgemein kann zwischen zwei Optimierungsarten unterschieden werden: Nominal Optimization und Yield Optimization [2]. Die Nominal Optimization versucht einen Designpunkt zu finden, der gleichzeitig Performances und Constraints erfüllt. Die mathematische Grundlage hierbei ist das Gradientenverfahren. Bild 1 zeigt den typischen Verlauf einer deterministischen Optimierung. Ausgehend vom Startpunkt x_0 nähert sich der Algorithmus sukzessive dem Minimum an. Unberücksichtigt in der Nominal Optimization bleiben zunächst Prozess- und Mismatch-Parameter. Diese Abweichungen können jedoch dazu führen, dass mit einer gewissen Wahrscheinlichkeit später gefertigte Designs außerhalb des Spezifikationsbereichs liegen und somit die Chipausbeute (Yield) reduziert wird. In Bild 2 ist beispielhaft der beschnittene Designraum im $\Re^2$ dargestellt. Nur die Punkte innerhalb des grau hinterlegten Bereichs erfüllen die Spezifikation. Um den Designpunkt herum sind in konzentrischen Ellipsen die Parameterstreuungen dargestellt, wobei hier der Design-Parameter W_2 stärker streut als W_1. Die Höhenlinien der Streuung entsprechen der Wahrscheinlichkeitsdichte der Zufallsvariablen. Die Streuungen lassen sich im Allgemeinen beschreiben durch

$$PDF_N(\mathbf{x}_s) = \frac{1}{\sqrt{2\pi}^{n_{xs}} \cdot \sqrt{det\ \mathbf{C}}} \cdot exp(-0,5 \cdot \beta^2(\mathbf{x}_s))$$

$$\beta^2(\mathbf{x}_s) = (\mathbf{x}_s - \mathbf{x}_{s,0})^T \cdot \mathbf{C}^{-1} \cdot (\mathbf{x}_s - \mathbf{x}_{s,0})$$

$$\beta \geq 0$$

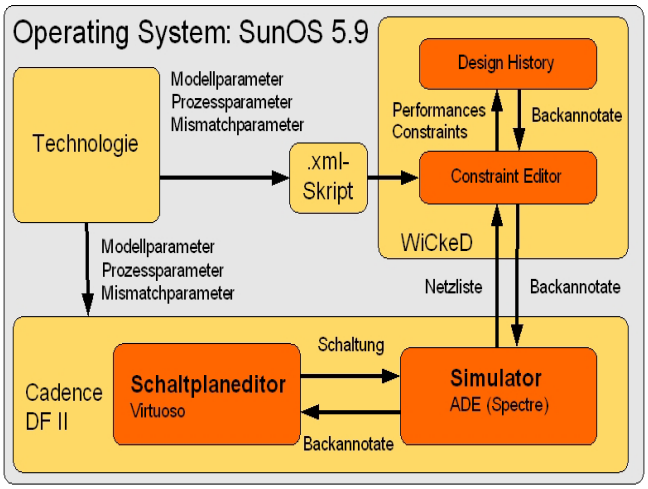


Bild 3: WiCkeD System

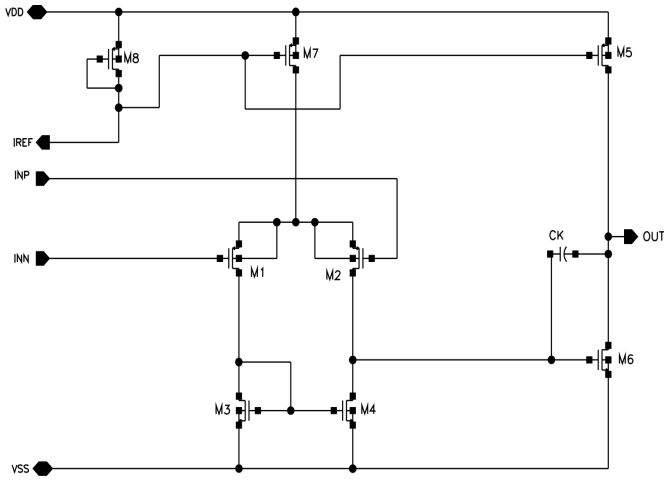


Bild 4: Standard Miller-OTA

Dabei beinhaltet die Kovarianzmatrix C die Parametervarianzen und deren Korrelationen. Der Mittelwertvektor x_{S0} beinhaltet alle Erwartungswerte der Parameter. Das Flächenintegral der Gauss'schen Normalverteilung über den beschnittenen Designraum führt zur geschätzten Chipausbeute. Darauf aufbauend versucht die Yield Optimization den Designpunkt derart zu verschieben, dass die Wahrscheinlichkeit einer Spezifikationsverletzung minimiert wird. Man spricht hierbei von einer Entwurfszentrierung. Liegen die Verteilungen der Parameterstreuungen vor, lässt sich allgemein für den multidimensionalen Fall die Chipausbeute über die Gauss'sche Normalverteilung berechnen.

Mittels dieser theoretischen Grundlagen wird im Folgenden die Systemumgebung und der Design Flow beschrieben.

3. WiCkeD Design Flow

3.1. Systemübersicht

WiCkeD ist ein modular aufgebautes Werkzeug, dass in Standard-Entwicklungsumgebungen wie Cadence Virtuoso Custom Design Platform oder Mentor Graphics DesignArchitect-IC integriert werden kann. Es werden hierbei bekannte Schaltungssimulatoren wie Spectre oder ELDO verwendet. WiCkeD unterstützt die Betriebssysteme Unix-SunOS oder Linux Red Hat.

Eine Übersicht des hier verwendeten WiCkeD Gesamt-Systems zeigt Bild 3. Als Hardware-Plattform wurde ein Sun Fire 440 Applikationsserver mit 4 Prozessoren (1,28 GHz) verwendet. Grundlage bildet das Unix-Betriebssystem SunOS 5.9, darauf aufbauend sind drei Blöcke zu erkennen: der WiCkeD-Block, die Entwicklungsplattform Cadence DFII (Version 5.11) und ein Technologie-Block, der das PDK (Process Design Kit) des Halbleiterherstellers repräsentiert. Die Anbindung von WiCkeD an das PDK wird über eine vom Halbleiterhersteller gelieferte .xml-Skriptdatei realisiert. In dieser Datei werden den einzelnen Bauelementen die zu dimensionierenden Design-Parameter (z.B. W, L eines MOSFET) und die relevanten Prozess- und Mismatch-Variationen zugewiesen. Des Weiteren lassen sich Constraints und typische Standardstrukturen (Stromspiegel) definieren. Typische Prozess-Parameter sind hierbei Variationen der Kanallänge L oder der Oxyddicke t_{ox} eines MOS-Transistors. Typische Mismatch-Parameter sind z.B. lokale Abweichungen der Ladungsträgerbeweglichkeit μ_0 oder der Schwellenspannung V_{th} eines MOS-Transistors. Die Anbindung von WiCkeD an Cadence Virtuoso wird bei der WiCkeD Installation automatisch eingerichtet und muss nicht weiter konfiguriert werden.

Im Folgenden wird die Dimensionierung und Optimierung eines Standard Miller-OTA mit WiCkeD exemplarisch nachvollzogen.

3.2. Entwicklungsumgebung

Im ersten Schritt einer Schaltungsoptimierung mit WiCkeD steht die Schaltplaneingabe. Mittels des Virtuoso Schaltplaneditors wurde der in Bild 4 dargestellte Miller-OTA eingegeben. Die Topologie beinhaltet 8 MOS-Transistoren und 1 Kapazität, dies entspricht 17 zu dimensionierenden Designvariablen (8 W, 8 L, 1 C). Charakterisiert wurde der Miller-OTA durch 9 Performances, diese sind in Tab. 1 mit entsprechenden Spezifikationen aufgeführt. Die Performances liegen im Klein- als auch im Großsignalbereich, zu deren Bestimmung AC-, Noise-, DC- und Transient-Analysen benötigt werden. Zur Bestimmung dieser Performances wurde die in Bild 5 gezeigte Testbench entworfen.

Tab. 1: Performances und Spezifikationen des Miller-OTA

Performance		**Spezifikation**
Verstärkung	A_0	> 90 dB
Stromaufnahme	I_{DD}	> 100 µA
Aussteuerung	U_O	> $\lvert\pm 1{,}5\ \mathrm{V}\rvert$
Transitfrequenz	f_T	> 4 MHz
Phasenreserve	φ_R	> 60°
Offsetspannung	U_{OS}	< $\lvert\pm 2\ \mathrm{mV}\rvert$
Slew Rate	SR	> 2 V/µs
Flicker Noise	N_f	< 400 $\mathrm{nV}/\sqrt{\mathrm{Hz}}$
Thermic Noise	N_{th}	< 20 $\mathrm{nV}/\sqrt{\mathrm{Hz}}$

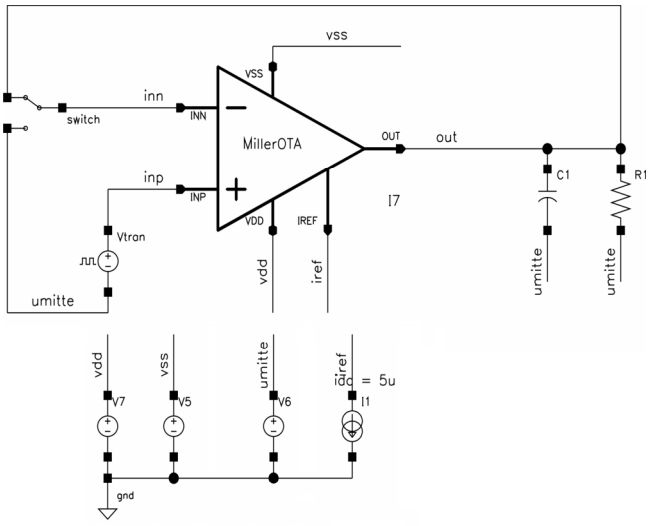


Bild 5: Testbench des Miller-OTA

Zu erkennen ist der im OP-Symbol hinterlegte Miller-OTA, der mit einer Versorgungsspannung von 3,3 V betrieben und eingangsseitig durch die Quelle Vtran beschaltet wird. Über einen BIAS-Strom kann der Arbeitspunkt geregelt werden, dieser Strom beträgt nominal 5 µA. Ausgangsseitig wird der Miller-OTA mit 1 MΩ und 5 pF belastet. Abhängig von der Analyseart können unterschiedliche Beschaltungen der Testbench erforderlich sein, z.B. offener Betrieb zur Bestimmung der Kleinsignalverstärkung A_0 oder rückgekoppelter Betrieb zur Bestimmung der Offsetspannung U_{OS}. Aus diesem Grund werden formal mehrere Testbenches benötigt, wobei WiCkeD (in unserer Version 5.2) nur in der Lage ist, jeweils eine Schaltung zu bearbeiten. Die dadurch entstehende Problematik kann mittels eines Simulationsschalters umgangen werden. Der in der Testbench verwendete Schalter wechselt seine Position innerhalb eines Simulationslaufes abhängig von der aktuell verwendeten Analyseart. Die Quelle Vtran liefert, entsprechend der jeweiligen Analyseart, eine Anregung mit AC-, DC- und Transient-Signalen am Eingang. Diese Funktionalität erlaubt eine „quasi“ multifunktionale Testbench, welche die Bestimmung aller Performances innerhalb eines Simulationslaufes ermöglicht.

Nach Eingabe der obigen Schaltungstopologien müssen die entsprechenden Simulatoreinstellungen im ADE (Analog Design Environment) getätigt werden. Die Parameter der Analysearten (AC, NOISE, DC, TRAN) sind für minimalen Rechenzeitbedarf zu wählen. Da WiCkeD im Zuge einer Optimierung mehrere 10.000 Simulationen durchführen kann, wird somit die benötigte Durchlaufzeit signifikant reduziert. Im Allgemeinen liefert der Halbleiterhersteller verschiedene Bauelement-Modelle, mittels derer Schaltungen unter Extrembedingungen (Worst Case Speed, Worst Case Power) simuliert werden können. Zusätzlich wird ein Monte Carlo-Modell mitgeliefert, das Simulationen unter Berücksichtigung von Prozess- und Mismatch-Variationen ermöglicht. Um WiCkeD eine spätere Optimierung auf diese Streuungen zu ermöglichen, muss dieses Monte Carlo-Modell (MC-Section) in der Cadence Simulationsumgebung (ADE) aktiviert werden. Eine Entscheidung, inwieweit die Performances der jeweiligen Dimensionierung die Spezifikationen erfüllen, kann nur anhand konkreter numerischer Werte erfolgen. Zur numerischen Bestimmung der Performances wurde hier das Cadence proprietäre OCEAN-Skript verwendet. Mit OCEAN lassen sich im ADE beliebige Werte aus den simulierten Datenkurven auslesen und nachträglich mathematisch bearbeiten. Anhand der durch OCEAN-Befehle bestimmten Performances entscheidet WiCkeD, ob die derzeitige Dimensionierung die geforderten Spezifikationen erfüllt. Ein typischer OCEAN-Befehl hat die Form:

```
value(dB20(mag(VF("/out"))) 10)
```

Er kann aus mehreren ineinander verschachtelten Funktionen bestehen. Mithilfe des obigen Befehls wird der Wert der Niederfrequenz-Kleinsignalverstärkung bestimmt. Der Befehl greift auf die gespeicherten Simulationsdaten des Ausgangsknotens zu und liest den Kleinsignal Spannungswert bei 10 Hz aus. Dieser wird dann in dB-Größenordnung ausgegeben.

In einem ersten Simulationslauf muss auf einen fehlerfreien Ablauf geachtet werden, da in diesem die später in WiCkeD verwendete Spectre-Netzliste generiert wird. Nach fehlerfreiem Simulationslauf kann WiCkeD direkt aus der ADE-Umgebung gestartet werden.

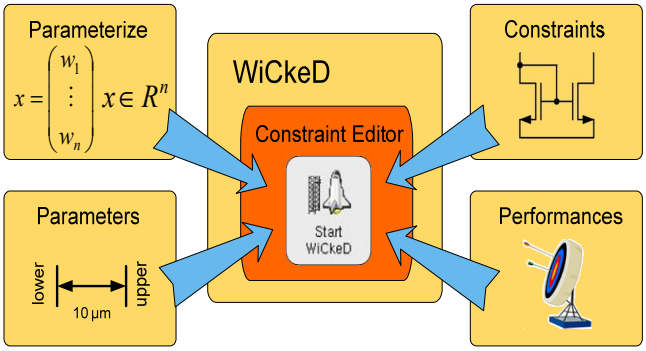


Bild 6: Übersicht des Constrains Editor

3.3. Constraints Editor

Wurde WiCkeD aus dem ADE gestartet, öffnet sich der Constraints Editor. Bevor mit einer Optimierung begonnen werden kann, müssen in WiCkeD die dazu erforderlichen Informationen eingegeben werden. Der Constraint Editor stellt das entsprechende Eingabeformular für diese Informationen dar. Zur Übersichtlichkeit sind die im Constraints Editor zu tätigenden Eingaben schematisch in Bild 6 abgebildet.

Im ersten Schritt Parameterize wird WiCkeD mitgeteilt, welche Bauelemente in die Optimierung miteinbezogen werden sollen. Dazu werden Designvariablen definiert, d.h. den jeweiligen Bauteilparametern (W, L, C) werden Variablen zugewiesen. Diese Designvariablen ersetzen die absoluten Parameterwerte der aktuellen Dimensionierung, z.B. werden W = 0,35 µm und L = 0,35 µm des MOSFET M1 durch die Variablen W = I7_M1_W und L = I7_M1_L ersetzt. In diesem Schritt Parameterize ist eine Kosten-zu-Nutzen-Abwägung zu treffen – viele Designvariablen bedeuten höheren Rechenaufwand aber eventuell ein besseres Endergebnis. Müssen zu viele Designvariablen (Anzahl > 30) dimensioniert werden, d. h das Optimierungsproblem besitzt zu viele Freiheitsgrade, kann WiCkeD erfahrungsgemäß nur mühsam eine verwendbare Lösung finden. Mittels Symmetrieüberlegungen kann dieses Problem entschärft werden. Da Stromspiegel und Differenzstufen (aus Matching-Gründen) gleiche Kanallängen und -breiten aufweisen, können diese Designparameter gleichgesetzt werden. Die Anzahl der Designvariablen kann durch solche Überlegungen signifikant reduziert werden. Dies ermöglicht auch Optimierungen von komplexeren Topologien.

Im zweiten Schritt Parameters werden für die vorher definierten Designvariablen zulässige Wertebereiche festgelegt. Beispielsweise können sich die Designvariablen I7_M1_W und I7_M1_L des MOSFET M1 im Bereich von 0,35 µm bis 30 µm bewegen. Die Festlegung dieser Wertebereiche erfordert gewisse Erfahrung mit den Schaltungen, da ein zu klein gewählter Wertebereich die Performance von Rauschen und Offset stark einschränken kann. Im Gegenzug kann es durch zu groß gewählte Bereiche zu einem großen Flächenbedarf kommen. Daher sollten schon Vorstellungen von Größenordnungen der Designvariablen vorhanden sein, z.B. können Eingangsdifferenzstufen aufgrund von Rauschen und Offset Kanalbreiten von mehreren hundert µm benötigen, Stromspiegel kommen hingegen mit Kanalbreiten von ca. 10 µm aus. An dieser Stelle empfiehlt es sich, die Dimensionierung auf minimale Kanallängen und –breiten zu setzen. Dieser Designpunkt dient der späteren Optimierung als Startpunkt. Dies hat den Vorteil, dass WiCkeD meist ein flächenminimales Optimum findet. Die späteren Ergebnisse der Dimensionierung werden von WiCkeD aufgrund der stetig arbeitenden Algorithmen bis auf fm genau angegeben. Eine solche Genauigkeit ist nur virtuell möglich, tatsächlich ist die Auflösung durch ein Raster (Grid) des Halbleiterherstellers nach unten beschränkt. Dieses Raster kann in die Dimensionierung miteinbezogen werden, was eine entsprechende Rundung der Werte erlaubt. Vorher definierte Versorgungsspannungen oder die Umgebungstemperatur können als Operating-Parameter eingegeben werden. Diese erlauben eine Berücksichtung von Worst Case-Schwankungen. Damit können z.B. Schwankungen der Versorgungsspannung V_{DD} = 3,3 V ± 50 mV während der Optimierung berücksichtigt werden. Die Einbindung der für die Prozess- und Mismatch relevanten Parameter erfolgt ebenfalls im Schritt Parameters. Im Allgemeinen werden diese Parameter durch das im vorigen Kapitel besprochene .xml-Skript automatisch geladen, jedoch kann jeder Parameter nachträglich manuell aktiviert oder deaktiviert werden. Die mittels der Prozessparameter berücksichtigten globalen Variationen gelten für alle Bauelemente gleichermaßen. Mismatchparameter berücksichtigen nur lokale Variationen und werden für jeden Transistor getrennt bestimmt. Dies kann schon bei Topologien mittlerer Komplexität hohe Rechenzeiten erfordern. Daher müssen Bauelemente (z.B. Differenzstufe) explizit zur Mismatch-Betrachtung hinzugefügt werden.

Im folgenden dritten Schritt Constraints können Beschränkungen (Nebenbedingungen) eingegeben werden. Diese Beschränkungen sind oftmals empirischer Natur: Gewisse MOS-Transistoren sollen sich in starker Inversion und in Sättigung befinden, Differenzstufen sollen gleiche Kanalbreiten besitzen, Stromspiegel sollen gleiche Kanallängen besitzen. Dabei lassen sich Standard-Strukturen durch das Programm automatisch erkennen. Für diese Strukturen wurden vom Halbleiterhersteller Constraints definiert, die beliebig aktiviert oder deaktiviert werden können.

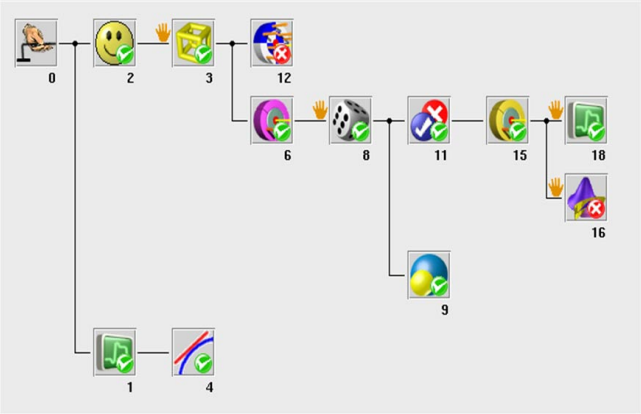


Bild 7: WiCkeD Design History

Durch die Nutzung von Constraints kommt es zu einer Verkleinerung des Designraumes, was zu einer Erleichterung der Optimierung führen kann. Durch falsch gesetzte Constraints hingegen können Designbereiche blockiert und somit die erzielte Performance reduziert werden. Die eigentlichen Ziele werden im letzten Schritt Performances eingegeben, hier werden die im ADE mittels OCEAN definierten Performances angezeigt. Jeder Performance kann eine obere und untere Spezifikationsgrenze zugewiesen werden, z.B. wurde im vorherigen Projekt eine Kleinsignalverstärkung von größer 90 dB und ein Offset von ± 1mV gefordert. Wurden alle gewünschten Spezifikationsbereiche eingegeben, kann die Optimierung gestartet werden.

3.4. Design History

Die eigentliche Schaltungsdimensionierung bzw. -optimierung mit WiCkeD findet in der so genannten Design History statt. In Bild 7 ist die komplette Design History des Miller-OTA dargestellt. Die einzelnen Bildsymbole stellen verschiedene Analyse- oder Optimierungsarten dar, welche aufeinander aufbauen oder sich wurzelförmig (parallel) verzweigen [3]. Diese Anordnung ermöglicht es von einem zentralen Wurzelknoten (Knoten 0) ausgehend verschiedene Optimierungsläufe zu verfolgen. Dabei kann ein einzelner Optimierungslauf sukzessiv aus verschiedenen Analysen und Optimierungen zusammengesetzt sein. Am Beginn eines Optimierungslaufes steht eine Feasibility Optimization (Knoten 2, Dauer 3 min.). Bei dieser Optimierung stehen die Constraints im Fokus. Die Designparameter soweit derart verändert, bis alle Constraints erfüllt werden, die Performances werden dabei vernachlässigt. Eine Feasibility Optimization versucht generell, die Dimensionierung in den durch die Constraints aufgespannten Designraum zu führen. In manchen Fällen wirken Constraints konträr zueinander, wodurch es unmöglich werden kann innerhalb des Designraumes zu gelangen. Prinzipiell informiert

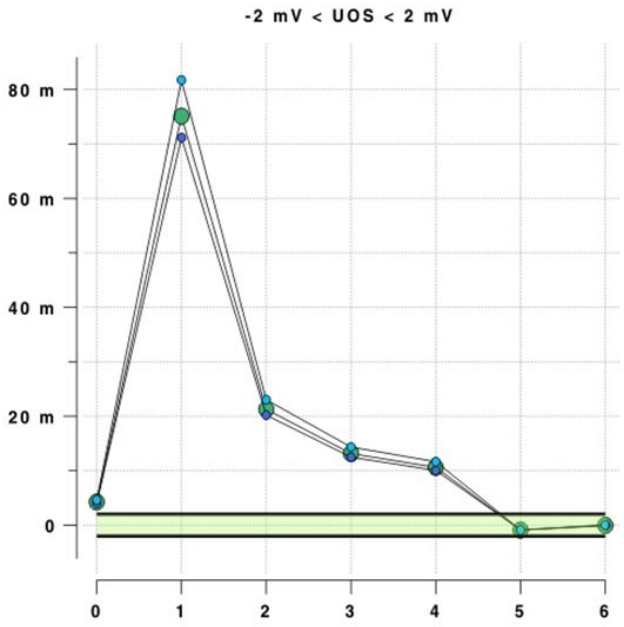


Bild 8: Optimierungsverlauf der Offsetspannung

eine Feasibility Optimization lediglich darüber, inwieweit es möglich ist die Constraints zu erfüllen. Eine darauf aufbauende Aussage über Performance und Yield wird von weiteren folgenden Optimierungen getroffen. Nach erfolgreicher Feasibility Optimization wird eine Deterministic Nominal Optimization (Knoten 6, Dauer 5,5 h) angehängt. Diese versucht, ausgehend von dem in der Feasibility Optimization erreichten Designpunkt, die einzelnen Performances in den definierten Spezifikationsbereich zu führen. Die Optimierung baut dabei auf dem Gradientenverfahren auf und bestimmt von jeder Performance den jeweiligen Gradienten. Einflussnahme auf die Gewichtung der Gradienten ist durch die Eingabe eines Scaling-Faktors möglich. Damit können z.B. 10 Mhz Bandbreite gleichwertig zu 5 dB Kleinsignalveratärkung gesetzt werden. Die Deterministic Nominal Optimization bietet dabei zwei Algorithmen an, den Least Square Algorithmus und den Parametric Distance Algorithmus. Je nach Optimierungsproblem kann sich einer von beiden als erfolgreicher erweisen. Bild 8 zeigt den Optimierungsverlauf der Offsetspannung des Miller-OTA über der Anzahl der Iterationsschritte. Es ist zu erkennen, dass sich der Offset zu Beginn der Optimierung weit von dem geforderten Spezifikationsbereich von ± 1 mV entfernt und diesen erst nach der 5. Iteration wieder erreicht. Im Diagramm lassen sich drei Kurven erkennen, diese repräsentieren den nominalen Offset (Mitte) und den durch die Operating-Parameter bestimmten Worst Case-Offset (obere und untere Kurve). Jede Performance zeigt einen eigenen Optimierungsverlauf. Mit ihnen lassen sich Korrelationen zwischen den Performances ablesen.

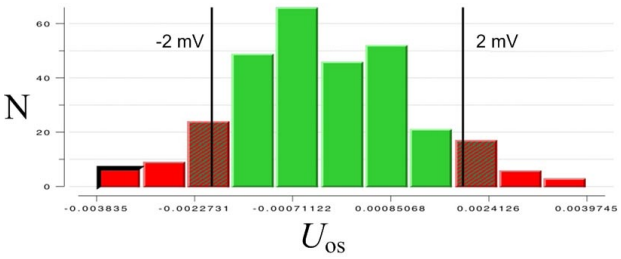


Bild 9: Histogramm der Offsetspannung

Endergebnis ist eine Dimensionierung, deren Nominal- und Worst Case-Performances sich im erfolgreichen Fall innerhalb des Spezifikationsbereiches befinden.

Das Ergebnis der Deterministic Optimization gibt noch keinen Aufschluss über die Sensibilität der Performances auf Fertigungstoleranzen. Diese wird im folgenden Schritt mittels einer Monte Carlo Analyse (Knoten 8, Dauer 15 h) geprüft. Ergebnis dieser Analyse ist die zu erwartende Ausbeute (Yield). Zusätzlich werden Informationen über Korrelationen der Performances oder Designparameter angegeben. Bild 9 zeigt ein typisches Histogramm einer Monte Carlo-Analyse mit 500 Simulationen. Des Weiteren ist der Spezifikationsbereich von ±2 mV zu erkennen; die Einzelausbeute ist dann das Verhältnis der Anzahl von Würfelungen innerhalb dieser Grenzen zur Gesamtzahl. Die Gesamt-Ausbeute (Total Yield) kann aus dem Produkt aller Einzelausbeuten berechnet werden.

Ist die Gesamt-Ausbeute an dieser Stelle zu gering, kann über eine Yield Optimization versucht werden diese zu erhöhen. Die Yield Optimization (Knoten 15, Dauer 17 h) versucht, ausgehend von dem in der Deterministic Nominal Optimization erzielten Designpunkt die Ausbeute zu verbessern und dabei alle Performances im spezifizierten Bereich zu halten. Oftmals liegen die in der Deterministic Nominal Optimization erzielten Performance-Werte direkt im Grenzbereich der Spezifikationen. Durch Fertigungstoleranzen verursachte Performanceschwankungen können die Ausbeute in diesem Fall stark reduzieren. Da die Möglichkeiten der Yield Optimization begrenzt sind, sollte in solchen Fällen eine Aufweitung des Spezifikationsbereiches vorgenommen werden – der Algorithmus wird die Ausbeute eines grenzgängigen Designs nicht von 2% auf 99% erhöhen können. Das Ergebnis der Yield Optimization des Miller-OTA ist in Bild 10 dargestellt. Es ist zu erkennen, dass die Gesamtausbeute des Miller-OTA von 82% auf 92% angehoben wurde. Die optimierte Dimensionierung wird automatisch von WiCkeD nach Cadence Virtuoso übertragen. Die erzielten Performances sind in Tabelle 2 aufgelistet, ein Auszug der zugehörigen Dimensionierung findet sich in Tabelle 3.

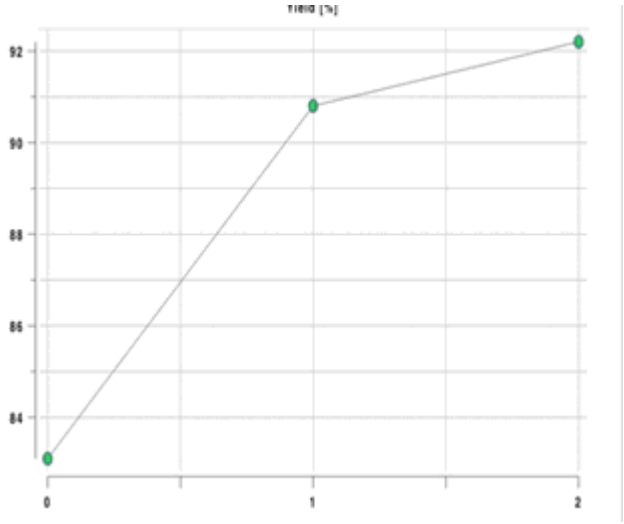


Bild 10: Optimierungsverlauf der Ausbeute

Tab. 2: Performances des Miller-OTA

Performance		**Ergebnis**
Verstärkung	A_0	104 dB
Stromaufnahme	I_{DD}	84.5 µA
Aussteuerung	U_O	$\pm 1{,}64$ V
Transitfrequenz	f_T	5,9 MHz
Phasenreserve	φ_R	63°
Offsetspannung	U_{OS}	± 2 mV
Slew Rate	SR	8,1 V/µs
Flicker Noise	N_f	88,7 nV/$\sqrt{\text{Hz}}$
Thermic Noise	N_{th}	21,4 nV/$\sqrt{\text{Hz}}$
Yield	92,2 %	

Tab. 3: Auszug der dimensionierten Designvariablen

Transistoren	**W**	**L**
MP1, MP2	50 µm	3,35 µm
MN3, MN4	33,5 µm	33,7 µm
MP5	50 µm	9,9 µm
MN6	35 µm	0,8 µm
MP7	12,1 µm	9,9 µm
MP8	4,15 µm	9,9 µm
Ck	1,275 pF	

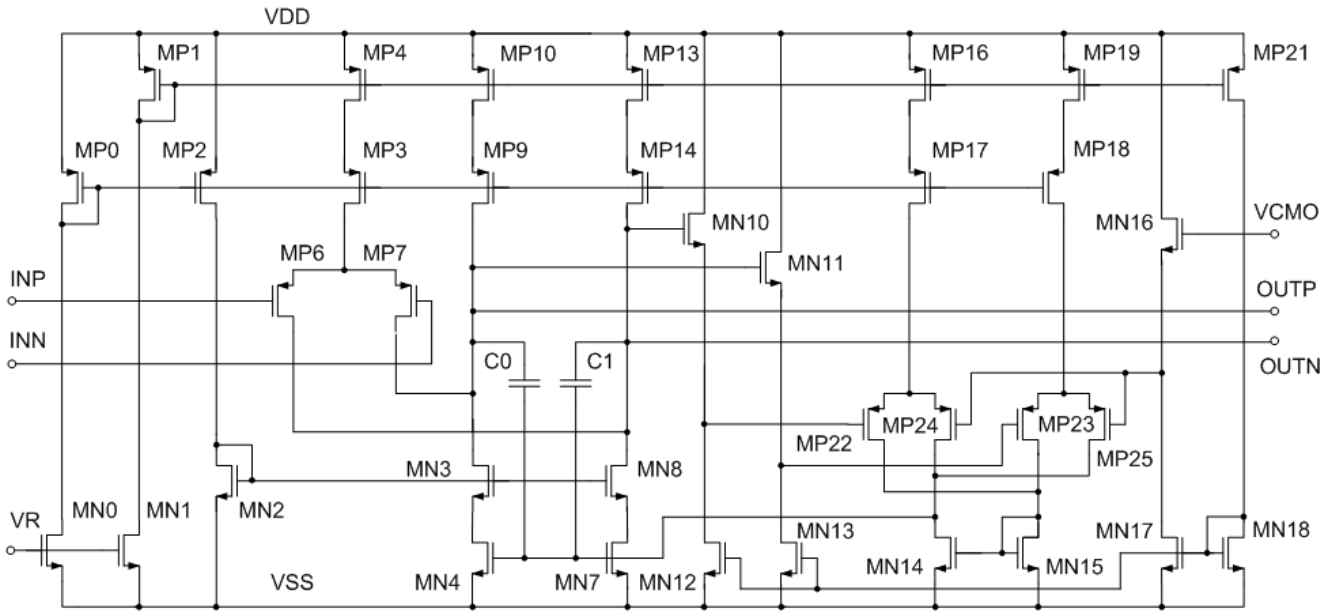


Bild 11: Undimensionierter vollsymmetrischer Operationsverstärker

4. Anwendung am Beispiel eines vollsymmetrischen OPV

4.1. Einführung

In einer früheren Arbeit wurde eine manuelle Dimensionierung eines volllsymmetrischen OPV durchgeführt. Aufgrund der schweren analytischen Fassbarkeit der Topologie war die Dimensionierung mit einem hohen Zeitaufwand verbunden. Ausgehend von Minimalstrukturen soll mit WiCkeD eine automatische Dimensionierung dieser komplexen Schaltung bei gleicher Spezifikation durchgeführt und die so erzielten neuen Ergebnisse mit den alten verglichen werden. So kann auch eine Aussage darüber getroffen werden, inwieweit WiCkeD fähig ist, mit komplexeren Topologien umzugehen. Die in Bild 11 dargestellte Topologie findet vor allem in differentiellen Systemen Einsatz, deren Grundlagen in [4, 5] ausführlich beschrieben sind. Die Schaltung lässt sich prinzipiell in einen Eingangsverstärker (gefaltete Kaskode) und eine Gleichtaktregelung zerlegen; dadurch entsteht ein rückgekoppeltes System, das Gleichtaktschwankungen am Ausgang unterdrücken soll. Unter Ausnutzung des Miller-Effekts sind Kapazitäten eingefügt, die eine Stabilisierung der Gleichtaktregelung bewirken. Um den Ausgangsknoten nicht zu belasten, wurde die Gleichtaktregelung über Impedanzwandler (Sourcefolger) entkoppelt. Die Schaltung umfasst 37 Bauelemente (35 Transistoren, 2 Kapazitäten) und damit 72 Designvariablen. Diese Anzahl ist deutlich zu hoch, so dass es später nötig sein wird durch Symmetrieüberlegungen Variablen zu eliminieren.

4.2. Testbench und Performances

Die in Bild 12 dargestellte Testbench zeigt den OPV in einer SC-Integrator-Umgebung, wie sie z.B. in Sigma-Delta-AD-Wandlern Anwendung findet. Über Simulationsschalter müssen auch hier simulationsspezifische Verdrahtungen eingerichtet werden. So muss bei Transient-Simulationen rückgekoppelt werden. Die Realisierung der symmetrischen Ansteuerung darf nicht über zwei AC-Quellen, sondern muss über eine erfolgen, da jede Übertragungsfunktion nur eine Bezugsquelle am Eingang besitzen darf. Das Signal wird über ideale Verstärker auf die beiden Eingänge INP und INN verteilt. Die Versorgungsspannung V_{DD} beträgt 3,3 V, V_{SS} beträgt 0 V. Um WiCkeD bei der Dimensionierung der Ruheströme möglichst freien Raum zu lassen, wird der Knoten VR auf 3,3 V gelegt. Der vom Benutzer gewünschte Gleichtakt am Ausgang muss an VCMO angelegt werden, in den allermeisten Fällen wird es sich um V_{DD}/2 (1,65 V) handeln. Die Zeitkonstante des SC-Integrators wird über das Verhältnis von C_8 zu C_5 bzw. C_9 zu C_4 bestimmt. Die Kapazitäten wurden nach der ursprünglichen Dimensionierung mit 500 fF angesetzt. Die Pulsflanken der Pulsansteuerung der idealen SC-Schalter werden mit einer Schutzzeit von 1 ns betrieben. Um die Performances numerisch berechnen zu können, muss wiederum auf das OCEAN-Skript zurückgegriffen werden. Tabelle 4 zeigt zunächst alle aus der alten Spezifikation übernommenen Performances, sowie einige weitere, deren Bedeutungen im Folgenden erläutert werden. Verstärkung, Stromaufnahme, Transitfrequenz, Phasenreserve und Thermic Noise bedürfen hingegen keiner weiteren Erläuterung.

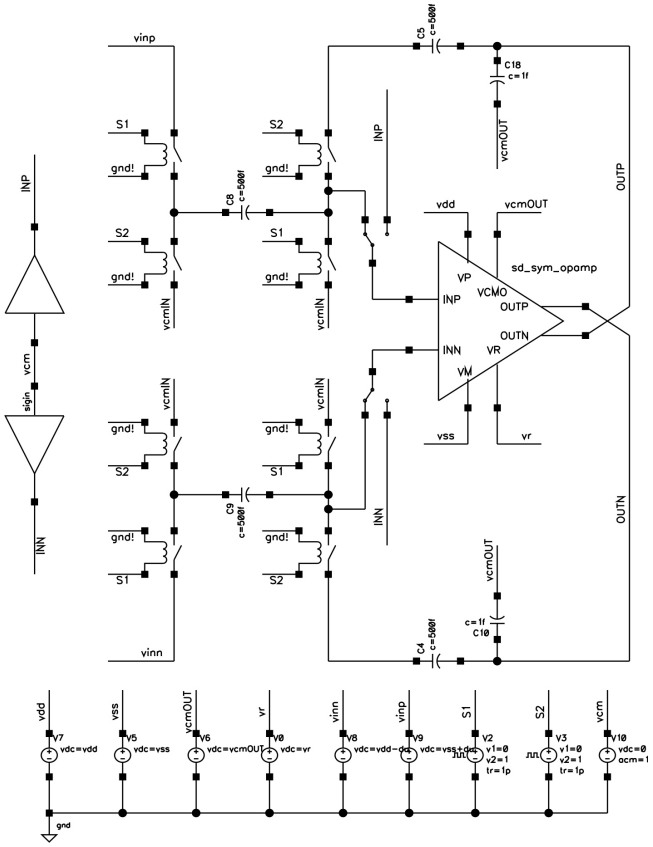


Bild 12: OPV-Testbench (SC-Integrator-Umgebung)

Um die Steilheit der Transferfunktion und den Gleichtaktfehler am Ausgang zu charakterisieren mussten neue Performances eingeführt werden. Dies war ein notwendiger Schritt, bevor die Topologie von WiCkeD optimiert werden konnte. Erst dadurch wurde eine vollständige Charakterisierung aller Eigenschaften der Schaltung erreicht. Bild 13 verdeutlicht diese neuen 5 Performances. Über einem DC-Sweep von -10 mV bis 10 mV am Eingang des Verstärkers wurden zwei Kurven aufgetragen: Einmal die Transferkennlinie (linke Y-Achse), für die es wichtig war, eine gewisse Steilheit zu gewährleisten. Hierzu wurden wie bereits erwähnt zwei diskrete Punkte U_{ON} und U_{OP} eingeführt, an denen es eine minimale Ausgangsspannung zu erreichen gilt. U_{ON} muss dabei kleiner als der geforderte untere Aussteuerbereich von 600 mV sein, U_{OP} hingegen größer als der geforderte obere Aussteuerbereich von 2,7 V. Damit kann am Ausgang ein Spannungshub von 2,1 V genutzt werden. Die andere Kurve ist die Fehlerkurve des Gleichtaktes am Ausgang. Der Fehler stellt die Abweichung von der am Knoten VCMO eingestellten Spannung (hier $V_{DD}/2$ = 1, 65 V) bei differentieller Anregung am Eingang dar. Selbst bei größeren Spannungen am Eingang darf der Fehler einen bestimmten Bereich nicht überschreiten, darum wurden drei Performances eingeführt: CM-Fehler (Max), CM-Fehler(Min) und CM-Fehler (static).

Tab. 4: Performance und Spezifikation des OPV

Performance		**Spezifikation**
Verstärkung	A_0	> 75 dB
Stromaufnahme	I_{DD}	< 1 mA
Transitfrequenz	f_T	> 140 MHz
Phasenreserve	φ_R	> 60°
Thermic Noise	N_{th}	< 25 $nV/\sqrt{Hz}$
Settling Time CM	ST_{cm}	< 40 ns
Settling Time DM	ST_{dm}	< 40 ns
CM-Fehler (Max)	$E_{\max}$	< 40 mV
CM-Fehler (Min)	$E_{\min}$	> -40 mV
CM-Fehler (static)	E_{stat}	> | ± 1 mV |
Uomax	$U_{o,\max}$	> 2,7 V
Uomin	$U_{o,\min}$	< 600 mV
Uon	U_{on}	< 600 mV
Uop	U_{op}	> 2,7 V

CM-Fehler (static) stellt dabei den Fehler im Nullpunkt dar, also einen immer vorhandenen, unerwünschten Offset der Regelspannung, auch ohne Ansteuerung. Er wurde auf ± 1 mV angesetzt. Der OCEAN-Befehle für CM-Fehler (static) lautet:

```
(((VDC("/OUTN")+VDC("/OUTP))/2)-1.65)
```

CM-Fehler (Min) und CM-Fehler (Max) definieren den zulässigen Fehlerbereich bei Ansteuerung. Der in Bild 13 dargestellte, grau hinterlegte Fehlerschlauch wurde mit ± 40 mV festgelegt. Der Fehler führt zu einem An- bzw. Absenken des Gleichtakts und damit zu einer Einschränkung des Aussteuerbereichs. Ursache des Fehlers ist die nicht-ideale Gleichtaktregelung.

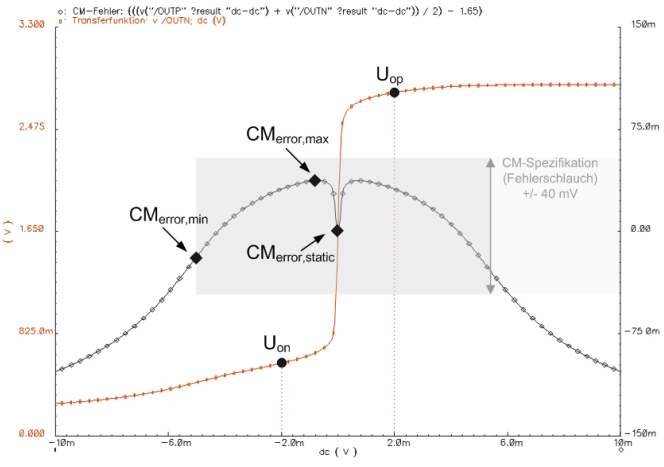


Bild 13: Transferfunktion und Gleichtakt-Fehlerkurve

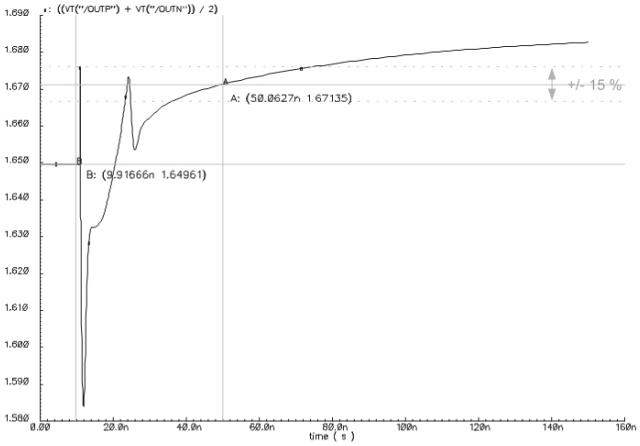


Bild 14: Gleichtaktfehler und Settling Time des CM

Die Stabilitätsbetrachtung der Reglerschaltung erfolgt im Zeitbereich über die Settling Time. Die Settling Time ist die Zeit, die der Verstärker bei differentieller Pulsanregung benötigt, um den Gleichtakt in einen vorgegebenen Fehlerbereich zu führen. Dazu wird in Bild 14 der OPV nach 10 ns differentiell mit einem Sprung ausgesteuert. Der Plot zeigt den Gleichtaktfehler über der Zeit. Das Schwingverhalten der Kurve ist auf den Regelvorgang zurückzuführen. Der Fehlerschlauch wurde bei 50 ns mit ± 15 % angesetzt. Der OCEAN-Befehle für CM-Fehler (max) lautet:

```
settlingTime(((VT("/OUTP")+VT("/OUTN"))
/2) 1e-08 t 5e-08 t 5)
```

Der Simulator Spectre benötigte mit dem verwendeten Server für einen Run mit allen Performances und Einstellungen 2,3 s.

4.3. Constraints Editor

Im WiCkeD Constraints Editor müssen vor der Optimierung diverse Einstellungen getätigt werden. Zunächst müssen die Designvariablen festgelegt werden. Bleiben Symmetrieüberlegungen unberücksichtigt verfügt die Schaltung über 72 Designvariablen, was aufgrund der Multidimensionalität des Optimierungsproblems und des damit verbundenen enormen Rechenaufwands (er wächst exponentiell mit der Anzahl an Designvariablen) ein weiteres Vorgehen ausschließt. Darum wurden neben der automatischen Strukturerkennung schaltungsspezifische Symmetrien ausgenutzt. C_0 und C_1 erhalten z. B. eine gemeinsame Variable I28_C0_c, die oberen PMOS-Transistoren MP1, MP4, MP10, MP13, MP16, MP19 und MP21 für die Kanallänge L eine gleiche Variable, die Differenzstufen-Transistoren der Gleichtaktregelung MP22 bis MP25 gleiches W und L. Tabelle 5 listet eine Auswahl aller ausgenutzten Symmetrien auf. Durch Symmetrieüberlegungen war es möglich, die Anzahl der Designvariablen auf 31 zu senken.

Tab. 5: Symmetrieüberlegungen für den OPV

Gleiches	Transistoren
L	MP1,MP4,MP10,MP13, MP16,MP19,MP21
L	MP0,MP2,MP3,MP9, MP14,MP17,MP18
W+L	MN12,MN13,MN17,MN18
C	C0,C1
W	MP6,MP7
W	MN14,MN15
W+L	MN10,MN11,MN16

Im nächsten Schritt müssen noch die Wertebereiche der Designvariablen festgelegt werden. Die Variable I28_C0_c repräsentiert beide Kapazitäten, sie startet bei 100 fF und darf sich zwischen 100 fF und 2 pF bewegen; das ihr zugeordnete Grid beträgt 1 fF (bei einer durchgeführten Rundung sind nur Werte innerhalb dieses Fertigungsrasters möglich). Alle Transistoren starten beim Technologieminimum von 0,35 µm und dürfen sich zwischen 0,35 µm und 300 µm bewegen; das Grid für Transistoren beträgt 0,05 µm.

4.4. Optimierung

Da die Design History mehrere Verzweigungen zulässt, wurden auch verschiedene Optimierungswege beschritten, die sich in ihrem Aufbau und der Wahl der Algorithmen unterscheiden. Bild 15 zeigt die Design History für den OPV. Ausgehend vom Root-Knoten (Knoten 0) wurden zunächst zwei Feasibility-Optimizations mit den Einstellungen closest-point bzw. central-point durchgeführt. Hierbei lieferte der central-point Algorithmus (Knoten 1, Dauer 30 min) Erfolg versprechende Ergebnisse, weshalb im weiteren Verlauf an diesen angeknüpft wurde.

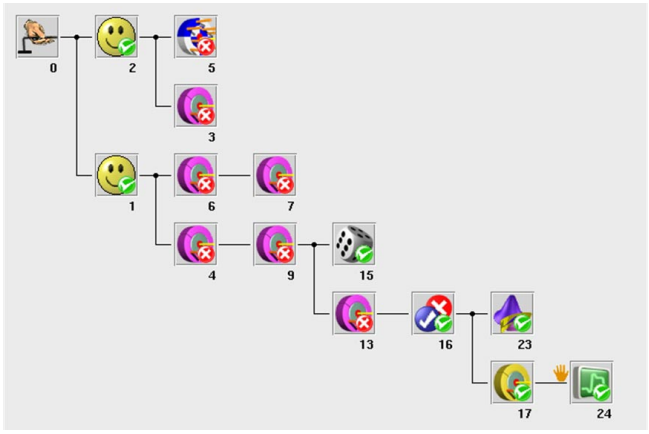


Bild 15: WiCkeD Design History für den OPV

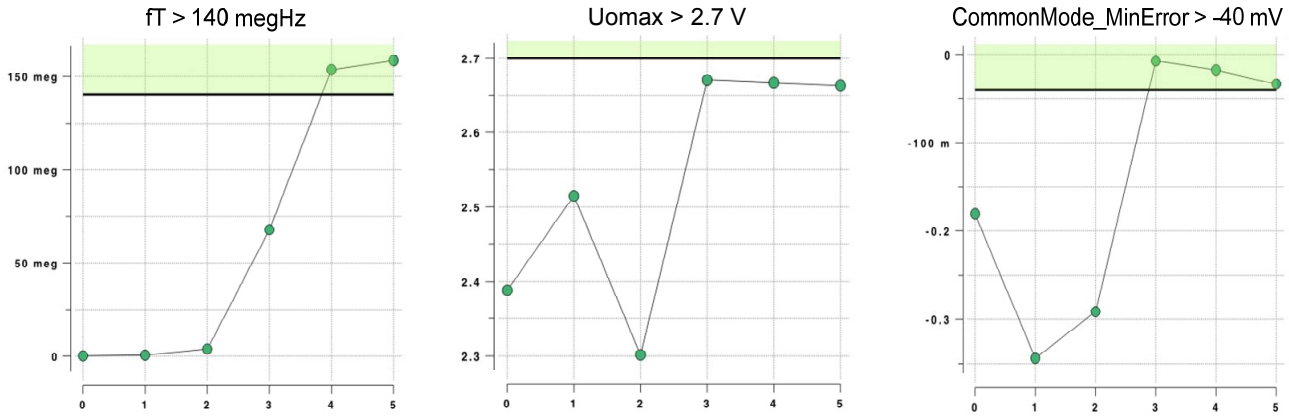


Bild 16: Exemplarische Optimierungsergebnisse Knoten 4 für Transitfrequenz, Uomax und CM-Fehler (min)

Da sich die Designvariablen zwischen 0,35 µm und 300 µm bewegen dürfen, kommen mit der Wahl des central-point Algorithmus eher Werte mittlerer Größenordnung zustande. Dadurch werden die Transistorflächen zwar größer aber WiCkeD kann dafür in einem größeren Designraum optimieren. Die Nominal Deterministic Optimization (Knoten 4, Dauer 12 h) wurde mit dem Algorithmus parameter-distance, Number of Iterations 10 und adaptive sensitivity calculation gestartet. Hier konnten bereits 10 von 14 Spezifikationen erreicht werden. Bild 16 zeigt exemplarisch 3 Performances und ihren Verlauf während der Optimierung. Die Transitfrequenz wurde nach 4 Schritten erreicht, der CM-Fehler (min) nach 3 Schritten. U_{omax} nähert sich auf 50 mV an die geforderten 2.7 V an und zählt also zu den Performances, die noch nicht erreicht werden konnten. Durch Anhängen zweier weiterer Deterministic Optimizations (Knoten 9, Dauer 10 h bzw. Knoten 13, Dauer 6 h) konnten dann jedoch 13 von 14 Spezifikationen erreicht werden. Die einzige verletzte Performance ist der diskrete Punkt U_{OP}, der jedoch bis auf 50 mV an die Spezifikation herangeführt wurde. Hier scheint eine prinzipielle Beschränkung der Topologie vorzuliegen, die nur durch einen Trade-Off mit anderen Spezifikationen erreicht werden könnte. Zu bemerken ist, dass es nicht ausreicht eine Deterministic Nominal Optimization mit hoher Iterationszahl einzustellen, sondern dass es notwendig ist, mehrere Knoten aneinanderzureihen. Treten ab einem gewissen Iterationsschritt keine Verbesserungen mehr auf, beendet der Algorithmus unabhängig von der eingegebenen Number of Iterations die Optimierung (hier: nach 5 Iterationsschritten). Das angehängte Screening (Knoten 16, Dauer 2 h) eliminiert diverse Prozessparameter, um die Rechenzeit zu verkürzen. 10 von 23 eingebunden Prozessparametern wurden wegen eines Einflusses kleiner 5 % vernachlässigt. Nach einer Rundung auf das Fertigungsgrid (Knoten 24, Dauer 1 min) lag eine fertige Dimensionierung mit einem geschätzten Yield von 84 % vor.

Die in das ursprüngliche Schematic zurück geschriebenen Bauteilwerte ermöglichen eine Nachsimulation in Cadence. Dabei zeigte sich interessanterweise, dass trotz erreichter Spezifikation, die Schaltung nicht wie gewünscht funktionierte. Das Problem lag in der Gleichtaktregelung, die bei differentieller Ansteuerung nicht wie gewünscht arbeitete (siehe Bild 15, Kurve ALT). Dem Optimierer wurden vor der Optimierung noch zu viele Freiheitsgrade gelassen. Durch Einführen zweier weiterer Performances TranStabil(Max) und TranStabil(min) sollte im Zeitbereich ein Toleranzschlauch hinzugefügt werden, den der Gleichtakt am Ausgang nicht über- oder unterschreiten darf. Die OCEAN-Befehle sind im Einzelnen:

```
TranStabil(Max)          =          (y-
max(clip(((VT('/OUTP')   +   VT("/OUTN"))
/2) 6e-08 2e-07)) - 1,65)

TranStabil(Min)          =          (y-
min(clip(((VT('/OUTP')   +   VT("/OUTN"))
/2) 6e-08 2e-07)) - 1,65)
```

In Bild 17 ist zweimal der Gleichtakt über der Zeit aufgetragen, einmal für die alte Dimensionierung, einmal

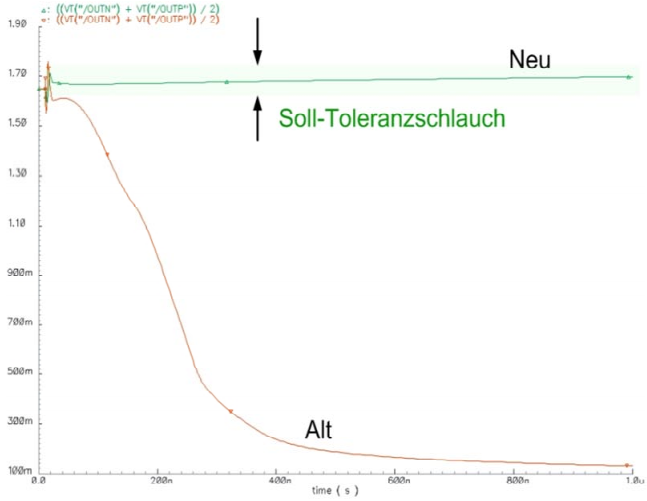


Bild 17: Gegenüberstellung der Gleichtaktregelung

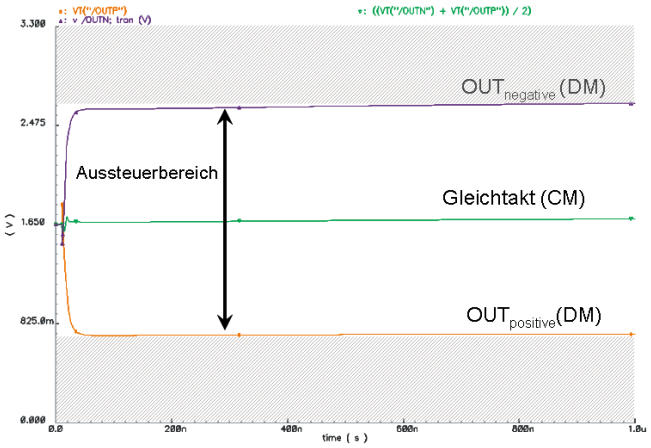


Bild 18: Differentieller Ausgang des OPV mit Gleichtakt

für die neue Dimensionierung, wobei letztere aus dem um TranStabil(Min) und TranStabil(Max) erweiterten Design Flow hervorgegangen ist. Der um diese zwei Performances erweiterte Design Flow führte folglich zu einer funktionierenden Schaltung. Damit liegt eine die Spezifikation erfüllende Dimensionierung vor, deren korrektes Verhalten in Bild 18 noch einmal abschließend dargestellt ist. Man erkennt deutlich das Regelverhalten auf $V_{DD}/2$ bei differentieller Anregung. Der Aussteuerbereich bewegt sich zwischen 600 mV und 2.7 V. Alle Spezifikationen werden eingehalten, unter Berücksichtigung der Streuparameter wird der Yield auf 84 % geschätzt.

Die Optimierung umfasste 34761 Simulationsläufe von je 2,32 s Dauer. Die gesamte Rechenzeit belief sich unter Berücksichtigung der Bedienungszeit in der Design History und sonstigen Zeiten auf etwa 72 h. Die Bauteilwerte der von WiCkeD dimensionierten Designvariabeln sind in Tabelle 6 in Auszügen aufgeführt. Die Dimensionierung erfolgte in Hinblick auf die Spezifikation und ist nicht flächenoptimal. Sollte die Gesamtfläche übergeordnete Bedeutung haben, müssen kleinere Wertebereiche gewählt werden.

Tab. 6: Auszug der dimensionierten Designvariablen

Transistoren	**W**	**L**
MP6,MP7	260,2 µm	1,45 µm
MN3, MN8	106,2 µm	0,55 µm
MN4, MN7	182,5 µm	22,8 µm
MP9, MP14	111,8 µm	4,45 µm
MN14,MN15	49,9 µm	172,4 µm
MP22,MP23,MP24,MP25	3,55 µm	6,8 µm
MN10,MN11,MN16	115,9 µm	0,85 µm
C0,C1	124 fF	

5. Vergleich weiterer Ergebnisse und Diskussion

Um ein umfassenderes Bild der Fähigkeiten von WiCkeD zu erhalten, wurden insgesamt fünf Topologien unterschiedlicher Art untersucht und bearbeitet. Eine Übersicht ist in Tabelle 7 dargestellt. Neben den bereits besprochenen Schaltungen Miller-OTA und OPV werden ein Transimpedanzverstärker (TIA) und Komparator für Highspeed-Anwendungen sowie eine Bandgap-Referenz vorgestellt. Jede Schaltung ist durch die Anzahl ihrer Bauelemente, der Anzahl an Designvariablen, der Schaltungsart, der Komplexität, der Vorentwicklungs- und Rechenzeit charakterisiert. Die Verwendbarkeit der Dimensionierung ist in der Zeile „Ergebnis“ dargestellt. Die Dimensionierung aller Schaltungen startete beim Technologieminimum (0,35 µm). Es zeigte sich, dass einfache, lineare Schaltungsprobleme wie Miller-OTA und TIA problemlos in WiCkeD dimensioniert werden können. Ergebnisse konnten hier bereits nach 16 h Vorentwicklungs- und 30 h Rechenzeit erzielt werden. Bei nichtlinearen Topologien muss mit einem erhöhten Zeitaufwand gerechnet werden. Für den Komparator waren 64 h Vorentwicklungs- und 40 h Rechenzeit notwendig, um zu einem Ergebnis zu gelangen. Für die Bandgap-Referenz hingegen konnte trotz erhöhten Aufwandes kein zufrieden stellendes Ergebnis erzielt werden – es zeigte sich, dass die Optimierung spezieller Topologien problembehaftet ist. Der OPV aus Kapitel 4 stellt in der Tabelle die komplexeste Schaltung dar und benötigte wie erwartet den höchsten Zeitaufwand. Nach 160 h Vorentwicklungs- und 72 h Rechenzeit war es möglich, eine verwendbare Dimensionierung zu erhalten.

Im Folgenden werden die in dieser Arbeit gewonnen Erfahrungen aus Anwendersicht diskutiert. WiCkeD kann problemlos in die Cadence DFII eingebunden werden. Die Benutzeroberfläche und der Design Flow erwiesen sich als übersichtlich und durchdacht. Die Nutzung des Werkzeugs erfordert allerdings einen hohen Einarbeitungsaufwand, jedoch kann ein Wissensgewinn durch die notwendigen Vorüberlegungen erzielt werden. Die Algorithmen von WiCkeD arbeiten sehr stabil und zuverlässig – in der einjährigen intensiven Nutzungszeit trat kein Programmabsturz auf. Selbst mit einer durchschnittlichen Rechenleistung bewegt sich die erforderliche Rechenzeit in einem akzeptablen Bereich. Zusammenfassend lässt sich sagen, dass eine Optimierung komplexer Schaltungen oftmals ein händisches, rekursives Eingreifen des Entwicklers erfordert – in diesem Zusammenhang muss von einem semi-automatisierten Design Flow gesprochen werden. Allerdings wird der Wunsch nach einer automatisierten Dimensionierung für einfache, lineare Schaltungen von WiCkeD erfüllt.

Tab. 7: Vergleich der mit WiCkeD erzielten Ergebnisse für unterschiedliche Topologien

	Miller-OTA	**TIA**	**Komparator**	**OPV**	**Bandgap**
Bauelemente	8	9	22	38	15
Designvariablen	12	19	34	31	17
Schaltungsart	linear	linear	nichtlinear	nichtlinear	nichtlinear
Komplexität	niedrig	niedrig	mittel	hoch	mittel
Vorentwicklungszeit	niedrig (16 h)	niedrig (16 h)	mittel (64 h)	hoch (160 h)	mittel (48 h)
Rechenzeit	niedrig (24 h)	niedrig (30 h)	hoch (70 h)	hoch (72 h)	mittel (40 h)
Ergebnis	ja	ja	ja	ja	nein

6. Zusammenfassung

Die hohe Komplexität und niedrige Produktivität im analogen Schaltungsbereich gründeten sich zum Teil auf eine fehlende Automatisierung des Entwicklungsprozesses. Der Wunsch nach Automatisierung war und ist seit langem ein Schwerpunkt der EDA-Entwicklung. Das neue Tool MunEDA® Wicked bietet einen solchen Lösungsansatz und wurde deshalb im Labor Mikroelektronik der Hochschule Ulm untersucht und bewertet.

WiCkeD wurde auf das Cadence Design-Framework II aufgesetzt und an die AMS C35B4 Technologie angebunden – es konnten alle relevanten Prozess- und Mismatchparameter berücksichtigt werden. Der Design Flow wurde sowohl für einfache als auch komplexe Topologien durchgeführt. Es zeigte sich, dass WiCkeD für einfache Topologien einen vollautomatisierten, für komplexe Topologien lediglich einen semiautomatisierten Design Flow ermöglicht. Auf die Erfahrung des Analogentwicklers kann dabei nicht verzichtet werden - WiCkeD ist als hilfreiche Unterstützung anzusehen. Besonders hervorzuheben ist, dass das Tool erstmals eine Verbesserung der Chipausbeute aus Entwicklersicht ermöglicht (Design for Yield). Nimmt man den Einarbeitungsaufwand einmal in Kauf, kann der Entwicklungsprozess nennenswert beschleunigt werden. Dieser Zusammenhang konnte in dieser Arbeit für eine Vielzahl an Topologien unterschiedlichster Art nachgewiesen werden.

WiCkeD ist ein vielversprechendes EDA-Werkzeug mit großem Zukunftspotential, eine Sicht, die viele Unternehmen der Halbleiterbranche bereits teilen. Diese moderne Methode des Schaltungsentwurfs wird auch in Zukunft an der Hochschule Ulm in Forschung und Lehre Anwendung finden.

7. Literatur

[1] International Technology Roadmap for Semiconductors, 2000, http://www.itrs.net/reports.html

[2] H. Graeb, „Analog Design Centering and Sizing“, Springer, 2007 (ISBN: 978-1-4020-6003-8)

[3] MunEDA GmbH, „WiCkeD Manual“, Version 5.2, 2007

[4] R. J. Baker, „CMOS Circuit, Layout and Simulation“, 2nd Edition, John Wiley & Sons Inc., 2005 (ISBN: 0-471-70055-X)

[5] P. Gray, P. Hurst, S. Lewis, R. Meyer, „Analyis and Design of Analog Integrated Circuits“, 4th Edition, John Wiley & Sons Inc., 2001 (ISBN: 0-471-32168-0)

FPGA Implementation of a HOG-based Pedestrian Recognition System

Sebastian Bauer, Ulrich Brunsmann, Stefan Schlotterbeck-Macht

Faculty of Engineering
Aschaffenburg University of Applied Sciences, Aschaffenburg, Germany

{sebastian.bauer, ulrich.brunsmann, stefan.schlotterbeck-macht}
@fh-aschaffenburg.de

With respect to road crash statistics, on-board pedestrian detection is a key task for future advanced driver assistance systems. In this paper, we describe the implementation of a real-time pedestrian recognition system that combines FPGA-based extraction of image features with a CPU-based object localization and classification framework. In terms of features, we have implemented the Histograms of Oriented Gradients (HOG) descriptor that is state-of-the-art in the field of human detection from a moving camera. While past HOG-related publications presented simplified FPGA-based HOG variants, often sacrificing classification performance, we implemented the original descriptor with minor modifications on dedicated hardware. Evaluation on the INRIA pedestrian database shows potential for deploying the system in practice. The descriptor computation runs on a PCIe frame grabber with embedded FPGA that can be directly integrated into an automotive computer of a test vehicle for evaluation purposes.

1. Introduction

1.1. Problem Scope

According to the preliminary report of the German Federal Statistical Office [Destatis 2009], 654 pedestrians were killed in road traffic crashes in Germany in 2008, a percentage of 15% of all road traffic related deaths (4,482). The global impact of the problem can be deduced from the first major international report on road traffic [WHO 2004]. Worldwide, an estimated 1.2 million people die in road traffic crashes annually, while the number injured could be as high as 50 million. According to model predictions, between 2000 and 2020, there will be a global increase of 67%. The majority of deaths are currently among vulnerable road users (VRU): pedestrians, pedal cyclists and motorcyclists. This is especially true in low-income and middle-income countries because of the greater variety and intensity of traffic mix and the lack of separation from other road users. In view of the economic impact, the direct costs of global road crashes have been estimated at more than US$ 500 billion per annum [WHO 2004].

1.2. Recognition of Traffic Participants

Advanced driver assistance systems (ADAS) have the potential to save numerous lives, for a survey see [Gandhi 2007]. Future pedestrian recognition systems will detect the person, predict the collision risk and warn the vehicle's driver or engage automatic braking, manoeuvring or safety devices. In the past decade, automotive night vision systems have been introduced as optional equipment on few premium vehicles. These systems are either based on near infrared (NIR, active) or far infrared (FIR, passive) sensors and increase the vehicle's driver perception distance in darkness. Traffic participants can be recognized and distinguished early when they are all but invisible by the vehicle's headlights. However, to our knowledge there is no ADAS commercially available yet that is able to recognize pedestrians in daylight situations.

Owing to the variety of possible applications in the field of surveillance, robotics and intelligent vehicles, among others, detection of humans in general has driven a surge of interest from the computer vision community over the past years, see [Gavrila 1999], [Moeslund 2006], [Poppe 2007]. As we focus on automotive applications, we use the term pedestrian in the remainder of the paper.

Vehicle-based pedestrian recognition systems are one component of the solution, but visibility from the vehicle is limited. We are developing an infrastructure-based system for ensuring crossroads safety. Such systems could complement vehicle-based hardware by monitoring the traffic and transferring the information through wireless communication channels. The fusion of complementary information of both units

can increase the virtual perception range of the vehicle and provide a more complete picture of the scene, being the foundation for proper judgements and reactions of the driver.

In this work we investigate the entropy of grayscale monocular video data for the recognition of objects. Our pedestrian detection concept is also applicable to moving platforms without algorithmic modification, for instance as a future ADAS. The system is running on a personal computer equipped with a frame grabber with an embedded field programmable gate array (FPGA) that can be used as an image pre-processor. Thus the framework can be integrated and evaluated directly in a test vehicle.

Our approach is based on a sliding-window method that evaluates image sections based on the HOG descriptor that is state-of-the-art in human detection (Fig. 1). A major contribution of this paper is the implementation of the descriptor on dedicated hardware with minor modifications compared to the scheme originally proposed by Dalal&Triggs [Dalal 2005]. While maintaining a similar classification performance level, the descriptor computation outperforms existing approaches in terms of speed. Besides, FPGA implementations are applicable for automotive applications thanks to established low cost production flows. To the best of our knowledge there has been no FPGA implementation of a HOG-based pedestrian recognition system presented in literature before.

The descriptor computation for the entire image is performed on a Xilinx Spartan 3 XC3S 4000 for the most part, leaving only a final normalization step to the CPU. Classification is performed on a graphics processing unit (GPU).

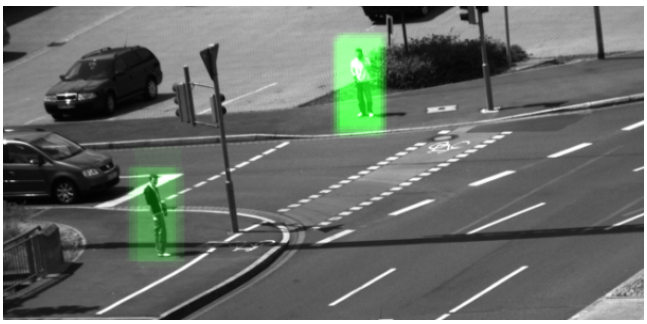


Fig. 1: Pedestrians detected with our HOG-based detection system.

The paper is organized as follows: We briefly discuss previous work in section 2. The description of the method in section 3 is followed by implementation details in section 4. Experimental results are presented in section 5, before we conclude with a summary and discuss open issues in section 6.

2. Related Work

2.1. State of the Art

Human detection in images is a challenge because of the wide variability in appearance that results from different poses and clothing, illumination conditions and complex backgrounds that are common in outdoor scenes. The excessive amount of interest in pedestrian detection led to a broad variety of different approaches in terms of system architecture, feature concepts and classification schemes.

In terms of architecture, systems can be divided into two major strategies. Part-based approaches [Ioffe 2001], [Felzenszwalb 2005], [Mikolajczyk 2004] employ individual detectors to locate single body parts that are interpreted as a human if they are arranged in a geometrically plausible configuration. Holistic approaches use a single detection window that is shifted over the image at dense positions and scales. Well known sliding-window systems rely on shape-based detection (comparison of edge images to a hierarchical exemplar dataset) [Gavrila 2007], Haar wavelet feature sets [Papageorgiou 2001], rectangle filters [Viola 2005] or the periodicity of the human walk [Fardi 2006]. Authors of both methods have offered a wide range of feature sets and classifier concepts in this domain, for a survey see [Wojek 2008b].

Recent experimental studies [Dollár 2009], [Enzweiler 2009] show that Histograms of Oriented Gradient descriptors are robust features for human detection and leading edge in terms of classification performance. Dollár et al. benchmarked seven promising pedestrian detectors on the Caltech Pedestrian Dataset [Caltech], providing an overview of state-of-the-art performance. All detectors are based on a sliding-window strategy, all use variants of HOG or Haar features. The experiments show that HOG remains competitive even on this challenging dataset. It is keeping pace with the MultiFtr and FtrMine feature sets that tend to outperform all other methods surveyed [Dollár 2009]. However, the latter two descriptors cannot compete with HOG in terms of computational load.

Enzweiler&Gavrila provide an extensive survey of pedestrian detection systems and also evaluate state-of-the-art detection systems. They use a database that is made publicly available as the Daimler pedestrian detection benchmark set [Daimler]. The experimental results show that the HOG approach outperforms all other systems they surveyed in terms

of classification performance at intermediate pedestrian image resolutions (48x96 pixels).

2.2. Histograms of Oriented Gradients

The HOG descriptor was introduced by Dalal&Triggs in 2005 as a feature set for object recognition tasks. At that time, this novel descriptor outperformed existing feature sets for human detection significantly. The basic idea is that local object appearance and shape is characterized by the distribution of local intensity gradients or edge directions, without precise knowledge of the corresponding gradient or edge positions [Dalal 2006].

The most important requirement for a pedestrian recognition system is the demand for real-time operation. However, according to Dalal&Triggs, the runtime of the object detection system lies in the range of a second for a 320x240 pixels image using a support vector machine (SVM) as classifier. For an image with twice the width and height, Dollár et al. report 13.3 seconds in their benchmark report. There have been several different approaches to speed up the HOG framework. Zhu et al. [Zhu 2006] accelerate the descriptor computation by employing an integral map (IMAP) and by replacing the SVM by a cascade-of-rejectors approach. In contrast to these software-based acceleration techniques, few authors presented HOG implementations on dedicated hardware. Besides Wojek et al. [Wojek 2008a] who take advantage of the emerging computing power of general purpose computation on graphic processing units (GPGPU), Cao&Deng [Cao 2008] presented an FPGA-based implementation of a simplified HOG framework. They realized a real-time stop sign detection system on a Xilinx Virtex-4 FPGA that is able to process 60 frames of 752x480 pixels per second by using a cascaded classifier.

3. Overview of the Method

3.1. Sliding-Window Framework

A common technique to obtain initial object location hypotheses is the sliding window scheme, where a detection window is shifted on a regular lattice over the image at various scales. For each window, a feature set (in this case HOG) is generated from the corresponding image patch and evaluated by a pre-trained classifier that categorizes unknown samples into one of the predefined classes, pedestrian or non-pedestrian in this case. Nearby detections of the same object are common with sliding-window frameworks and are typically merged using non-maxima suppression approaches in order to yield bounding boxes with confidence levels for the final detections. Sliding-window-based detection techniques showed promising performance for human recognition and have become very popular in this domain in recent literature. However, they are often considered unfeasible for real-time operation due to the immense costs in terms of resources and processing time. Significant speed-ups can be obtained by incorporating a priori information (scene geometry, target object) or by employing special classifier concepts e.g. a cascade of weak classifiers with increasing complexity. Early rejection often comes with a performance loss and methods that sacrifice classification performance to achieve speed-ups do not stand in the long term [Wojek 2008a]. As a consequence, we use a kernel support vector machine as classifier, providing the best results in Dalal&Triggs' performance study.

3.2. Descriptor Computation

The HOG descriptor is computed for each detection window with the process chain illustrated in fig. 2.

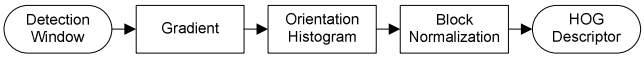
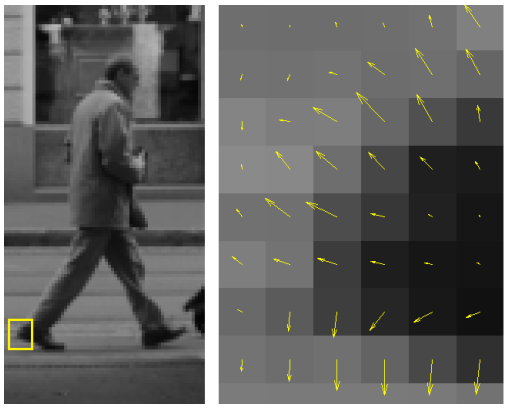


Fig. 2: HOG descriptor computation scheme.

The default detector introduced by Dalal&Triggs is based on a detection window that covers 64x128 pixels. First of all, the intensity gradient in x- and y-direction and the resulting magnitude and orientation angle is computed for the respective image patch. For an illustration of the two-dimensional gradient vectors, see fig. 3.



(a) (b)

Fig. 3a: Pedestrian example from [INRIA]. Fig. 3b: Gradients computed for an image section. The yellow arrows represent the two-dimensional gradient vector $[g_x \; g_y]^T$. Magnitude is encoded by the vector length, orientation by the vector angle.

The next step is called orientation binning and represents the fundamental nonlinearity of the HOG descriptor. The detection window is divided into 8x16 rectangular local spatial regions called cells (Fig. 4). The 8x8 cell pixels are then discretized into 9 angular bins according to their gradient orientation. Each pixel contributes a weighted vote for its corresponding angular bin, the vote is a function of the gradient magnitude at the pixel. This way the information is compressed to a 9-dimensional space per cell. The angular histogram bins are evenly spaced over 0°-180°.

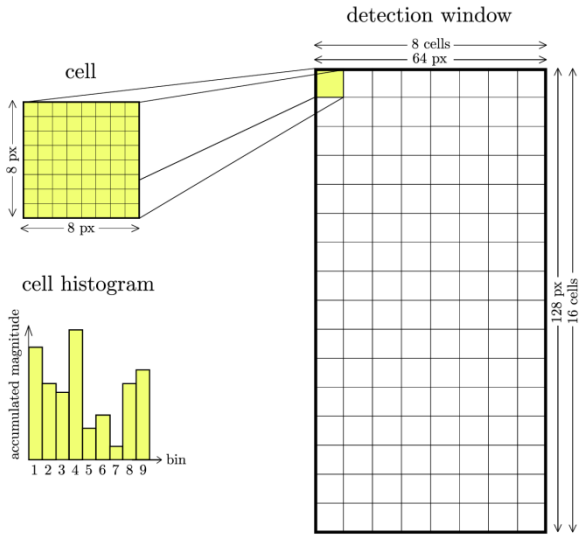


Fig. 4: Division of detection window into cells. Histogram generated for a single cell.

Gradient strengths vary over a wide range due to shadows, local variations in illumination and foreground-background contrast. Therefore local contrast normalization is essential for good performance. For this purpose, groups of 2x2 adjacent cells are considered as spatial regions called blocks. Each block is represented by a concatenation of the corresponding four cell histograms, resulting in a 36-D feature vector that is normalized to unit length, using the L2 norm.

The final HOG descriptor is represented by a concatenation of all these normalized block responses. In fact, blocks typically overlap with each other in a sliding-window fashion so that each cell response appears several times in the final feature vector, each normalized with respect to a different block. The default block stride is 8 pixels (1 cell), resulting in a fourfold coverage of each cell. To summarize: Each detection window is represented by 7x15 blocks, a block consisting of 2x2 cells, a cell is represented by a 9-bin histogram, giving a total of $(7x15) \cdot (2x2) \cdot 9 = 3780$ features.

3.3. Classification

The generated HOG descriptor is used to categorize the image patch into one of the predefined classes, pedestrian or non-pedestrian. For this classification step, Dalal&Triggs employ a support vector machine (SVM) that learns an implicit representation of the classification object from examples. In case of HOG, the classifier is pre-trained with the 3780-dimensional descriptor that is generated for all positive (pedestrian) and negative (non-pedestrian) samples, available from established datasets.

There are many different classifier concepts in the domain of supervised learning. Owing to the popularity in literature - in particular for pedestrian recognition approaches, see [Gandhi 2007], [Munder 2006] for a comparison - we also decide in favour of support vector machines.

4. Implementation

4.1. System Overview

The proposed system is based on an FPGA-CPU-GPU network. We outsourced the corpus of the HOG descriptor computation to an FPGA. The descriptor normalization step is then performed on the CPU, being the central entity of our system. The SVM-based sliding-window evaluation is currently running on a publicly available GPGPU solution [cuSVM]. An overview of the framework is shown in fig. 5. This paper focuses on the FPGA-based HOG descriptor generation.

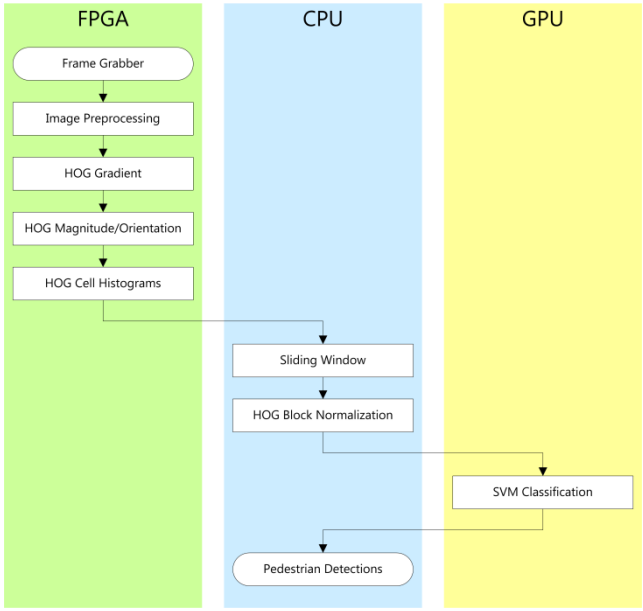


Fig. 5: Overview of the implemented FPGA-CPU-GPU framework.

4.2. Rapid Prototyping Platform

Our implementation is developed and runs on the commercially available microEnable IV-FULLx4 image acquisition and pre-processing frame grabber board from [SiliconSoftware]. It is equipped with two FPGAs, in our case a Xilinx Spartan 3 XC3S 2000 and a Spartan 3 XC3S 4000. The former provides data transfer interfaces to the camera (CameraLink) and to the host (PCIe), the latter and on-board memory is available for programming individual real-time image processing applications. The block diagram of this rapid prototyping platform is shown in fig. 6.

The board provides two external CameraLink ports that can operate with any CameraLink conform camera with Base, Medium or Full configuration. CameraLink is a serial communication protocol in the field of industrial image processing that works at a pixel frequency of 85 MHz and can transmit up to 680 MB/s by using parallel streams (taps). We use a CCD camera from [Sentech] with UXGA resolution (1600x1200 pixels) that operates at a frame rate of 30 Hz and a pixel frequency of 73.636 MHz.

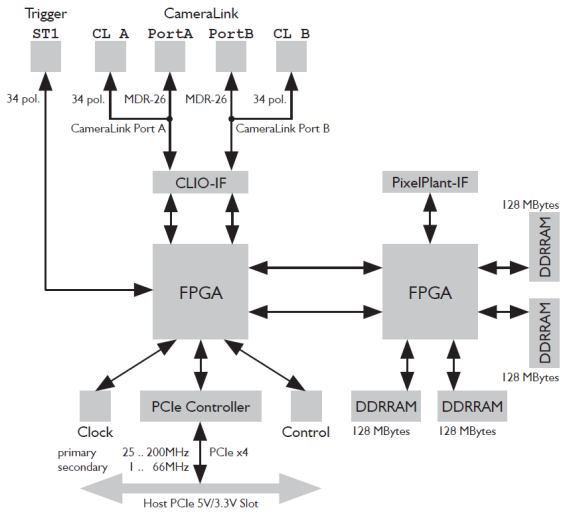


Fig. 6: Block diagram of the rapid prototyping platform microEnable IV-FULLx4 from [SiliconSoftware].

The HOG descriptor computation is implemented on the Spartan 3 XC3S 4000 that can access 4x128=512 MB on-board DDR-RAM memory. Our design is running at a frequency of 63 MHz, while parallelization enables processing of up to 32 pixels per clock cycle. For transferring the computed HOG descriptor into the main memory of the host computer, a PCIe x4 (quad lane) interface is used that achieves a bandwidth of 760 MByte/s.

With this platform, our system can be tested directly in a test vehicle by integrating the board into an automotive computer that is connected to an on-board CameraLink conform camera.

Based on a rapid prototyping idea, the Spartan 3 XC3S 4000 can be programmed with VisualApplets (VA), a graphic-oriented hardware development software from [SiliconSoftware] that is based on tools from the Xilinx ISE design suite [Xilinx]. VA enables an abstract high-level hardware implementation of customized image processing applications operating in real-time. The individual design is arranged by image processing operators and transport links that form a graphical data flow. Each operator/link can be parameterized graphically.

4.3. FPGA Design Components

Our design computes the HOG descriptor for all window positions (pixel-wise) of the entire frame. In addition to the HOG descriptor, a reference copy of the current frame is transferred through an individual direct memory access (DMA) channel. In the following, we present each individual system component. We report difficulties we faced and show our way of tackling these issues with respect to the limitations of the rapid prototyping platform.

Timing synchronization

The pixel frequency of the stream delivered by the camera is higher than the design frequency, hence buffering the image first into on-board DDR-RAM is necessary in terms of timing synchronization. Nonetheless the frequency decrease from camera to FPGA is no bottleneck, since for further processing a set of pixels can be read from the image buffer in a parallel fashion.

Scaling

With the current experimental arrangement of our infrastructure-based system, the distance of the camera to the surveillance zone is large compared to the dimensions of the surveillance zone itself. As a consequence of this arrangement, variations in pedestrian size are negligible and the descriptor is computed for one single scale level. The scale factor was set to a manually chosen value that shrinks the actual pedestrian size to the dimension of the image patches used for training [INRIA]. For applications that require multiple scale levels, the FPGA design can be modified accordingly. The maximum number of scale levels is limited by the available hardware resources.

Gradient Computation

The first step for generating the HOG descriptor is to compute the 1-D point derivatives G_x and G_y in x- and y-direction by convolving the gradient masks M_x and M_y with the raw image I:

$$G_x = M_x * I \qquad M_x = [-1\ 0\ 1]$$
$$G_y = M_y * I \qquad M_y = [-1\ 0\ 1]^T$$

On the basis of the derivatives G_x and G_y we then compute the gradient magnitude $|G(x,y)|$ and orientation angle $\phi(x,y)$ for each pixel. The gradient magnitude expresses the gradient strength at a pixel:

$$|G(x,y)| = \sqrt{G_x(x,y)^2 + G_y(x,y)^2}$$

We do not leave out the extraction of the square root since the performance study of Dalal&Triggs reports best results with this Euclidean metric. In our FPGA implementation, the square root operator cuts off decimal places, as we do not observe effects on classification accuracy (see section 5.1). The gradient magnitude is merely employed as a weighting factor for the orientation histogram.

The gradient orientation angle can be calculated straightforward from:

$$\tan\big(\phi(x,y)\big) = \frac{G_y(x,y)}{G_x(x,y)}$$

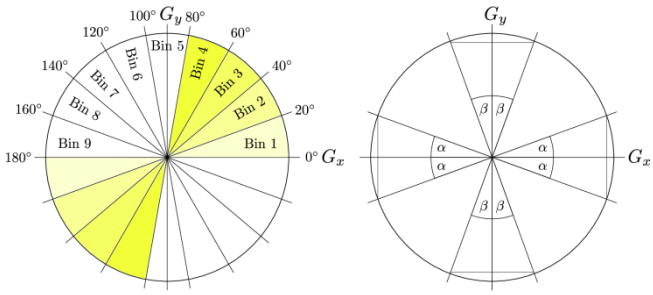
However, calculating the $\arctan()$ on an FPGA is expensive. As reported by Cao&Deng, there are hardware friendly approximation algorithms available, but they are generally iterative and slow down the system's speed. On the contrary, using lookup tables (LUTs) requires large amounts of memory which would increase system's costs. They propose to combine the gradient orientation computation with the angular binning step. Thus we are able to directly discretize the pixel's gradient angle into bins without computing the angular value explicitly.

Following this approach [Cao 2008], we introduced two improvements. We increased the number of bins from 4 to 9 in order to achieve better classification performance. In addition to that, we introduced a scheme for quantizing the pixel's gradient angle that avoids the use of signs and reduces the required bit width for relational operators.

Gradient Orientation Binning

As stated in section 3.2, the step following the gradient computation is to discretize each pixel's gradient orientation angle into 9 evenly spaced angular bins over 0°-180° (unsigned gradient), see fig. 7a. Based on the previously computed horizontal and vertical gradients G_x and G_y we first determine the angle's corresponding quadrant according to the following rule set:

$$G_x(x,y) > 0 \ \wedge\ G_y(x,y) > 0 \ \Leftrightarrow\ quadrant\ I$$
$$G_x(x,y) < 0 \ \wedge\ G_y(x,y) > 0 \ \Leftrightarrow\ quadrant\ II$$
$$G_x(x,y) < 0 \ \wedge\ G_y(x,y) < 0 \ \Leftrightarrow\ quadrant\ III$$
$$G_x(x,y) > 0 \ \wedge\ G_y(x,y) < 0 \ \Leftrightarrow\ quadrant\ IV$$



(a) (b)

Fig. 7a: Angular quantization into 9 evenly spaced orientation bins over 0°-180° (unsigned gradient). Fig. 7b: Quadrant-respective angular binning with respect to the horizontal and vertical principal axis of the unit circle.

We then compute the pixel's gradient orientation angle α with respect to its respective quadrant (0°-90°). Hence the corresponding orientation bin can be determined by quadrant and angle. The following scheme for bin 1 (0°-20°) illustrates how the $\arctan()$ computation is replaced by simple integer multiplications:

$$Cond.I:\ G_x(x,y) > 0 \ \wedge\ G_y(x,y) > 0 \ \Leftrightarrow\ quadrant\ I$$
$$\vee\quad G_x(x,y) < 0 \ \wedge\ G_y(x,y) < 0 \ \Leftrightarrow\ quadrant\ III$$

$$Cond.II: \quad 0 < \tan(\alpha) < \tan(20°)$$
$$0 < \left|\frac{G_y(x,y)}{G_x(x,y)}\right| < \tan(20°)$$
$$0 < \big|G_y(x,y)\big| < \tan(20°) \cdot |G_x(x,y)|$$
$$0 < \big|G_y(x,y)\big| < 0.364 \cdot |G_x(x,y)|$$

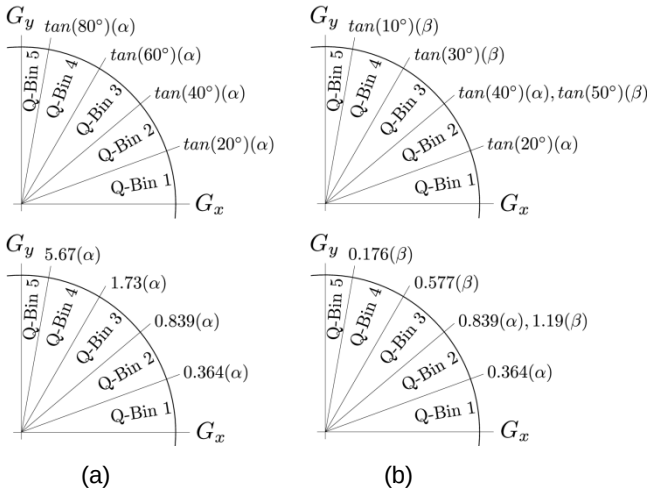
The multiplication in the right part of the last inequation above is a floating point operation. For FPGA implementation, the inequation is to be multiplied by a scalar >> 1 in order to replace the floating point by fixed point / integer operations. Superior is the use of bit shift operations.

The quadrant-respective angular binning is performed with respect to the horizontal and vertical principal axis of the unit circle, respectively. In case the angle α

to the horizontal axis is less than 40°, the binning is based on $\tan(\alpha)$, else on $\tan(\beta)$, see fig. 7b.

$$\tan(\alpha) = \left|\frac{G_y(x,y)}{G_x(x,y)}\right| \qquad \tan(\beta) = \left|\frac{G_x(x,y)}{G_y(x,y)}\right|$$

The advantage of this differentiation is illustrated in fig. 8. The comparison range in terms of values for condition II is reduced significantly, resulting in an equivalent reduction of bit width for the relational operators.



(a) (b)

Fig. 8a: Standard angular binning of the gradient vector $[G_x(x,y)\ G_y(x,y)]^T$. Fig. 8b: Using both α and β for angular binning reduces range of values for condition II.

Histogram Generation

At this stage, we already know the angular bin (1-9) for each pixel. Then 9 binary single-channel images O_i are generated for each bin i, where the value 1 denotes that the pixel's gradient orientation lies within the corresponding angular range, 0 denoting the opposite. In a second step, we multiply each of these 9 binary bin images O_i with the gradient magnitude $|G(x,y)|$, providing 9 non-binary magnitude-weighted bin images M_i.

For each sliding window position, the histogram entry of a specific bin i in a particular cell can be easily computed by accumulating the pixel intensity values (representing the magnitude) over the cell region within M_i. In order to calculate these histogram entries for all potential sliding window positions over the entire image efficiently, recent publications work with IMAPs [Zhu 2006], [Cao 2008]. We found that this approach is not feasible with our development platform due to constraints in terms of buffers. Instead we convolute the following sum filter kernel K_S with the nine magnitude-weighted bin images M_i:

$$B_i = K_S * M_i \qquad K_S = \begin{bmatrix} 1 & \cdots & 1 \\ \vdots & \ddots & \vdots \\ 1 & \cdots & 1 \end{bmatrix}$$

4.4. What remains for the CPU/GPU

The 9 bin images B_i are to be transferred to the PC via DMA. On the CPU, the detection window is shifted over the entire image in a sliding-window fashion. The histogram entries for a specific detection window can be easily read out from the FPGA output. After that, only the final HOG descriptor block normalization step remains. At first the 4 cell histogram vectors of the current block are concatenated to a vector $\boldsymbol{v}$ with $4 \cdot 9 = 36$ components. The normalization is then performed by dividing $\boldsymbol{v}$ by the L2 norm:

$$\boldsymbol{v} \longrightarrow \frac{\boldsymbol{v}}{\sqrt{\|\boldsymbol{v}\|_2^2 + \epsilon^2}}$$

where ϵ is a small constant inhibiting divisions by zero. The values of the normalized vector are then clipped to a certain limit ([Dalal 2005]: 0.2) and the entire vector is re-normalized. This type of normalization is also known as L2-Hys.

We use a GPGPU Gaussian kernel SVM [cuSVM] for classification, operating on the normalized HOG feature vectors that are stored line by line in a matrix. The GPU can thus parallelize the classification of all windows.

5. Evaluation

5.1. Classification Performance

SVM training is performed using the INRIA dataset [INRIA] that remains the most widely used benchmark set. For the purpose of evaluation, in the following we use Dalal&Triggs' original work as a reference. Hence, a per-window evaluation is performed. Though in practice, per-window performance measures can fail to predict actual per-image performance [Dollár 2009].

In terms of positive training examples, the INRIA dataset contains 2,416 image crops (64x128 pixels) of people in roughly upright poses. 12,180 negative training examples are generated by randomly extracting image patches of the same dimension from 1,218 pedestrian-free photos. The HOG descriptors of the training examples are extracted using our FPGA-based implementation that is modified in a way that

the frame grabber module is replaced by a buffer that is able to load an image from the PC to the frame grabber's on-board memory. We performed the training process on the GPU. It took a few seconds using [cuSVM].

For evaluation, we classified 1,132 positive and 4,530 negative unseen examples (randomly sampled from 453 pedestrian-free images) from [INRIA], visualized with Detection Error Tradeoff (DET) curves on a log-log scale. DET curves plot miss rate versus false positives per window (FPPW). Miss rate is the proportion of pedestrian instances that were wrongly classified as non-pedestrian. On the contrary, FPPW is the proportion of non-pedestrian instances that were wrongly classified as pedestrian, denoted as false alarm.

We verified each individual step of our FPGA implementation by comparison to an equivalent C/C++ reference implementation running on a CPU. Fig. 9 shows that the classification performance of both our implementations is roughly consistent. In addition, DET curves from Dalal&Triggs' performance study are presented.

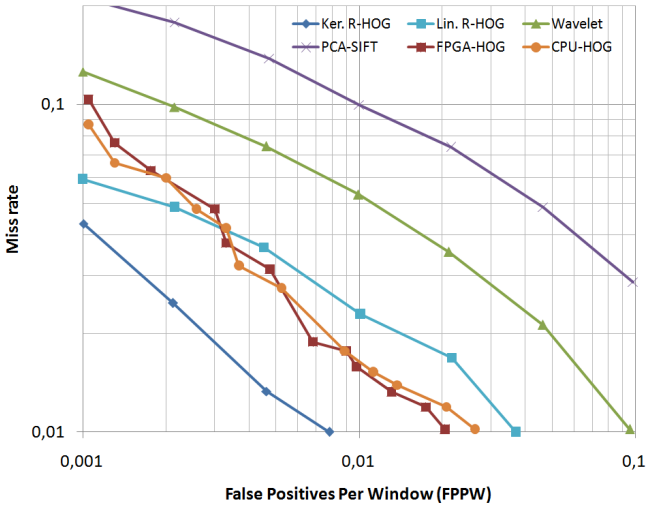


Fig. 9: Classification performance of our FPGA- and CPU-HOG implementation. The curves for Kernel SVM R-HOG, Linear SVM R-HOG, Wavelet and PCA-SIFT are extracted from [Dalal 2005].

Compared to Dalal&Triggs' original kernel R-HOG performance curve, we approach a miss rate of +6% at 10^{-3} FPPW. We deduce this difference as follows:

Essentially, in contrast to Dalal&Triggs, we have not yet applied retraining techniques such as bootstrapping that can improve the classifier performance by one order of magnitude [Munder 2006]. Furthermore, our testing set of 4,530 negative samples cannot produce positive FPPWs less than $2.2 \cdot 10^{-4}$.

We expect minor declines in classification performance due to the following implementation differences to Dalal&Triggs:

- We currently work on grayscale images, leading to a 1.5% lower detection rate at 10^{-4} FPPW compared to color images.
- Gradient magnitude is calculated with integer accuracy, i.e. all decimal places are lost when the square root is extracted.
- Our implementation does not interpolate votes between neighbouring histogram bin centres in both orientation and position to reduce aliasing effects with the histogram generation step.
- We do not down-weight pixels near the edges of the block by applying a Gaussian spatial window before accumulating orientation votes into cells. The resulting loss is about 1% at 10^{-4} FPPW.

5.2. Computation Time and Resources

With respect to computation time, we evaluate our work by measuring latency and throughput time. For latency measurements, VA operators can be used to count the number of clock cycles that elapse between the time a pixel enters a design section and the time the result is available at the section's output.

Table 1 shows latencies of the HOG descriptor computation steps, obtained by that measurement, for a UXGA camera input downscaled to a processing resolution of 800x600 pixels (SVGA) and a design frequency of 63 MHz. The histogram generation step accounts for about 2/3 of the entire latency due to the costs of convolution with the 8x8 filter kernel K_S.

Table 1: Latencies of HOG steps for a UXGA camera input downscaled to a processing resolution of 800x600 pixels and a design frequency of 63 MHz.

HOG step	Latency [µs]
Image Buffer	26.2
Scaling	26.0
Gradient	52.0
Magnitude/Orientation	0.14
Histogram	207.2
TOTAL	**311.54**

The throughput-time for one entire frame that is processed on-the-fly while grabbing the pixels is 30.8 ms for a SVGA processing resolution, the full 30 fps delivered by the camera are available via DMA.

As an indication for computation speed, one may relate the total sum of 312µs to one of the fastest implementations for HOG descriptor generation, recently published [Wojek 2008a] and running on a GPU. For the HOG steps we perform on hardware (one-time down-scaling, gradient computation, gradient magnitude, orientation binning, histogram generation) Wojek et al. report about 13 ms for an image of 320x240 pixels, computing the HOG descriptor for multiple scales.

For the SVGA image dimensions used for the latency measurements presented before, the resource usage level for the Xilinx Spartan 3 XC3S 4000 is shown in table 2.

Table 2: Xilinx Spartan 3 XC3S 4000 resource usage level for a processing resolution of 800x600 px.

Resource Type	Total number	Usage Level
4-input LUTs	42,435	76%
Internal Block RAM (18 kbit)	60	62%
Embedded Multipliers (18x18)	18	18%

6. Summary and Conclusions

As shown in section 5.2, the proposed HOG descriptor computation fits into a low-cost Xilinx Spartan 3 XC3S 4000 device. On a commercial PCIe frame grabber with embedded FPGA, it runs already fast enough that it can be directly integrated into an automotive computer of a test vehicle for evaluation purposes. The latency of the HOG descriptor generation is 312 µs and we approach classification levels in terms of pedestrian detection close to Dalal&Triggs (miss rate +6% at 10^{-3} FPPW).

Essential improvements of classification performance are expected from applying retraining techniques which have not yet been used. While Gaussian down-weighting has not yet been implemented, the histogram bin interpolation scheme does not fit into our computation approach.

Our pedestrian recognition framework based on an FPGA-CPU-GPU network currently runs in real-time in an infrastructure-based crossroads assistance system with a limited surveillance zone. The computation time for a detection window is below 150 µs, including the proportionate time for the FPGA-based HOG descriptor generation (312 µs for the entire image), CPU-based normalization (25µs) and GPU-based Kernel SVM classification (~110µs).

The system is running with 20 fps and can evaluate more than 300 detection window hypotheses per frame. This rate meets the demands of our application, as we perform a pre-selection of hypothesis windows.

There is still room to further speed up the framework. Outsourcing further parts of our recognition framework to the FPGA requires additional resources that are provided by available extension boards with additional FPGA processor and RAM. We are working on integrating the HOG descriptor normalization and SVM prediction [Irick 2008] into the FPGA, and on applying the framework for the classification of other traffic participants.

Acknowledgements

The authors acknowledge the support of the *Bayerisches Staatsministerium für Wissenschaft, Forschung und Kunst* in the context of the *Forschungsschwerpunkt Intelligente Sensorik* at Aschaffenburg University of Applied Sciences.

References

[Caltech] Caltech Pedestrian Dataset, 2009. *http://www.vision.caltech.edu/Image_Datasets/CaltechPedestrians/*

[Cao 2008] T. P. Cao and G. Deng. Real-time vision-based stop sign detection system on FPGA. In *Proc. of the International Conference on Digital Image Computing: Techniques and Applications (DICTA)*, pages 465-471, IEEE Computer Society, 2008.

[cuSVM] *http://www.patternsonascreen.net*

[Daimler] Daimler Pedestrian Detection Benchmark, 2009.*http://www.science.uva.nl/research/isla/downloads/pedestrians/*

[Dalal 2005] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *Proc. of the IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, pages I:886-893, 2005.

[Dalal 2006] N. Dalal. Finding People in Images and Videos. PhD thesis, Institut National Polytechnique de Grenoble, 2006.

[Destatis 2009] Statistisches Bundesamt Deutschland, 2009.

[Dollár 2009] P. Dollár, C. Wojek, B. Schiele, P. Perona. Pedestrian detection: A benchmark. In *Proc. of the IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2009.

[Enzweiler 2009] M. Enzweiler and D. M. Gavrila. Monocular Pedestrian Detection: Survey and Experiments. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 2008.

[Fardi 2006] B. Fardi, I. Seifert, G. Wanielik, J. Gayko. Motion-based pedestrian recognition from a moving vehicle. In *Proc. of the IEEE International Symposium on Intelligent Vehicles*, pages 219-224, 2006.

[Felzenszwalb 2005] P.F. Felzenszwalb and D.P. Huttenlocher. Pictorial structures for object recognition. *International Journal of Computer Vision (IJCV)*, 61(1):55-79, 2005

[Gandhi 2007] T. Gandhi and M.M. Trivedi. Pedestrian protection systems: Issues, survey, and challenges. *IEEE Transactions on Intelligent Transportation Systems*, 8(3):413-430, 2007.

[Gavrila 1999] D. M. Gavrila. The visual analysis of human movement: A survey. *Computer Vision and Image Understanding (CVIU)*, 73(1):82–98, 1999.

[Gavrila 2007] D. M. Gavrila and S. Munder. Multi-cue pedestrian detection and tracking from a moving vehicle. *International Journal of Computer Vision (IJCV)*, 73(1):41-59, 2007.

[INRIA] INRIA Person Dataset, 2005. *http://lear.inrialpes.fr/data/human*

[Irick 2008] K.M. Irick, M. DeBole, V. Narayanan, A. Gayasen. A Hardware Efficient Support Vector Machine Architecture for FPGA. In *Proc. of the IEEE International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 304-305, 2008.

[Ioffe 2001] S. Ioffe and D.A. Forsyth. Probabilistic methods for finding people. *International Journal of Computer Vision (IJCV)*, 43(1):45-68, 2001.

[Mikolajczyk 2004] K. Mikolajczyk, C. Schmid, A. Zisserman. Human detection based on a probabilistic assembly of robust part detection. In *Proc. of the IEEE European Conference on Computer Vision (ECCV)*, pages I:69-82, 2004.

[Moeslund 2006] T.B. Moeslund, A. Hilton, V. Kruger. A survey of advances in vision-based human motion capture and analysis. *Computer Vision and Image Understanding (CVIU)*, 103(2-3):90-126, 2006.

[Munder 2006] S. Munder and D.M. Gavrila. An experimental study on pedestrian classification. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 28(11):1863-1868, 2006.

[Papageorgiou 2000] C. Papageorgiou and T. Poggio. A trainable system for object detection. *International Journal of Computer Vision (IJCV)*, 38(1):15-33, 2000.

[Poppe 2007] R. Poppe. Vision-based human motion analysis: An overview. *Computer Vision and Image Understanding (CVIU)*, 108(1-2):4-18, 2007.

[Sentech] Sensor Technologies America, Inc. *http://www.sentechamerica.com*

[SiliconSoftware] Silicon Software GmbH, 2009. *http://www.silicon-software.com*

[Viola 2005] P. Viola, M. J. Jones, D. Snow. Detecting pedestrians using patterns of motion and appearance. *International Journal of Computer Vision (IJCV)*, 63(2):153-161, 2005.

[WHO 2004] World Health Organization (WHO) and World Bank. World report on road traffic injury. Geneva, 2004.

[Wojek 2008a] C.Wojek, G. Dorko, A. Schulz, B. Schiele. Sliding-windows for rapid object class localization: A parallel technique.
In *Deutsche Arbeitsgemeinschaft für Mustererkennung (DAGM)*, 2008.

[Wojek 2008b] C. Wojek and B. Schiele. A Performance evaluation of single and multi-feature people detection. In *Deutsche Arbeitsgemeinschaft für Mustererkennung (DAGM)*, 2008.

[Xilinx] Xilinx, Inc. *http://www.xilinx.com/tools/designtools.htm*

[Zhu 2006] Q. A. Zhu, M. C. Yeh, K. T. Cheng, and S. Avidan. Fast human detection using a cascade of histograms of oriented gradients. In *Proc. of the IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, pages II:1491-1498, 2006.

Cache-Speicher für den Softprozessor SIRIUS mit DDR-Interface

Florian Zowislok, M.Eng.
Hochschule Offenburg, Badstraße 24, 77652 Offenburg
florian.zowislok@fh-offenburg.de

Zusammenfassung

Der Cache-Speicher für den Softprozessor SIRIUS ist ein 4-fach assoziativer Cache-Speicher, der mit einem DDR-Interface auf einen externen Speicher zugreifen kann. Er verwaltet und beschleunigt Zugriffe vom Prozessor auf diesen Speicher. Der Cache-Speicher arbeitet intern mit 32 Bit und der doppelten Prozessortaktfrequenz und ermöglicht Systeme mit größeren Speicheranforderungen ohne signifikante Performanceverluste.

Der Cache-Speicher wurde mit der Hardware-beschreibungssprache VHDL erstellt und mit dem bestehenden Mikrocontrollersystem verbunden. Das Gesamtsystem wurde zunächst simuliert und anschließend mit dem Cyclone III FPGA Starter Kit von Altera, welches ein 32 MB DDR-RAM-Modul zur Verfügung stellt, durch Ausführen eines Test-programms erfolgreich verifiziert. Für den kompletten Cache-Speicher werden inklusive der Pins für den externen Oszillator und des Reset-Tasters 3805 Logik-Zellen, 27 M9K-Blöcke, 44 Pins und eine PLL benötigt.

1. Das Mikrocontrollersystem

In mehreren vorangegangenen Projekten und Abschlussarbeiten wurde am Institut für Angewandte Forschung ein Mikrocontrollersystem entwickelt. Kernstück dieses Systems ist der 32-Bit SIRIUS-Softcore [1] [2], der zusammen mit verschiedenen Peripherieeinheiten auf einem Cyclone III FPGA emuliert wird. Dieses FPGA stellt zudem 64 KB RAM zur Verfügung. Der SIRIUS Prozessor kann mit der dazugehörigen SIRIUS IDE sowohl in Assembler als auch in C programmiert werden. Die Bedienung des Systems erfolgt mit einem Terminalprogramm, welches auf einem PC ausgeführt wird, der über USB mit dem Mikrocontrollersystem verbunden ist. Mittlerweile verfügt das System auch über eine SD-Karte und einen Touchscreen. Mit steigender Komplexität der Anwendungsprogramme stellt der verfügbare RAM des FPGAs den Flaschenhals dar. Aus diesem Grund soll ein größerer externer RAM verwendet werden. Um die langen Zugriffszeiten des externen RAMs zu kompensieren, sollen die Speicherblöcke des FPGAs als schneller Zwischen-speicher verwendet werden – ein sogenannter Cache. An diesen Cache soll mit einem DRAM-Controller der externe Speicher angebunden werden.

2. Anforderungen

2.1. Zielsystem

Das komplette Mikrocontrollersystem wird in einem Cyclone IIIFPGA der Firma Altera emuliert. Der Cache soll zusammen mit dem DRAM-Controller in das bestehende System integriert werden. Dabei wird der interne RAM durch den Cache ersetzt. Für den Cache können somit die Speicherblöcke des FPGAs verwendet werden, die zuvor für den RAM benötigt wurden.

2.2. Schnittstelle zum Mikrocontrollersystem

In *Abbildung 2.1* ist die ursprüngliche Version des Gesamtsystems dargestellt. Im mittleren Block ist das gesamte Mikrocontrollersystem mit allen Peripherien ohne den RAM zusammengefasst. Der rechte Block zeigt den RAM, der mit mehreren Leitungen an das Mikrocontrollersystem angebunden. Ist. Die Taktfre-quenz von einem externen Oszillator wird mit der links oben dargestellten PLL auf die gewünschte System-frequenz gesetzt. Die Reset-Logik links unten erzeugt das Reset-Signal beim Einschaltvorgang des Systems bzw. wenn ein externer Resetknopf betätigt wird.

Damit das bestehende System, insbesondere die CPU, so wenig wie möglich verändert werden muss, soll die Ansteuerung des Caches identisch mit der des RAMs sein. Der RAM ist 64 KB groß und verfügt über eine Wortbreite von 16 Bit, womit 15 Bit für die Adressierung der einzelnen Worte benötigt werden. Über eine separate Byte-Enable-Leitung können beim Schreibenvorgang die beiden Bytes des 16-Bit-Wortes ausmaskiert werden. Der Zugriff auf den RAM erfolgt dabei innerhalb von einem Taktzyklus.

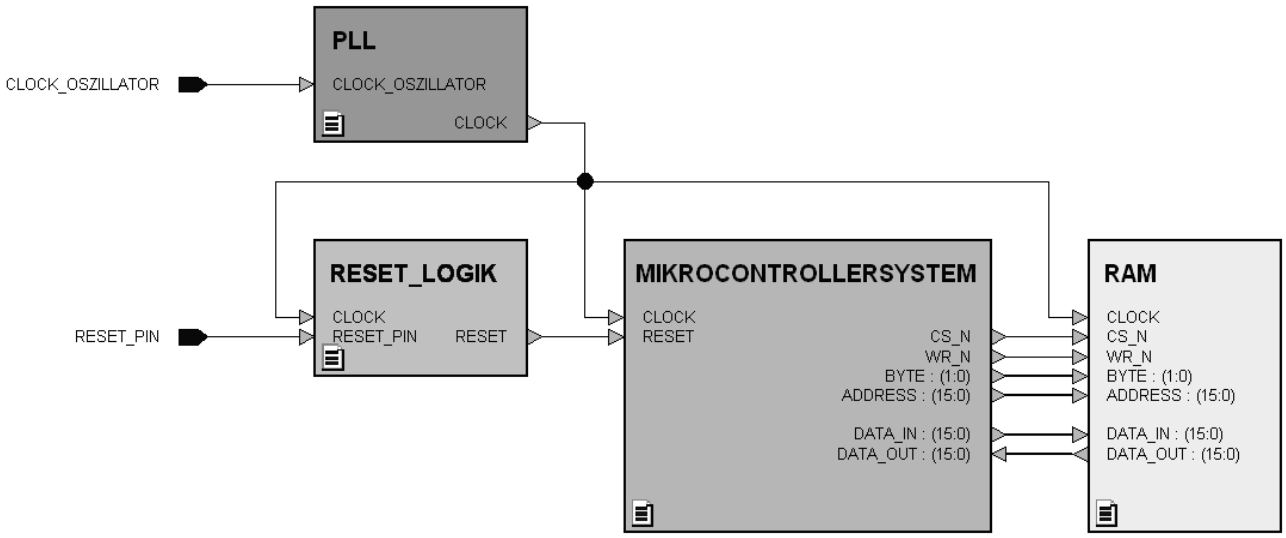


Abbildung 2.1: -Mikrocontrollersystem mit FPGA RAM

Für den Cache-Speicher soll der komplette 32 Bit breite Adressbus verwendet werden, mit welchem das Mikrocontrollersystem intern arbeitet. Zusätzlich wird die Schnittstelle um ein READY-Signal erweitert. Mit diesem Signal ist es möglich, das Mikrocontroller-system anzuhalten, falls eine Speicheranforderung nicht in einem Taktzyklus bedient werden kann.

2.3. Externes Speichermodul

Als externes Speichermodul soll ein geeigneter SD-RAM zum Einsatz kommen, der deutlich mehr Speicher als der interne FPGA-RAM zur Verfügung stellt. Die Größe soll dabei 16 MB oder mehr betragen, um auch zukünftigen Anforderungen gerecht zu werden.

3. Funktionsweise von Cache-Speichern

3.1. Basisinformationen

Um den wiederholten Zugriff auf große Speicher mit langen Zugriffszeiten zu beschleunigen, werden häufig Cache-Speicher eingesetzt. Diese kleinen schnellen Zwischenspeicher enthalten Kopien kleiner Daten-bereiche des großen Speichers. Wird auf Daten zuge-griffen, die sich nicht im Cache-Speicher befinden, handelt es sich um einen sogenannten Fehlschlag, der als „Miss“ bezeichnet wird. Die Daten müssen zunächst mit einer langen Zugriffszeit aus dem großen Speicher in den Cache geladen werden, bevor darauf zugegriffen werden kann. Bei einem anschließenden Zugriff auf dieselben Daten sind diese bereits im Cache verfügbar. In diesem Fall handelt es sich um einen Treffer, welcher als „Hit“ bezeichnet wird. Die Daten können mit der verhältnismäßig kurzen Zugriffs-zeit des Caches gelesen oder geschrieben werden. Bei jedem Zugriff kann es somit entweder zu einem Hit oder zu einem Miss kommen. Der prozentuale Anteil wird als Hit-Rate bzw. Miss-Rate bezeichnet. Idealerweise sollte die Hit-Rate möglichst hoch sein. [3]

In der Speicherhierarchie eines Rechnersystems befindet sich der Cache zwischen den CPU-Registern und dem Hauptspeicher. Dies ist in *Abbildung 3.1* dargestellt. Obwohl der Zugriff auf den Hauptspeicher nur über den Cache erfolgt, bekommt der Programmierer davon nichts mit, denn im Gegensatz zum Hauptspeicher kann der Cache nicht direkt adressiert werden. Es handelt sich also um einen versteckten Speicher, der im Verborgenen die Systemperformance erhöht. [4]

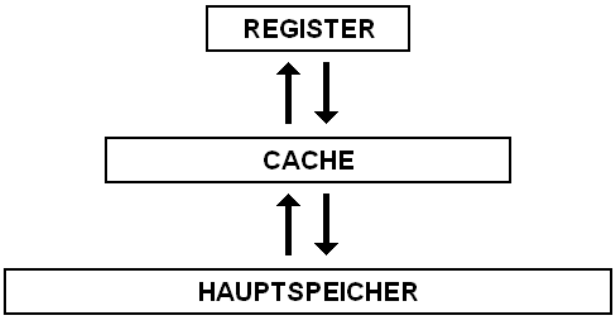


Abbildung 3.1: Speicherhierarchie eines Rechnersystems

3.2. Cache-Organisation

Für Cache-Speicher gibt es drei verschiedene Organi-sationen. Dabei wird der Cache-Speicher in einen oder mehrere Blöcke unterteilt. Jeder Block enthält die

gleiche Anzahl von Zeilen, wobei jede dieser Zeilen eine bestimmte Anzahl zusammenhängender Datenbytes enthält. Diese liegt üblicherweise im Bereich von 4 bis 128 Bytes. [5]

Besteht der Cache nur aus einem Block, gibt es zwei verschiedene Organisationstypen: „vollassoziativ“ und „direkt verdrahtet“. Bei einem vollassoziativen Cache kann jede Datenzeile aus dem großen Speicher an jeder beliebigen Stelle im Cache abgelegt werden. Bei einem direkt verdrahteten Cache kann jede Zeile, abhängig von der Adresse, nur an einer einzigen Position gespeichert werden. Dadurch vereinfacht sich zwar der Implementierungsaufwand, sobald jedoch nacheinander abwechselnd auf Speicherbereiche zugegriffen wird, die auf dieselbe Position im Cache verdrahtet sind, kommt es jedes Mal zu einem Miss. Um diesem Problem entgegenzuwirken, können mehrere direkt verdrahtete Cache-Blöcke parallel verwendet werden. [5]

Bei n Blöcken spricht man von einem n-fach assoziativen Cache. Jede Adresse ist auch hier mit einer bestimmten Zeile innerhalb eines Blockes verdrahtet, jedoch kann der Block selbst frei gewählt werden. Somit ergeben sich für n Blöcke auch n mögliche Speicherpositionen im Cache. Diese n Speicherpositionen, die mit denselben Adressen verdrahtet sind, werden als Satz bezeichnet. Die Anzahl der Sätze in einem Cache ist gleich der Anzahl der Cache-Zeilen in einem Block. [5] *Abbildung 3.2* zeigt einen n-fach assoziativen Cache mit m Sätzen.

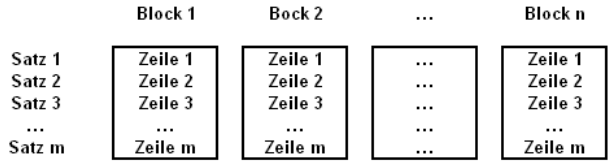


Abbildung 3.2: n-fach assoziativer Cache mit m Sätzen

Für die Entwicklung des Caches war vorgegeben, dass der Cache 4-fach assoziativ ausgelegt sein soll. Damit ist sichergestellt, dass mindestens vier unterschiedliche Datenbereiche gleichzeitig zwischengespeichert werden können. Außerdem stellt dies einen guten Kompromiss aus Effizienz und Implementierungsaufwand dar.

3.3. Adressierung und Steuerung

Bei Cache-Speichern wird die Adresse in drei Adressen unterteilt, die Tag-Adresse, die Selektor-Adresse und die Wort-Adresse, z.B. stellen bei der Adresse „11001001“ die drei MSBs „110“ die Tag-Adresse, die zwei mittleren Bit „01“ die Selektor-Adresse und die drei LSBs „001“ die Wort-Adresse dar. Die Wort-Adresse adressiert ein Wort innerhalb einer Cache-Zeile, die mit der Selektor-Adresse ausgewählt wird. Die zusammenhängenden Daten innerhalb einer Cache-Zeile haben somit immer eine gemeinsame Tag- und Selektor-Adresse, die dazu gehörige Tag-Adresse wird zu jeder Cache-Zeile separat gespeichert. [5]

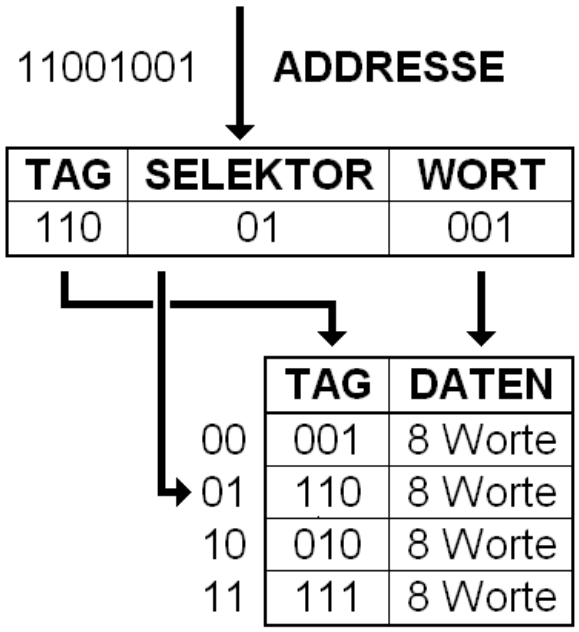


Abbildung 3.3: Adressierung im Cache

In *Abbildung 3.3* ist die Adressierung eines Cache-Blockes dargestellt. Wenn die Tag-Adresse einer über den Selektor ausgewählten Cache-Zeile mit der gespeicherten Tag-Adresse in einem der Blöcke übereinstimmt, kommt es zu einem Hit. Die adressierten Daten sind im Cache verfügbar und können direkt gelesen oder geschrieben werden. Andernfalls kommt es zu einem Miss und die entsprechende Datenzeile muss aus dem großen Speicher in einen Cache-Block nachgeladen werden. Dabei wird auch die entsprechende gespeicherte Tag-Adresse aktualisiert. Welcher Cache-Block dabei gewählt wird, ist im nachfolgenden Kapitel beschrieben.

3.4. Verdrängungsstrategien

Beim Nachladen einer Daten-Zeile in einen 4-fach assoziativen Cache kann diese theoretisch in jedem der vier Cache-Blöcke abgelegt werden, wobei jedoch die an der betreffenden Position befindliche Zeile aus dem Cache verdrängt wird. Idealerweise sollte dabei der Cache-Block gewählt werden, dessen über die Selektor-Adresse ausgewählte Zeile am längsten nicht benutzt wird. Diese ist jedoch meist im Voraus nicht bekannt. Aus diesem Grund gibt es verschiedene Verdrängungsstrategien. Es kann beispielsweise eine zufällige, die älteste, die am seltensten genutzte oder

die am längsten nicht genutzte Zeile verdrängt werden. [3]

Der beste Kompromiss zwischen Implementierungsaufwand und Effizienz ist es, die Zeile zu verdrängen, die am längsten nicht genutzt wurde. Dazu wird zu jeder Zeile ein 2-Bit-Zähler als Attribut gespeichert. Dabei darf jede Bitkombination des Zählers in allen vier Cache-Blöcken in den jeweiligen Zeilen mit der gleichen Selektor-Adresse nur einmal verwendet werden. Für die zuletzt verwendete Zeile beträgt die Bitkombination „00“, für die vorletzte „01“, für die am zweitlängsten nichtverwendete „10“ und für die am längsten nicht verwendete Zeile „11“.

3.5. Sicherstellen der Konsistenz

Da der Cache Kopien von Daten enthält, die durch Schreib-Zugriffe verändert werden können, muss die Konsistenz zwischen dem Cache und dem großen Speicher sichergestellt sein. Dazu gibt es mehrere Strategien. Beim „write-through“ werden die Daten direkt in den Hauptspeicher geschrieben und gegebenenfalls auch in den Cache, wenn die Daten im Cache vorhanden sind. Dabei entstehen jedoch auch langsame Schreib-Zugriffe auf den Hauptspeicher, wenn die Daten im Cache verfügbar sind. Dies kann mit der „write-back“-Strategie vermieden werden. Falls die Daten nicht im Cache verfügbar sind, werden sie wie beim „write-through“ direkt im Hauptspeicher verändert. Sind die Daten jedoch im Cache verfügbar, werden sie nur im Cache verändert und die entsprechende Zeile mit dem Attribut „Dirty-Bit“ gleich „1“ gekennzeichnet. Bei dem Verdrängen durch eine andere Datenzeile muss die verdrängte Cache-Zeile dann zuerst in den Speicher zurückgeschrieben werden, falls das Dirty-Bit gesetzt ist. Ist es nicht gesetzt, so kann die verdrängte Zeile verworfen werden. Die „write-allocation“-Strategie funktioniert im Prinzip genauso, jedoch wird im Gegensatz zur „write-back“-Strategie die Datenzeile, in welche geschrieben werden soll, immer zuerst komplett in den Cache geladen und anschließend nur im Cache verändert. [4]

Um die Anzahl der langsamen Zugriffe auf den Hauptspeicher zu minimieren, wird hier die „write-allocation“-Strategie gewählt. Dies hat zur Folge, dass unabhängig davon, ob es sich um einen Lese- oder Schreib-Zugriff handelt, bei einem Miss immer eine Datenzeile vom Hauptspeicher in den Cache geladen werden muss und die verdrängte Zeile dabei gegebenenfalls zurückgeschrieben werden muss.

3.6. Zugriffszeiten

Die Zugriffszeit auf Daten, die sich im Cache befinden, beträgt normalerweise nur einen Taktzyklus. Befinden sich die Daten nicht im Cache, müssen diese aus dem Speicher nachgeladen werden. Da dabei immer eine ganze Zeile geladen wird und gegebenenfalls zurückgeschrieben wird, setzt sich diese Zeit aus der Zugriffszeit und der Transferzeit für die Daten zusammen. Die Zugriffszeit hängt dabei von dem verwendeten Speicher und die Transferzeit noch zusätzlich von der Länge einer Cache-Zeile ab. Werden für die Zugriffszeit beispielsweise 8 und für die Transferzeit 16 Taktzyklen benötigt, dauert der gesamte Nachladevorgang 24 Taktzyklen. [5]

3.7. Beispiel

In diesem Beispiel ist die Funktionsweise des Caches in den *Abbildungen 3.4 bis 3.7* dargestellt. Dazu wird ein Hauptspeicher angenommen, der 32 Datenworte enthält und somit 5 Adressbit benötigt. Der Cache ist 2-fach assoziativ mit 2 Zeilen je Block und 4 Worten pro Zeile ausgelegt. Somit bilden die 2 MSBs des Hauptspeichers die Tag-Adresse, das mittlere Bit die Selektor-Adresse und die 2 LSBs die Wortadresse. Das „C“ in den Abbildungen steht für einen 1-Bit-Zähler, der anzeigt, welche Zeile innerhalb eines Satzes am längsten nicht verwendet wurde. Das „D“ steht für das Dirty-Bit. Zu Beginn sind im Cache schon Daten geladen, wie in *Abbildung 3.4* dargestellt. Die Datenzeile mit der Tag-Adresse „00“ und der Selektor-Adresse „0“ befindet sich beispielsweise in der oberen Zeile des oberen Caches. Wird nun beispielsweise das Wort mit der Adresse „00110“ gelesen, sind diese Daten im Cache verfügbar. Die Selektor-Adresse „1“ (mittleres Bit) wählt die untere Zeile in den Blöcken aus. Im oberen Block stimmt die gespeicherte Tag-Adresse mit der aktuellen Tag-Adresse überein und es kommt zu einem Hit. Mit der Wort-Adresse „10“ wird nun das zweite Wort von links innerhalb der betreffenden Cache-Zeile ausgewählt und kann direkt gelesen werden. Da der Zähler „C“ (COUNTER) dieser Cache-Zeile bereits den Wert „0“ hat, müssen die Zähler innerhalb des Satzes nicht verändert werden.

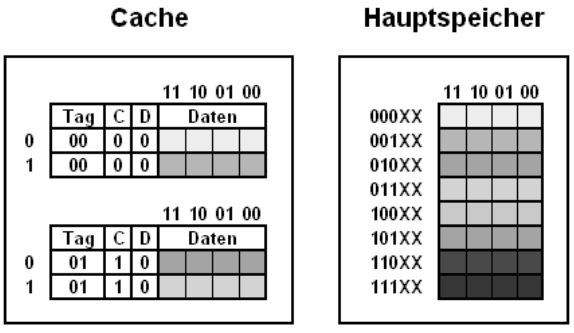


Abbildung 3.4: Aufbau des Beispiel-Caches

Als nächstes wird in Adresse „01001“ geschrieben. Die Daten sind in der oberen Zeile des unteren Cache-Blocks verfügbar und können direkt geschrieben wer-

den. Dabei muss der Zähler innerhalb des Satzes verändert und das Dirty-Bit der betreffenden Cache-Zeile gesetzt werden, wie in *Abbildung 3.5* dargestellt. Die veränderten Daten werden durch ein „X“ markiert und sind nun nicht mehr mit dem Hauptspeicher konsistent.

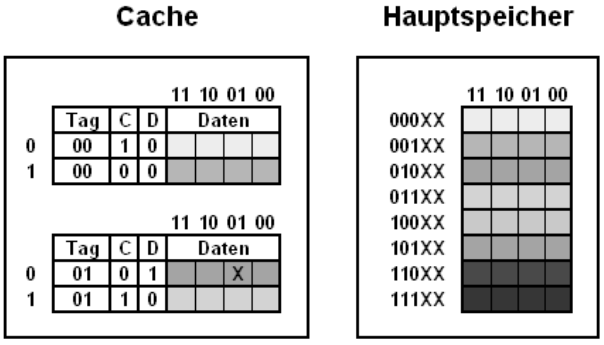


Abbildung 3.5: Beispiel-Cache nach dem Schreib-Zugriff

Der nächste Lese-Zugriff erfolgt auf Adresse „10010“. Über die Selektor-Adresse werden die jeweils oberen Cache-Zeilen ausgewählt, wobei es in keinem der Blöcke zu einer Übereinstimmung der Tag-Adressen kommt. Die angeforderten Daten müssen also zunächst in den Cache geladen werden. Der Zähler „C“ zeigt an, dass die Cache-Zeile des oberen Blocks innerhalb des Satzes am längsten nicht verwendet wurde. Da keine Daten verändert wurden (Dirty-Bit = 0), kann die Cache-Zeile verworfen werden und die benötigte Zeile aus dem Speicher geladen werden. Dies ist in *Abbildung 3.6* dargestellt. Beim Nachladevorgang wird die neue Tag-Adresse in den Tag-Speicher geschrieben und das Dirty-Bit gegebenenfalls auf „0“ gesetzt. Die Daten sind nun im Cache verfügbar und können gelesen werden. Dabei wird der Zähler der Cache-Zeile auf „0“ gesetzt und der anderen Zähler des gleichen Satzes auf „1“ gesetzt.

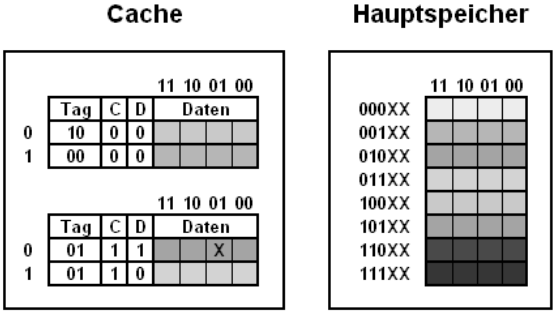


Abbildung 3.6: Nachladen in den Beispiel-Cache

Der nächste Schreib-Zugriff wird mit 11000 adressiert. Wieder sind die oberen Cache-Zeilen selektiert und es kommt zu keiner Übereinstimmung der Tag-Adresse, weshalb die angeforderten Daten aufgrund des Zählerstandes in die obere Zeile des unteren Cache-Blocks geladen werden müssen. Hier ist jedoch das Dirty-Bit gesetzt, womit die Cache-Zeile zunächst in Hauptspeicher geschrieben werden muss. *Abbildung 3.7* zeigt den Cache und den Hauptspeicher, nachdem die alte Cache-Zeile in Hauptspeicher geschrieben, die neue Datenzeile geladen und das adressierte Wort geschrieben wurde.

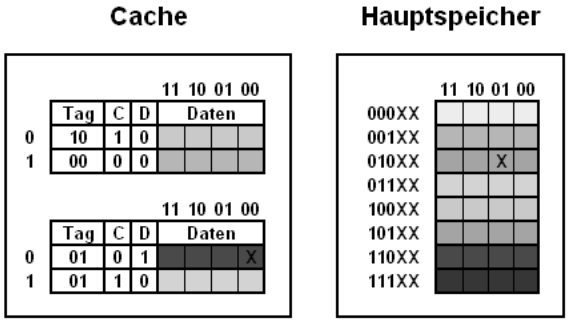


Abbildung 3.7: Beispiel-Cache nach dem Rückschreiben in den Hauptspeicher

4. Aufbau des kompletten Caches

4.1. Daten- und Adressbus

Wie in *Kapitel 2.2* beschrieben, soll der Cache mit der kompletten Busbreite von 32 Adressbit angesteuert werden, damit die Speichergröße des externen Speichers bis zu einer Größe von maximal 4 GB erweitert werden kann. Obwohl für die Schnittstelle zum Mikrocontrollersystem ein 16-Bit-Datenbus vorgesehen ist, soll der Cache-Speicher intern mit einem 32-Bit-Datenbus arbeiten, damit das Mikrocontrollersystem später ohne großen Aufwand ebenfalls auf einen 32 Bit breiten Datenbus umgestellt werden kann.

4.2. Dimensionierung der Cache-Blöcke

Die Größe eines Cache-Blockes wird durch zwei Parameter bestimmt, die Länge einer einzelnen Cache-Zeile, sowie die Anzahl von Zeilen innerhalb eines Blockes. Das Produkt aus diesen beiden Parametern ergibt die Größe eines Blockes.

Da der Cache-Speicher zukünftig auch in einem ASIC zum Einsatz kommen soll, bei dem die Speicherzellen den größten Teil der Chipfläche benötigen, sollte die Größe des Caches möglichst gering gehalten werden. Andererseits können mit einem großen Cache auch mehr Daten zwischengespeichert werden, was die Hit-Rate und somit die Systemperformance erhöht. Eine Cachegröße von insgesamt 16 KB zuzüglich der benötigten Speicher für die Tag-Adresse und Attribute zu jeder Cache-Zeile sind das Maximum für den

geplanten ASIC. Somit stehen für jeden der vier Cache-Blöcke 4 KB Datenspeicher zur Verfügung.

Auch bei der Länge der Cache-Zeilen muss ein Kompromiss gefunden werden. Die Daten innerhalb der Cache-Zeile stellen das Abbild aus einem zusammenhängenden Speicherbereich des externen DRAMs dar. Beim Nachladen einer Cache-Zeile aus dem DRAM muss diese immer komplett als Burst gelesen oder geschrieben werden. Beim Zugriff auf den DRAM wird dabei vor jedem Burst eine gewisse Latenzzeit benötigt. Um möglichst wenig dieser Latenzzeiten zu verursachen, sollte die Zeilenlänge möglichst groß gewählt werden. Beim Zugriff auf einzelne Bytes, die nicht innerhalb einer Zeile liegen, kommt es dabei jedoch dazu, dass lange Zeilen aus dem DRAM geladen werden müssen, obwohl immer nur ein einzelnes Byte benötigt wird. In diesem Fall würde eine kurze Zeilenlänge die Systemperformance erhöhen. Die ideale Zeilenlänge kann letztendlich nur über Benchmarks ermittelt werden und hängt stark von der Programmierung der Anwendung ab. Für dieses Cache-System wird eine Zeilenlänge von 64 Byte verwendet. Ein 4-KB-Cache-Block setzt sich somit aus 64 Zeilen zusammen.

4.3. Adressierung eines Cache-Blockes

Vom 32-Bit-Adressbus werden theoretisch die untersten 6 Bit zum Adressieren der 64 Bytes innerhalb einer Zeile verwendet. Da der Cache intern jedoch mit 32-Bit-Worten arbeitet, werden je 4 Bytes zu einem Wort zusammengefasst. Somit bleiben die untersten 2 Bit ungenutzt. Mit den übrigen 4 Bit können die daraus resultierenden 16 Worte innerhalb einer Cache-Zeile adressiert werden. Die darüber liegenden 6 Bit des Adressbusses werden als Selektor-Adresse zum Auswählen der Cache-Zeile verwendet. Die höchsten 20 Bit vom Adressbus ergeben somit die Tag-Adresse, die neben den Attributen zu jeder Cache-Zeile gespeichert wird. In *Abbildung 4.1* ist die Adressierung eines Cache-Blockes dargestellt.

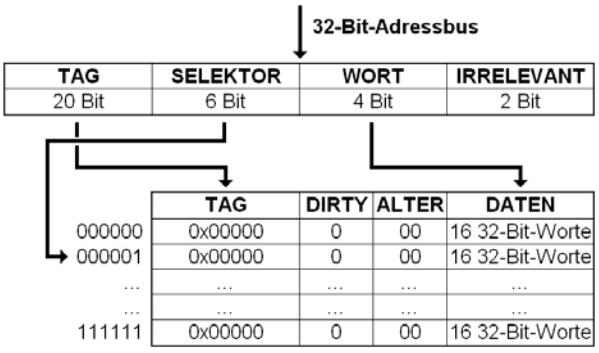


Abbildung 4.1: Adressierung eines Cache-Blockes

4.4. Aufbau des gesamten Cache-Systems

Abbildung 4.2 zeigt das gesamte Cache-System. Es besteht aus den vier Cache-Blöcken und einer Steuereinheit, die die Zugriffe auf die einzelnen Cache-Blöcke verwaltet. Mit dem im Cache enthaltenen DRAM-Controller kann die Steuereinheit auf den externen DRAM zugreifen. Die Kommunikation mit dem Mikrocontrollersystem erfolgt ebenfalls über die Steuereinheit.

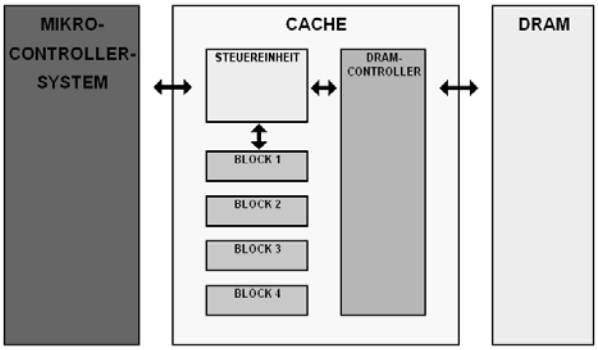


Abbildung 4.2: Aufbau des Cache-Systems

5. Auswahl der Hardware

Bevor mit der Implementierung des Systems begonnen werden kann, muss festgelegt werden, welche Hardware verwendet werden soll, da der DRAM-Controller an das externe Speichermodul angepasst werden muss. Um den Aufwand möglichst gering zu halten, wird zunächst nach einer fertigen Lösung gesucht. Dazu bietet Altera ein preiswertes Entwicklungsboard an, das Cyclone III FPGA Starter Kit, welches in *Abbildung 5.1* dargestellt ist.

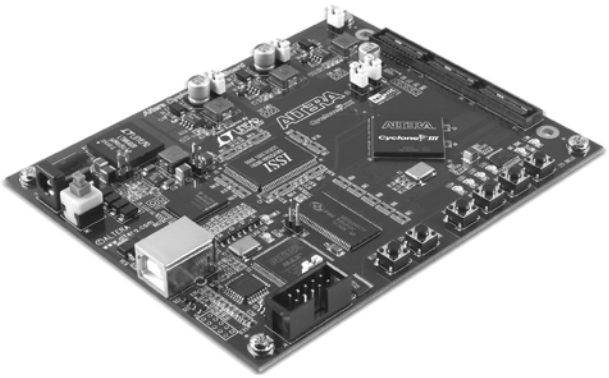


Abbildung 5.1: Cyclone III FPGA Starter Kit

Das Entwicklungsboard stellt zahlreiche Hardwarekomponenten zur Verfügung. Die relevanten Elemente sind im Folgenden aufgelistet:

- Cyclone III EP3C25 FPGA

- 32 MB DDR-RAM
- USB-Blaster zum Programmieren des FPGAs
- 50 MHz Oszillator
- Jeweils 4 Taster und LEDs
- HSMC-Anschlussleiste für Zusatzplatinen

5.1. DDR-RAM

Bei DDR-RAMs handelt es sich um dynamische Speicher mit großen Kapazitäten, sogenannte DRAMs. Um diese Kapazitäten auf kleinen Raum zu erhalten, werden die Datenbit mit Kondensatoren gespeichert. Da sich die Kondensatoren jedoch über unerwünschte Leckströme entladen, geht der Speicherinhalt nach kurzer Zeit verloren. Aus diesem Grund müssen die Daten im Speicher zyklisch mit einem sogenannten „Refresh“ aktualisiert werden. Im DRAM sind die Speicherzellen zeilenweise angeordnet.

Um Adressleitungen zu sparen, wird beim Zugriff über die gleichen Leitungen zuerst die Zeilenadresse und dann die Wortadresse mit denselben Leitungen übertragen. Durch den internen Aufbau dauert der Zugriff auf DRAMs deutlich länger als auf statische RAMs, jedoch können die zusammenhängenden Daten innerhalb einer Zeile als “Burst“ ausgelesen werden. Dabei werden nach der relativ langen Zugriffszeit für die erste Speicherzelle die folgenden mit jedem Taktzyklus gelesen bzw. geschrieben.

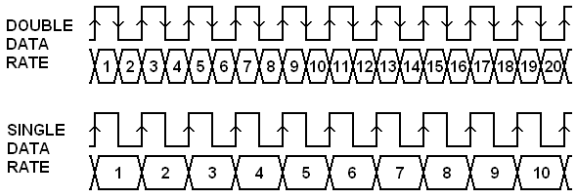


Abbildung 5.2: DDR im Vergleich zu SDR

Die Besonderheit bei DDR-RAM liegt darin, dass die Daten sowohl mit steigender als auch mit fallender Flanke übertragen werden, womit die doppelte Daten-Rate von herkömmlichen DRAMs erreicht wird, bei denen die Daten nur mit der steigenden Taktflanke übertragen werden. Dies ist in *Abbildung 5.2* dargestellt. Bei dem verwendeten DDR-RAM-Modul handelt es sich um einen Speicherchip mit einem 16 Bit breiten Datenbus. Der Speicherchip kann theoretisch im Bereich zwischen 83 und 200 MHz betrieben werden, jedoch unterstützt der verwendete DRAM-Controller im Cyclone III FPGA nur Frequenzen bis maximal 167 MHz. Somit ergeben sich bei den unterschiedlichen Betriebsfrequenzen theoretisch die folgenden maximalen Datenraten:

83 MHz * 16 Bit * 2 (DDR) = 317 MB/s
100 MHz * 16 Bit * 2 = 381 MB/s
133 MHz * 16 Bit * 2 = 507 MB/s
150 MHz * 16 Bit * 2 = 572 MB/s
167 MHz * 16 Bit *2 = 637 MB/s

Durch den zyklischen Refresh und vor allem den hohen Zugriffszeiten werden in der Praxis jedoch nur Datenraten erreicht, die bei etwa 50% dieser Angaben liegen. Die Speichergröße des verwendeten DDR-RAMs ist mit 32 MB auch für zukünftige Anforderungen ausreichend.

5.2. Weitere Hardware-Komponenten

Da das Mikrocontrollersystem in der bisherigen Version den Flash-Speicher M25P40 der Firma ST zum Booten des Systems benötigt, soll dieser an das System angeschlossen werden. Dazu wird eine Zusatzplatine der Firma Bitec verwendet, die über die HSMC-Anschlussleiste mit dem Entwicklungsboard verbunden wird. 158 Anschlüsse des FPGAs sind direkt auf dem Board verfügbar, sowie Versorgungsspannungen von 3,3 und 12 Volt. *Abbildung 5.3* zeigt Vorder- und Rückseite der Zusatzplatine.

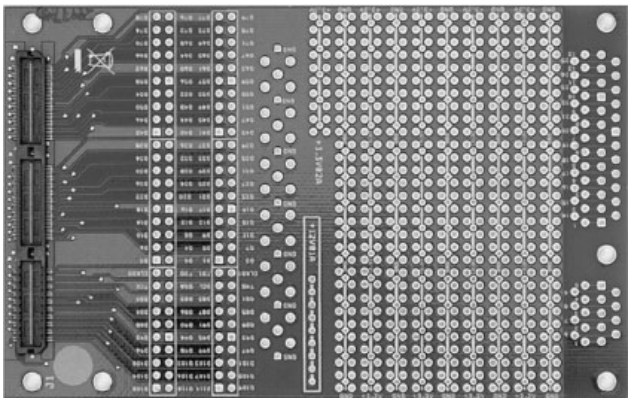


Abbildung 5.3: Zusatzplatine für das Cyclone III FPGA Starter Kit

Da die Versorgungsspannung des FPGAs 2,5 Volt beträgt und der Flash mit den 3,3 Volt der Zusatzplatine betrieben wird, wird zum Konvertieren der Signale der Pegelwandler ADG3308 von Analog Devices eingesetzt. Des Weiteren wird zur Tonausgabe ein Piezo-Lautsprecher verwendet.

6. Implementierung in VHDL

6.1. Verwendete Entwicklungssoftware

Für die Implementierung und Simulation des VHDL-Codes werden die Programme HDL Designer und ModelSim verwendet, die im FPGA Advantage Softwarepaket von Mentor enthalten sind. Die VHDL-Codes für FPGA-spezifische Funktionsblöcke werden mit dem in der Quartus II Software von Altera enthaltenen MegaWizzard Plug-In Manager generiert.

6.2. Altera MegaCore Funktionen

Um das auf dem Entwicklungsboard vorhandene DDR-Speichermodul mit dem Cyclone III FPGA ansprechen zu können, wird die Altera MegaCore Funktion „DDR SDRAM High Performance Controller“ verwendet. Altera MegaCore Funktionen sind optimierte Funktionsblöcke für häufig verwendete Anwendungen, welche mit dem MegaWizzard Plug-In konfiguriert werden können.

6.3. Festlegung der Taktfrequenzen

Bei dem bisherigen Mikrocontrollersystem erfolgte ein Speicherzugriff innerhalb von einem Taktzyklus. Diese Bedingung soll auch beim Cache erfüllt werden, sofern die angeforderten Daten im Cache verfügbar sind. Bei einem Schreib-Zugriff auf den Cache müssen jedoch zuerst die Tag-Adressen des angeforderten Speicherbereichs mit den gespeicherten Tag-Adressen in den selektierten Zeilen der Cache-Blöcke verglichen werden, bevor die Daten in den richtigen Block geschrieben werden können. Dafür sind zwei Taktzyklen erforderlich. Aus diesem Grund wird für den Cache die zweifache Taktfrequenz des Mikrocontrollersystems gewählt. Für einen Taktzyklus des Mikrocontrollersystems ergeben sich somit zwei Taktzyklen im Cache

6.4. Übersicht über das Gesamtsystem

Das gesamte Cache-System, in *Abbildung 6.1* dargestellt, ist in sieben verschiedene Einheiten unterteilt. Die Anschlussports links bilden die Schnittstelle zum Mikrocontrollersystem, die Ports auf der rechten Seite die Schnittstelle zum DRAM-Modul.

Das Kernstück bildet die Steuereinheit, die direkt und über den Multiplexer mit den vier Cache-Blöcken verbunden ist. Während des Zugriffs auf den externen Speicher mit dem DRAM-Controller übernimmt Nachladeeinheit die Steuerung. Die Anpassung zwischen den unterschiedlichen Datenbusbreiten von Mikrocontrollersystem und Cache erfolgt mit dem Datenbus-Wrapper. Die Reset-Logik wurde vom übrigen Mikrocontrollersystem in den Cache verlegt, da damit direkt der DRAM-Controller angesteuert wird..

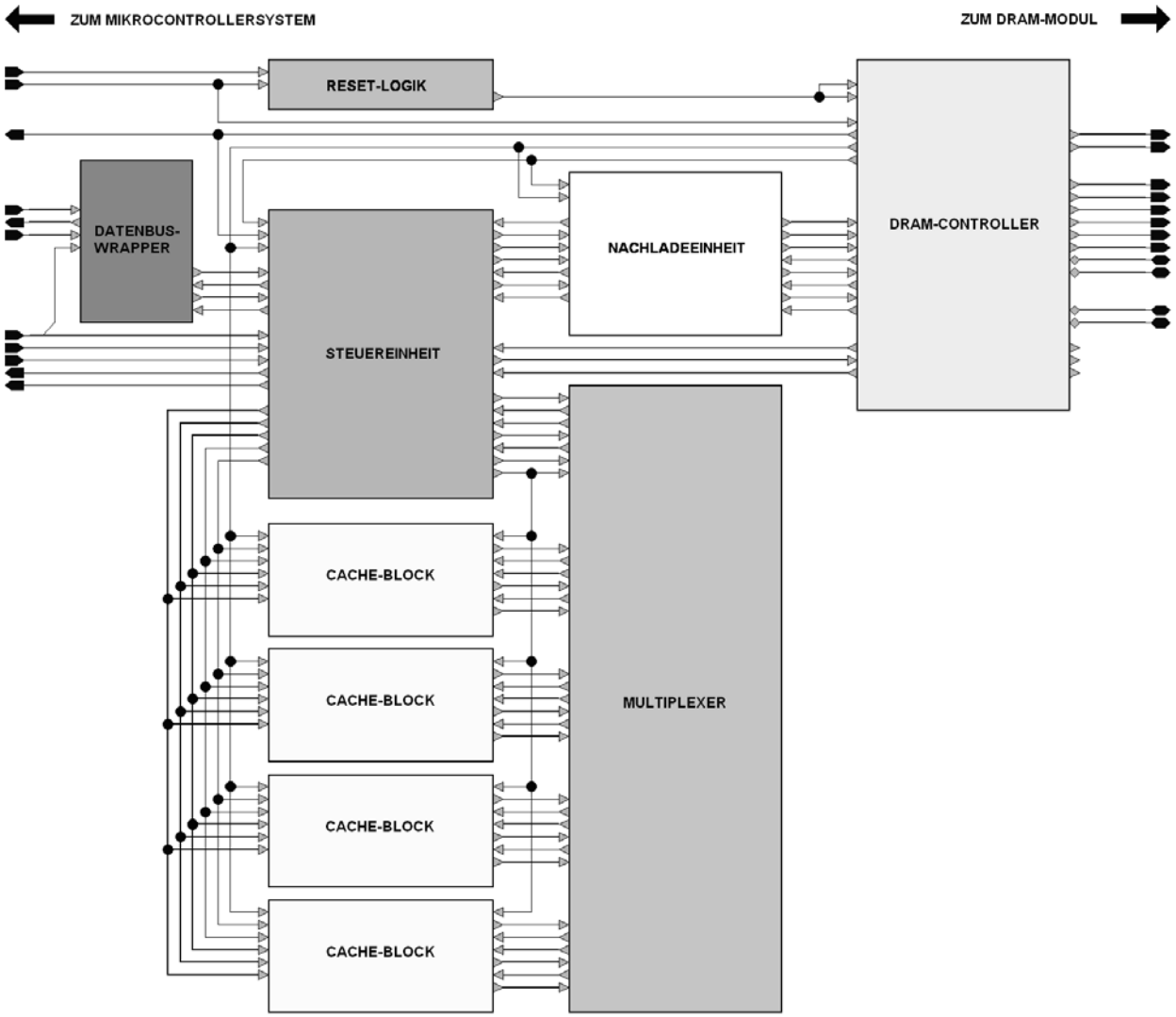


Abbildung 6.1: Übersicht über das Gesamtsystem

6.5. Steuereinheit

Die Steuereinheit ist das Kernstück des Caches. Abgesehen von der Reset-Logik ist die Steuereinheit mit allen Einheiten des Caches direkt verbunden. Die komplexe Cache-Steuerung wird durch einen einfachen Zustandsautomaten realisiert. Da die Schnittstelle zum Mikrocontrollersystem nur mit der halben Taktfrequenz des Caches betrieben wird, benötigt die Steuereinheit beide Taktsignale. Durch den DRAM-Controller ist zwar sichergestellt, dass die beiden Taktsignale synchronisiert sind, aber es ist nicht bekannt, ob zum Zeitpunkt der steigenden Taktflanke vom schnelleren Cache-Takt das langsamere Taktsignal eine steigende oder fallende Flanke aufweist. Aus diesem Grund wird zum Zeitpunkt der fallenden Flanke des Cache-Taktes der Zustand des langsamen Taktes in dem internen Signal „SYS_CLK_LEVEL“ gespeichert. Ist „SYS_CLK_LEVEL“ zum Zeitpunkt der steigenden Taktflanke des Cache-Taktes gleich „0“, so ist die Taktflanke des langsamen Taktes ebenfalls steigend. Dies ist in *Abbildung 6.2* veranschaulicht.

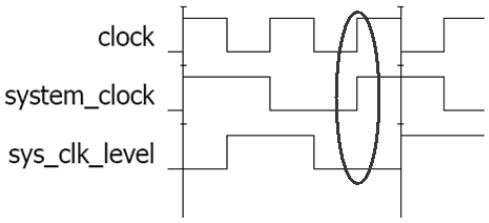


Abbildung 6.2: Phasensynchronisation des Takts

Der Zustandsautomat hat fünf Zustände, „RESET“, „LEERLAUF“, „TAG LESEN“, „LESEN/SCHREIBEN“ und „NACHLADEN“. Nach dem Einschalten befindet sich der Zustandsautomat im „RESET“-Zustand und der DDR-RAM enthält zufällige Daten. Deshalb können die ebenfalls zufälligen Daten im Cache-Speicher als gültiges Abbild des DDR-RAMs angenommen werden. Beim Booten des Systems wird zunächst der

untere Speicherbereich beschrieben. Um das unnötige Nachladen aus dem DDR-RAM von zufälligen Daten zu vermeiden, werden die Daten im Cache als Abbild des unteren Speicherbereichs angenommen. Dazu müssen im „RESET“-Zustand die Tag-Adressen und Attribute der Tag-Speicher korrekt initialisiert werden.

Mit den vier Cache-Blöcken, die je 4 KB Daten zwischenspeichern, können die unteren 16 KB Daten abgebildet werden. Dies entspricht dem Adressraum von 0x00000000 bis 0x00003FFF. Dies ergibt für die Cache-Blöcke folgende Adressbereiche:

Block 0: 0x00000000 bis 0x00000FFF
Block 1: 0x00001000 bis 0x00001FFF
Block 2: 0x00002000 bis 0x00002FFF
Block 3: 0x00003000 bis 0x00003FFF

Die Tag-Adressen innerhalb eines Blockes sind für alle Zeilen identisch und unterscheiden sich nur in den zwei LSBs gegenüber den anderen Blöcken. Die oberen 18 Bit sind dabei immer gleich „0“. Das Attribut für das Alter einer Cache-Zeile kann beliebig gewählt werden, jedoch muss dabei sichergestellt sein, dass bei den Cache-Zeilen mit gleicher Selektor-Adresse jeder Zählerwert nur einmal verwendet wird. Zur Vereinfachung wird das Attribut entsprechend den zwei LSBs der Tag-Adressen initialisiert. Das Dirty-Bit wird immer auf „0“ gesetzt. Somit ergeben sich für die Tag-Speicher der Cache-Blöcke die in *Abbildung 6.3* gezeigten Werte für die Initialisierung.

	TAG-ADRESSE	ALTER	DIRTY
BLOCK_0	00000000000000000000	00	0
BLOCK_1	00000000000000000001	01	0
BLOCK_2	00000000000000000010	10	0
BLOCK_3	00000000000000000011	11	0

Abbildung 6.3: Initialisierung der Tag-Speicher

Nach der Initialisierung wechselt er in den „LEERLAUF“-Zustand. Sobald während einer steigenden Taktflanke des langsamen System-Takts der Cache mit dem Chip-Select-Signal angesteuert wird, wechselt er in den „TAG LESEN“-Zustand, in welchem die Tag-Adressen aus den adressierten Cache-Zeilen ausgelesen werden. Danach wird in den „LESEN/SCHREIBEN“-Zustand gewechselt. In diesem Zustand werden die Tag-Adressen in den Cache-Blöcken miteinander verglichen und gegebenenfalls ein Hit-Signal erzeugt. Kommt es zu einem Hit, können die Daten aus dem entsprechenden Cache-Block gelesen oder geschrieben werden. Bei einem Miss wird in den „NACHLADEN“-Zustand gewechselt, in dem die benötigte Datenzeile aus dem DDR-RAM nachgeladen wird. Nach dem Nachladevorgang wird mit steigender Taktflanke wieder in den „TAG LESEN“-Zustand und darauf folgend in den „LESEN/SCHREIBEN“-Zustand gewechselt. Da die nachgeladenen Daten nun auf jeden Fall im Cache verfügbar sind, kommt es zu einem Hit. In diesem Fall wird mit steigender Taktflanke des System-Taktes wieder in den „TAG LESEN“-Zustand gewechselt, wenn der Cache mit dem Chip-Select-Signal erneut angesteuert wird. Ansonsten kehrt der Zustandsautomat in den „LEERLAUF“-Zustand zurück, bis ein erneuter Cache-Zugriff erfolgt. Das Zustandsdiagramm ist in *Abbildung 6.4* dargestellt.

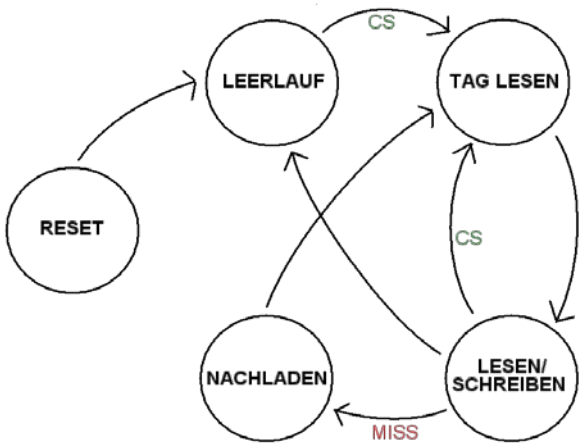


Abbildung 6.4: Zustandsautomat

7. Simulation

Zur Simulation des Cache-Systems wird das Programm ModelSim verwendet. Dabei kann das Verhalten der Schaltung genau analysiert werden. Damit die Eingangssignale nicht für jeden Taktzyklus einzeln festgelegt werden müssen, wird das Cache-System zusammen mit dem kompletten Mikrocontrollersystem simuliert. Sollte es im Cache-Speicher zu einem Fehler kommen, ist dies meist leicht am Fehlverhalten des Mikrocontrollersystems zu erkennen.

Bei der simulierten Systemfrequenz von 50 MHz erfolgt der Zugriff auf den Cache gemäß der Anforderung innerhalb eines System-Taktes. Dies entspricht einer Zugriffszeit von 20 ns. Beim Nachladen einer Datenzeile aus dem DDR-RAM werden für den gesamten Datenzugriff 23 Systemtakte benötigt, womit sich eine Zugriffszeit bei einem Miss von 460 ns ergibt. Falls dabei eine Cache-Zeile zuvor in den RAM zurückgeschrieben werden muss, dauert der Zugriff 35 Systemtakte, womit sich die Zugriffszeit auf 700 ns verlängert. Wird während des Nachladevorgangs ein Refresh des DDR-RAMs vom DRAM-Controller ausgelöst, verlängert sich der Zugriff um weitere 4 System-Takte bzw. 80 ns.

8. Verifikation im FPGA

In der Simulation werden die Durchlaufzeiten der Gatter nicht berücksichtigt. Um herauszufinden, ob die erstellte Schaltung mit der gewünschten Frequenz betrieben werden kann, wird diese im Cyclone III FPGA des Entwicklungsboards verifiziert. Dazu muss zunächst das fertige Design der Software Quartus II kompiliert werden. Dies erfolgt in mehreren Schritten.

Im ersten Schritt wird der VHDL-Code analysiert und in eine Gatternetzliste umgewandelt. Im nächsten Schritt müssen die Pins des FPGAs konfiguriert werden. Dazu wird die beim Entwicklungsboard mitgelieferte Pin-Beschreibung verwendet. Anschließend werden mit dem „Fitter“ die benötigten Gatter des FPGAs verdrahtet. Zuletzt erstellt der Assembler eine Konfigurationsdatei, die in den FPGA geladen werden kann. Nach dem Kompilieren kann der Software entnommen werden, welche Ressourcen des FPGAs für den Cache benötigt werden. Diese sind in *Abbildung 8.1* aufgelistet.

	Logik-Zellen	M9K-Blöcke	Pins	PLLs
CACHE_TOP	3805	27	44	1
CACHE_CONTROL	154	0	0	0
DDR_RAM_LOADER	73	0	0	0
DDR_CONTROLLER	2523	3	42	1
CACHE_MUX	131	0	0	0
CACHE_BLOCK (0)	226	6	0	0
CACHE_BLOCK (1)	225	6	0	0
CACHE_BLOCK (2)	225	6	0	0
CACHE_BLOCK (3)	226	6	0	0
RESET_LOGIC	33	0	1	0
SIRIUS_WRAPPER	20	0	0	0
OSZILLATOR	0	0	1	0

Abbildung 8.1: Verwendete Ressourcen des FPGAs

Um das Cache-System zu testen, wurde mit der SIRIUS IDE ein Programm geschrieben, das zuerst den kompletten DDR-RAM beschreibt und anschließend ausliest, wobei die gelesenen Daten überprüft werden. Während des Vorgangs kommt es dabei auch dazu, dass der Programmcode in den DDR-RAM geschrieben und wieder ausgelesen wird. Nach einem erfolgreich durchlaufenen Test wird eine bestimmte Melodie abgespielt.

9. Technische Daten

- 16 KB Cache-Speicher 4-fach assoziativ
- Wahlweise 16- oder 32-Bit-Datenbus
- Maximal 32-Bit-Adressbus (bis zu 4 GB Speicher adressierbar)
- DDR-RAM-Controller
- 100 MHz Betriebsfrequenz
- PLL zur Erzeugung der Taktsignale
- 50 MHz Taktgeber für die Systemfrequenz
- Resetlogik für das Mikrocontrollersystem
- Lese- und Schreib-Zugriffe auf Daten im Cache in einem System-Takt (20 ns)
- Lese- und Schreib-Zugriffe auf Daten im Hauptspeicher in 23 Takten ohne „write-back“ (460 ns) bzw. 35 Takten mit „write-back“ (700 ns)

10. Quellen

[1] Dirk Jansen, Nidal Fawaz, Daniel Bau, Marc Durrenberger: „A Small High Performance Microprocessor Core SIRIUS For Embedded Low Power Designs, Demonstrated in a Medical Mass Application Of an Electronic Pill (ePille®)“

Embedded System Design Topic, Techniques&Trends, p. 363-372 2007, USA Hanser Verlag, Deutschland, ISBN 978-0-387-72257-3

[2] Marc Durrenberger: „Überarbeitung eines 32 Bit Prozessor-Kerns mit Optimierung der ALU für die Synthese in eine 0.35 µm CMOS Bibliothek und Erprobung in einem FPGA“

Diplomarbeit, Hochschule Offenburg

[3] Deutschsprachige Wikipedia: „Cache“

http://de.wikipedia.org/wiki/Cache

[4] Bernd Becker, Paul Molitor: „Technische Informatik“

Oldenbourg Verlag München Wien, ISBN 978-3-486-58650-3

[5] John L. Hennessy, David A. Patterson: „Rechnerarchitektur“

Vieweg Verlag, ISBN 3-528-05173-6

[6] „DDR and DDR2 SDRAM High Performance Controller User Guide“

http://www.altera.com/literature/ug/ug_ddr_sdram.pdf

Vom UML-Modell einer Zustandsmaschine zu deren VHDL- und SystemC-Architektur

M. Holzer, T. Greiner, F. Schumacher, F. Kesel

Hochschule Pforzheim, MERSES[1] - Zentrum für Angewandte Forschung

www.merses.de, merses@hs-pforzheim.de

Diese Arbeit entstand im Rahmen des Zentrums für Angewandte Forschung MERSES[1] an der Hochschule Pforzheim und beschäftigt sich mit der automatischen Erzeugung von synthetisierbaren VHDL- und SystemC-Architekturen aus funktionalen UML-Zustandsmaschinenmodellen. Dabei kommen Methoden und etablierte Werkzeuge der Modellgetriebenen Softwareentwicklung (*eng.* Model Driven Software Development: MDSD) zum Einsatz.

Im ersten Schritt des Generierungsprozesses wird mittels Modell-zu-Modell-Transformation die UML-Spezifikation der Zustandsmaschine, unter Berücksichtigung eines selbst entworfenen Metamodells, in ein zielsprachenunabhängiges Zwischenmodell übersetzt. Während der Transformation wird das UML-Modell gegen die durch das Metamodell definierten Restriktionen geprüft und dem Benutzer gegebenenfalls Warnungen oder Fehlermeldungen angezeigt. War die Erzeugung des Zwischenmodells erfolgreich, wird Quellcode für jede Zielsprache mittels eigenem Satz von Generator-Templates erzeugt. Zusätzlich zu den erzeugten Architektur-Quellcode-Dateien, wird eine VHDL-Testbench für den funktionalen Test der Zustandsmaschine und ein Simulationsskript für den automatisierten Testablauf mit dem HDL-Simulationswerkzeug Modelsim (Mentor Graphics) erzeugt. Um eine bessere Strukturierung und Skalierbarkeit zu gewährleisten, werden hierarchische UML-Zustandsmaschinen unterstützt. Dabei sind sowohl einfache als auch parallele Unterzustandsmaschinen möglich.

Der beschriebene Generator-Workflow wurde mittels der Programmiersprache Java und dem Generator-Framework openArchitectureWare (oAW) in ein eigenständiges EDA-Werkzeug mit grafischer Benutzeroberfläche integriert.

1. Einleitung

Um komplexe Signalverarbeitungssysteme - oft im mobilen Umfeld - in eingebettete Systeme zu integrieren, kommen heutzutage immer öfter sogenannte System-on-Chip-Lösungen (SoC) zum Einsatz, bei denen das gesamte System, bzw. Subsysteme auf einem einzigen Chip integriert werden. Um die Entwicklung solcher komplexer VLSI-Designs effizienter zu gestalten und der Forderung nach kürzeren Entwicklungszyklen (time to market) folge leisten zu können, sind die Schlüsselfaktoren Abstraktion [1] und Wiederverwendbarkeit [2] von einzelnen Modulen besonders wichtig. Dazu werden immer mehr modellgestützte Beschreibungsmöglichkeiten für die einzelnen Algorithmenblöcke verwendet. Dabei führt eine Anhebung des Abstraktionsniveaus einzelner Systemkomponenten zu übersichtlicheren und kompakteren Systemspezifikationen, die eine geringere Fehleranfälligkeit aufweisen. Um dem zweiten Schlüsselfaktor zur Produktivitätssteigerung im Bereich des Hardwareentwurfs gerecht zu werden, sollten im Entwicklungsprozess möglichst viele vorhandene und getestete Komponenten, sogenannte IP-Module (eng. Intellectual Property: IP), wiederverwendet werden. Die wichtigsten Eigenschaften von IP-Modulen sind Rechenleistung, Energieverbrauch und Flächenbedarf auf dem Chip und meist gegenläufig [1]. Deren Bedeutung ist vom Einsatzfeld und den damit verbundenen Anforderungen an das Gesamtsystem abhängig. Hieraus resultiert, dass je stärker ein IP-Modul parametrisierbar bezüglich dieser äußeren, von seinem Einsatzfeld getriebenen Systemanforderungen ist, desto universeller kann es an die individuellen Erfordernisse eines Systems angepasst werden und somit seine Fähigkeit zur Wiederverwendung erhöht werden, was ebenfalls für

[1] MERSES: Modellgestützte Entwurfs- und Realisierungsmuster für Signalverarbeitende Eingebettete Systeme - Zentrum für Angewandte Forschung. Das ZAFH MERSES wird gefördert durch die Europäische Union, Europäischer Fonds für regionale Entwicklung und das Land Baden-Württemberg, Ministerium für Wissenschaft, Forschung und Kunst.

eine abstrakte Beschreibung der einzelnen IP-Module spricht.

Mit diesen abstrakten und kompakten Systemspezifikationen lassen sich modellbasiert einzelne Funktionsblöcke automatisch erzeugen. Im Bereich der Softwareentwicklung hat sich dieses Vorgehensparadigma bereits in einer Vielzahl von Anwendungsfeldern als sogenannte Modellgetriebene Softwareentwicklung (eng. Model Driven Software Development MDSD) als adäquate Methode zur Steigerung der Produktivität etabliert. MDSD ist hierbei ein Oberbegriff für Techniken, die aus formalen Modellen automatisiert lauffähige Software erzeugen [3]. Hierzu werden Teile der Software auf einem hohen Abstraktionsniveau beschrieben und mit entsprechenden Codegeneratoren und weiteren Informationen, beispielsweise über die Softwarearchitektur, lauffähige Module erzeugt. Dabei handelt es sich meist um einen mehrstufigen Prozess, welcher Modell-zu-Modell-Transformationen und Modell-zu-Code-Transformationen beinhaltet.

Zur Beschreibung solcher funktionaler Systemspezifikationen hat sich im Softwareumfeld die Modellierungssprache UML (eng. Unified Modeling Language, spezifiziert von [4]) als Quasi-Standard etabliert. Ihr Anwendungsfeld hat sich hierbei vom anfänglichen Einsatz zu Softwaredokumentationszwecken hin zur Modellgetriebenen Softwareentwicklung und deren Modell bzw. Metamodellmodellierung, sowie Analyse, Dokumentation und Validierung von Gesamtsystemen gewandelt. Eine Vielzahl von kommerziellen UML-Werkzeugen wie zum Beispiel Rhapsody (IBM), Enterprise Architect (Sparx Systems) und MagicDraw (NoMagic) versuchen sich im Bereich der automatischen Quellcodegenerierung aus UML-Modellen zu etablieren. Jedoch unterstützen diese Werkzeuge bisher keine automatische Erzeugung von Hardware-Beschreibungszielsprachen wie VHDL oder SystemC, sondern beschränken sich meist auf die Quellcodegenerierung objektorientierter Programmiersprachen wie C++, C# und Java. Alternativ zu kommerziellen gibt es frei erhältliche Werkzeuge, wie openArchitectureWare (oAW) [5], welches auf dem Eclipse Modeling Framework (EMF) basiert und ein Framework für die modellgetriebene Software- und Testentwicklung darstellt. Dabei bietet oAW im Wesentlichen die Möglichkeit der Template-basierten Codegenerierung für verschiedene Eingangsmodelltypen, wie EMF-Modelle, UML-Modelle im XMI (XML Metadata Interchange)-Format [4], aber auch Visio-Modelle oder textuelle DSL-Spezifikationen. Aus diesen Eingangsmodellen kann mittels einzelner Templates beliebiger Quellcode generiert werden, wobei die Validierung des Eingangsmodells und Modell-zu-Modell-Transformationen unterstützt werden.

Zustandsorientierte Modelle repräsentieren nach [6] ein System als Menge von Zuständen und Zustandsübergängen und sind als kontrollflussdominant einzustufen. Sie sind am besten zur Modellierung von Steueraufgaben geeignet und stellen im Bereich reaktiver Systeme, insbesonders im Bereich eingebetteter Systeme, ein weitverbreitetes Instrument zur Verhaltensbeschreibung von zeitlichen Abläufen dar. Beim Entwurf von synchronen Steuerwerken in digitalen Schaltungen werden sie ebenfalls häufig eingesetzt. Aufgrund dessen wurde in dieser Arbeit die automatische, Template-basierte Erzeugung von synthetisierbarem VHDL- und SystemC-Umsetzungen von Zustandsmaschinen (finite state machines FSM) aus funktionalen UML-Modellen untersucht und mittels oAW in einem eigenständigen EDA (Electronic Design Automation)-Werkzeug implementiert. Der dazu umgesetzte Generator-Workflow wird im Folgenden detailliert beschrieben.

2. Umsetzung von Zustandsmaschinen in VHDL und SystemC

Bei der Umsetzung (Grundkonzept siehe Listing 1) der Zustandsmaschinen in die Zielsprachen VHDL wird eine *Entity,* die alle relevanten Ein-/Ausgangssignale beinhaltet, sowie eine *Architecture* mit mindestens einem getakteten und einem kombinatorischen Prozess benötigt. Über den getakteten Prozess werden alle Register für Ausgangs- und Steuersignale, wie z. B. das Zustandsspeicherregister, umgesetzt. Das Zustandsspeicherregister wird normalerweise mit einem Aufzählungstyp realisiert, der alle Zustände der FSM sowie einen Leerlaufzustand beinhaltet. Der kombinatorische Prozess der FSM besteht üblicherweise aus einer *Case*-Anweisung, welche auf das Zustandsspeicherregister zugreift und je nach Eingangssignalbelegung dessen Zustandswert und die mit dem Zustand bzw. der Transition verknüpften Ausgangssignalwerte ändert. Weiterhin ist bei einer Umsetzung von hierarchischen FSM eine Einbindung der Unterzustandsmaschinen (welche in unsern Fall eigenständige Module sind) mittels VHDL-Schlüsselwort *COMPONENT* und deren Bindung an die Ein-/Ausgangssignale (mittels *PORT MAP)*, sowie Steuersignale und Kombinatorik zu deren Steuerung notwendig.

Bei der Umsetzung der Zustandsmaschinen in die Zielsprachen SystemC ist eine SystemC-Modul *SC_MODULE* mit ebenfalls einem getakteten und einem kombinatorischen Prozess sinnvoll (was einer

Kombination aus Entity und Architecture in VHDL entspricht [7]). Dieser beinhaltet alle relevanten Ein-/Ausgangssignale, sowie ein Zustandsspeicherregister, welches ebenfalls über einen Aufzählungstyp realisiert wird. Die Umsetzung des getakteten und kombinatorischen Prozesses ist identisch zu der in VHDL. Bei der Umsetzung von hierarchischen Zustandsmaschinen müssen die Konstruktoren der Unterzustandsmaschinen-Module (SystemC spezifisch) über den Konstruktor der Hauptzustandsmaschine aufgerufen und in diesem die Ein-/Ausgangs-, sowie die Steuersignalzuweisung von der Haupt- zu den Unterzustandsmaschinen vorgenommen werden.

```
entity Main is
Port (
        Input1 : in std_logic;
        Input2 : in std_logic;
        ...

        );
end Main;

architecture behaviour of Main is
type STATES is (STATE_IDLE,
                STATE_START,
                STATE_Q0,
                STATE_Q1,
                ...
                STATE_STOP);
signal StateHandler, next_StateHandler : STATES := STATE_IDLE;

Main_reg : process(clk,rst)
        begin
            if rst = '1' then
                StateHandler <= STATE_IDLE;
                ...
            elsif clk'event and clk = '1' then
                StateHandler <= next_StateHandler;
                ...
            end if;
end process Main_reg;

Main_comb : process(Input1,Input2,...)

        begin

            case StateHandler is
            ...
                when STATE_Q0 =>

                    next_StateHandler <= STATE_Q0;
                        if Input1 = '1' then
                        next_StateHandler <= STATE_Q1;
                        ...
                        end if;
                ...
            end case;
end process Main_comb;
```

Listing 1: Grundkonzept einer Zustandsmaschine in VHDL

3. Automatischer Entwurf von Zustandsmaschinen mit UML

Der Generator-Workflow des automatischen Entwurfs von Zustandsmaschinen mit UML besteht aus den folgenden drei Hauptbestandteilen

- Metamodell der Zustandsmaschine
- Modell-zu-Modell-Transformation des UML-Eingangsmodell in ein Zwischenmodell für die nachgeschaltete Quellcodeerzeugung
- Template-basierte Quellcode- und Testbench-Erzeugung

3.1. Das Metamodell der Zustandsmaschine

Die Grundlage des mehrstufigen Generierungsprozesses stellt ein Metamodell (Modell des Modells) der Zustandsmaschine dar. Dessen Aufgabe ist die Spezifikation der Struktur einer Zustandsmaschine und aller weiteren für den Generatorprozess relevanten Eigenschaften des UML-Eingangsmodells. Mithilfe dieses Metamodells kann die Vollständigkeit des UML-Eingangsmodells, im Hinblick auf den Generierungsprozess, verifiziert und gegebenenfalls vorhandene Mängel dem Benutzer mittels Warnungen oder Fehlermeldungen angezeigt werden. Das Metamodell der Zustandsmaschine ist ebenfalls in UML umgesetzt worden und besteht aus einem Struktur-/Klassendiagrammen, welches in Bild 1 dargestellt ist. Die wichtigsten semantischen/funktionalen Aspekte des Metamodells der Zustandsmaschine sollen im Folgenden kurz vorgestellt werden. Ausgangspunkt ist die Betrachtung, welche spezifischen Eigenschaften eines Zustandes berücksichtigt werden müssen. Dies sind der eigentliche Zustand selbst, ein Start- und ein Stopzustand sowie Transitionen und Ereignisse. Diese Anforderungen werden entsprechend im Metamodell berücksichtigt. Die Hauptklasse *StateMachine* muss hierzu mindestens einen *State* beinhalten (existenzielle Abhängigkeit), wobei es sich bei *States* um eine abstrakte Klasse handelt, welche ihre Eigenschaften an die spezialisierten Klassen *SimpleState*, *StartState* und *StopState* vererbt. Außerdem besitzt *StateMachine* Abhängigkeiten (Aggregationen) zu den Klassen *Action* und *Signal*. Jeder *State* kann Transitionen besitzen, jedoch nur *SimpleState* darf Aktionen auslösen, da die Pseudozustände (Start-/Stoppzustände) in UML lediglich zur Ablaufsteuerung verwendet werden. Die Aktionen, die aus einem normalen Zustand (*SimpleState*) aus möglich sind, werden unterschieden in *doActions* und *entry-/exitActions*. Die ersteren sind während der gesamten Ausführungsdauer des Zustands aktiv, während die letzteren lediglich beim Betreten oder vor dem Verlassen des Zustands ausgeführt werden. Jeder normale Zustand kann außerdem wieder beliebig viele Unterzustandsmaschinen (*subStateMachine*) beinhalten. Jede Transition kann genau ein *Event* besitzen, auf das sie sensitiv ist. Ein Event ist genau

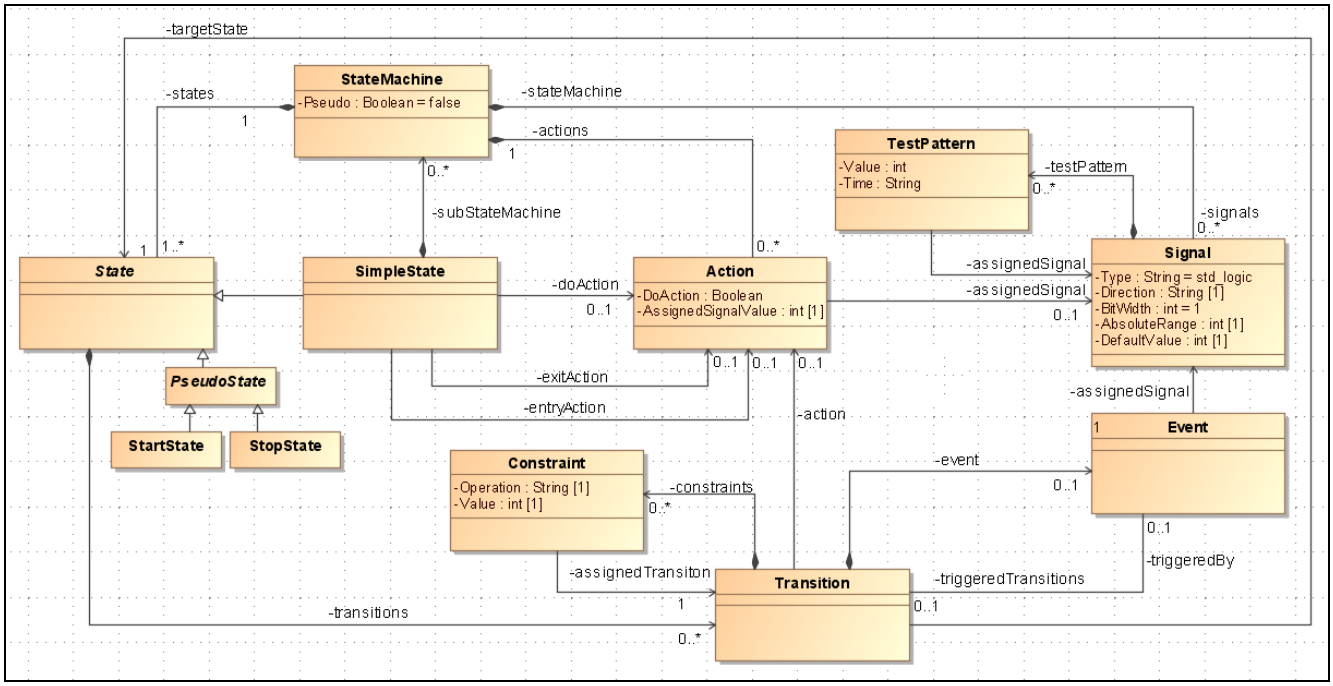


Bild 1: Metamodell der Zustandsmaschine

einem Signal zugewiesen (*assignedSignal*). Weiterhin ist jede Transition an beliebig viele Weiterschaltbedingungen (*Constraint*) gebunden, welche sich auf das über den Event zugewiesene Eingangssignal beziehen. Jedem Signal können für die automatische Erstellung der Testbench für die Zustandsmaschine, beliebig viele TestPattern zugewiesen werden. Diese Testsequenzen legen die zeitlichen Stimuli der Eingangs- und die dazu gehörenden Verifikationsmuster der Ausgangssignale für den automatischen Testablauf fest.

3.2. Modell-zu-Modell-Transformation

Das UML-Eingangsmodell der Zustandsmaschine wird im ersten Arbeitsschritt in ein zielsprachenunabhängiges Zwischenmodel (EMF Ecore-Modell) überführt. Bild 2 stellt ein solches UML-Eingangsmodell dar. Die darin enthaltene Zustandsmaschine *Main* besteht aus vier Hauptzuständen (*SimpleState*): Q_0, Q_1, Q_{cb}, Q_{ot}, den schwarz gekennzeichneten Startzuständen und den Stoppzuständen. Bei den Zuständen Q_1, Q_{cb} und Q_{ot} handelt es sich um zusammengesetzte Zustände. Q_1 enthält die Unterzustandsmaschine *Sub1* (siehe Bild 3), welche ein eigenes Element vom UML-Typ *State Machine* mit eigenen Ein- und Ausgängen, darstellt. Q_{cb} enthält eine sogenannte Pseudo-Unterzustandsmaschine, welche keine eigene Ein-/Ausgänge definieren kann, sondern sich diese mit der hierarchisch übergeordneten Zustandsmaschine teilt. Q_{ot} besitzt zwei parallele Unterzustandsmaschinen, von denen die untere eine Pseudo-Unterzustandsmaschine ist und beide terminiert sein müssen (das heißt den Endzustand erreicht haben) bevor der übergeordnete Zustand verlassen werden kann. An den Transitionen zwischen den Zuständen sind die Trigger-Signale und Weiterschaltbedingungen (in geschweiften Klammern), sowie (mittels Schrägstrich getrennt) eventuelle Aktionen vermerkt. Außerdem sind optional in den einzelnen Zuständen die auszulösenden Aktionen aufgeführt. Dabei wird, wie bereits erwähnt, zwischen *do*-Aktionen und *entry/exit*-Aktionen unterschieden (*Anmerkung*: alle Aktionen haben das Setzen eines zugewiesenen Ausgangssignals zur Folge). Da sowohl Transitionen als auch Zustände Aktionen ausführen können bzw. Zustände *entry-/exit*-Aktionen auslösen können, ist es möglich, sowohl **Moore-** als **Mealy-Automaten** aus dem UML-Eingangsmodell zu generieren. Zur Zeit wird der UML-Event-Typ *SignalEvent* als Transitions-Trigger-Ereignis unterstützt. In Bild 4 ist das erzeugte Zwischenmodell für das hierarchische UML-Eingangsmodell (nach Bild 2 und Bild 3) dargestellt. Zu erkennen sind die Unterzustandsmaschinen und deren Referenzierung im Zustand Q_{ot}. Die letzten beiden Zustandsmaschinen in Bild 4 sind Pseudo-Unterzustandsmaschinen, was über den angehängten Suffix PSSM<*KNOTENNAME*> angezeigt wird. Die Weiterschaltbedingungen beziehen sich immer auf das mit der Transition verknüpfte Eingangsignal. Es sind die Vergleichsrelationen: *gleich*, *ungleich*, *größer*, *kleiner*, *größergleich* und *kleinergleich* bezogen auf einen konstanten Wert im dezimalen oder hexadezi-

malen Zahlenformat (Präfix „0x“) möglich. Den Ein-/Ausgangssignalen der Zustandsmaschinen können beliebige Bitbreiten kleiner 32 Bit zugewiesen werden. Außerdem können ihnen wie bereits erwähnt, mittels UML-Constraint sogenannte *TestPattern* zugewiesen werden. Diese bilden die Datengrundlage für die automatische Generierung der Testbench.

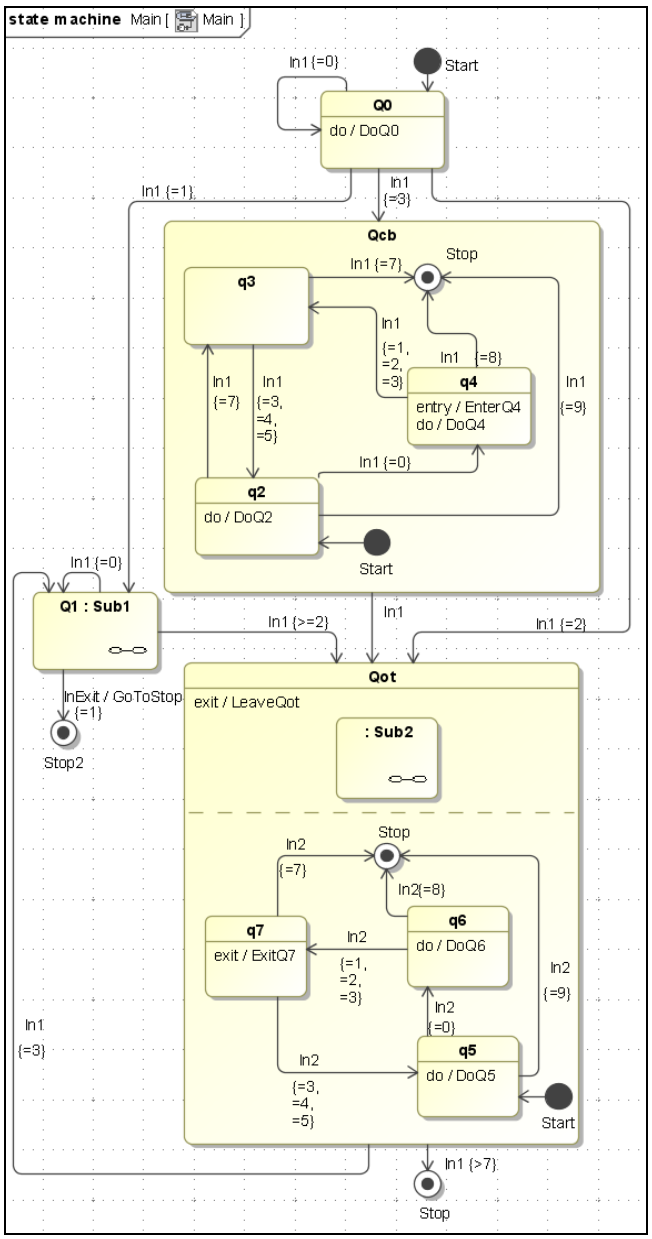


Bild 2: Eingangsmodell der Zustandsmaschine Main in UML

Vor der eigentlichen Erzeugung des zielsprachenunabhängigen Zwischenmodels wird das UML-Eingangsmodell auf seine Vollständigkeit geprüft. Dabei werden alle durch das Metamodell definierten Restriktionen, sowie selbst definierte Regeln überprüft. Eine selbstdefinierte Regel kann zum Beispiel überprüfen, ob der Zahlenwert der Weiterschaltbedingungen einer Transition den, über die Bitbreite definierten, Wertebereich des Trigger-Signals, überschreitet. Eine solche komplexe Abhängigkeit dieser beiden Attribute voneinander wäre nur sehr schwer über das UML-Metamodell abbildbar.

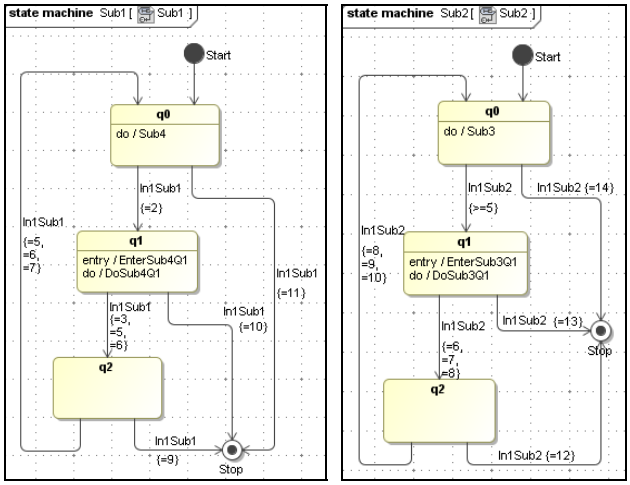


Bild 3: UML-Modell der Zustandsmaschine Sub1 und Sub2

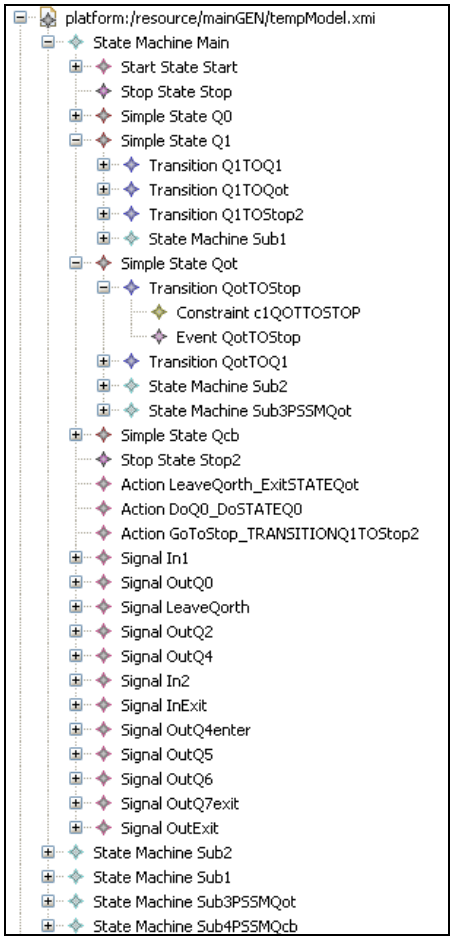


Bild 4: Zielsprachenunabhängiges Zwischenmodel

3.3. Template-basierte Quellcode- und Testbench-Erzeugung

In diesem Arbeitsschritt des Generatorprozesses wird basierend auf dem zuvor erzeugten Zwischenmodell und zwei Sätzen Generator-Templates der VHDL- und SystemC-Quellcode erzeugt. Ein Satz von Generator-Templates besteht aus Erstellungs-Skripten für

- die hierarchische Zustandsmaschine
- die Testbench mit optionalen Test-Pattern der Ein-/Ausgangssignale
- das Do-Skript zur automatischen Ausführung des Komponententests mit Modelsim

Für alle im UML-Modell enthaltenen Zustandsmaschinen wird ein separates VHDL-/SystemC-Modul, eine VHDL-Testbench (gekennzeichnet mit Präfix „TB“), sowie ein separates Modelsim-Do-Skript für VHDL-/SystemC generiert (siehe Bild 5).

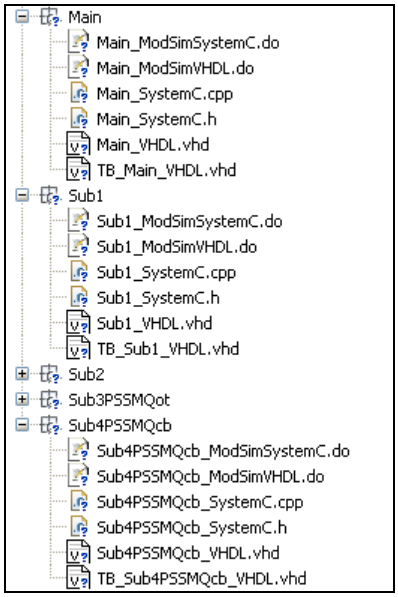


Bild 5: Generierte Quelldateien des UML-Eingangsmodell der hierarchischen-Zustandsmaschine Main

Jede einzelne Zustandsmaschine kann somit eigenständig, basierend auf den durch das UML-Modell zugeordneten Testsequenzen ihrer Ein-/Ausgangssignale, mit Modelsim simuliert werden.

Die Entity der Top-Level-Zustandsmaschine *Main,* (siehe Listing 2) beinhaltet außer den eigenen Signalen, alle Ein-/Ausgangssignale der Unterzustandsmaschinen. Dabei ist zu beachten, dass die Ausgangssignale der Unterzustandsmaschinen einen eindeutigen Namen mittels Präfix (Name der Zustandsmaschine) erhalten. Dies vermeidet, dass ein Ausgangssignal von zwei Quellen getrieben wird. Die Top-Level-Zustandsmaschine *Main* bindet weiterhin alle Unter-Zustandsmaschinen als Komponenten/Instanzen ein (siehe Listing 3) und sorgt für deren Steuerung, das heißt, wenn der mit der Unterzustandsmaschine verknüpfte Knoten erreicht wird, wird diese gestartet und ihre Terminierung überwacht. Die Terminierung aller Zustandsmaschinen eines Zustandes ist obligatorisch, damit dieser über seine ausgehenden Transitionen verlassen werden kann.

```
entity Main is
Port (  -- sub state machine relevant in/out signals
        SIGNAL_In1Sub1 : in std_logic_vector (3 downto 0);
        SIGNAL_In1Sub2 : in std_logic_vector (3 downto 0);
        -- unique output signals for all sub state machines
        SIGNAL_EnterQ1_Sub1 : out std_logic;
        SIGNAL_DoSub4_Sub1 : out std_logic_vector (1 downto 0);
        SIGNAL_DoQ1_Sub1 : out std_logic_vector (1 downto 0);
        SIGNAL_EnterSub3Q1_Sub2 : out std_logic;
        SIGNAL_DoSub3_Sub2 : out std_logic_vector (1 downto 0);
        SIGNAL_DoSub3Q1_Sub2 : out std_logic;
        -- main state machine relevant in/out signals
        clk : in std_logic;
        rst : in std_logic;
        enable : in std_logic; -- enable state machine, go from idle state to start state
        SIGNAL_In1 : in std_logic_vector (3 downto 0);
        SIGNAL_In2 : in std_logic_vector (3 downto 0);
        SIGNAL_InExit : in std_logic_vector (3 downto 0);
        stop  : out std_logic; -- stop flag, is set when one stop state is reached
        SIGNAL_OutQ0 : out std_logic;
        SIGNAL_LeaveQorth : out std_logic;
        -- signal OutQ2 is mapped to pseudo state machine Sub4PSSMQcb
        SIGNAL_OutQ2 : out std_logic;
        -- signal OutQ4 is mapped to pseudo state machine Sub4PSSMQcb
        SIGNAL_OutQ4 : out std_logic;
        -- signal OutQ4enter is mapped to pseudo state machine Sub4PSSMQcb
        SIGNAL_OutQ4enter : out std_logic_vector (1 downto 0);
        -- signal OutQ5 is mapped to pseudo state machine Sub3PSSMQot
        SIGNAL_OutQ5 : out std_logic_vector (3 downto 0);
        -- signal OutQ6 is mapped to pseudo state machine Sub3PSSMQot
        SIGNAL_OutQ6 : out std_logic_vector (2 downto 0);
        -- signal OutQ7exit is mapped to pseudo state machine Sub3PSSMQot
        SIGNAL_OutQ7exit : out std_logic_vector (1 downto 0);
        SIGNAL_OutExit : out std_logic
        );
end Main;
```

Listing 2: VHDL-Entity der Top-Level-Zustandsmaschine

```
COMPONENT Sub1 Port (
        -- Sub1 state machine relevant in/out signals
        clk : in std_logic;
        rst : in std_logic;
        enable : in std_logic; -- enable state machine, go from idle state to start state
        SIGNAL_In1Sub1 : in std_logic_vector (3 downto 0);
        stop  : out std_logic; -- stop flag, is set when one stop state is reached
        SIGNAL_EnterQ1 : out std_logic;
        SIGNAL_DoSub4 : out std_logic_vector (1 downto 0);
        SIGNAL_DoQ1 : out std_logic_vector (1 downto 0)
    );
END COMPONENT;

...
```

Listing 3: Komponenten-Deklaration der Unterzustandsmaschinen in Top-Level-Zustandsmaschine (VHDL)

Mit der Top-Level-Testbench kann die komplette hierarchische Zustandsmaschine simuliert werden, da die Top-Level-Zustandsmaschine alle benötigten Signale an die Unterzustandsmaschinen durchreicht

```
-- input signal stimulation
Stim_In1: process
begin
    -- wait for reset to occur  (4*clk_period)
    wait for 6 * clk_period;
    -- start stimulation of input signal: In1
    SIGNAL_In1 <= "0000"; wait for 5 * clk_period; -- assigned value is: 0
    SIGNAL_In1 <= "0011"; wait for 1 * clk_period; -- assigned value is: 3
    SIGNAL_In1 <= "0000"; wait for 5 * clk_period; -- assigned value is: 0
    SIGNAL_In1 <= "0011"; wait for 1 * clk_period; -- assigned value is: 3
    SIGNAL_In1 <= "0111"; wait for 1 * clk_period; -- assigned value is: 7
    SIGNAL_In1 <= "0011"; wait for 51 * clk_period; -- assigned value is: 3
    SIGNAL_In1 <= "0001"; wait for 10 * clk_period; -- assigned value is: 1
    wait; -- end of stimulation
end process;

...

-- output signal verification main state machine
Verify_OutQ0: process
begin
    -- wait for reset to occur  (4*clk_period)
    wait for 6 * clk_period;
    -- start verification of output signal: OutQ0
    wait for 2 * clk_period;
    if SIGNAL_OutQ0 = '1' then report "SIMULATION> Test Ok! Signal: SIGNAL_OutQ0 = 1 ";
    else
        report "SIMULATION> Test failed! Signal: SIGNAL_OutQ0 = "
        & STD_LOGIC'image(SIGNAL_OutQ0) & " ( /= 1) " severity FAILURE;
    end if;
    Verification_OutQ0_END <= true; --  this verification process is finished
    wait;
end process;
```

Listing 4: Auszug Testbench Stimulation und Verifikation von Ein-/Ausgangssignalen

(in VHDL mittels `PORT MAP` Befehl) und diese, aufgrund der im UML-Modell definierten Testsequenzen, von ihr stimuliert bzw. validiert werden (siehe Listing 4).

Die Modelsim-Do-Skripte für die automatische Ausführung des Komponententests veranlassen Modelsim alle VHDL-/SystemC-Module, die zur hierarchischen Zustandsmaschine gehören, sowie deren Testbench, mittels *vcom/sccom* Befehl zu übersetzen und in eine Library namens *workVHDL/ workSystemC* abzulegen. Anschließend wird die Simulation gestartet und mittels eines Labels (welches nach Ablauf aller Tests aus der Testbench gesetzt wird) das Ende des Komponententests überwacht und die Simulation angehalten.

4. EDA-Werkzeug mit grafischer Benutzeroberfläche

Das mit Java umgesetzte Generator-Werkzeug *FSMgen 2.0* erlaubt es, den Generator- und Simulationsprozess in einer eigenständigen Anwendung zu verwalten und zu steuern, da alle benötigten Bibiliotheken und externen Abhängigkeiten (hauptsächlich oAW und EMF betreffend) integriert wurden. Die GUI des EDA-Werkzeug wird in ***Bild 6*** dargestellt. Über die Auswahlboxen „Target Language“ kann die Zielsprache ausgewählt werden.

Mit dem Button „Start Generator“ wird der Generator-Vorgang und mit dem Button „Start Simulation“ der Simulationsvorgang gestartet. Über einen weiteren Dialog kann das Do-Skript ausgewählt werden, mit dem Modelsim dann direkt gestartet wird.

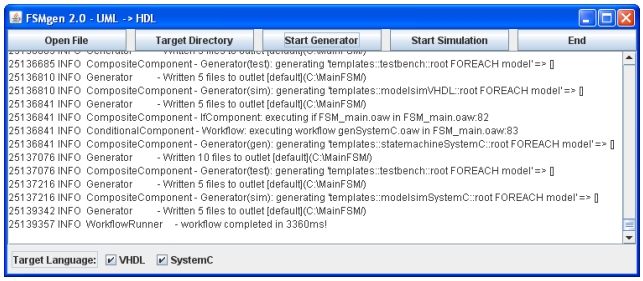


Bild 6: GUI des Generator-Werkzeug FSMgen 2.0

Das Ergebnis des funktionalen Komponententests mit Modelsim ist grafisch in Bild 7 und textuell in Listing 4 dargestellt. In Bild 7 sind die zeitlichen Verläufe der Ein-/Ausgangssignale dargestellt, sowie einige wichtige Steuersignale der Haupt- und Unterzustandsmaschinen. Besonders interessant sind die statehandler-Signale der einzelnen Zustandsmaschinen und deren enable/stop-Signale. Sie geben Aufschluss darüber, welche Unterzustandsmaschine zu welchen Zeitpunkten aktiv ist und in welchem Zustand sie sich befindet. Für jede in der Testbench definierte Ausgangssignal-Verifikation wird zu Dokumentationszwecken eine Meldung im Modelsim-Transcript-Fenster (Listing 4) und in der Modelsim-Protokoll-Datei *transcript* ausgegeben.

```
# ** Note: SIMULATION> Test Ok! Signal: SIGNAL_OutQ0 = 1
#    Time: 80 ns  Iteration: 0  Instance: /tb_main
# ** Note: SIMULATION> Test Ok! Signal: SIGNAL_OutQ2 = 1
#    Time: 150 ns  Iteration: 0  Instance: /tb_main
# ** Note: SIMULATION> Test Ok! Signal: SIGNAL_OutQ4enter = 3
#    Time: 160 ns  Iteration: 0  Instance: /tb_main
# ** Note: SIMULATION> Test Ok! Signal: SIGNAL_OutQ4 = 1
#    Time: 160 ns  Iteration: 0  Instance: /tb_main
# ** Note: SIMULATION> Test Ok! Signal: SIGNAL_DoSub3_Sub2 = 3
#    Time: 240 ns  Iteration: 0  Instance: /tb_main
# ** Note: SIMULATION> Test Ok! Signal: SIGNAL_OutQ5 = 15
#    Time: 240 ns  Iteration: 0  Instance: /tb_main
# ** Note: SIMULATION> Test Ok! Signal: SIGNAL_EnterSub3Q1_Sub2 = 1
#    Time: 250 ns  Iteration: 0  Instance: /tb_main
# ** Note: SIMULATION> Test Ok! Signal: SIGNAL_OutQ6 = 7
#    Time: 260 ns  Iteration: 0  Instance: /tb_main
# ** Note: SIMULATION> Test Ok! Signal: SIGNAL_OutQ7exit = 3
#    Time: 320 ns  Iteration: 0  Instance: /tb_main
# ** Note: SIMULATION> Test Ok! Signal: SIGNAL_DoSub3Q1_Sub2 = 1
#    Time: 460 ns  Iteration: 0  Instance: /tb_main
# ** Note: SIMULATION> Test Ok! Signal: SIGNAL_LeaveQorth = 1
#    Time: 570 ns  Iteration: 0  Instance: /tb_main
# ** Note: SIMULATION> Test Ok! Signal: SIGNAL_DoSub4_Sub1 = 3
#    Time: 600 ns  Iteration: 0  Instance: /tb_main
# ** Note: SIMULATION> Test Ok! Signal: SIGNAL_EnterQ1_Sub1 = 1
#    Time: 610 ns  Iteration: 0  Instance: /tb_main
# ** Note: SIMULATION> Test Ok! Signal: SIGNAL_DoQ1_Sub1 = 3
#    Time: 620 ns  Iteration: 0  Instance: /tb_main
# ** Note: SIMULATION> Test Ok! Signal: SIGNAL_DoSub4_Sub1 = 3
#    Time: 660 ns  Iteration: 0  Instance: /tb_main
# ** Note: SIMULATION> Test Ok! Signal: SIGNAL_OutExit = 1
#    Time: 730 ns  Iteration: 0  Instance: /tb_main
# Simulation positiv finished!
# Simulation stop requested.
# Simulation Breakpoint: Simulation stop requested.
# MACRO ./Main_ModSimSystemC.do PAUSED at line 55
```

Listing 5: Ergebnis der Modelsim Simulation (Transcript-Fenster)

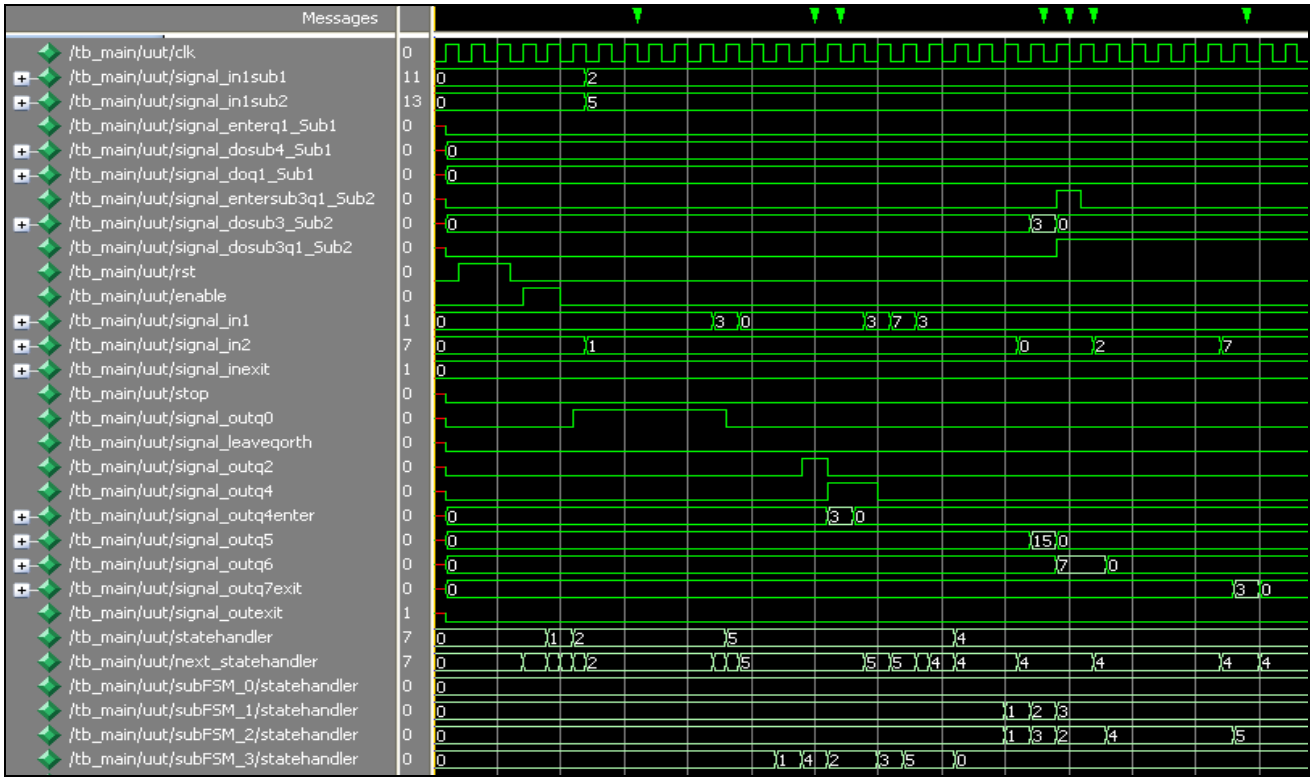


Bild 7: Auszug des Wave-Fensters der Modelsim-Simulation (SystemC-Architektur)

Die Synthesefähigkeit der erzeugten VHDL-Architektur wurde mittels Xilinx ISE 10.1 getestet. Der Synthesevorgang verlief erfolgreich und warnungsfrei.

5. Zusammenfassung

In der hier vorliegenden Arbeit wurde die Umsetzung eines zweistufigen, systematischen Generatoransatzes zur Erzeugung von synthetisierbaren VHDL- und SystemC-Architekturen aus funktionalen UML-Zustandsmaschinenmodellen vorgestellt. Der zweistufige Ansatz hat dabei den Vorteil, dass zielsprachenunabhängige, funktionale Aspekte von der eigentlichen Zielsprachengenerierung getrennt werden und somit dasselbe Zwischenmodell für verschiedene Zielsprachen, in diesem Fall für die Erzeugung von VHDL- und SystemC-Quellcode, benutzt werden kann. Ein weiteres Augenmerk lag auf der automatischen Generierung einer Testumgebung aus dem mit Testdaten angereicherten UML-Modell, welche in Ergänzung mit den ebenfalls generierten Modelsim-Do-Skripten eine gute Grundlage für den schnellen Komponententest des funktionalen UML-Zustandsmaschinenmodells erlaubt.

Die Integration des Generator- und Simulationsprozesses in ein eigenständiges Werkzeug erleichtert hierbei die Handhabung für den Benutzer erheblich und sorgt für schnellere Iterationszyklen bzw. Verifikationszyklen beim Design von hierarchischen Zustandsmaschinen.

6. Literaturverzeichnis

1. **Bartholomä R.** *Automatische Synthese von Hardware-Architekturen aus Lifting-basierten Filterbankspezifikationen.* München : Hut, 2008.

2. **Vörg A., Madrid N. Martínez, Seepold R., Rosenstiel W.** IP-Qualifizierung wiederverwendbarer Schaltungsbeschreibungen,Wolfgang Rosenstiel, *VDE.* 2002.

3. **Stahl T., Völter M., Efftinge S.** *Modellgetriebene Softwareentwicklung. Techniken, Engineering, Management.* Heidelberg : dpunkt, 2007.

4. **The Object Management Group (OMG).** http://www.omg.org/. *The Object Management Group (OMG).* [Online]

5. **openArchitectureWare.** http://www.openarchitectureware.org/. [Online]

6. **Teich J., Haubelt C.** *Digitale Hardware/Software-Systeme: Synthese und Optimierung.* Berlin : Springer, 2007. 3540468226.

7. **Kesel F., Bartholomä R.** *Entwurf von digitalen Schaltungen und Systemen mit HDLs und FPGAs.* s.l. : Oldenbourg, 2006. 3486575562.

Continuous Time Sigma-Delta ADC für High Performance Class-D Verstärker

Prof.
Christian Münker

Hochschule München

XLII. MPC Workshop

Karlsruhe 10. Juli 2009

Übersicht

BMBF-Projekt ***Energieeffiziente intelligente Class-D Verstärker für flexible ELA- Systeme (EiffELA)***

Förderlinie „IngenieurNachwuchs“

Schwerpunkte:

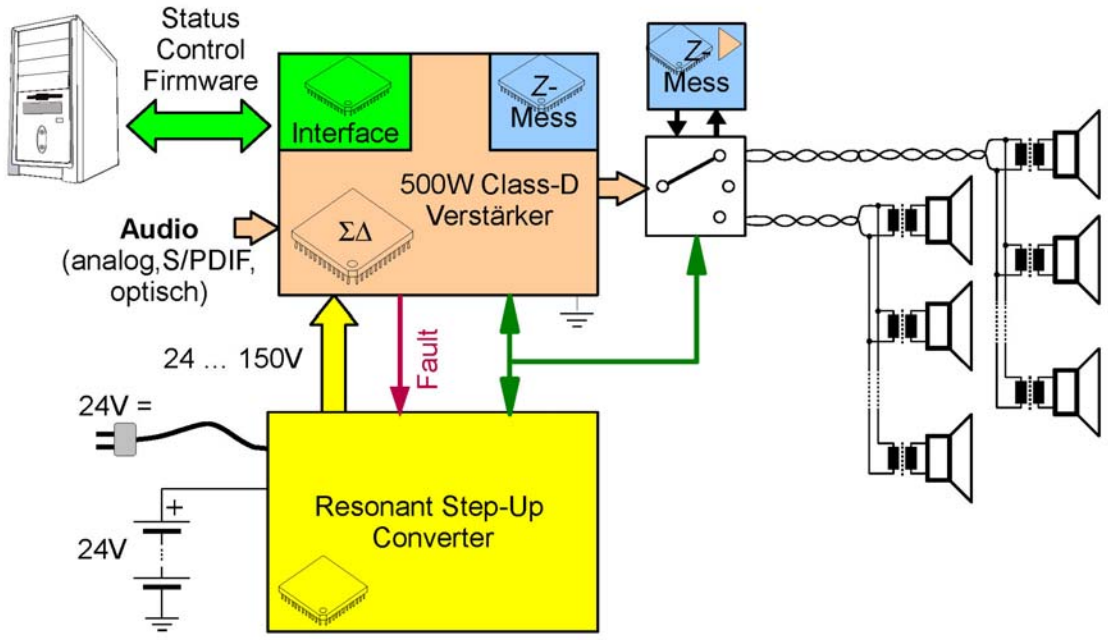
- ► Robuste Class-D Verstärker
- ► Frequenzabhängige Impedanzmessung
- ► Effiziente, notstromfähige Stromversorgungen
- ► Entwurf gemischt analog-digitaler Systeme

Industriepartner: Stemin GmbH, Königsdorf, mit ca. 25 Mitarbeitern
Weltweit aktiv auf dem Gebiet Konferenzsysteme und öffentliche Durchsageanlagen (U-Bahn etc.)

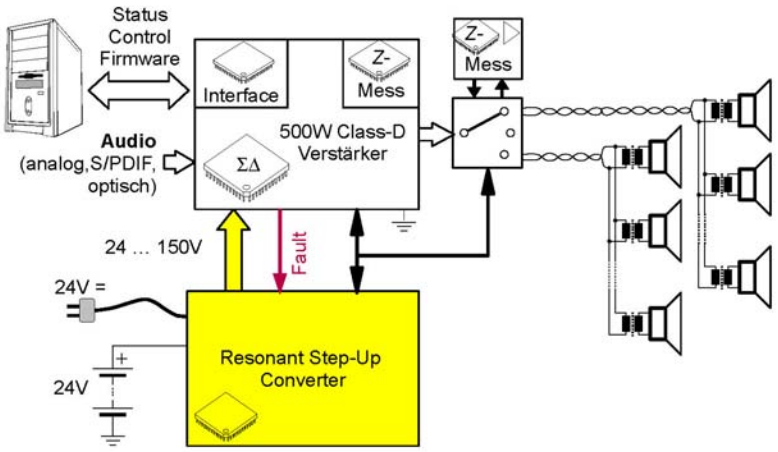
Projektvorgaben

- **Kompatibilität** zu bestehender Infrastruktur (100V - Technik)
- **Permanente Überwachung** des gesamten Signalpfads im laufenden Betrieb
  - Ausfall einzelner Lautsprecher muss detektiert werden
- **Notstromversorgung mit hoher Energieeffizienz** (> 90 % Wirkungsgrad) und **kompakter Aufbau**
  - Längere Notstromversorgung / kleinere Batterien
  - Passive Kühlung, möglichst ohne Kühlkörper
  - Verzicht auf Ausgangsübertrager → 150 V Betriebsspannung
- **Hohe Übertragungsqualität** und Zuverlässigkeit bei 500 W Dauerleistung an unterschiedlichsten Lastimpedanzen
  - Kurzschlussfest / leerlauffest / überlast- und übertemperaturfest
  - SNR > 100 dB, THD > 80 dB
  - Bandbreite 20 ... 20 kHz
- **Langlebigkeit** (> 20 Jahre): Robuster Aufbau und Rekonfigurierbarkeit

Übersicht Gesamtsystem



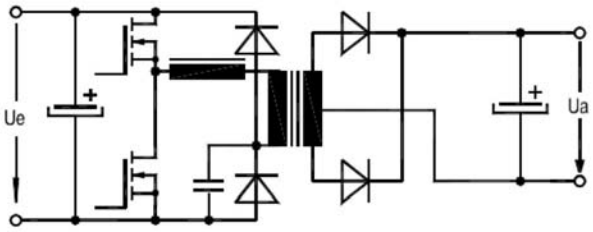
Übersicht



- **Resonanzwandler**
- Class-D Verstärker
- Impedanzüberwachung
- (Interfaces und Bussteuerung)

Resonanzwandler

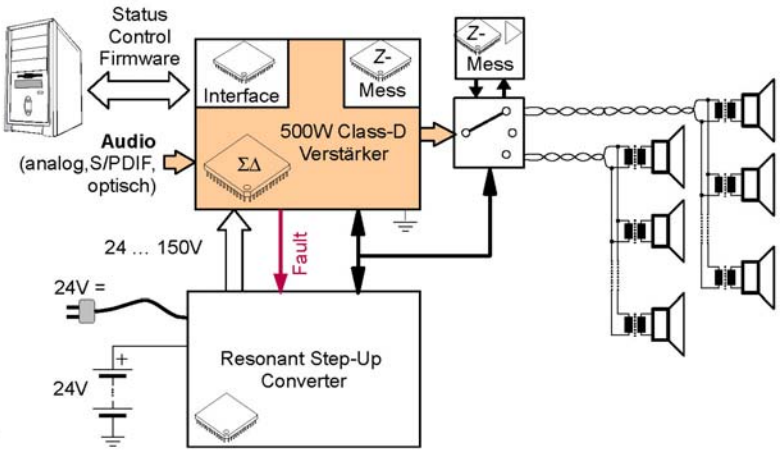
- Ausgangsspannung: 24 … 150 V, Ausgangsleistung: 0 … 500 W
- Eingangsspannung: 19 … 28 V → Primärströme bis 30 A!
- Effizienz > 90% bei Volllast



Eigenschaften

- Geringeres „Klingeln“ im passiven Gleichrichter durch sinusförmige Signalformen
- Kompakte und günstige Realisierung mit Print-Trafo, aber größere Parametervariation
- Adaptive Signalverarbeitung (FPGA) ermöglicht Selbstabgleich und Betriebsmodi oberhalb und unterhalb der Resonanzfrequenz

Übersicht

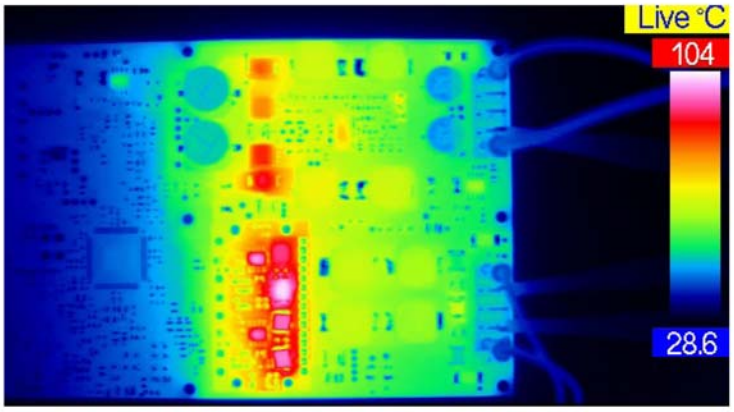


- Resonanzwandler
- **Class-D Verstärker**
- Impedanzüberwachung
- (Interfaces und Bussteuerung)

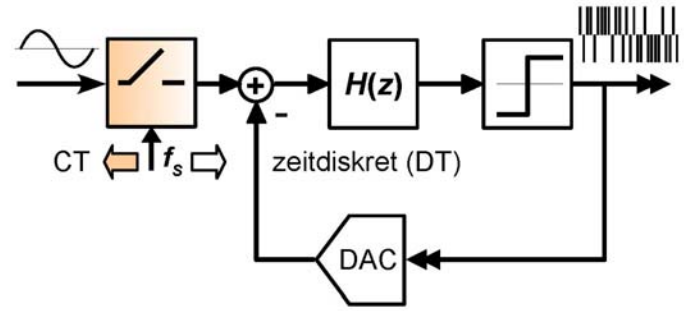
Energieeffizienter Class-D Verstärker

Voriges Projekt:

- 97 % Wirkungsgrad
- 300W
- Ohne Kühlkörper!

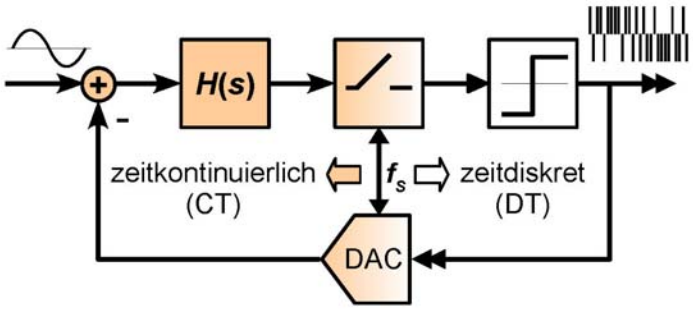


Vergleich DT und CT ΣΔ – ADC



DT ΣΔ – ADC

+ Präzises *H(z)* (SC-Filter)
+ Unempfindlich auf Jitter
- AA-Filter erforderlich
- Sampler begrenzt Auflösung
- OPV-Settling begrenzt f_S

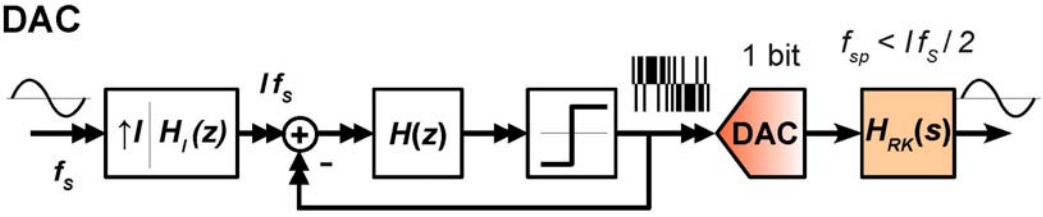


CT ΣΔ – ADC

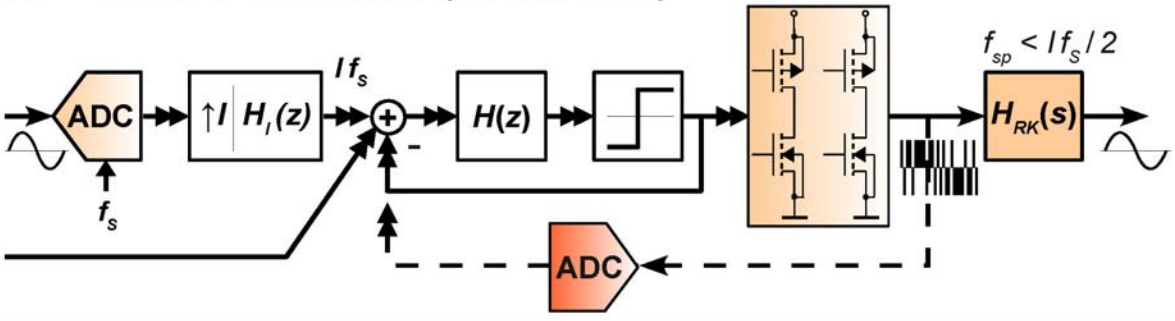
+ *H(s)* ist schneller als *H(z)*
+ AA-Filter meist unnötig
- Jitterempfindlich (DAC)
- DAC & Q.-Latenz kritisch
- Hohe Linearität 1. Stufe
- Starke *H(s)* Variation → keine Multiloop-Topologien

Prinzip digitaler ΣΔ Class-D Verstärker

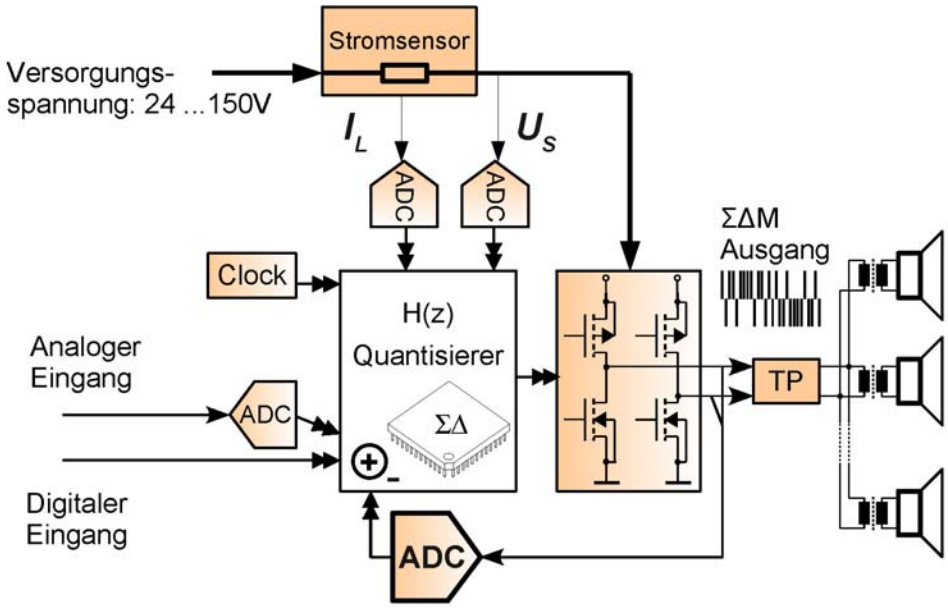
ΣΔ – DAC



ΣΔ – Class-D Verstärker (Power DAC)

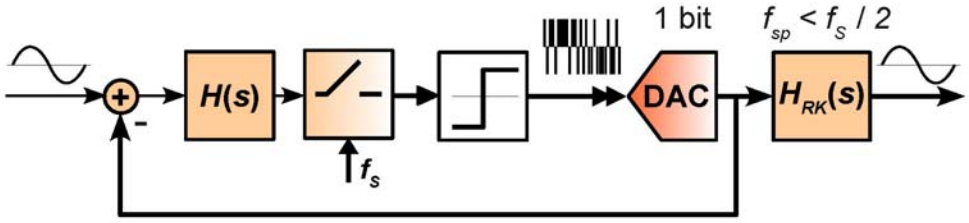


Digitaler Class-D Verstärker

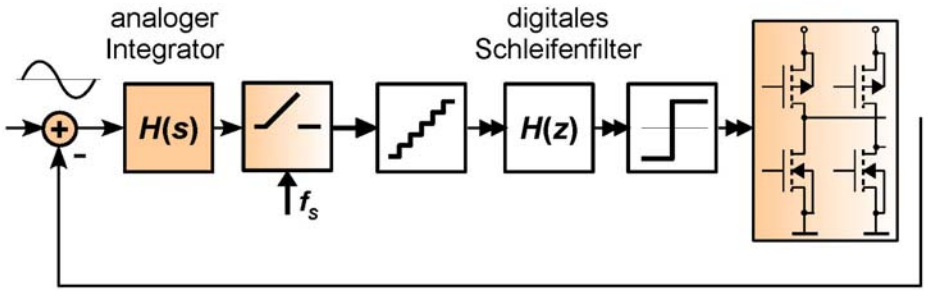


Prinzip analoger ΣΔ Class-D Verstärker

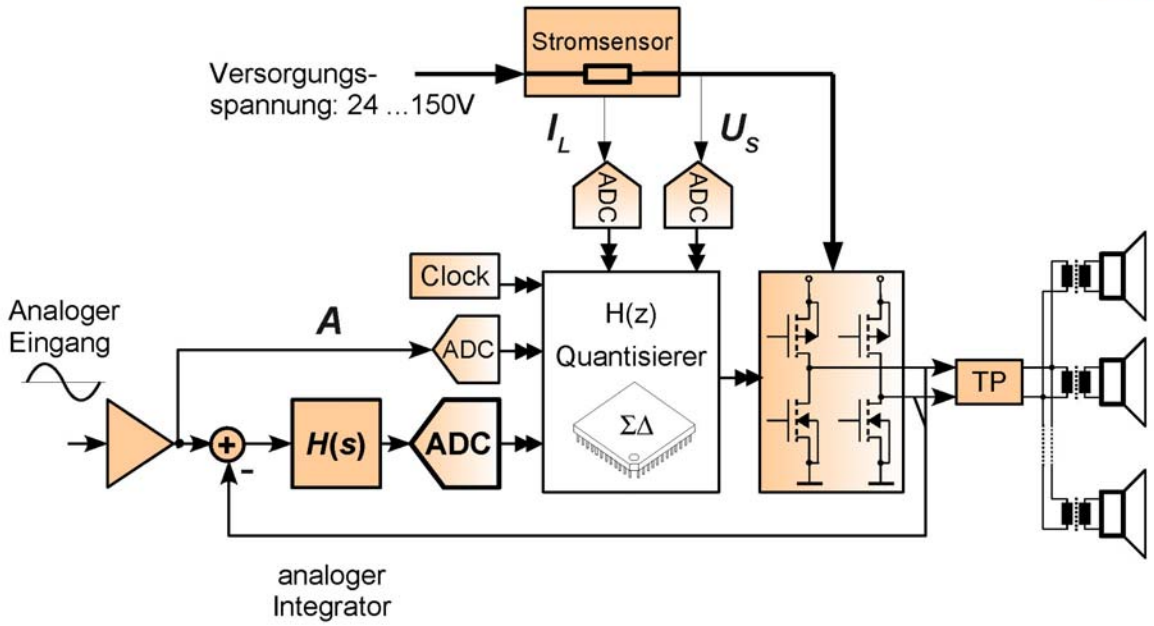
ΣΔ – Class-D Verstärker mit analogem Schleifenfilter



ΣΔ – Class-D Verstärker mit mixed-signal Schleifenfilter



Analoger Class-D Verstärker



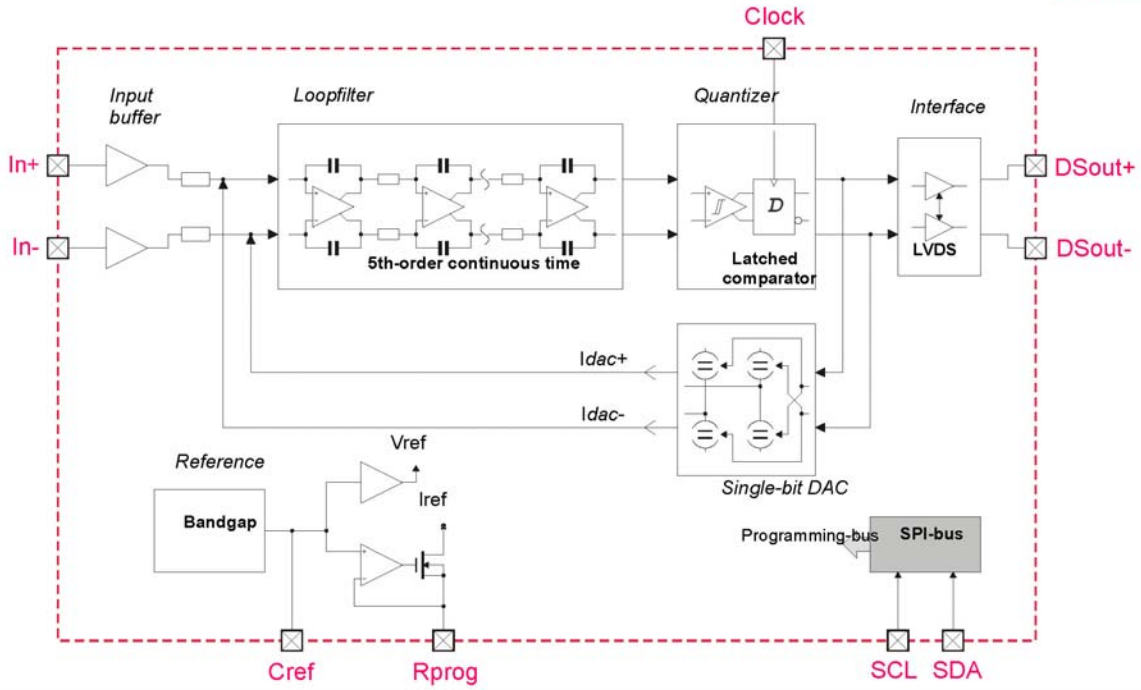
Vergleich analoge / digitale Implementierung

Schleifenfilter	analog / digital	digital
Auflösung Haupt - ADC	gering (10 bit)	**hoch (18 bit)**
Taktrate Haupt - ADC	gering	hoch
Digitale Signalbeeinflussung	kaum möglich	sehr flexibel
Adaptive Systemauslegung	kaum möglich	sehr flexibel

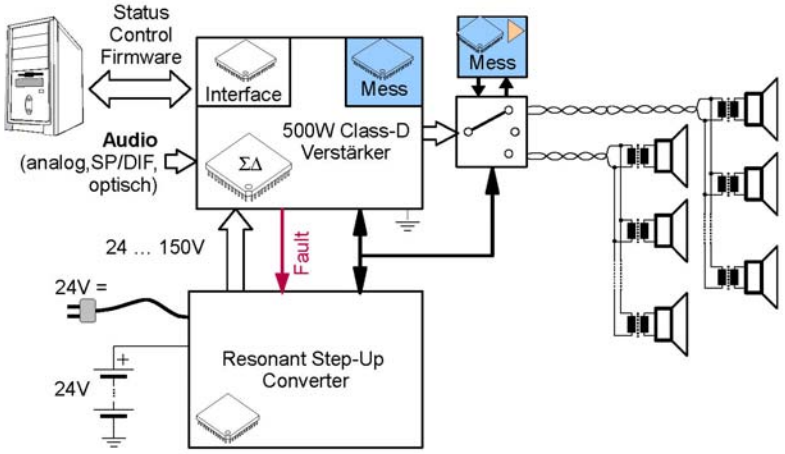
Anforderungen ΣΔ - ADC für *digitalen* Class-D Verstärker

- SNR > 100 dB, THD > 80 dB → 18 b Auflösung
- $f_{S,ADC}$ = 32 MHz (= 10 * $f_{S,Amp}$)
- **geringe Latenzzeit**
- **bezahlbar**

Blockschaltbild CT-ΣΔ ADC

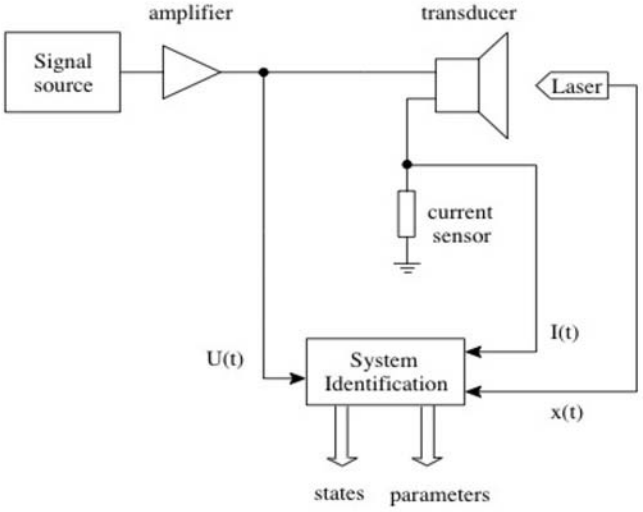


Übersicht



- Resonanzwandler
- Class-D Verstärker
- **Impedanzüberwachung**
- (Interfaces und Bussteuerung)

Lautsprecher-Messtechnik (1)

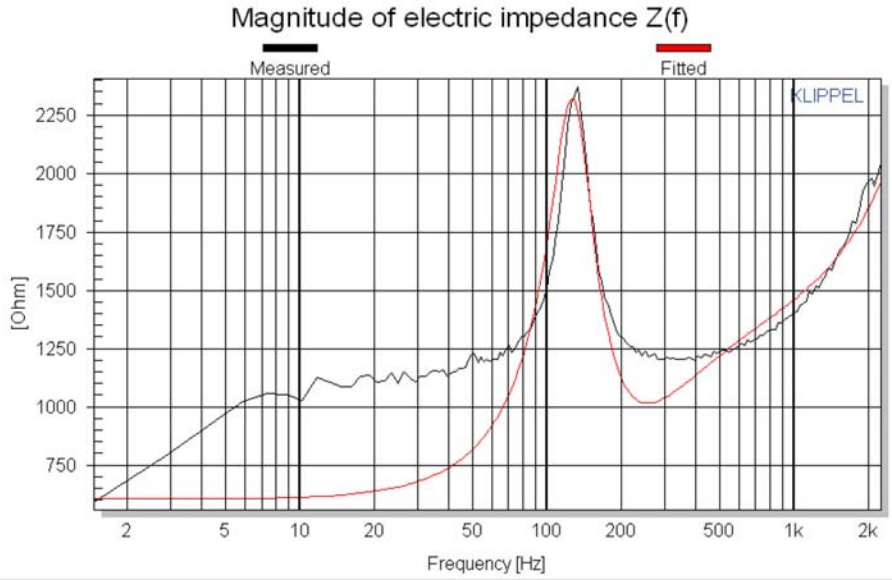


Messsystem der Fa. Klippel

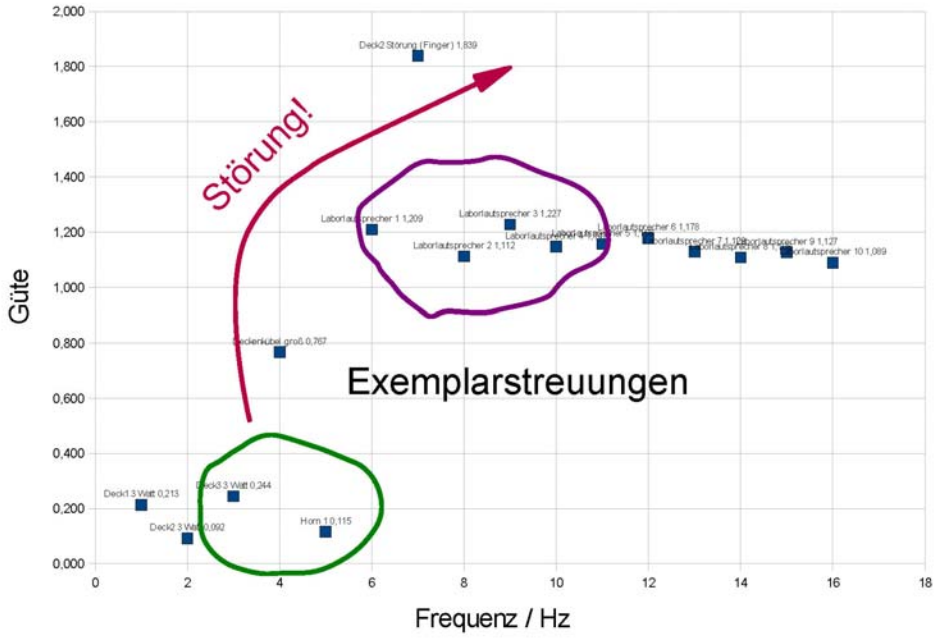
- LS-Impedanz wird ermittelt durch Spannungs- und Strom-Messung an den Klemmen
- Auslenkung der Membran wird mit Laser-Messsystem bestimmt
- Automatische Generierung von Thiele-Small-Parametern

Lautsprecher-Messtechnik (2)

Ziel: Detektion von Ausfällen / Degradation einzelner Lautsprecher einer gesamten Lautsprecherlinie

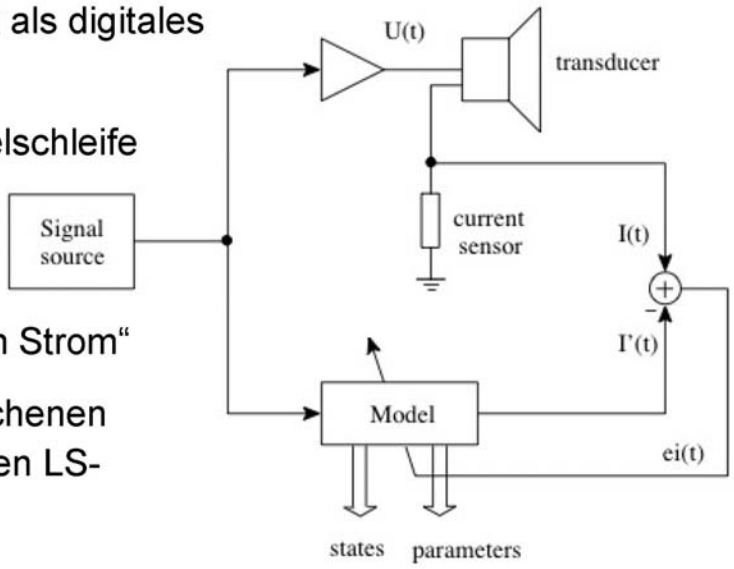


Lautsprecher-Messtechnik (3)



Integrierte Impedanzmessung

- Lautsprecher modelliert als digitales adaptives Filter
- Filter wird in einer Regelschleife abgeglichen auf minimalen Fehler zwischen gemessenem und „simuliertem Strom“
- Parameter des abgeglichenen Filters sind die gesuchten LS-Parameter !



Untersuchungen zur Impedanzmessung

- Typ des adaptiven Filters: FIR / IIR / Bandpass-Resonator
- Abgleichalgorithmus
- Stimulus-Signal (Rauschen / Musik / Multiton …)
- Effiziente Implementierung in FPGA

Zusammenfassung

- Class-D Verstärker bieten aufgrund der hohen Energieeffizienz auch für ELA – Technik entscheidende Vorteile:
  - ➔ Einfachere Klimatechnik und Notstromversorgung
  - ➔ Geringeres Volumen und Gewicht
- Umfassende Systemüberwachung durch Kombination von analoger und digitaler Schaltungstechnik
- Weitgehende Digitalisierung ermöglicht adaptive Signalverarbeitung:
  - ➔ Impedanzmessung mit adaptiven Filtern
  - ➔ ΣΔ-DAC und -ADC mit großem Eingangsbereich
  - ➔ Resonanzwandler mit hoher Effizienz in unterschiedlichen Arbeitspunkten

MULTI PROJEKT CHIP GRUPPE

Hochschule Aalen
Prof. Dr. Bartel, (07361) 576-4182
manfred.bartel@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. Freudenberger, (07531) 206-647
juergen.freudenberger@htwg-konstanz.de

Hochschule Mannheim
Prof. Dr. Paul, (0621) 292-6351
g.paul@hs-mannheim.de

Hochschule Offenburg
Prof. Dr. Jansen, (0781) 205-267
d.jansen@fh-offenburg.de

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

Hochschule Ravensburg-Weingarten
Prof. Dr. Ludescher, (0751) 501-9685
ludescher@hs-weingarten.de

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

Hochschule Ulm
Prof. Dipl.-Phys. Forster, (0731) 50-28180
forster@hs-ulm.de

www.mpc.belwue.de