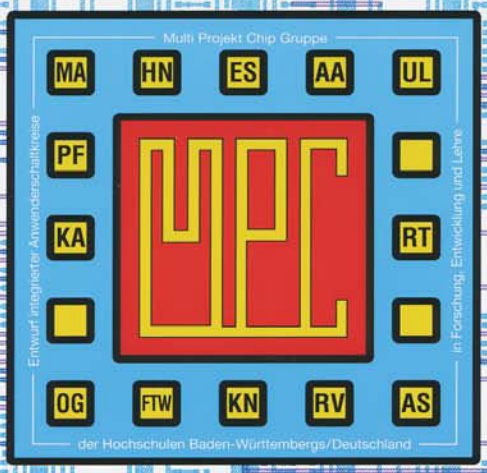
MULTIPROJEKTCHIP GRUPPE

BADEN-WÜRTTEMBERG

MPC-Workshop Februar 2006

Esslingen



MULTIPROJEKTCHIP GRUPPE

BADEN-WÜRTTEMBERG

MPC-Workshop Februar 2006

Esslingen

Cooperating Organization
Solid-State Circuits Society Chapter
IEEE Germany Section

ISSN 1862-7102

Herausgeber: Fachhochschule Ulm

Adressen der

MULTIPROJEKT-CHIP-GRUPPE (MPC-Gruppe)

BADEN - WÜRTTEMBERG

http://www.mpc.belwue.de

Hochschule Aalen
Prof. Dr. Bartel, Postfach 1728, 73428 Aalen
Tel.: 07361/576-107, Fax: -324, Email: manfred.bartel@htw-aalen.de

Hochschule Albstadt-Sigmaringen
Prof. Dr. Rieger, Johannesstr. 3, 72458 Albstadt-Ebingen
Tel.: 07431/579-124, Fax: -149, Email: rieger@hs-albsig.de

Hochschule Esslingen
Prof. Dr. Kampe, Flandernstr. 101, 73732 Esslingen
Tel.: 0711/397-4221, Fax: -4212, Email: gerald.kampe@fht-esslingen.de

Hochschule Furtwangen
Prof. Dr. Rülling, Postfach 28, 78113 Furtwangen
Tel.: 07723/920-503, Fax: -610, Email: rue@hs-furtwangen.de

Hochschule Heilbronn
Prof. Dr. Clauss, Max-Planck-Str. 39, 74081 Heilbronn
Tel.: 07131/504400, Fax: /252470, Email: clauss@hs-heilbronn.de

Hochschule Karlsruhe
Prof. Dr. Koblitz, Postfach 2440, 76012 Karlsruhe
Tel.: 0721/925-2238, Fax: -2259, Email: rudolf.koblitz@hs-karlsruhe.de

Hochschule Konstanz
Prof. Dr. Voland, Brauneggerstraße 55, 78462 Konstanz
Tel.: 07531/206-644, Fax: -559, Email: voland@fh-konstanz.de

Hochschule Mannheim
Prof. Dr. Albert, Speyerer Str. 4, 68136 Mannheim
Tel.: 0621/2926-351, Fax: -454, Email: g.albert@fh-mannheim.de

Hochschule Offenburg
Prof. Dr. Jansen, Badstr. 24, 77652 Offenburg
Tel.: 0781/205-267, Fax: -242, Email: d.jansen@fh-offenburg.de

Hochschule Pforzheim
Prof. Dr. Kesel, Tiefenbronner Str. 65, 75175 Pforzheim
Tel.: 07321/28-6567, Fax: -6060, Email: frank.kesel@fh-pforzheim.de

Hochschule Ravensburg-Weingarten
Prof. Dr. Ludescher, Postfach 1261, 88241 Weingarten
Tel.: 0751/501-9685, Fax: -9876, Email: ludescher@hs-weingarten.de

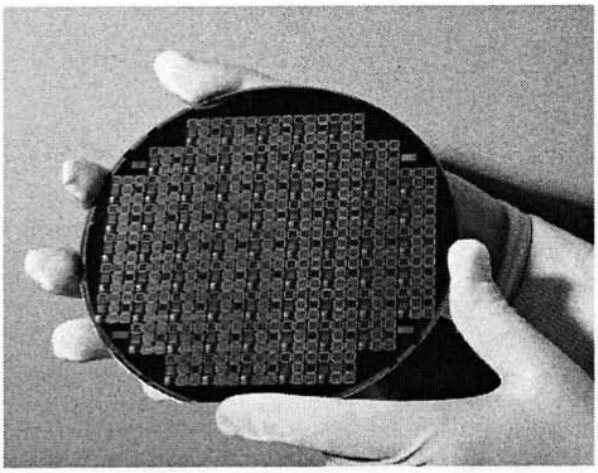
Hochschule Reutlingen
Prof. Dr. Kreutzer, Alteburgstraße 150, 72762 Reutlingen
Tel.: 07121/341-108, Fax: -100, Email: hans.kreutzer@reutlingen-university.de

Hochschule Ulm
Prof. Führer, Postfach 3860, 89028 Ulm
Tel.: 0731/50-28338, Fax: -28363, Email: fuehrer@fh-ulm.de

Inhaltsverzeichnis

Schaltungsentwurf für GATE FOREST – Chips

Prof. Dr.-Ing. Joachim Burghartz, Dipl.-Ing.(FH) Thomas Deuble
Institut für Mikroelektronik Stuttgart, Allmandring 30a, 70569 Stuttgart
Telefon 0 711 / 21 855 -200, Fax -222, E-Mail burghartz@ims-chips.de



Das Institut für Mikroelektronik Stuttgart arbeitet seit langem eng mit der Multiprojekt-Chip-Gruppe Baden-Württemberg zusammen. Eine ganze Reihe MPC-Chips wurden bereits am IMS gefertigt. Dieser Artikel zeigt die aktuellen Möglichkeiten und Neuerungen beim Schaltungsentwurf mit dem IMS GATE FOREST auf.

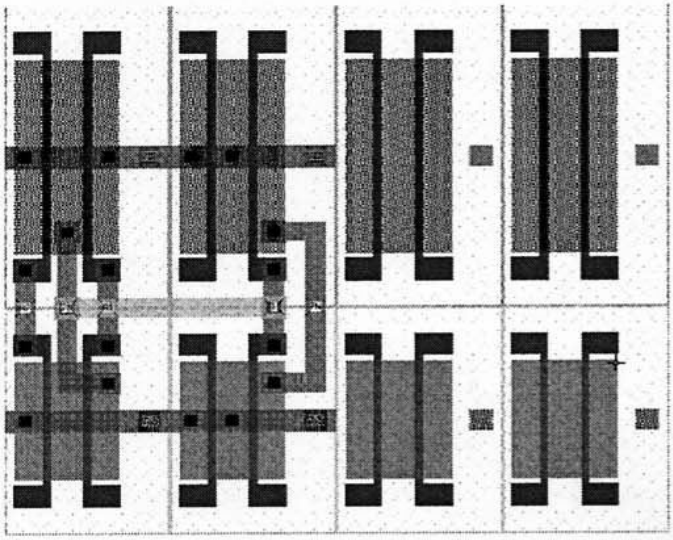
Mit der gänzlich am IMS etablierten 0,8um CMOS-Technologie GFN ("GATE FOREST N") steht für Forschung und industrielle Entwicklung ein stabiler und zuverlässiger Herstellprozess zur Verfügung. Damit lassen sich in kurzer Zeit Chips mit bis zu 100.000 digitalen Gattern in Kombination mit umfangreichen analogen Funktionen herstellen. Der Prozess ist nach mehreren internationalen Normen und Qualitätsstandards zertifiziert und unterliegt strengen Qualitätskontrollen. Das bedeutet, dass jeder vom IMS ausgelieferte Chip Industriequalität aufweist. Dies wissen industrielle und wissenschaftliche Anwender seit Jahren zu schätzen. IMS-Chips finden sich in zahlreichen Produkten und Anwendungen, von der einfachen Infrarot-Fernbedienung bis hin zum Einsatz in Radar-Satelliten zur Erderkundung.

Großen Wert legt das IMS auf eine intensive und persönliche Betreuung von Entwicklungspartnern und Kunden. Satzungsgemäß unterstützt das IMS als wirtschaftsnahes Institut des Landes Baden-Württemberg besonders die kleinen und mittelständischen Unternehmen bei der Bewältigung von mikroelektronischen Herausforderungen. Spezielle und nur für einzelne Unternehmen produzierte ASICs ("anwendungsspezifische integrierte Schaltungen") ermöglichen innovative Problemlösungen auf kleinstem Raum und schützen wertvolles Know-how. Ein stets aktuelles Thema ist der Nachbau von Standard-ICs, die beispielsweise in langlebigen Produkten eingebaut sind. Der internationale Halbleitermarkt entwickelt sich sehr dynamisch und die großen Chiphersteller nehmen viele ihrer Standardprodukte bereits nach wenigen Jahren als "obsolete" wieder vom Markt, was in der Folge zu Versorgungsproblemen für die industriellen Anwender führt. Das IMS kann hier regelmäßig helfen und liefert vollkompatible Ersatzbausteine, die im Kundenauftrag kurzfristig und auch in kleinen Stückzahlen produziert werden können.

1. Das GATE FOREST – Prinzip

Die GATE FOREST – Chips lassen sich rasch und kostengünstig fertigen, weil sie auf vorgefertigten Master-Scheiben basieren. Das bedeutet, dass diese Silizium-Wafer in einem ersten zeitaufwändigen Teilprozess mit einem "Meer" von einzelnen Transistoren versehen werden, was man auch als "Sea-of-Gates" bezeichnet. Diese vorprozessierten Scheiben werden

dann nach Kundenwunsch in einem zweiten Teilprozess "personalisiert", das heißt die vorhandenen Transistoren werden über Leiterbahnen elektrisch so miteinander verbunden, dass die geforderte Schaltungsfunktion entsteht.



Ausschnitt aus einem GFN-Layout: rechte Bildhälfte zeigt vier Transistoren ohne Personalisierung; links sind Transistoren über Leiterbahnen verbunden.

Derzeit stehen acht unterschiedliche Master-Typen zur Verfügung, die sich in Chipfläche, Zahl der realisierbaren analogen und digitalen Funktionen und in der Anzahl der Anschlusspads unterscheiden. Der kleinste Master, GFN001 ist ein rein digitaler Chip mit etwa 300 Gattern und maximal 26 I/O-Pads. Der größte ist der GFN120, mit ca. 100.000 digitalen Gattern und 46 Analogfeldern und über 400 I/O-Pads. Die Analogfelder sind mit optimierten Transistoren, Widerständen und Kapazitäten versehen, um Komparatoren, Operationsverstärker, A/D- und D/A-Wandler sowie Oszillatoren und andere Analogschaltungen realisieren zu können. Eine umfangreiche digitale Zellbibliothek bietet dem Entwickler Zugriff auf Logikgatter und komplexe Funktionen bis hin zum "PIC-kompatiblen" 8-Bit-Mikrocontroller-Kern.

Die technischen Daten der GFN-Chips entsprechen international üblichen Standards bei CMOS-Prozessen. Die typische Betriebsspannung ist 3...5V, die Leistungsaufnahme liegt bei 4 uW/Gatter/MHz. Eingangssignale können bis zu einer Frequenz von 300MHz verarbeitet werden, Ausgangssignale bis zu 145 MHz. Die GFN-Chips des IMS decken alle üblichen Temperaturbereiche ab, einschließlich des erweiterten Einsatzbereichs bei -55° C...+125°C. Das heißt, IMS-Chips können für alle Einsatzbereiche qualifiziert werden. Durch die Zusammenarbeit mit internationalen Zulieferern können die Chips in einer breiten Palette von IC-Gehäusen montiert werden, vom einfachen Dual-Inline-Gehäuse über diverse SMD-Gehäuse bis hin zu ultrakompakten Ball-Grid-Array-Gehäusen, die nur unwesentlich größer sind, als der Chip selbst. Dass die Gehäuse den aktuellen Anforderungen nach Schadstoffarmut (RoHS-Konformität) entsprechen, versteht sich von selbst. Für Kleinserien und Muster können alle Chips in der hauseigenen Chipmontage in verschiedene Keramikgehäuse oder auch direkt auf Leiterplatten montiert werden.

2. Ablauf einer Chipentwicklung

Die Schaltungsentwicklung für GATE FOREST-Chips kann vom Kunden weitgehend selbst durchgeführt werden; das IMS stellt dann entsprechende Zell-Bibliotheken für marktübliche Entwicklungsprogramme zur Verfügung. Es besteht auch die Möglichkeit, lediglich die Anforderungen an den Chip zu definieren und die gesamte Entwicklungsarbeit dem erfahrenen IMS-Team zu überlassen. Darüber hinaus gibt es auch die Option eines Einstiegs auf "halber Strecke", etwa, wenn der Kunde seine Schaltung auf der Basis eines feldprogrammierbaren Bausteins bereits erfolgreich selbst entwickelt hat und nun eine Umsetzung dieser Schaltung in einen ASIC wünscht. Die Entwicklung läuft immer strukturiert ab, jeder Arbeitsschritt wird dokumentiert und gegenüber dem Kunden transparent gemacht. Gemeinsame Freigabeprozeduren zwischen IMS und Kunden an allen entscheidenden Stellen im Entwurfsablauf stellen sicher, dass die Schaltung exakt den Kundenanforderungen entspricht. Grundsätzlich wird für jede Chipentwicklung am IMS ein interner Produktleiter benannt, der das gesamte Chip-Projekt für einen Kunden vom ersten Gespräch bis zur Serienfertigung begleitet.

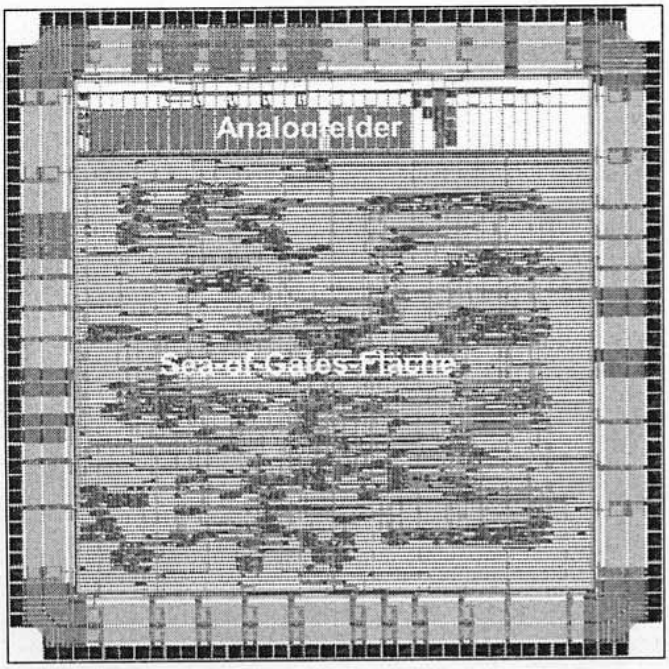
2.1. Von der Idee zur Spezifikation

Gemeinsam mit dem Kunden erstellt das IMS-Team eine Spezifikation, die die Schaltung des Chips sowie alle Randbedingungen (Chipgehäuse, Testmethode, Prüfstandards usw.) definiert. Eine unverbindliche und ausführliche Beratung des Kunden ermöglicht im Vorfeld die Klärung der Realisierbarkeit einer Schaltungsidee. Auf Wunsch wird die Machbarkeit in einer entsprechenden Studie dokumentiert. Sobald die Spezifikation festgelegt wurde, beginnt die eigentliche Entwurfsphase.

2.2. Der Schaltungsentwurf

Digitale Schaltungsteile werden vorzugsweise in der Hardwarebeschreibungssprache VHDL beschrieben, simuliert und synthetisiert. Alternativ zu VHDL kann auch mit der Sprache Verilog gearbeitet werden. Die Synthese bezeichnet die automatische Umsetzung einer Schaltungsbeschreibung in eine ausgewählte Zieltechnologie, zum Beispiel in die des IMS GATE FOREST. Für analoge Komponenten kann auf die

umfangreiche Modulbibliothek des IMS zurückgegriffen werden, GATE FOREST Chips haben spezielle Flächen für Analogschaltungen, um eine hochgenaue Verarbeitung analoger Signale neben digitalen Schaltungen zu ermöglichen. Auch die analogen Komponenten werden vollständig simuliert, bevor sie gemeinsam mit dem eventuellen Digitalteil im Layout des Chips realisiert werden.



Typisches Layout eines GFN-Chips mit Analogfeldern im oberen Bereich und darunter der Sea-of-Gates-Fläche für Digitalschaltungen.

Parallel zur Erstellung des Chiplayouts werden umfangreiche Tests vorbereitet, die später den produzierten Chip auf Herz und Nieren prüfen, die Einhaltung der Spezifikation nachweisen und fehlerfreie Chip-Produkte garantieren. Für den Chip-Test werden so genannte Testmuster erzeugt, die den Chip in allen denkbaren Betriebszuständen aktivieren und die korrekte Funktion überprüfen. Diese Testmuster werden einerseits mit Hilfe der Synthesesoftware automatisch generiert, andererseits aber auch vom Schaltungsentwickler in Absprache mit dem Kunden als "Testbench" in VHDL beschrieben, um die spezifizierten Betriebszustände des Chips direkt testen zu können. Hierzu wird eine Testspezifikation erstellt, in der alle notwendigen "Test Cases" vereinbart werden.

Am Ende der Entwurfsphase erhält der Kunde den IMS-Designreport, in dem alle Entwurfsschritte nachvollziehbar und verständlich dokumentiert werden. Mit der Annahme des Designreports ist die Freigabe des Chip-Entwurfs durch den Kunden verbunden und stellt gleichzeitig den Startpunkt für die Herstellung des Chips dar.

2.3. Die Chipproduktion

Das freigegebene Chiplayout ist die Basis für die automatische Erstellung der Fertigungsdaten. Mit den Fertigungsdaten wird dann einer der derzeit modernsten Elektronenstrahl-Schreiber, ein Leica SB350, im Reinraum des IMS angesteuert. Der Elektronenstrahl-Schreiber kann zur Anfertigung von Belichtungsmasken für die Waferfertigung genutzt werden.

Mit den Masken werden in weiteren Prozessschritten die Metall-Ebenen auf den Wafern hergestellt. Für Prototypen oder Kleinserien kann der Elektronenstrahl-Schreiber aber auch zur "Direktbelichtung" eines einzelnen Wafers verwendet werden. In diesem Fall werden die Strukturen der Metallebenen des Wafers direkt mit dem Elektronenstrahlschreiber belichtet. Man spart sich dann den zeit- und kostenaufwändigeren Weg über die Maskenherstellung.

Grundsätzlich ist die Maskenbelichtung jedoch bei der Herstellung mehrerer Wafer mit gleichen Chips ökonomisch sinnvoller, da die Fertigungszeiten bei Maskenbelichtung wesentlich kürzer sind, sobald die wieder verwendbaren Masken einmal hergestellt wurden. Die GATE FOREST Chips werden vollständig am IMS produziert. Durch die Zusammenarbeit mit dem Freiburger Chiphersteller MICRONAS kann das IMS eine zum IMS-Prozess kompatible Second Source anbieten, die die rasche Fertigung großer Produktionsvolumen erlaubt und die ohnehin hohe Verfügbarkeit der IMS-Chips auch bei eventuellen Produktionsstörungen sicherstellt.

2.4. Test, Gehäuseaufbau, Qualifizierung und Auslieferung

Sobald die Wafer mit den GATE FOREST Chips aus der Produktion kommen, werden sie von der IMS-Testabteilung mit den im Schaltungsentwurf erstellten Testmustern geprüft. Diese Wafer mit geprüften Chips kommen dann in die Chipmontage. Für die Bemusterung oder Klein- und Sonderserien werden die Chips im IMS unter Reinraumbedingungen in die gewünschten Keramikgehäuse verpackt. Serienprodukte mit Kunststoffgehäusen lässt das IMS bei qualifizierten Zulieferern in Europa und Asien verpacken.

Sind die verpackten Chips aus der Montage zurückgekehrt, werden sie erneut, in einem Bauelemente-Test, geprüft, um eventuelle Fehler bei der Chipmontage zuverlässig ausschließen zu können. Sobald erste Muster zur Verfügung stehen, bekommt der Kunde "seinen" ASIC erstmals in die Hand und kann ihn in der vorgesehenen Anwendung testen.

Parallel dazu führt die IMS-interne Qualitätssicherung laufend Untersuchungen über Lebensdauer und Zuverlässigkeit der produzierten Chips durch. Bei diesen "Torturen" werden die Bausteine unter extremen Be-

dingungen an die Grenzen in der Anwendung gebracht, um nachzuweisen, dass sie die strengen Qualitätsanforderungen des IMS erfüllen. Neben elektrischen Tests werden zyklische Temperaturwechsel, Feuchtigkeitstests und mechanische Untersuchungen durchgeführt. Diese und andere etablierte Qualitätsmaßnahmen spiegeln sich auch in den einschlägigen Standards wieder, nach denen die Entwicklung und Herstellung von IMS-Chips zertifiziert ist: ISO9001 und DIN EN 100 114, letztere qualifiziert das IMS als "echten" Chiphersteller in Industriequalität.

3. Innovative Ideen umsetzen

IMS GATE FOREST Chips sind eine ideale Plattform, um rasch und kostengünstig eigene Schaltungsideen zu realisieren. Ganz gleich, ob es um einige wenige Chips zu Forschungszwecken geht oder, ob eine konkrete Produktidee dahinter steht und der Chip das Potenzial zu einem Serienprodukt besitzt.

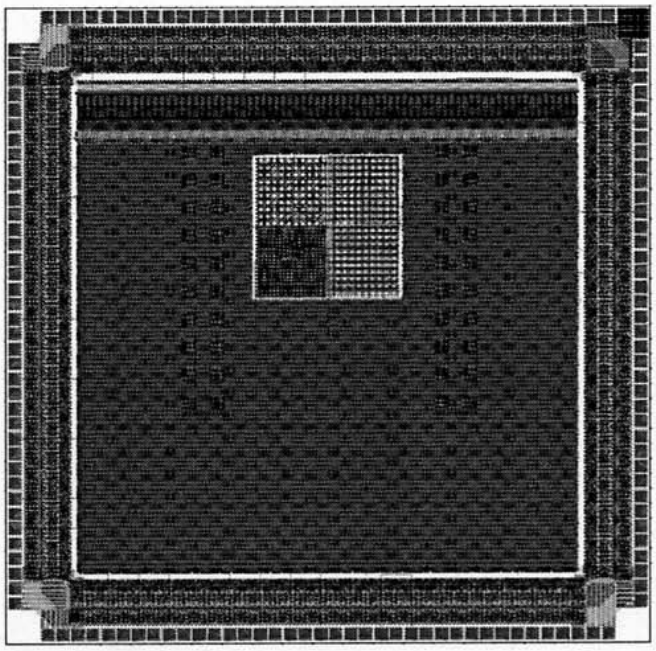
In vielen Unternehmen existieren viele innovative Ideen für neue Produkte. Oft scheitert die Realisierung an Hemmnissen wie hohen Entwicklungskosten für eigene Mikrochips oder einfach am fehlenden Knowhow, speziell bei der Anwendung neuer Nano- und Mikrotechnologien. Das IMS sieht es als seine Aufgabe an, Unternehmen bei der Umsetzung neuer Ideen zu unterstützen. Entwicklungsmitarbeiter von kleinen und mittleren Unternehmen können am IMS den so genannten "Walk-in-Service" nutzen und am IMS begleitet vom IMS-Team eigene Schaltungsentwürfe realisieren.

3.1. IC-Lab Entwicklungsumgebung

Aus dem Gebiet der beruflichen Weiterbildung stammt das neue IC-Lab-Paket des IMS. Damit steht ein flexibles Starter-Kit zur Entwicklung integrierter Schaltungen mit modernsten Methoden kostengünstig zur Verfügung. Mit dem IC-Lab können komplexe Schaltungen beschrieben, simuliert und auf einem mehrfach programmierbaren FPGA-Baustein erprobt werden. Ist die Schaltungsidee dann soweit gereift, dass sie auf einem eigenen Mikrochip realisiert werden kann, steht die IMS GATE FOREST Chiptechnologie zur Verfügung und die FPGA-Schaltung kann in kurzer Zeit in ein ASIC-Design überführt werden.

3.2. On-chip Sensorarrays

Die GATE FOREST-Strategie mit ihren einheitlich vorgefertigten Master-Wafern und der rasch und günstig durchführbaren Personalisierung hin zu kundenspezischen Chips ist eine etablierte und erfolgreiche Sparte im Angebot des IMS für Industrie und Forschung. Im Zuge der Weiterentwicklung des GATE FOREST zu einem 0,5µm-Prozeß entsteht eine "Structured ASIC"-Plattform, welche bei gleicher Chipgröße neben wesentlich erhöhter Gatterkapazität auch dedizierte Bereiche für Sensorarrays bietet. Auf diesen Sensorarrays lassen sich zum Beispiel Detektoren für Licht und Magnetfelder direkt im Chip neben Schaltungslogik und analoger Signalaufbereitung integrieren – kompakte Mikrosysteme für industrielle Anwendungen.



Die Abbildung zeigt integrierte pn-Photodioden in CMOS-Technologie auf dem neuen 0,5µm GATE FOREST – Master.

4. Integrierte Mikrosysteme

4.1. Nanoschalter

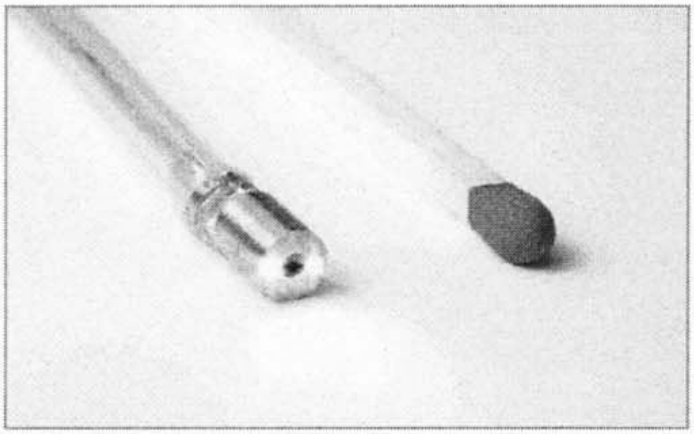
Durch die enge Zusammenarbeit mit einigen jungen Unternehmen aus der Nano- und Mikromechanik rücken kleinste in Silizium gefertigte mechanische Komponenten verstärkt in den Fokus industrieller Anwendungen beim IMS. So ist das Institut beispielsweise an der Entwicklung mikromechanischer Schaltelemente beteiligt, die ihren Zustand stromlos speichern, jedoch durch Anlegen einer Steuerspannung "umprogrammiert" werden können. Diese im Grunde rein mechanischen Schalter sind so klein, dass sie als Arrays auf Chips integriert werden können und wie nichtflüchtige Datenspeicher einsetzbar sind. Untersuchungen zeigen, dass diese mikromechanischen Speicher erheblich robuster gegenüber externen Einflüssen sind, als zum Beispiel Speicherelemente auf der Basis von Flash-Technologien.

In Verbindung mit der bei GATE FOREST Chips realisierbaren Robustheit gegen Strahlung und elektro-

magnetischen Feldern eröffnen sich völlig neue Anwendungen für solche robusten Mikrosysteme. Dies gilt insbesondere bei Anwendungen unter extremen Bedingungen wie etwa in der Raumfahrt.

4.2. Welt-kleinste Endoskopkamera

Nicht nur im High-End-Bereich sind solche Mikrosysteme interessant. Gerade auch bei Anwendungen, die bisher mit sehr teuren Spezialgeräten realisiert wurden, lassen sich Mikrosysteme schon sehr bald als preiswerte Alternative realisieren. So hat das IMS-Team im Rahmen eines Verbundforschungsprojektes die derzeit kleinste Endoskopkamera vorgestellt, die künftig als "bildgebendes Einweg-Endoskop" Medizinern völlig neue Diagnosemöglichkeiten geben könnte. Auf einem Chip mit nur 1.8 mm Durchmesser wurde ein Videosensor mit 40.000 Pixeln verwirklicht. Der Chip wurde komplett am IMS entwickelt und in einer externen Fertigungsstätte in einem 0,25µm-CMOS-Prozess hergestellt.



Die IVP1-Endoskopkamera im Vergleich mit einem Streichholz. Der Endoskopkopf enthält einen IMS HDRC-Farbsensor und zwei Lichtquellen.

Für praktische Versuche wurde der Mini-Bildsensor auf einer 2x3 mm^2 kleinen Platine direkt montiert und auf der Rückseite der Anschluss der Versorgungs- und Datenleitungen realisiert. Weitere Forschungen im Bereich Aufbau- und Verbindungstechnik laufen am IMS, um kostengünstige und zuverlässige Methoden der Verbindung solcher Winzlinge mit der Außenwelt weiterzuentwickeln.

4.3. Drahtlose Mikrosysteme

Neben drahtgebundenen Mikrosystemen drängen sich immer mehr die drahtlosen Systeme in den Vordergrund. Ein bekanntes Beispiel sind die schlagzeilenträchtigen RFID-Tags, die als drahtlos auslesbare Identifikationsmöglichkeit schon sehr bald viele Logistikaufgaben in Handel und Industrie dominieren werden. Auch komplexere drahtlose Mikrosysteme werden realisierbar: zum Beispiel drahtlos ansprechbare Sensoren, die in Autoreifen laufend den Druck messen und bei gefährlichem Druckabfall eine Alarmmeldung erzeugen. Ebenso sind Sensoren denkbar, die in Bauteilen von Gebäuden fest eingebaut sind und laufend Druck, Temperatur und Kräfte messen, die aus der Distanz drahtlos erfassbar sind.

Zusammenfassung

Das Institut für Mikroelektronik Stuttgart versteht sich als Partner für kleine und mittelständische Anwender bei der Umsetzung von Schaltungskonzepten von der Idee bis zum Chip in Industriequalität. Über 300 Kunden mit teils langjährigem Kontakt zum IMS wissen das zu schätzen. Die qualifizierten hauseigenen CMOS-Prozesse bieten eine breite Basis für ASICs und integrierte Mikrosysteme in den Bereichen Digitaltechnik, Mixed-Signal und hochwertigen Sensoranwendungen. Durch die enge Kooperation mit anderen Foundries wie Micronas stehen sowohl Second Source-Fertigungslinien als auch Sonderprozesse für Bildsensoren und Smartpoweranwendungen zur Verfügung.

Ausführliche Informationen sind im Internet zu finden.

www.ims-chips.de

Über die Autoren

Prof. Dr.-Ing. Joachim Burghartz leitet seit Oktober 2005 das Institut für Mikroelektronik Stuttgart (IMS CHIPS). Davor war er sieben Jahre lang an der TU Delft tätig, wobei er von 2001 bis 2005 Wissenschaftlicher Direktor des Forschungsinstituts DIMES war. Dem ging eine elfjährige Tätigkeit bei IBM am T. J. Watson Research Center in New York in verschiedenen Funktionen voraus. Er promovierte 1987 zum Dr.-Ing. an der Universität Stuttgart und erhielt den Dipl.Ing. Grad 1982 von der RWTH Aachen.

Dipl.-Ing.(FH) Thomas Deuble studierte an der FH Ravensburg-Weingarten und ist seit 1997 Produktmanager ASICs im Geschäftsfeld Systeme am IMS und betreut Kundenprojekte sowie Förderprojekte im Bereich Chipentwicklung.

Simulation und Layout eines schnellen CMOS-Serializers

Christian Bradler, Prof. Dr.-Ing. H.-P. Bürkle

Hochschule Aalen, Beethovenstr. 1, 73430 Aalen

Tel. 07361 / 576 - 240, Fax 07361 / 576 - 324

christian.bradler@gmx.de, heinz-peter.buerkle@htw-aalen.de

Im Rahmen eines Projekts zur Entwicklung von LVDS-Zellen wurde ein schneller CMOS-Serializer bis zum Chip-Layout entwickelt. Dazu wurde zunächst eine eigene Bibliothek von Standard-zellen erstellt und dann im Serializer verwendet. Langsame stromsparende Zellen und schnelle Zellen ermöglichen eine gute Balance zwischen Geschwindigkeit und Leistungsverbrauch. Inhalt der hier vorgestellten Arbeit ist die systematische Darstellung der Layoutentwicklung und eine Systemsimulation mit innerhalb des Projekts bereits erstellten Zellen.

1. Einleitung

Die vorliegende Arbeit über einen schnellen Serializer entstand im EDA-Zentrum der Hochschule Aalen im Zusammenhang eines Testchipprojektes, das sich mit dem Entwurf von LVDS-Zellen für die Technologie von IHP Microelectronics [1] befasst. (IHP: Innovation for High Performance, Frankfurt/Oder). LVDS steht für Low Voltage Differential Signaling und ist ein digitaler Schnittstellenstandard, der lediglich auf der physikalischen Schicht spezifiziert ist [2]. Niedrige Spannungshübe (maximal 400mV) sorgen für geringen Leistungsverbrauch und hohe Bitraten, der Gegentaktbetrieb (Differential Mode) bewirkt eine minimale Störaussendung und geringen Einfluss von (Gleichtakt)-Störungen auf das LVDS-Signal.

Anwendung findet LVDS heute zunehmend für schnelle serielle Übertragung auf der Leiterplatte, da parallele Bussysteme aufgrund der endlichen Ausbreitungsgeschwindigkeit der Signale (etwa 20cm/ns) schon heute im Bereich mehrerer GBit/s oft an ihre Grenzen stoßen (Laufzeitversatz aufgrund der unterschiedlichen Leitungslängen, Reflexionen und Überkoppeln zwischen eng benachbarten Leitungen). Aktuelle Einsatzgebiete für LVDS sind PCI-Express, DVI, RapidIO, FireWire, SCSI und S-ATA. Darüber hinaus denkt man auch daran LVDS zur Kommunikation zwischen Subsystemen On-Chip zu verwenden. Daher spielt LVDS auch für die System-On-Chip (SoC) Integration eine große Rolle.

1.1. Gesamtprojekt

Abbildung 1 zeigt das Gesamtprojekt. Neben LVDS-Leitungstreiber (der aus einer Weiterentwicklung von [3] entstand) und Empfängereingangsstufe [4] sind der Parallel-Seriell-Wandler („Serializer“) und der Seriell-Parallel-Wandler („Deserializer“) besonders wichtig. Daneben werden noch Schaltungen zur Erzeugung der Takte benötigt (Phasenregelschleife PLL mit Frequenzteiler). Im Empfänger muss zudem eine Abtastung des Signals zum Zeitpunkt mit maximaler Augenöffnung (Retiming) und eine Synchronisation zum Sender vorgenommen werden, damit die einzelnen Signale nach dem Deserializer wieder auf den richten Leitungen erscheinen.

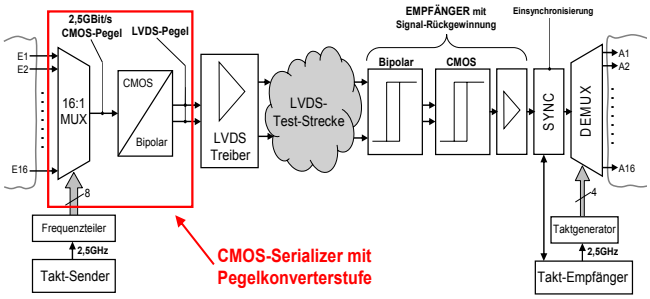


Abbildung 1 Gesamtsystem zur LVDS-Übertragung

Der aktuelle Stand der Entwicklung des Gesamt-projekts umfasst Leitungstreiber, Empfänger, Deserializer und erste Blöcke zur Erzeugung der notwendigen Takte. Als Ziel für die Bitrate werden 2.5GBit/s angestrebt.

1.2. Technologie

Als konkrete Technologie liegt ein Prozess von IHP Microelectronics vor [1]. Es handelt sich hierbei um einen 0.25µm SiGe:C-HBT-BiCMOS Mixed-Signal Prozess. Zur Erfüllung höchster Anforderungen bezüglich Geschwindigkeit und Rauschen können auch Silizium-Germanium-Hetero-Bipolartransistoren verwendet werden. Aufgrund der Minimierung der Chipfläche und der elektrischen Leistungsaufnahme wird man jedoch bestrebt sein, die CMOS Technologie so weit als irgend möglich einzusetzen. Daher finden im hier behandelten Serializer, anders

als in Leitungstreiber und Empfängereingangsstufe, keine SiGe-HBTs Verwendung.

2. Entwurf des Serializers

Zielvorgabe für den Schaltungsentwurf ist ein 16:1 Serializer, der eine Bitrate von mindestens 2.5GBit/s am Ausgang aufweist.

2.1. Schaltungskonzept

Die Schaltungsidee basiert auf einer geradlinigen hierarchischen Struktur aus elementaren 2:1 Multiplexern mit vier Taktbereichen (Abbildung 2). Hierfür lässt sich ein besonders einfacher Taktgenerator mittels eines Frequenzteilers realisieren. Der Serializer enthält damit sowohl langsame als auch schnell getaktete Schaltungsteile. Damit die Schaltung stromsparend ausgelegt werden kann, ist es ratsam einen langsamen MUX am Eingang und einen schnellen MUX am Ausgang einzusetzen. Da bei den 2:1 Multiplexern sowohl der jeweilige originale Takt als auch der invertierte Takt genutzt werden, reicht als Maximaltakt der 8-fache Basistakt aus, sofern man am Ausgang der letzten Stufe das Signal nicht abtasten muss.

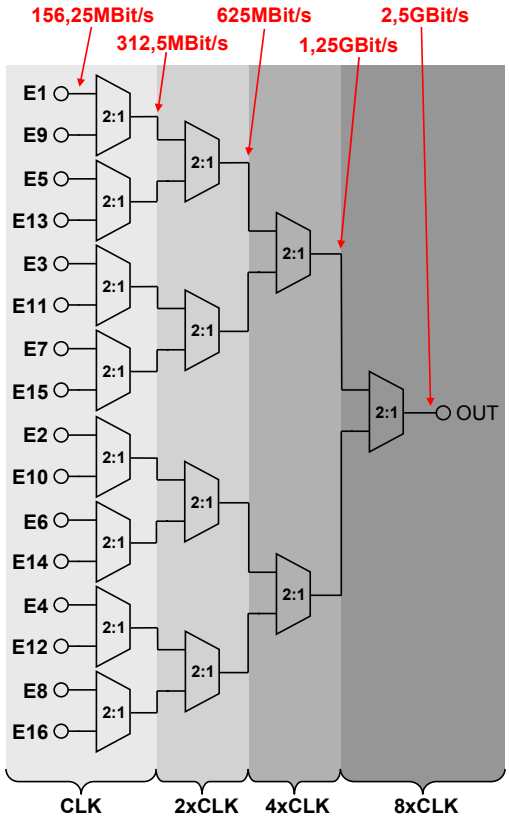


Abbildung 2 Serializer Schaltungskonzept

2.2. Synchronisation der Multiplexerstufen

Aufgrund von Laufzeiten müssen die Multiplexerstufen synchronisiert werden. Andernfalls kommt es zu Laufzeitkonflikten, wie nachfolgend dargestellt: Abbildung 3 zeigt ein Modell, das die Laufzeiten der Multiplexerstufen berücksichtigt. Der nachgeschaltete MUX3 erwartet das Signal X1 zu einem Zeitpunkt, bei dem noch kein neues Datum aufgrund der realen Verzögerungszeit des davor geschalteten MUX1 anliegen kann. Daraus ergeben sich die in Abbildung 4 dargestellten Laufzeitkonflikte.

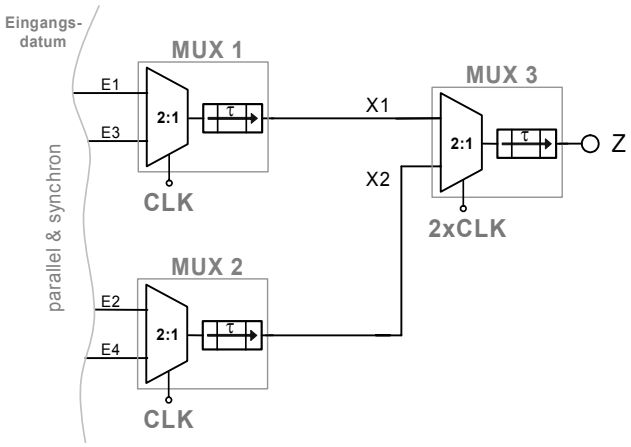


Abbildung 3 Laufzeitmodell einer Teilschaltung

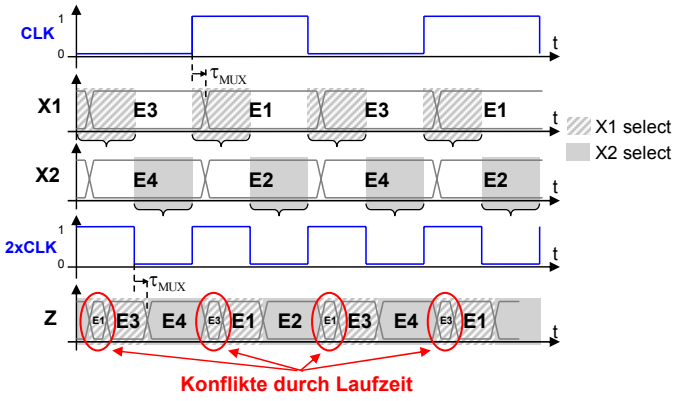


Abbildung 4 Impulsdiagramm mit Darstellung der Laufzeitkonflikte

Um dieses Problem zu lösen, müssen die Eingangssignale am MUX3 rechtzeitig bereitgestellt werden. Abhilfe kann durch folgende Schaltungsmaßnahme geschaffen werden (Abbildung 5). Nach jeder Multiplexerstufe werden taktzustandsgesteuerte Doppelspeicherflipflops (DFF) eingesetzt, die mit der doppelten Taktfrequenz des davor geschalteten Multiplexers angesteuert werden. Die Signale X1.0 und X2.0 werden darauf eindeutig in einem gültigen Zustand abgetastet. Im unten dargestellten Signalpfad sorgt ein D-Latch dafür, dass das Signal um eine halbe Taktperiode des schnellen Taktes (bzw. um ein

Viertel der Basistaktperiode) verzögert wird. Somit steht am Eingang des nachfolgenden MUX3 abwechselnd auf den beiden Zweigen jeweils ein stabiles Signal zur Verfügung, das an den Ausgang Z weitergereicht werden kann. Das Impulsdiagramm hierzu zeigt Abbildung 6.

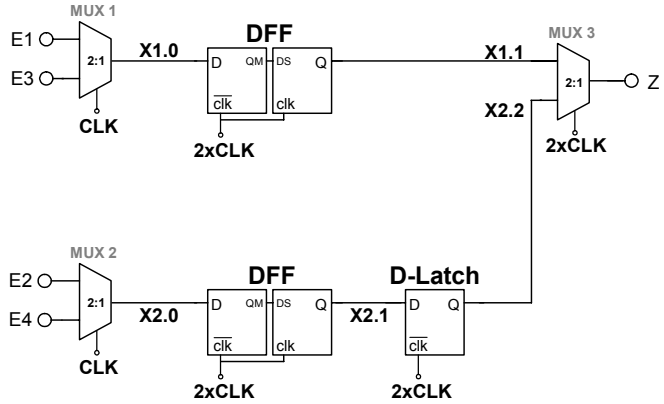


Abbildung 5 Schaltungsmodell zur Lösung der Laufzeitkonflikte

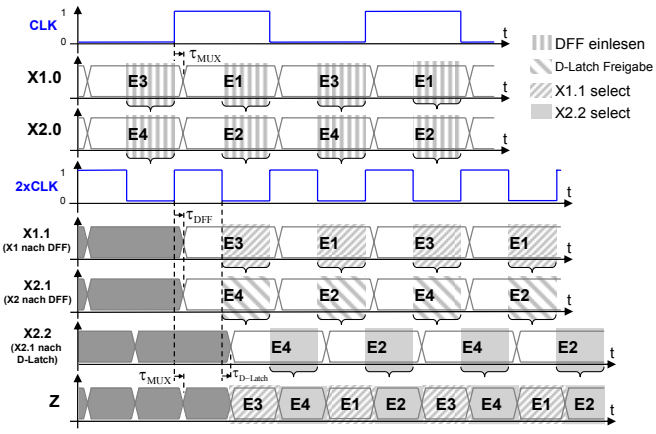


Abbildung 6 Impulsdiagramm der synchronisierten Multiplexerschaltung

3. Entwurf von Standardzellen

Die benötigten Zellen zum Aufbau des Multiplexers wurden als Standardzellen entworfen. Die im Design-Kit vorhandene Standardzellen-Bibliothek kann keiner Extraktion parasitärer Komponenten unterzogen werden, da die Layouts zum Schutz von Intellectual Property lediglich als so genannte „Abstracted View“ zur Verfügung stehen. Um eine möglichst exakte Simulation auch der parasitären Effekte nach dem Layout zu erhalten, wurden die für den Multiplexer relevanten Zellen neu entworfen. Dies hat zudem den Vorteil, dass die Zellen bezüglich Geschwindigkeit und Stromverbrauch für diese spezielle Applikation optimiert werden konnten.

3.1. Schaltplan und Layout der Standardzellen

Die folgenden Schaltungen sind für den Aufbau eines 16:1-MUX notwendig: 2:1-Multiplexer-Zellen, D-Latch Speicherzellen und Doppelspeicher-Flip-Flop-Zellen (DFF). Da die Geschwindigkeitsanforderungen je nach Multiplexerstufe (Anordnung am Eingang oder Ausgang) recht unterschiedlich sind, wurden sowohl langsame und somit stromsparende Standardzellen, sowie schnelle Varianten entworfen.

Den prinzipiellen Layoutaufbau der entworfenen Standardzellen und deren Anordnung in einem Array zeigt Abbildung 7. Die Zellen werden an der Horizontalen gespiegelt. Dadurch können die Stromversorgungspfade besonders platzsparend ausgeführt werden. Die Taktsignale werden über Takt-leitungspfade herangeführt.

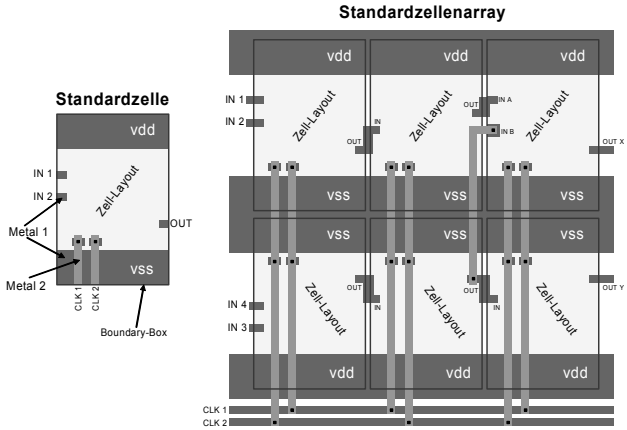


Abbildung 7 Schematische Darstellung der Standardzellen und Standardzellenarray

Es erfolgt nun eine kurze Beschreibung der entwickelten Standardzellen: Der elementare Multiplexer (hier in Abbildung 8 als schnelle Variante) besteht aus jeweils einem Transmission-Gate an jedem Eingang, die im Gegentakt durchgeschaltet werden. Anschließend übergibt ein einfacher CMOS-Inverter einem nachgeschalteten schnellen Inverter das gemultiplexte Signal.

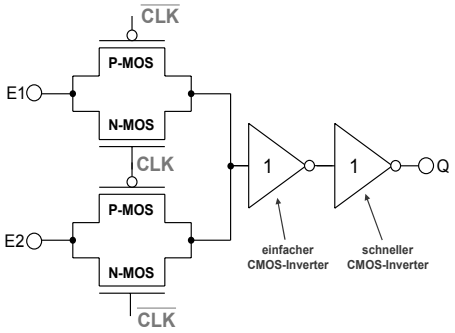


Abbildung 8 Schaltplan schnellen Multiplexerzelle

Die Umsetzung dieser Schaltung in ein Layout zeigt Abbildung 9. Dieses Layout wurde einer Extraktion parasitärer Komponenten unterzogen und neu simuliert. Den transienten Spannungsverläufen wurden die charakteristischen Laufzeiten sowie die Anstiegs- und Abfallzeiten entnommen und vergleichend für Schematic- und Analog-Extracted-Simulation in die Tabelle (Abbildung 10) eingetragen. Man erkennt einen geringfügigen Einfluss des Layouts.

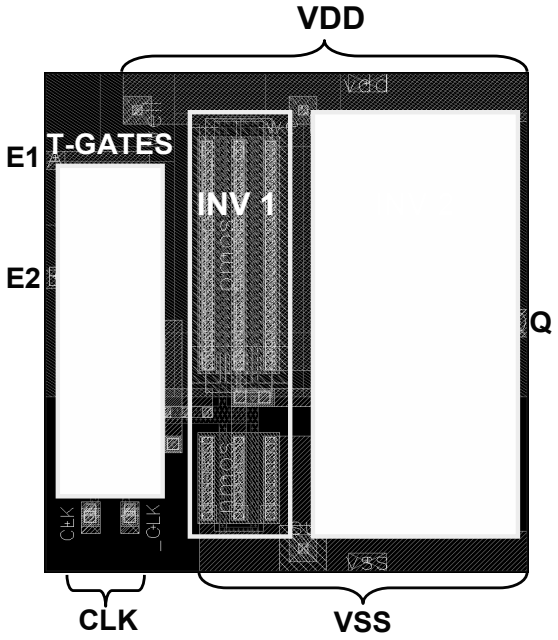


Abbildung 9 Layout der schnellen Multiplexerzelle

Messwert	Schematic-Simulation		Analog-Extracted-Simulation	
Eingang A (Signallaufzeit)	$T_{01} = 189ps$	$T_{10} = 184ps$	$T_{01} = 202ps$	$T_{10} = 198ps$
Eingang B (Signallaufzeit)	$T_{01} = 189ps$	$T_{10} = 184ps$	$T_{01} = 204ps$	$T_{10} = 198ps$
Clock-To-Output-Delay (T_{co})	$T_{co01} = 201ps$	$T_{co10} = 196ps$	$T_{co01} = 219ps$	$T_{co10} = 214ps$
Anstiegszeit (T_{rise})	$T_{rise} = 62ps$		$T_{rise} = 71ps$	
Abfallzeit (T_{fall})	$T_{fall} = 52ps$		$T_{fall} = 60ps$	
Zusammenfassung: $\tau_{MUX} = 220ps$ (Maximale Zeit T_{COmax})				

Abbildung 10 Charakteristische Zeiten für Signale an der schnellen Multiplexerzelle

Die anderen Zellen werden hier nun weniger ausführlich beschrieben. Abbildung 11 und Abbildung 12 zeigen Schaltplan und Layout der schnellen D-Latch-Speicherzelle.

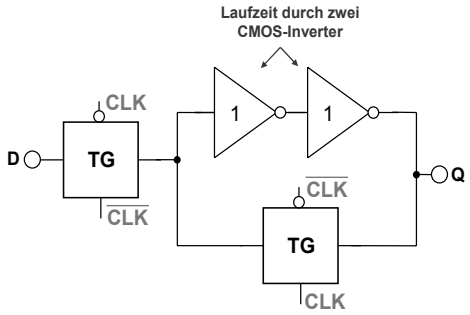


Abbildung 11 Schaltplan schnelles D-Latch

Das D-Latch besteht aus einem Transmission-Gate am Eingang, sowie einem Transmission-Gate im Rückkopplungspfad. Zwei Inverter, die in Serie geschaltet sind, sorgen für einen steilen Pegel am Ausgang sowie für die erforderliche Laufzeit vom Eingang zum Ausgang.

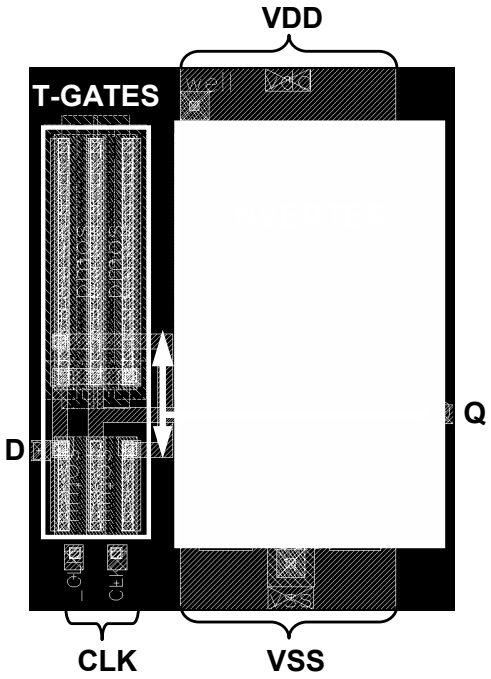


Abbildung 12 Layout des schnellen D-Latch

Das Doppelspeicher-Flipflop wird als Master-Slave-Flipflop aus zwei hintereinander geschalteten D-Latch-Zellen, die zueinander mit invertierten Takten betrieben werden (Abbildung 13, Abbildung 14), realisiert.

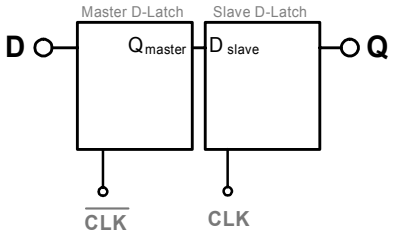


Abbildung 13 Schaltplan des schnellen DFF

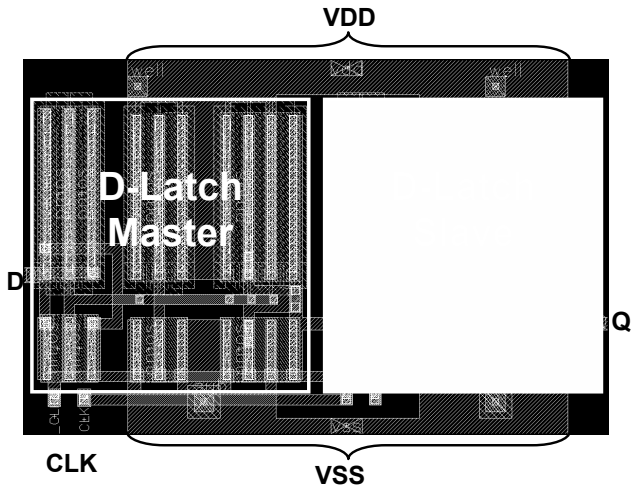


Abbildung 14 Layout des schnellen DFF

3.2. Evaluation der entwickelten Standardzellen

Die Standardzellen wurden zunächst auf Schaltplanebene simuliert und optimiert. Nach der Layouterstellung wurden die parasitären Elemente extrahiert und in eine Post-Layout Simulation eingebunden. Diese Simulationswerte flossen dann in das Laufzeitmodell ein. Daraus lassen sich die folgenden Aussagen für die einzelnen Komponenten hinsichtlich ihrer Geschwindigkeit treffen:

Zeit	Zeit (analog-extracted)	maximale Basisfrequenz	schnellster Takt (8xCLK)	maximale Übertragungsrate
τ_{MUX}	**220 ps**	**284 MHz**	**2,272 GHz**	**4,544 GBit/s**
τ_{Latch}	**205 ps**	**305 MHz**	**2,44 GHz**	**4,88 GBit/s**
τ_{DFF}	**190 ps**	**329 MHz**	**2,632 GHz**	**5,264 GBit/s**
t_{Setup} D-Latch	**162 ps**	**355 MHz**	**2,84 GHz**	**5,68 GBit/s**
t_{Setup} DFF	**150 ps**	**338 MHz**	**2,704 GHz**	**5,408GBit/s**

Abbildung 15 Zusammenstellung der kritischen Zeiten für Multiplexer (MUX), D-Latch und DFF und der dadurch jeweils limitierten Übertragungsrate.

Abbildung 15 zeigt, dass die am stärksten begrenzende Zeit die Laufzeit der Multiplexerzelle ist. Diese erlaubt eine maximale Übertragungsrate von ca. 4.5GBit/s. Da lediglich 2.5GBit/s anvisiert wurden, erfolgten an dieser Stelle keine weiteren Optimierungen mehr.

4. Layoutentwicklung des gesamten Serializers

Insgesamt sieben Basiszellen bilden den 4:1-Multiplexer (in Abbildung 16 dargestellt). Er besteht aus drei elementaren 2:1 Multiplexerzellen mit den jeweils nachgeschalteten Doppelspeicher-Flip-Flops und einem D-Latch am Ausgang.

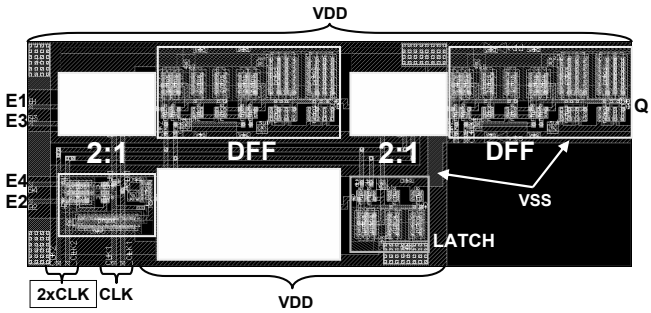


Abbildung 16 Layout des 4:1 Multiplexers

Die Anordnung dieser Zellen im Layout sollte so erfolgen, dass die Standardzellen-Designregeln eingehalten werden können.
Um den Stromversorgungspfad VSS gemeinsam nutzen zu können, werden die unteren Basiszellen an der Horizontalen gespiegelt, so dass der VSS-Anschluss nach oben herausgeführt wird. Zwischen dem VSS-Stromversorgungspfad wird ein Taktleitungspfad geführt, der die Multiplexer- und DFF-Zellen mit dem benötigten Takt versorgt. Der Ausgang des D-Latches wird über die Metall2 Verdrahtungsebene an den oberen E2-Eingang des letzten 2:1-MUX herangeführt.

Für die 8:1-Multiplexerzelle (Abbildung 17) werden zwei komplexe 4:1-Multiplexerzellen miteinander verschaltet. Das Ausgangsignal der beiden 4:1-MUX-Zellen wird abermals an einen 2:1-MUX geführt. Die zweite 4:1-Multiplexerzelle in der unteren Bildhälfte ist zur anderen Zelle horizontal gespiegelt. Die Taktleitungen CLK1 und CLK2 können daher vertikal über das Zelllayout hinweg durch den Metal2 Layer verdrahtet werden. Ab hier wird erstmals in diesem Entwurf die schnelle 2:1-MUX-Zelle mit nachfolgendem schnellen DFF eingesetzt, da das DFF schon mit dem achtfachen Basistakt (CLK4) getaktet wird. Für das D-Latch wird ebenfalls die schnelle Variante verwendet. Es ist ein deutlicher Flächenbedarfsunterschied zwischen langsamen und schnellen Zelltypen zu sehen. Die schnellen Varianten beanspruchen fast das Doppelte an Chipfläche, als die einfachen Standardzellen.

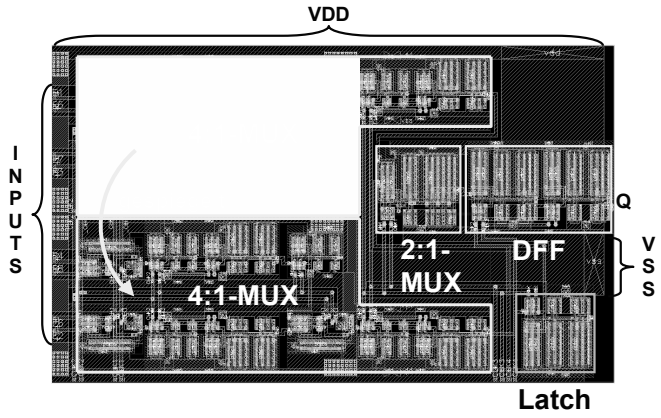


Abbildung 17 Layout des 8:1 Multiplexers

Der Aufbau der 16:1 MUX-Zelle erfolgt analog zu der 8:1-MUX-Zelle. Das Ausgangssignal kann direkt, d.h. ohne eine Einsynchronisierung mit dem nachfolgenden (hier nicht betrachteten) CMOS/Bipolar-Pegelkonverter verbunden werden. Der schnelle 2:1-Multiplexer ist also dafür ausgelegt, eine nachfolgende Konverterstufe treiben zu können. Auch hier wurde die zweite 8:1-Multiplexerzelle an der Horizontalen nach unten gespiegelt. Zwischen den beiden 8:1 MUX-Zellen wird nun ein wenig mehr Platz für einen Taktleitungspfad gelassen, der die Taktleitungen der kompletten Schaltung zusammenfasst.

Eine übersichtliche Darstellung des kompletten Multiplexerlayouts ist in Abbildung 18 gezeichnet. Um das komplette Layout herum wurde ein Graben aus Metall1 Substratkontaktierungen gezogen, um Störeinkopplungen von innen nach außen und umgekehrt zu vermeiden. Die Substratkontaktierungen werden auf die negative Betriebsspannung (VSS) gelegt. Die Kontaktierung der positiven Betriebsspannung (VDD) erfolgt von oben über den Metall2 Layer. Sämtliche Signalleitungen müssen ebenfalls durch den Metall2 Layer von außen angeschlossen werden, um über die Metall1 Substratkontaktierungen zu gelangen.

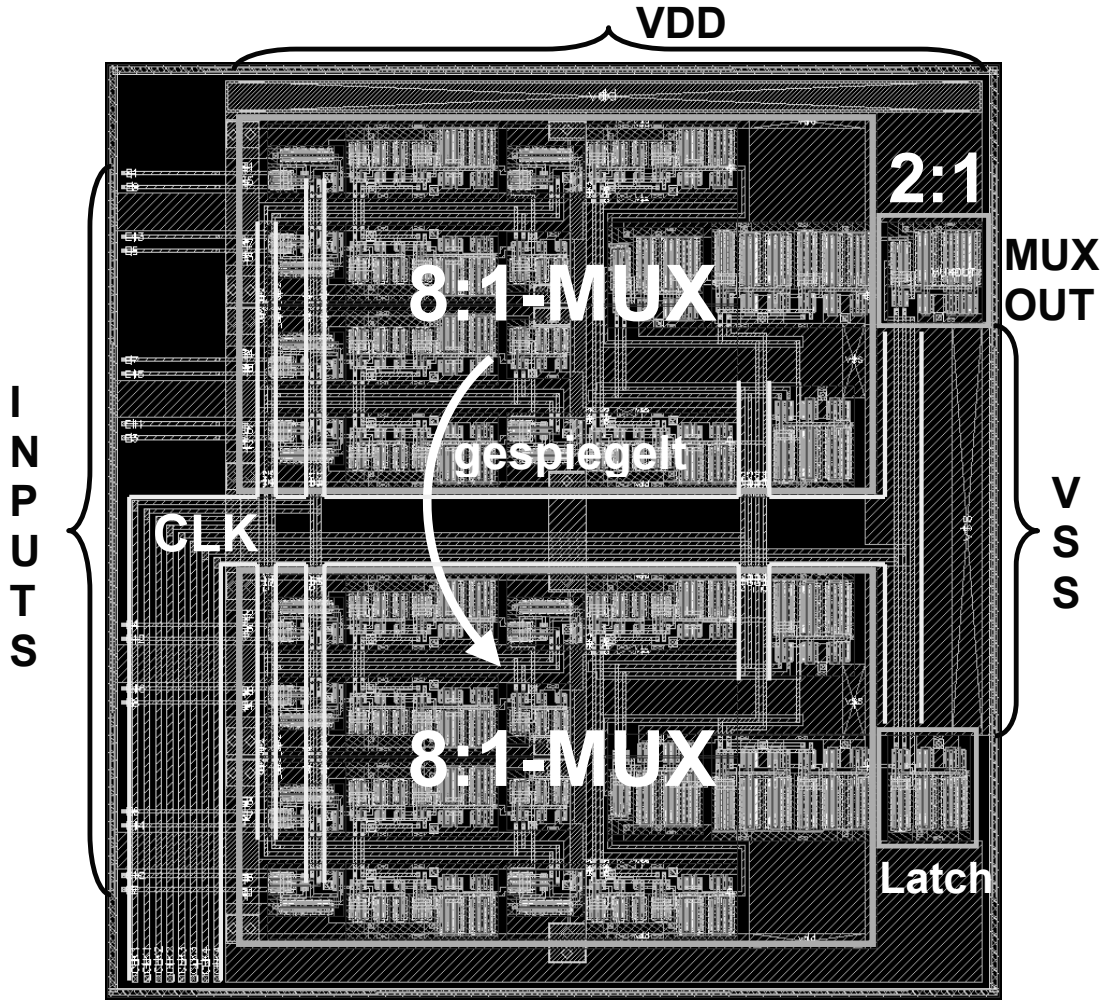


Abbildung 18 Layout des 16:1 Multiplexers

5. Teilsystem-Simulation

Der 1:16 Serializer wurde in einer Teilsystem-simulation zusammen mit anderen Komponenten durch eine Postlayout-Simulation einschließlich extrahierter parasitärer Elemente verfiziert. Der im Rahmen dieser Arbeit entwickelte Serializer befindet sich am Eingang. Danach folgt ein Pegelkonverter von CMOS auf LVDS, anschließend der LVDS-Leitungstreiber, die Übertragungsstrecke, die Empfänger-Eingangsstufe mit LVDS-Abschluss und Flankenregeneration sowie eine Pegelkonvertierung nach CMOS. Die Simulation erfolgte einschließlich der parasitären Bondinduktivitäten für die Eingänge der Spannungsversorgung sowie die Ein- und Ausgänge der Signalleitungen. Die LVDS-Übertragungsleitung auf der Platine wurde durch eine Kapazität nach Masse für jede der beiden Gegentakt-LVDS-Leitungen sowie eine Koppelkapazität zwischen den beiden Leitungen erfasst.

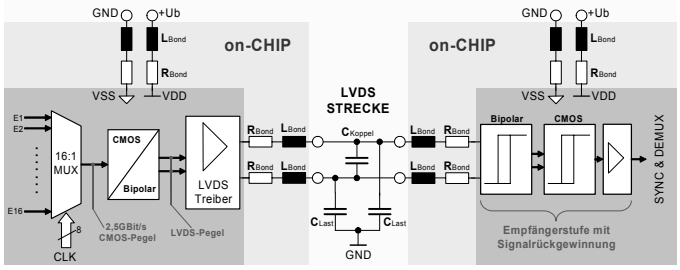


Abbildung 19 Teilsystemsimulation der LVDS-Übertragungstrecke

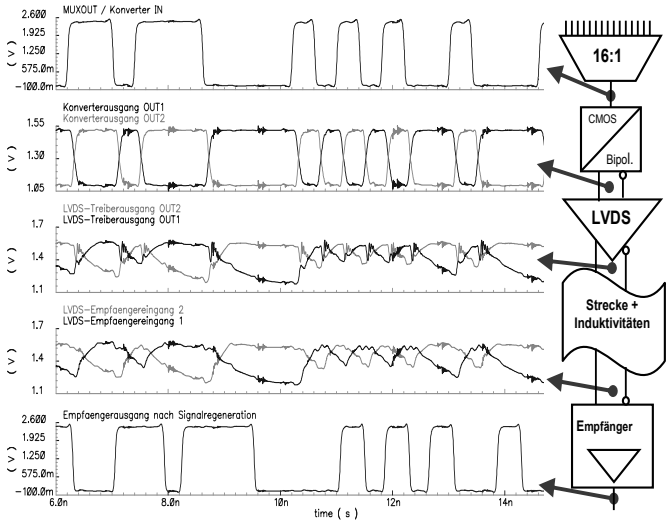


Abbildung 20 Simulationsergebnis (Postlayout)

Als Zahlenwerte für die hier beispielhaft dargestellte Simulation wurde für die Leitungskapazität jeweils 7pF, als Koppelkapazität zwischen den Leitungen 0.7pF und als Bondinduktivität 0.5nH angenommen. Anhand der Postlayout-Simulation in Abbildung 20 sieht man überlagerte Störungen, die sich auch auf die Ausgangspegel des CMOS/Bipolar-Pegel-konverters auswirken. Da aber die Auswertung der LVDS-Signale immer differentiell erfolgt, spielen diese relativ kleinen Gleichtaktstörungen keine Rolle.

Der Leitungstreiber ist offensichtlich hier an seiner Belastungsgrenze. Im Augendiagramm am Ausgang der LVDS-Treiberstufe ergäbe sich lediglich eine geringe Augenöffnung. Trotzdem kann der Empfänger aufgrund seiner empfindlichen bistabilen Eingangsstufe die Impulsform wieder regenerieren. Das Eingangs- und Ausgangsignal stimmt somit - bis auf eine Verzögerungszeit - logisch überein.

6. Ergebnis und Ausblick

Als Ergebnis lässt sich festhalten, dass gemäß der durchgeführten Post-Layout-Simulationen mit der entwickelten Schaltung eine Datenrate von bis zu 4.5GBit/s möglich ist. Die Datenrate übersteigt dabei bei weitem die angestrebten 2.5GBit/s. Andere Schaltungsteile kommen nur wenig über diese Datenrate hinaus, so dass der Serializer nicht die eigentliche Begrenzung des Designs darstellt. Da der Prozess inzwischen abgekündigt wurde, muss vor einer Fertigung des Schaltkreises auf den Nachfolgeprozess umgestellt werden, der jedoch im Wesentlichen dem für das Design verwendeten Prozess entspricht.

7. Literatur

[1] http://www.ihp-ffo.de

[2] ANSI/TIA/EIA-644-A

[3] I. Kurz, H.-P. Bürkle: Entwurf eines LVDS-Leitungstreibers für On-Chip-Kommunikation, Multiprojekt-Chip-Gruppe Baden-Württemberg: MPC-Workshop. - Ulm: Fachhochsch. 2004, Juli, Albstadt-Sigmaringen

[4] I. Kurz: Entwicklung eines integrierten LVDS-Empfängers mit Phasenregelkreis für eine Datenrate von 2.5GBit/s, Diplomarbeit an der Hochschule Aalen, 2005

Designstudie für einen 10 Bit-Analog-Digital-Umsetzer

Michael Diebold*, Arnold Erni+, Karl Hofmann+, Gerhard Forster*

+AIM INFRAROT-MODULE GmbH, Söflinger Straße 100, 89077 Ulm
*Fachhochschule Ulm, Prittwitzstraße 10, 89075 Ulm

MichaelDiebold@gmx.de, Forster@fh-ulm.de

In Zusammenarbeit mit der Firma AIM INFRAROT-MODULE GmbH wurden an der Hochschule Ulm zwei unterschiedliche Konzepte für einen AD-Umsetzer zur Anwendung in infrarot-optischen Sensoren untersucht. Ziel ist es, Sensorik und Datenverarbeitung einschließlich des AD-Umsetzers später auf einem Chip zu integrieren, um geringe Baugröße, geringen Energieverbrauch und hohe Zuverlässigkeit zu erreichen. Die Spezifikation für den AD-Umsetzer sieht eine Auflösung von 10 Bit bei einer Abtastrate von 20 MS/s vor. Die Realisierung ist in Form eines IP-Cores auf Basis der 0,35 µm CMOS-Technologie von AMS vorgesehen. Die Simulation mit Cadence zeigt, dass eine der beiden Schaltungstopologien die Spezifikation mit vertretbarem Aufwand an Chipfläche und Verlustleistung erfüllt.

1. Einleitung

Aus der Literatur ist eine Vielzahl von AD-Umsetzern bekannt [1, 2, 3]. Sie lassen sich grob in drei verschiedene Kategorien aufteilen:

- Parallelverfahren
- Wägeverfahren
- Zählverfahren

Der Vorteil des Parallelverfahrens liegt in der Erreichbarkeit hoher Abtastraten. Nachteilig ist, dass ein erheblicher Platzbedarf benötigt wird und nur eine geringe Auflösung erreicht werden kann. Mit dem Zählverfahren hingegen erreicht man eine hohe Auflösung bei einem geringen Platzbedarf, dafür aber nur kleine Abtastraten. Somit entschied man sich für den Mittelweg, das Wägeverfahren. Aus dieser Kategorie wurden zum Vergleich zwei verschiedene AD-Umsetzer entwickelt, ein zyklischer AD-Umsetzer nach dem RSD-Verfahren (Redundant Signed Digit) und ein AD-Umsetzer nach dem Prinzip der sukzessiven Approximation, auf den im vorliegenden Bericht detaillierter eingegangen wird.

2. RSD-Verfahren

Das RSD-Verfahren wurde in den 90iger Jahren entwickelt [4]. Somit ist es ein sehr neues Verfahren, das auf einem speziellen Algorithmus beruht.

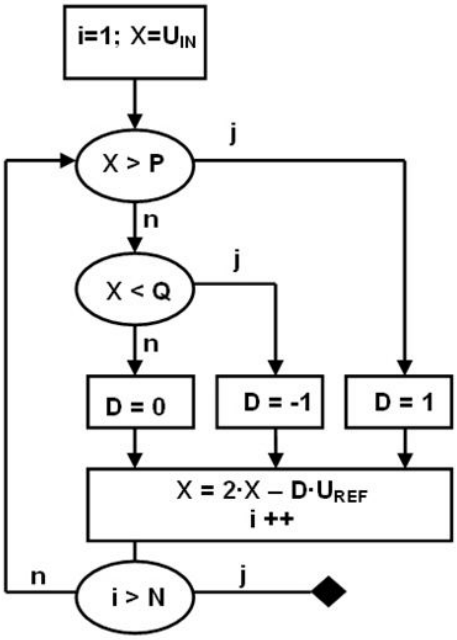


Abb.1 Signalflussplan des RSD Verfahrens

In Abb.1 ist der Signalflussplan des RSD-Verfahrens dargestellt. Zuerst wird die Variable *X* mit der Eingangsspannung versehen. Danach durchläuft die Variable *X* zwei Schwellen. Abhängig davon wird das Bit *D* gesetzt. Nach Beendigung des Vergleichs und Neuberechnung von *X* wird entschieden, ob ein weiterer Zyklus durchlaufen wird. Bei einer 10 Bit-Auflösung benötigt man somit 10 Schleifendurchläufe. In Abb.2 ist das Blockdiagramm des RSD Verfahrens dargestellt. Das Abspeichern der Eingangsspannung erfolgt durch eine Sample and Hold-Schaltung. Der darauf folgende Vergleich mit zwei Schwellen wird mit zwei Komparatoren realisiert. Das Ergebnis der Komparatoren ist gleichzeitig die Ausgangsbitfolge für einen Addierer. Danach wird der Algorithmus aktiv.

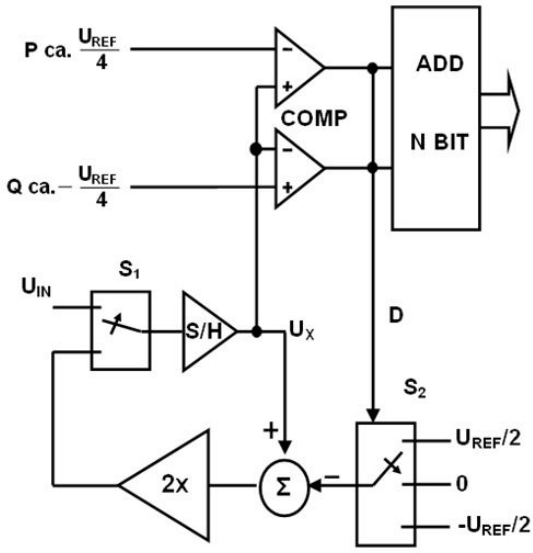


Abb.2 Blockdiagramm des RSD Verfahrens

Dieser Algorithmus, $X = 2 \cdot X - D \cdot U_{REF}$, wird durch den Schalter S_2, einer Summation und einer Verstärkung um den Faktor 2 realisiert. Der neu berechnete Wert wird wiederum in der Sample and Hold-Schaltung gespeichert, woraus eine neue Abtastung erfolgt. Dies wird solange durchgeführt, bis eine Auflösung von 10 Bit erreicht ist.

Der Vorteil dieses Verfahrens ist, dass der AD-Umsetzer einen geringen Platzbedarf benötigt (unabhängig von der Auflösung) und die Komparatoren unkritisch hinsichtlich ihrer Offset- Anforderung sind. Der Nachteil liegt bei der Multiplikation mit dem Faktor 2, da diese die Linearität (INL und DNL) bestimmt. Um die Auflösung von 10 Bit zu erzielen, muss der Verstärker innerhalb kurzer Zeit auf einen Spannungswert einschwingen, der weniger als 1 ‰ Gesamtfehler aufweist.

Mit einem einzelnen zyklischen AD-Umsetzer wurde eine Abtastrate von 500 kS/s bei einer Auflösung von 10 Bit erreicht. Durch ein Pipeline-Verfahren mit mehreren solcher AD-Umsetzer lässt sich die Abtastrate vergrößern. Bei diesem Verfahren wird die Schleife des zyklischen AD-Umsetzers an der S/H-Schaltung aufgetrennt und auf den nächsten AD-Umsetzer geführt, so dass jeder AD-Umsetzer ein Bit abhandelt. Somit werden 10 AD-Umsetzer, bei einer 10 Bit-Auflösung, in Reihe geschaltet. Dadurch wird eine Abtastrate von 5 MS/s erreicht, was der Spezifikation allerdings noch nicht entspricht.

In der Simulation zeigte sich, dass bei der vorgegebenen Technologie die geforderte Spezifikation mit dem RSD-Verfahren nur mit hohem Aufwand an Fläche und Verlustleistung zu erreichen ist.

3. SAR-Verfahren

Das Verfahren nach der sukzessiven Approximation (SAR-Verfahren) ist altbekannt für seine Einsatzmöglichkeit bei mittleren Anforderungen an Abtastrate und Auflösung. Wegen seiner Empfindlichkeit, vor allem gegenüber Offsetspannungen, kann dieses Verfahren nur dann in einer CMOS-Technologie realisiert werden, wenn aufwändige Methoden der Selbstkalibration eingesetzt werden.

3.1. Gesamtsystem

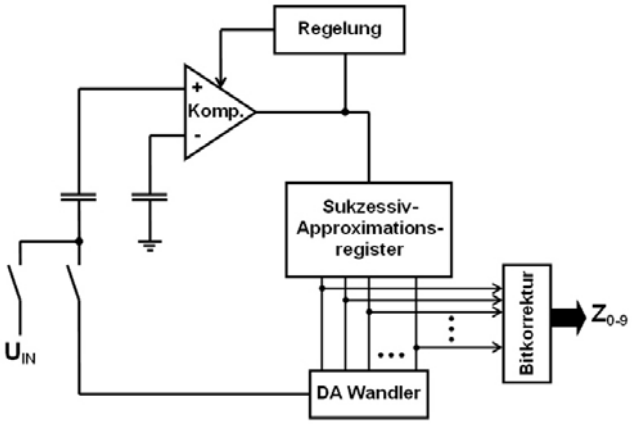


Abb.3 Blockdiagramm des SAR Verfahrens

Der Grobablauf des Verfahrens soll an Abb.3 erklärt werden. Die Eingangsspannung U_{IN}, (z. B. U_{IN} = 0,7 V) wird über eine Sample and Hold-Schaltung abgespeichert. Diese wird hier mit einem Schalter und Kondensator dargestellt. Gleichzeitig wird ein Abgleich durchgeführt, der die beiden Eingänge des Komparators kurzschließt und gemeinsam auf eine Referenzspannung von z.B. 1 V legt. Somit wird die Differenzspannung von 0,3 Volt auf dem linken Kondensator abgespeichert. Der rechte Kondensator wird auf 1 Volt aufgeladen. Nun erfolgt die Abtastung des abgespeicherten Signals. Nachdem die Abtastung erfolgt ist, sollte der DA-Wandler die Eingangsspannung von 0,7 V nachgebildet haben. Das parallele Wort am Eingang des DA-Wandlers repräsentiert daher das Ergebnis der AD-Umsetzung. Auf Regelung und Bitkorrektur wird später eingegangen.

3.2. Parallelisierung des SAR- Verfahrens

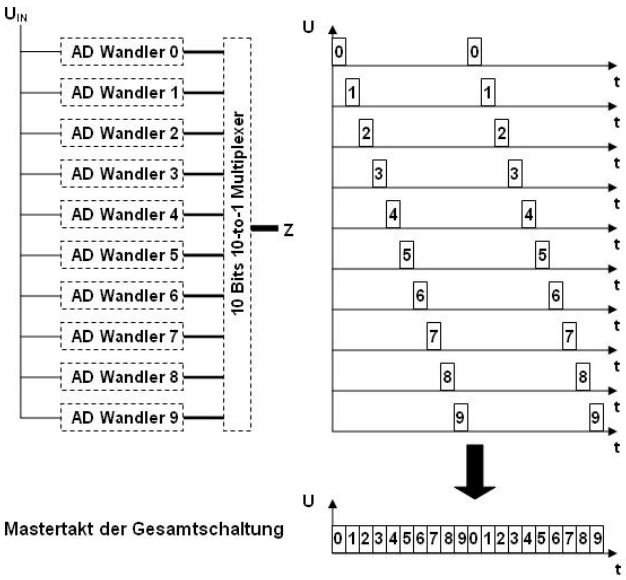


Abb.4 Parallelisierung des SAR Verfahrens

Auch in diesem Verfahren lässt sich durch Parallelisierung die Abtastrate erhöhen. Ein einzelner AD-Umsetzer nach der sukzessiven Approximation speichert mit der Sample and Hold-Schaltung die Eingangsspannung ab und benötigt danach bei einer 10 Bit Auflösung 10 Schritte, um das Signal zu wandeln. Dadurch entsteht eine gewisse Umsetzzeit von der Aufnahme des Eingangssignals bis zur Ausgabe der Bitfolge. Um diese Umsetzzeit zu nutzen, werden wie in Abb.4 beispielsweise zehn AD-Umsetzer parallel geschaltet. Der AD-Wandler 0 speichert im ersten Takt das Eingangssignal ab. Im zweiten Takt beginnt dieser mit der eigentlichen Abtastung. Gleichzeitig speichert im zweiten Takt der AD Wandler 1 das Eingangssignal ab. So werden alle 10 AD-Umsetzer nach und nach angesteuert. Durch diese Parallelisierung wird eine zehnfach höhere Abtastrate erreicht. Werden über die entsprechende Auflösung hinausgehend weitere Umsetzer integriert, so entstehen weitere Taktphasen, die zur Autokalibration genutzt werden können. Im vorliegenden Beispiel werden 16 AD-Umsetzer parallelisiert und nur eine Auflösung von 10 Bit angestrebt. Somit bleiben sechs Phasen für die Regelung zur Verfügung. Die einzelnen Teilschaltungen des AD-Umsetzers und Funktionen werden nachfolgend beschrieben.

3.3. Sample and Hold

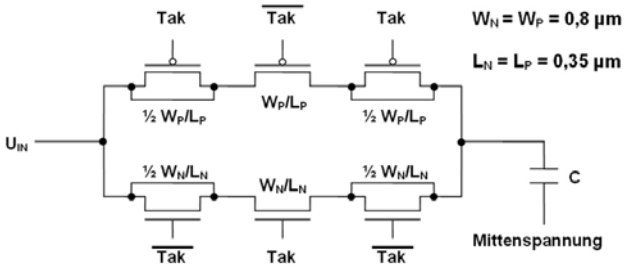


Abb.5 Sample and Hold-Schaltung

Eine ideale Sample and Hold-Schaltung (S/H-Schaltung) besteht aus einem Schalter und einem Kondensator. Der hier verwendete Schalter wurde mit einem Transmission Gate (TGate) und vier Dummy Transistoren realisiert. Dabei sind zwei Kenngrößen des TGates zu beachten, zum einen die Ladungsinjektion und zum anderen der Durchschaltewiderstand. Da die Ladungsinjektion von der Größe der Transistoren abhängt, wurde das TGate mit der Minimalgröße von AMS implementiert. Durch zusätzliche Dummy-Transistoren, die kurzgeschlossen sind und wie Kapazitäten wirken, wird bei gegenphasiger Ansteuerung zusätzlich die Ladungsinjektion minimiert. Bei einer Kapazität von 240 fF verursacht die Ladungsinjektion eine maximale Fehlerspannung von 204 µV.

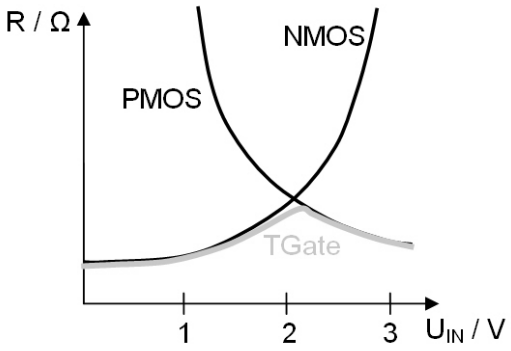


Abb.6 Eingangsimpedanz eines TGates

Die Eingangsimpedanz resultiert aus der Parallelschaltung der beiden Widerstandsverläufe von NMOS- und PMOS Transistor. Somit wird ein relativ konstanter Widerstandsverlauf erreicht.

3.4. Komparator

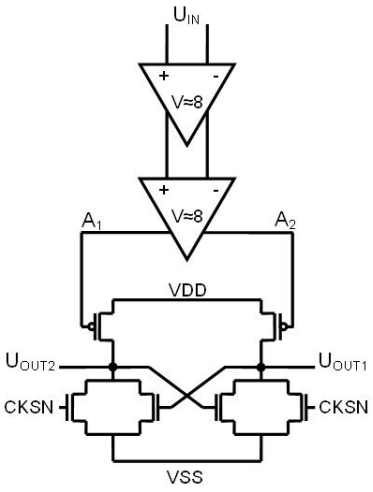


Abb.7 Komparator

Der Komparator besteht aus zwei Vorverstärkern und einem Latch (Abb.7). Das Latch wird über das Taktsignal CKSN angesteuert. Ist das Taktsignal high, so geht das Latch in den differentiellen Betrieb und vergleicht die beiden Eingangssignale. Ist das Taktsignal hingegen low, so ist das Latch im Flip-Flop-Betrieb und speichert das Ergebnis des vorigen Vergleichs in Form einer Null oder Eins (*VSS* oder *VDD*) ab.

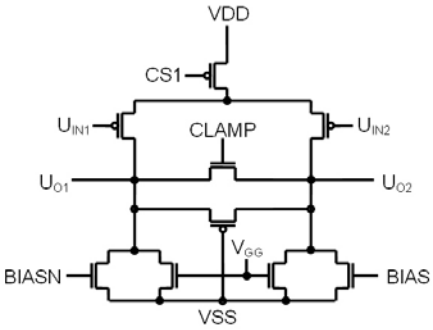


Abb.8 Vorverstärker

Zur Erhöhung der Komparator- Empfindlichkeit muss das Eingangssignal U_{IN} mindestens um den Faktor 50 verstärkt auf das Latch gegeben werden. Diese Verstärkung erfolgt in zwei Stufen, um möglichst hohe Bandbreite bei gegebener Verlustleistung zu erzielen. Jede Stufe besteht aus einem PMOS- Differenzverstärker mit aktiver Last und zwei Clamp-Transistoren. Die erste Stufe (Abb.8) enthält zusätzlich zwei weitere Transistoren zum Offsetabgleich, angesteuert über die Signale BIAS und BIASN. Der passive Clamp, der mit *VSS* angesteuert wird, begrenzt den Ausgangsspannungshub auf 1 Volt. Somit muss der Verstärker nicht bis zur Versorgungsspannung aussteuern, wodurch sich seine Erholzeit verkürzt.

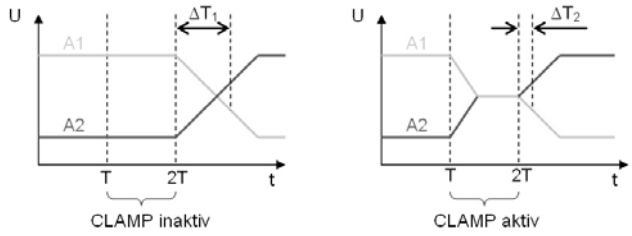


Abb.9 Funktion des aktiven Clamps

Der aktive Clamp-Transistor, der mit dem Taktsignal CLAMP geschaltet wird, erzwingt einen Kurzschluss an den Ausgängen, wodurch die Erholzeit noch weiter verkürzt werden kann. In Abb.9 wird die Funktion des aktiven Clamp ersichtlich. Bei minimalen Differenzspannungen am Eingang des Vorverstärkers und inaktivem Clamp benötigt es lange (ΔT_1), bis das Ergebnis durch das Latch detektiert werden kann, wenn das Ausgangssignal der Vorverstärker seine Polarität ändern muss. Durch den aktiven Clamp wird der Vorgang beschleunigt ($\Delta T_2 < \Delta T_1$), da die beiden Ausgangsspannungen zuerst durch den Kurzschluss auf Mittenspannung gebracht werden. Somit kann auch das Latch auf minimale Spannungsänderungen reagieren, da dieses kurz nachdem CLAMP aktiv war, in den Flip-Flop-Betrieb geschaltet wird.

Die Offsetspannung des Komparators wird über die BIAS-Transistoren ausgeregelt, die die Symmetrie der aktiven Last beeinflussen. Auf die Ansteuerung wird in Abschnitt 3.6 eingegangen.

Die Vorverstärker haben zusammen eine Gesamtverstärkung V_{ges} = 35 dB und eine Grenzfrequenz f_{3dB} = 60 MHz. Daraus lässt sich eine Anstiegszeit $t_{10/90}$ = 5,8 ns abschätzen. Bei einer Abtastrate von 20 MS/s stehen 25 ns zur Verfügung, so dass die Anstiegszeit ausreichend kurz sein sollte. Die Einschwingzeit des Komparators ist allerdings von der Höhe der Eingangsspannung abhängig, da das Latch bei großer Eingangsspannung einen geringen, bei kleiner Eingangsspannung dagegen einen größeren relativen Hub benötigt, um richtig zu entscheiden.

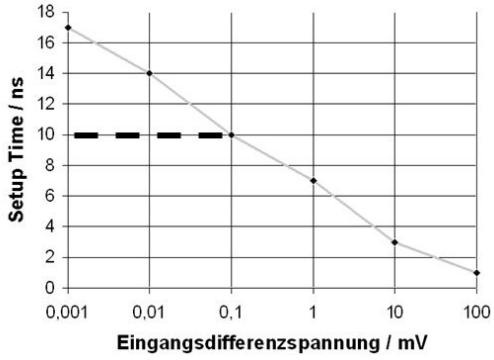


Abb.10 Setup-Time des Komparators

Dazu wurde die Setup-Time simuliert. Diese sagt aus, wie lange die Differenzspannung am Eingang des Komparators anliegen muss, bis das Latch richtig entscheidet. Je kleiner die Differenzspannung, umso größer ist die Setup-Time, da es länger dauert, bis alle parasitären Kapazitäten umgeladen sind. Durch die Integration des aktiven Clamp wird die Setup-Time bei kleinen Eingangsdifferenzspannungen auf 10 ns begrenzt. Die Verzögerungszeit des Komparators, d.h. die Zeit zwischen Freigabe des Clamp und eingeschwungenem Ausgangsignal, beträgt in der Simulation τ = 3 ns bei U_{IN} = 1 µV.

3.5. Digital-Analog-Wandler

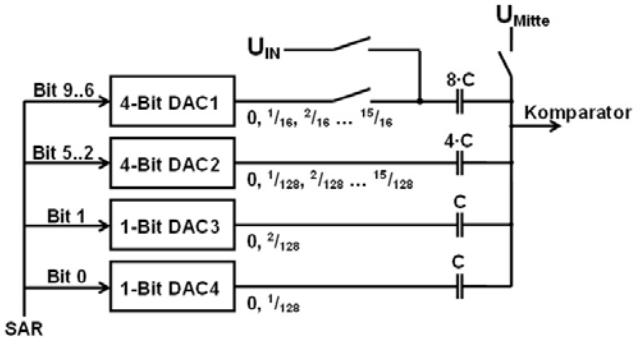


Abb. 11 Digital-Analog-Wandler

Im vorliegenden Konzept wird nicht ein einzelner DA-Wandler eingesetzt, der 1024 Teilspannungen der Referenzspannung U_{REF} zu erzeugen hätte, sondern es wird nur eine Untermenge von Spannungen erzeugt und stattdessen am Komparatoreingang mit mehreren dual abgestuften Kondensatoren gearbeitet (Abb. 1). Mit den gewählten Abstufungen C, C, 4C, 8C wird als kleinste Teilung nur noch $^1/_{128}\cdot U_{REF}$ benötigt wegen $^1/_8 \cdot ^1/_{128} = ^1/_{1024}$. Dies vereinfacht ganz entscheidend die Realisierung des Referenzspannungsteilers. Es werden 4 einfache DA-Wandler benötigt, zwei 4-Bit-Wandler und zwei 1-Bit-Wandler, um in Kombination mit den unterschiedlichen Kondensatoren alle Referenzspannungen zu realisieren. Die DA-Wandler bestehen aus einfachen Transmission Gates, die die Teilspannungen einer Widerstandsleiter durchschalten. Benötigt werden lediglich die Spannungswerte $^1/_{16} \ldots ^{15}/_{16}\cdot U_{REF}$ und $^1/_{128} \ldots ^{15}/_{128}\cdot U_{REF}$. Dennoch ist die Dimensionierung des Teilers kritisch, da dieser beim Schalten der Transmission Gates durch die Ladeströme der Kondensatoren belastet wird und somit eine endliche Einschwingzeit aufweist. Da aus Gründen der Chipfläche der gleiche Widerstandsteiler nicht für einen, sondern für alle 16 AD-Umsetzer genutzt werden soll, verschärft sich das Problem der endlichen Einschwingzeit. Die genannten Probleme wurden wie folgt gelöst:

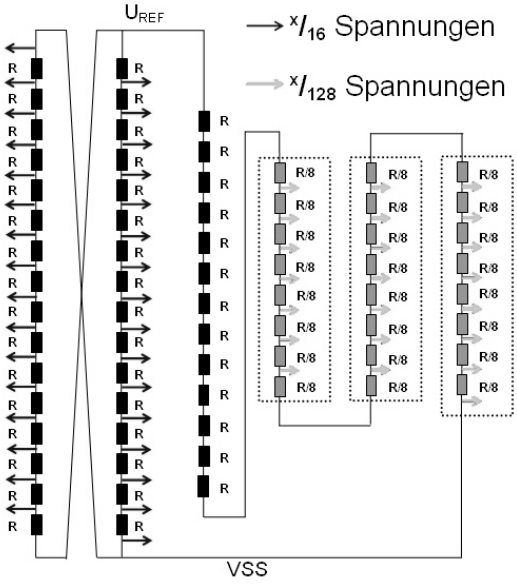


Abb.12 Widerstandsleiter

Es werden drei Reihenschaltungen benutzt, zwei Reihenschaltungen für die $^x/_{16}$ Spannungen, die kreuzgekoppelt sind, und eine Widerstandskette, die die $^x/_{128}$ Spannungen erzeugt (Abb.12). Durch die drei Reihenschaltungen sind die jeweiligen Widerstandsketten unabhängig voneinander. Dadurch wirkt sich eine starke Belastung an einer Stelle nicht auf die andere Widerstandskette aus. Die einzelnen Widerstände wiederum bestehen aus Reihen- und Parallelschaltung von Einheitswiderständen. Damit wird optimales Matching im Layout ermöglicht. Ein Einheitswiderstand wurde auf $R = 104\,\Omega$ $(2\cdot 52\,\Omega)$ dimensioniert. Somit beträgt der Gesamtwiderstand:

$$\frac{1}{R_{ges}} = 2\cdot\frac{1}{16\cdot 104\,\Omega} + \frac{1}{13\cdot 104\,\Omega + 3\cdot(8\cdot 13\,\Omega)}$$

$$R_{ges} = 555\ \Omega$$

Mit U_{REF} = 2 V resultiert ein Gesamtstrom von 3,6 mA, was noch akzeptabel ist. Der hochohmigste Anzapfpunkt liegt bei U_{REF} = 1 V.

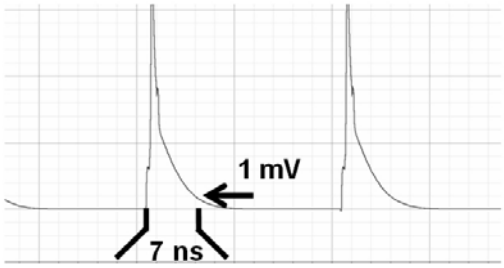


Abb.13 Referenzspannung $^8/_{16}\cdot U_{REF}$ = 1 V

Abb.13 zeigt das Einschwingverhalten dieses Knotens bei gleichzeitigem Zuschalten aller 16 AD-Umsetzer.

Die Einschwingzeit auf weniger als 1 mV Abweichung beträgt 7 ns. Es zeigt sich, dass mit diesem Widerstandsteiler ein guter Kompromiss zwischen Einschwingzeit und Verlustleistung gefunden wurde. Im Vergleich dazu hätte sich mit einer Widerstandsleiter, die nur eine Reihenschaltung bei gleichem Gesamtwiderstand enthält, eine Einschwingzeit von über 10 ns ergeben.

3.6. Regelung

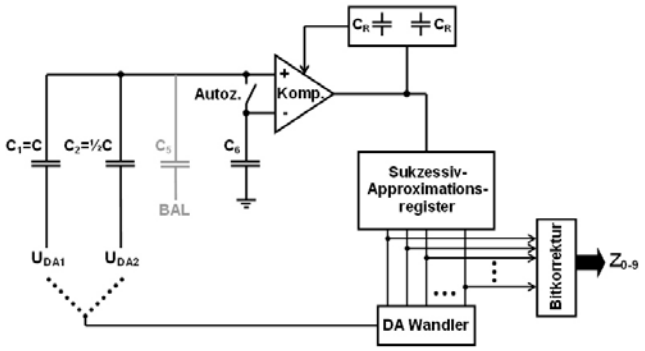


Abb.14 Regelung

Da die Linearität (INL und DNL) beim SAR-Verfahren durch den DA-Wandler und den Komparator bestimmt wird, musste eine Abgleichschaltung integriert werden, die die Offset- bzw. Kapazitätsfehler kompensiert. Das Prinzip der Regelung ist in Abb.14 mit zwei Kondensatoren C_1 und C_2 symbolisch dargestellt.

Im ersten Schritt der Regelung wird ein Autozero durchgeführt. Dabei wird der Eingang des Komparators mit 1 Volt kurzgeschlossen. Gleichzeitig wird vom DA-Wandler eine Spannung an den Kondensator C_1 angelegt. Diese Spannung beträgt $^9/_{16}\cdot U_{REF}$. Folglich wird der Kondensator C_1, bei einer Referenzspannung $U_{REF} = 2$ V, auf die Differenzspannung von $^1/_2\cdot U_{REF} - {}^9/_{16}\cdot U_{REF}$ aufgeladen. Die beiden anderen Kondensatoren C_2 und C_6 werden jeweils auf 1 Volt aufgeladen.

Im nächsten Schritt wird der Autozero inaktiv und die beiden DA-Spannungen bleiben bestehen. Dadurch sollte an beiden Eingängen des Komparators 1 Volt anliegen. Bei einem idealen Komparator werden dann über mehrere Zyklen im Mittel gleich viele Nullen wie Einsen als Ergebnis ausgegeben. Der zeitliche Mittelwert wird auf einer Kapazität abgespeichert. Die Sollspannung ist somit die Mittenspannung. Wenn nun ein Offsetfehler im Komparator vorliegt, werden mehr Nullen als Einsen ausgegeben bzw. mehr Einsen als Nullen. In diesem Fall weist die Kondensatorspannung eine Abweichung von der Mittenspannung auf. Diese Spannung steuert nun die beiden BIAS-Transistoren an und beeinflusst dadurch die aktive Last im Vorverstärker (Abb.8). Somit wird der Offsetfehler kompensiert.

Im dritten Schritt werden nun die beiden DA-Spannungen verändert. U_{DA1} beträgt nun $^8/_{16}\cdot U_{REF}$ und $U_{DA2} = {}^2/_{16}\cdot U_{REF}$. Durch die Veränderung der beiden Spannungen U_{DA1} und U_{DA2} findet ein Ladungsausgleich zwischen beiden Kondensatoren C_1 und C_2 statt.

$$\frac{8}{16}\cdot U_{REF} + \frac{1}{2}\cdot\frac{2}{16}\cdot U_{REF} = \frac{9}{16}\cdot U_{REF}$$

Durch das Kapazitätsverhältnis und durch die beiden Spannungen sollte nun wieder an den Eingängen des Komparators jeweils 1 V anliegen. Liegt nun ein Kapazitätsfehler vor oder eine geringe Abweichung in den Spannungen U_{DA1} und U_{DA2}, wird dies auf einen weiteren Kondensator in der Regelung abgespeichert. Die Spannung dieses Kondensators regelt nun das Signal BAL. Dieses Signal steuert nun einen weiteren Kondensator C_5 an. Ist $C_1 > 2\cdot C_2$ wird das Potential von BAL vergrößert. Somit wird die fehlende Ladung am Kapazitätsverhältnis, das die 1 V am Komparatoreingang ergeben sollte, durch den Kondensator C_5 eingebracht. Ist hingegen $C_1 < 2\cdot C_2$, so wird das Potential von BAL kleiner und Ladung wird vom Kapazitätsverhältnis abgezogen. Dadurch wird ein möglicher Kapazitätsfehler kompensiert. Gleichzeitig werden auch die beiden U_{DA} Spannungen überprüft, da sie die Vergleichspannungen für den Komparator vorgeben. Somit wird auch ein Widerstandsleiterabgleich durchgeführt.

Die Einschwingzeit der Regelspannungen ist von der Größe der Kondensatoren C_R abhängig. Mit der vorgenommenen Dimensionierung benötigt der AD-Umsetzer ca. 50000 Taktzyklen, bis die Regelspannungen die Mittenspannung erreicht haben.

3.7. Sukzessiv- Approximationsregister

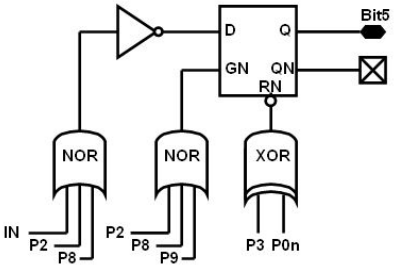


Abb.15 Das Sukzessiv-Approximationsregister

Das SAR besteht aus 11 D-Flip-Flops (D-FF), die je nach Takt angesteuert werden. In Abb.15 ist die Ansteuerung von Bit 5 dargestellt. In Phase 2 (P2) wird das D-FF aktiviert. Dabei speichert es eine Eins ab und gibt diese am Ausgang Q aus. Während Phase 2 wird dieses Bit für den Kapazitätsabgleich benötigt. Während der Phase 3 (P3) wird diese Eins wieder ge-

löscht. In Phase 8 (P8) wird wieder eine Eins im D-FF abgespeichert. Dadurch wird nun eine Spannung für den Vergleich an den Eingangskondensatoren angelegt. Während Phase 9 (P9) wird das Ausgangssignal des Komparators (IN) abgespeichert, das das Ergebnis des Vergleichs der vorigen angelegten Spannung ist. Lag die Spannung des vorigen Vergleichs oberhalb der Vergleichsspannung, so wird eine Null abgespeichert; lag sie darunter, wird eine Eins abgespeichert. Dieses Ergebnis bleibt solange im D-FF gespeichert, bis in Phase 15 die Bitfolge ausgegeben wird. Danach werden während der Phase 0 (P0n) alle D-Flip-Flops wieder gelöscht.

3.8. Bitkorrektur

Das Ergebnis des Sukzessiv-Approximationsregisters wird nicht sofort ausgegeben, sondern durchläuft vorher eine Bitkorrektur, mit der die Auswirkung der folgenden Maßnahme berücksichtigt wird. Um die Settling Time bei höherwertigen Bits zu verkürzen, wird bis zur Phase 8 die Vergleichsspannung um eine geringe Offsetspannung abgesenkt (*Abb.16*).

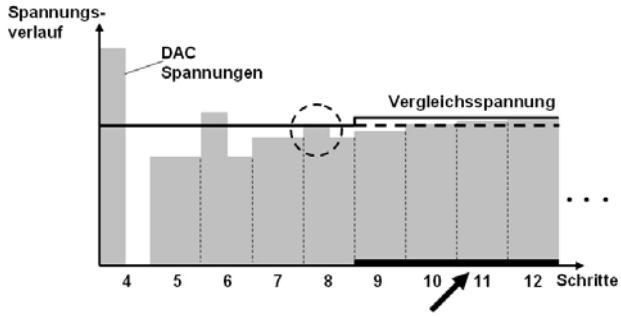


Abb.16 Bitkorrektur

Dabei kann in Schritt 8 ein Fehler entstehen wie in dieser Abbildung dargestellt (mit Kreis markiert). Die DA-Spannung wäre normalerweise unterhalb der Vergleichsspannung, es würde eine Eins abgespeichert und die Spannung würde bestehen bleiben. Wegen der Offsetspannung liegt aber die DA- Spannung oberhalb der Vergleichsspannung. Nach Schritt 8 wird die Offsetspannung wieder abgeschaltet und somit vergrößert sich die Vergleichsspannung. Nun können die DA-Spannungen die Vergleichsspannung nicht mehr erreichen, sondern nur noch die Vergleichsspannung mit Offset (gestrichelte Linie). Deshalb wird im Schritt 9 zusätzlich geprüft, ob zuvor ein Fehler gemacht wurde oder nicht. Sobald ein Fehler entstanden ist, wird eine zusätzliche Spannung in den darauf folgenden Schritten angelegt (mit Pfeil markiert). Dadurch werden die folgenden Spannungen um die Offsetspannung angehoben. Somit können die DA-Spannungen die Vergleichsspannungen wieder erreichen. Durch die Anhebung der DA-Spannung ab Schritt 9 entsteht ein Fehler bei der Ausgangsbitfolge. Um diesen auszugleichen, wird einfach eine Bitfolge, die der Offsetspannung entspricht, auf die Ergebnisbitfolge addiert. Dadurch wird der Fehler, der durch den künstlichen Offset entstehen kann, kompensiert.

3.9. Phasensequenz

Der gesamte Ablauf der Umsetzung vollzieht sich in 16 Taktphasen. Die Aktion innerhalb der einzelnen Phasen ist in der folgenden Tabelle dargestellt.

Phasen	Aktion
P0	Autozero
P1	Offsetabgleich
P2	Kapazitäts- bzw. Widerstandsabgleich
P3	Autozero Anlegen der Eingangsspannung U_{IN}
P4-P14	eigentliche Abtastung des Eingangssignals
P15	Bitkorrektur, wenn BitD04 = 1 Ausgabe des Bitfolge

3.10. Simulationsergebnisse

Schaltungseingabe und Simulation erfolgten mit Cadence- Werkzeugen. Nach der vollständigen Dimensionierung und Verifikation aller Teilschaltungen wurde eine Testbench für den gesamten AD-Umsetzer erstellt (Abb.18). Für die Erzeugung des Eingangssignals wurde ein DA-Wandler eingesetzt, der eine rampenförmige Spannung zwischen 0 und 2 V in LSB-Schritten quantisiert. Die entsprechenden Ausgangssignale des AD-Umsetzers werden in Analogspannungen umgerechnet und mit den um die Umsetzzeit verzögerten Eingangsspannungen verglichen. Somit kann bequem die Abweichung dargestellt werden (Abb.17).

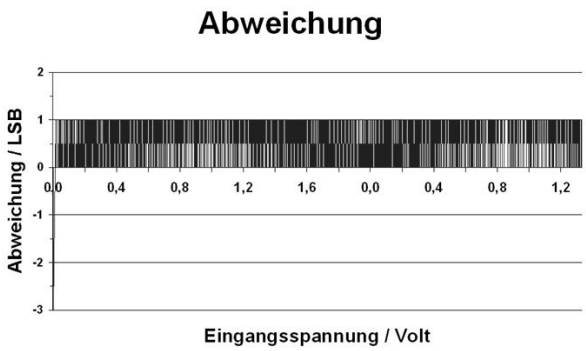


Abb.17 Abweichung

Die Abweichung liegt zwischen 0 und 1 LSB. Diese Simulation wurde mit einem AD-Umsetzer durchgeführt, der mit einer Taktrate von 312,5 kS/s betrieben wurde. Dies entspricht bei 16 AD-Umsetzern einer Abtastrate von 5 MS/s. Derselbe Versuch, durchgeführt mit einer Abtastrate von 1,25 MS/s bzw. 20 MS/s, ergab die gleiche Abweichung. In dieser Analyse werden 1600 Zyklen durchlaufen, was eine Simulationszeit von etwa 2 Wochen erfordert.

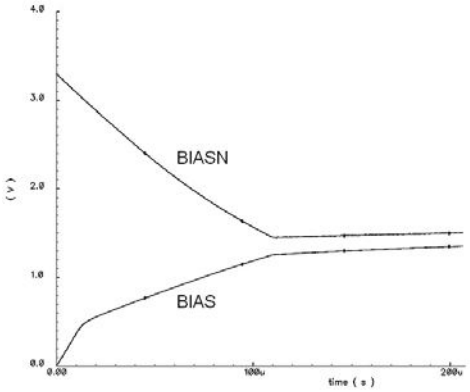


Abb.19 Regelspannungen BIAS und BIASN

In dieser Simulation wurden die Regelspannungen mit Startspannungen versehen. Somit waren diese im eingeschwungenen Zustand, und das Ergebnis konnte sofort ausgewertet werden. In einer weiteren Analyse wurden die Regelspannungen ohne Startspannungen simuliert. In Abb.19 sind die beiden Regelspannungen BIAS und BIASN dargestellt. Die Regelspannungen CAL und CALN verhalten sich in ihrem Einschwingverhalten gleich.

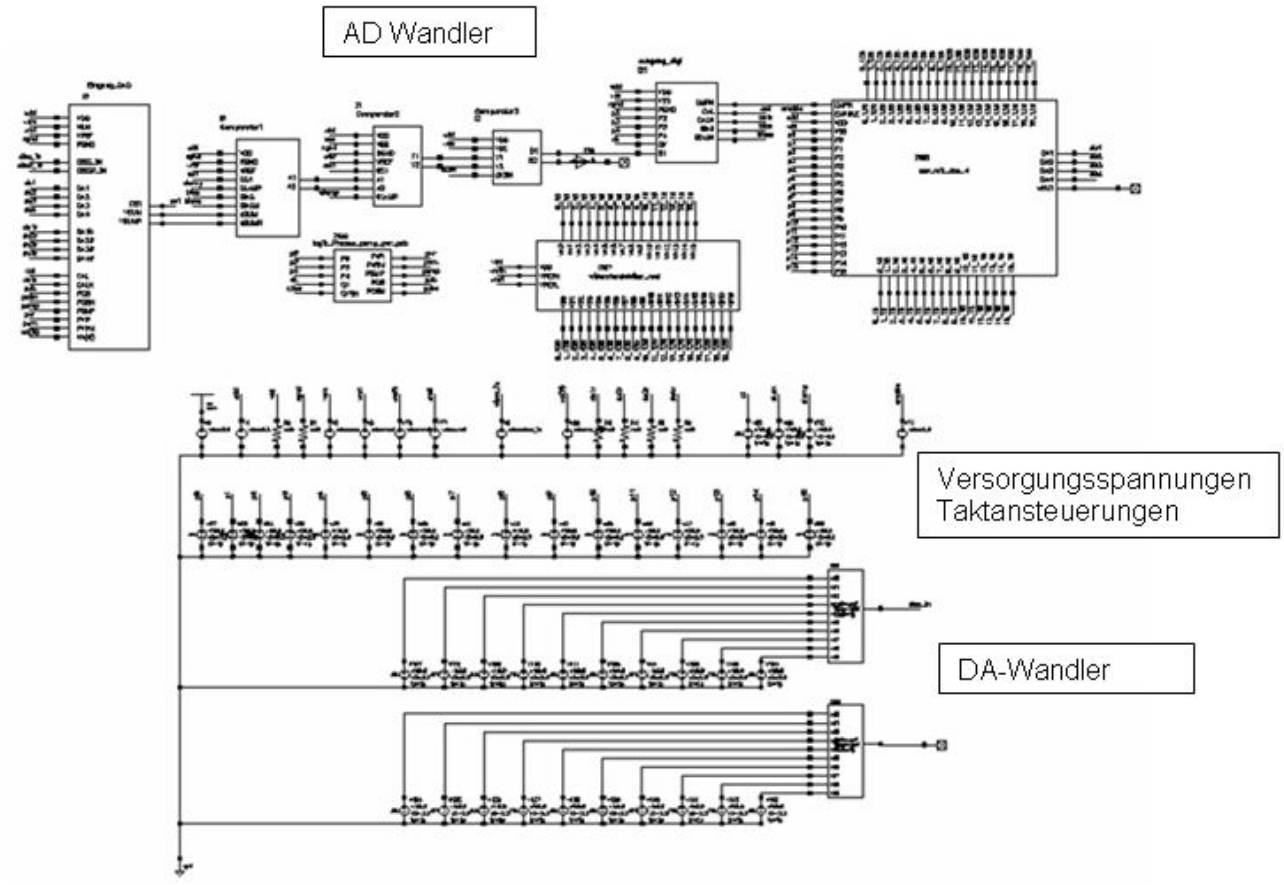


Abb.18 Testbench des AD Wandlers

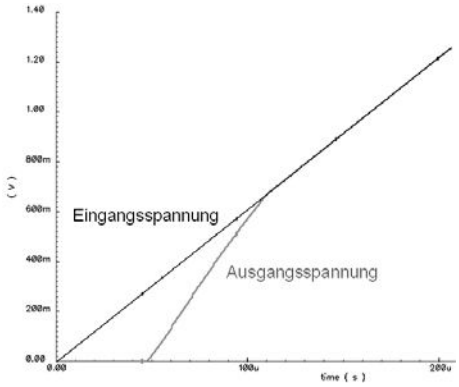


Abb.20 Ein- bzw. Ausgangsspannung

An der Ein- bzw. Ausgangsspannung des AD-Umsetzers (*Abb.20*) kann man erkennen, wie sich die Ausgangsspannung der Eingangsspannung annähert. Der Einschwingvorgang der Regelung dauert etwa 120 µs.

4. Layoutabschätzung

Zum Abschluss wurde im Hinblick auf die Realisierung eine Layoutabschätzung durchgeführt. In der folgenden Tabelle sind die einzelnen Komponenten des AD-Umsetzers aufgeführt. Dabei wurde die Fläche ermittelt, die die Transistoren auf dem IP-Core einnehmen. Für die Verdrahtung der einzelnen Transistoren wurden bestimmte Faktoren angenommen: Für analoge Schaltungen der Faktor 10, für digitale Teilschaltungen der Faktor 1,4 und für den DA-Wandler der Faktor 5, da dieser aus digitalen und analogen Teilschaltungen besteht.

	Transistorfläche (μm^2)	Faktor für die Verdrahtung	Endfläche (μm^2)
Komparator	305	10	3050
Regelung	5233	10	52330
Sample and Hold	1,12	10	11
SAR	4684	1,4	6558
DA Wandler	13676	5	68380
Taktgenerierung	2228	1,4	3119
Bitkorrektur	2723	1.4	3812
Gesamtfläche eines AD-Umsetzers			137260

Somit kommt man auf eine Fläche eines einzelnen AD-Umsetzers von 0,137 mm^2. Da der komplette AD-Umsetzer aus 16 einzelnen AD-Umsetzern (2,196 mm^2) besteht und einer Widerstandsleiter (0,067 mm^2), kommt man auf eine gesamte Coreflä-che von 2,263 mm^2.

5. Zusammenfassung und Ausblick

Zwei unterschiedliche Ausgestaltungen für einen 10 Bit-Analog-Digital-Umsetzer nach dem Wägeverfahren wurden untersucht. Mit dem RSD-Verfahren wurde mit vertretbarem Aufwand eine Abtastrate von 5 MS/s erreicht. Durch weitere Parallelisierung wäre eine Abtastrate von 10 MS/s denkbar, allerdings nur mit überhöhtem Aufwand an Chipfläche und Verlustleistung. Die Auflösung von 10 Bit wurde auch mit dem SAR-Verfahren erreicht. Erforderlich dazu sind allerdings Maßnahmen zur Autokalibration, um Fehler durch Offsetspannungen sowie Widerstands- und Kapazitätstoleranzen zu kompensieren. Mit der vorliegenden Schaltung wurde die Abtastrate von 20 MS/s erreicht. Auch bei 50 MS/s ist die Schaltung noch lauffähig, zeigt dann aber einen Auflösungsfehler von 4 LSB. Entscheidend für die Auflösung ist die Genauigkeit des Widerstandsteilers. Hierzu sind noch weitere Untersuchungen erforderlich, ebenso zu Temperaturabhängigkeit, Versorgungsabhängigkeit und Rauschen. Da die Simulationen sehr zeitaufwändig sind, soll nunmehr ein Testchip realisiert werden.

6. Literaturverzeichnis

[1] Behzad Razavi: Principles of Data Conversions. IEEE Computer Society Press 1994

[2] Rudy J. van de Plassche: CMOS Integrated Analog-to-Digital and Digital-to-Analog Converters. Kluwer Academic Publishers 2003

[3] Mikko E. Waltari; Kari A.I. Halonen: Circuit Techniques for Low-Voltage and High-Speed A/D Converters. Springer Verlag 2002

[4] Reinhard Kindt; Richard Ižák; Jan Strömer: Erfahrungen bei der Nutzung einer AHDL beim Entwurf eines zyklischen AD-Wandlers. http://www.imms.de/db/publikationen/2000_itg_ahdl.pdf 2000

[5] R. Jacob Baker; Harry W. Li; David E. Boyce: CMOS Circuit Design, Layout, and Simulation. Wiley 2005

Modellierung und Synthetisierung des Peripheriebausteins SAB8279

Christof Voelkle

Fachhochschule Ulm, Prittwitzstr. 10, 89075 Ulm

cvoelkle@web.de

Auf einer älteren Maschinensteuerung der Firma Handtmann wird unter anderem der Tastatur- und Anzeigecontroller SAB8279 eingesetzt. Dieser Baustein wird schon seit einigen Jahren nicht mehr produziert, und die noch vorhandenen Lagerbestände schwinden langsam aber sicher. Ziel dieser Arbeit ist der Entwurf einer VHDL Beschreibung der von der Steuerung benötigten Funktionen, die Synthetisierung der Beschreibung auf einem FPGA und die Entwicklung einer Platine auf der dieses, und Modelle anderer Bausteine evaluiert werden können.

1. Einleitung

1.1. Motivation

Die erwähnte Maschinensteuerung wird seit 1989 produziert und ist als klassisches Microcontrollerkonzept aufgebaut. Als Prozessor kommt ein 8085 der Firma Intel zum Einsatz. Da dieser Prozessor keinen Speicher und keine Peripherie besitzt, müssen diese Funktionen von externen Bausteinen übernommen werden. Zu diesen Bausteinen gehören:

- RAM
- ROM
- Multifunktionsbaustein SAB8256
- Tastatur- und Anzeigecontroller SAB8279

Die beiden letzten Bausteine werden schon seit einigen Jahren nicht mehr produziert und sind nur noch als Restbestände zu beziehen. Da noch sehr viele Maschinen mit dieser Steuerung in Betrieb sind, muss aber die Versorgung mit Ersatzteilen gesichert sein.

Eine Möglichkeit wäre nun die Steuerung mit aktueller Hardware auszustatten und die Software auf diese Hardware zu portieren. Dagegen spricht allerdings ein gravierender Punkt. Die Software ist auf einem mittlerweile etwas betagten Entwicklungssystem der Firma Hewlett Packard in Pascal und Assembler verfasst worden, was den Portierungs- und Testaufwand sehr hoch ausfallen lassen würde.

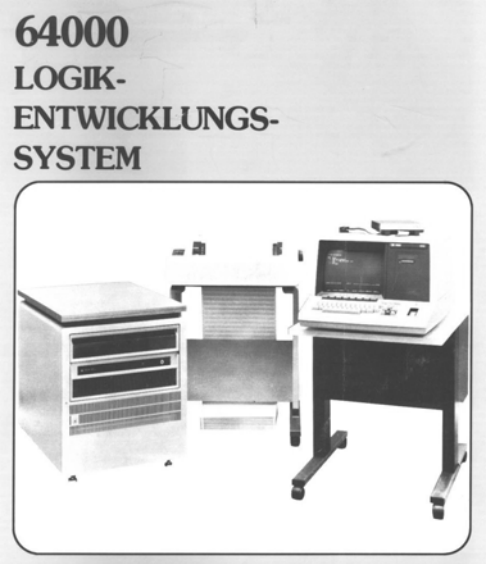


Abb. 1: HP64000 Entwicklungssystem

Eine andere Möglichkeit wäre die Nutzung einer aktuellen Steuerung unter Verwendung einer angepassten aktuellen Software. Die Anpassung der aktuellen Software, um die Funktionen der älteren Maschinen zu steuern ist allerdings ebenfalls mit einem hohen Aufwand verbunden.

Aus diesen Gründen wurde beschlossen die betroffenen Bausteine in VHDL zu beschreiben und auf einem FPGA nachzubilden. Dadurch ist sichergestellt, dass die original Software weiterhin ohne Anpassung verwendet werden kann, und die Steuerung verhält sich nach außen, wie eine Originalsteuerung.

1.2. Ziele

Schlussendlich soll der komplette Digitalteil der Steuerung durch eine neue Platine ersetzt werden, auf der möglichst alle Funktionen der alten Platine auf einem FPGA realisiert sind. Dadurch sinkt das Risiko, dass Bauteile abgekündigt werden, auf ein Minimum. Sollte dennoch einmal das FPGA abgekündigt

werden, lässt sich dies durch eine Layoutänderung beheben, da die eigentliche Funktion Hardwareunabhängig als VHDL Beschreibung existiert.

2. Der Baustein SAB8279

2.1. Funktionen

In einer klassischen Microcontrollerschaltung übernimmt der Baustein SAB8279 zwei elementare Funktionen. Zum einen wird von ihm die Abtastung und Auswertung der Tastatur bzw. der Tasten und Schalter übernommen, zum anderen wird von ihm die zyklische Ausgabe der Anzeigedaten übernommen. Diese beiden Funktionen, das Einlesen von Tasten bzw. Schaltern und das Ausgeben von Informationen z.B. auf 7-Segmentanzeigen, waren von damaligen Prozessoren nur mit hohem zeitlichem Aufwand zu realisieren, was den Einsatz dieser Bausteine rechtfertigt.

Das Funktionsprinzip ist in Abb. 2 angedeutet. Die einzulesenden Tasten sind in einer Matrix angeordnet. Über die Scanlines, die zyklisch durchgezählt werden, wird immer eine Spalte adressiert und über die Returnlines in das interne RAM zurückgelesen. Die Anzeige funktioniert nach dem Selben Prinzip. Mithilfe der Scanlines werden die 7-Segment-Anzeigen nacheinander adressiert und die jeweiligen Daten aus dem Anzeige-RAM darauf ausgegeben.

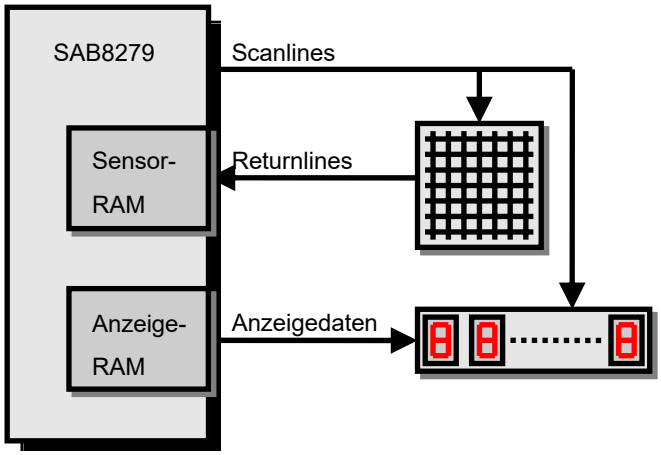


Abb. 2: Funktionsprinzip des Bausteins SAB8279

Der SAB8279 besitzt intern einige Konfigurations-register. Mithilfe dieser Register können die grundlegenden Funktionen des Bausteins auf die eigenen Anforderungen angepasst werden. Beispielsweise gibt es verschiedene Möglichkeiten der Auswertung einer angeschlossenen Tastatur. Zum einen können die Tastendrücke wie bei einer gewöhnliche PC Tastatur ausgewertet werden, das heißt, bei jedem Tastendruck wird der Tastencode in einem FIFO Speicher abgelegt, der vom Prozessor gelesen werden kann. Bei dieser Möglichkeit kann allerdings nichts über die Dauer und nichts über die Anzahl der momentan gedrückten Tasten ausgesagt werden. Bei der anderen Betriebsart dagegen ist das interne RAM als 8x8 Bit Feld organisiert, bei dem jedes Bit dem Zustand einer Taste entspricht. Bei dieser Art ist somit immer bekannt welche Taste(n) gerade gedrückt ist(sind).

Bei jedem Start der Steuerung werden diese Modi immer gleich initialisiert und ändern sich während des Betriebs nicht. Aus diesem Grund ist es nicht nötig die nicht verwendeten Modi in VHDL zu beschreiben. Solange sich das Modell des Bausteins gegenüber dem Prozessor wie das Original verhält ist für den Prozessor auch kein Unterschied feststellbar

2.2. Ermittlung der benötigten Funktionen

Um herauszufinden welche Funktionen des 8279 tatsächlich verwendet werden, und somit implementiert werden müssen, ist es nötig den Zustand der internen Konfigurationsregister zu kennen. Die einfachste Lösung wäre den Sourcecode der Steuerungssoftware nach der Initialisierung durchzusehen. Dieser ist allerdings in Papierform leider nicht mehr vollständig vorhanden und die Inbetriebnahme des alten Entwicklungssystems hätte einigen Aufwand bedeutet. Eine andere Möglichkeit ist die Aufzeichnung der Initialisierungssequenz mithilfe eines Logicanalyzers. Zum Einsatz kam hier das Modell TLA5203 der Firma Tektronix. Mithilfe der Initialisierungssequenz und dem Datenblatt des Bausteins ließen sich nun die benutzten Funktionen leicht bestimmen.

Drei wichtige Dinge werden zu Beginn Festgelegt: Die Eingabebetriebsart, die Ausgabebetriebsart und der Eingangstaktteiler. Die Eingabebetriebsart wird auf "Kodierte-Sensor-Matrix" eingestellt. Wie weiter oben beschrieben ist das RAM in dieser Betriebsart als 8x8 Bit Feld ausgelegt, bei dem jedes Bit den Zustand eines Schalters bzw. Tasters repräsentiert. Die andere Betriebsart, bei der das RAM als FIFO-Speicher organisiert wird, muss also nicht realisiert werden. Die Ausgabebetriebsart wird auf "sechzehn 8-bit-Zeichen-Anzeige" initialisiert. Das bedeutet, es können 16 Stellen mit jeweils 8 Bit angesteuert werden. Dies entspricht den verwendeten 7-Segment-Anzeigen mit zusätzlichem Dezimalpunkt. Schlussendlich wird der Taktvorteiler auf 31 initialisiert. Dieser Wert kann somit ebenfalls fest

programmiert werden und muss nicht konfigurierbar sein.

3. Realisierung in VHDL

3.1. Aufteilung in Module

Das Gesamte Projekt ist in 4 Module aufgeteilt:

- Busanbindung inkl. aller benötigten Register
- Timinggenerator
- Eingabeteil
- Ausgabeteil

In dem Busanbindungsteil wird der gesamte Datenaustausch mit dem Prozessor abgewickelt. Weiterhin werden dort die entsprechenden Kommandos verarbeitet, gegebenenfalls gespeichert und den anderen beiden Modulen zur Verfügung gestellt. Der Timinggenerator generiert aus dem 5MHz Taktsignal alle benötigten Signale wie z.B. die Scanlines oder das Dunkeltastsignal, und stellt sie den anderen Modulen zur Verfügung. Im Eingabeteil werden die Zustände der angeschlossenen Taster bzw. Schalter abgefragt und gespeichert. Weiterhin wird hier auch überprüft ob sich Veränderungen ergeben haben, und gegebenenfalls wird ein Interrupt Request ausgelöst. Der Ausgabeteil übernimmt die Aktualisierung der 7-Segment Anzeigen und löscht das Anzeige-RAM wenn das entsprechende Kommando vom Prozessor gesendet wird. Diese 4 Module werden in einem übergeordneten Modul zusammengefasst und bilden zusammen den Baustein SAB8279VHDL. Abb. 3 zeigt den modularen Aufbau des Projektes.

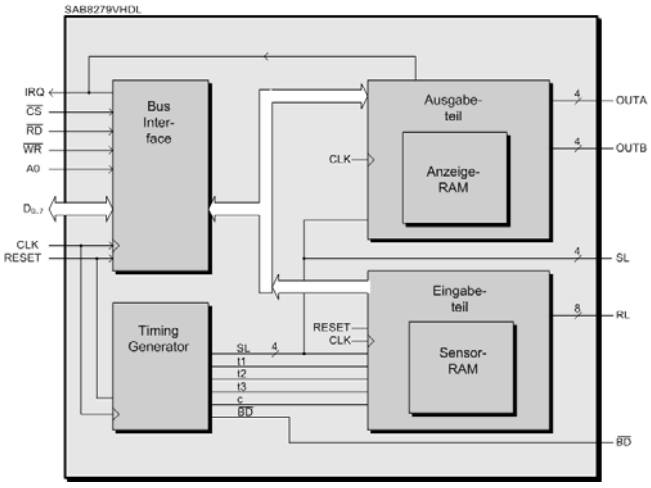


Abb. 3: Modularer Aufbau des VHDL-Projektes

Alle Module die ein Reset Signal benötigen werden mit einem synchronen Reset ausgestattet. In vielen VHDL Projekten die eigentlich streng synchron gehalten werden, wird ein asynchroner Reset eingesetzt. Dies kann allerdings zu teils schwerwiegenden Problemen führen: Zum einen können bereits sehr kurze Spikes einen Reset auslösen, oder noch schlimmer: Einen "halben" Reset. Dabei werden Teile der Register zurückgesetzt, andere bleiben aber gesetzt da der Resetpuls zu kurz war. Bei einem synchronen Reset kann dieses Verhalten nicht auftreten.

Der zweite Punkt weshalb in einem ansonsten synchronen Design kein asynchroner Reset verwendet werden sollte ist mehr FPGA spezifisch. Die FlipFlops auf einem XILINX FPGA können entweder synchron oder asynchron betrieben werden. Ein mix dieser beiden Anforderungen ist somit mit zusätzlicher Kombinatorik verbunden da die geforderte Funktionalität nachgebildet werden muss. Ein synchroner Reset trägt somit teilweise erheblich dazu bei die Anzahl benötigter Logikzellen zu minimieren.

3.2. Das Businterface

Über das Businterface kommuniziert der Baustein zum einen mit dem Prozessor, zum anderen wird die Kommunikation mit dem Eingabe- und dem Ausgabeteil abgewickelt. Ein Zugriff auf den Baustein wird immer dann erkannt, wenn, während /CS LOW ist, /WR eine steigende Flanke aufweist bzw. /RD LOW ist. Eine steigende Flanke an /WR leitet einen Schreibzugriff ein, ein LOW-Pegel an /RD einen Lesezugriff. Die Auswahl ob Daten oder ein Kommando/Zustandswort geschrieben bzw. gelesen werden, erfolgt über die einzige Adressleitung A0.

Um eine Flanke zu erkennen wird /WR kontinuierlich in ein 2 Bit breites Schieberegister eingelesen. Wenn nun das Schieberegister den Wert "01" annimmt, wurde eine steigende Flanke erkannt. Diese Art der Flankenerkennung ist zuverlässig, allerdings kommt dadurch auch eine Taktverzögerung zustande, da ja immer erst im folgenden Takt entschieden werden kann ob eine Flanke auftrat. Aus diesem Grund müssen die Werte die auf dem Datenbus anliegen ebenfalls zwischengespeichert werden, da die Werte die zur Taktflanke übernommen werden sollen, beim Erkennen der Flanke bereits nicht mehr gültig sind.

3.3. Der Timinggenerator

Der Timinggenerator stellt alle zum Betrieb nötigen Signale zur Verfügung. Als Eingang bekommt er nur den Eingangstakt und das Resetsignal. Der

Eingangstakt wird dann über einen Zähler um den Faktor 1984 heruntergeteilt. Dieser Faktor setzt sich aus dem programmierten Teilerfaktor 31 und einem festen Teilerfaktor von 64 zusammen. Mit diesem neu gewonnenen Takt werden dann die 4 Scanlines binär von 0 bis 15 durchgezählt. Das Signal /BD ist ein direkter Ausgang des Bausteins und dient dazu, die Anzeige während der Umschaltvorgänge dunkelzutasten, um ein Flackern der Anzeige zu verhindern. Die Signale t1, t2 und t3 werden zur internen Ablaufsteuerung benötigt. Abb. 4 Zeigt den Ablauf des erzeugten Timings.

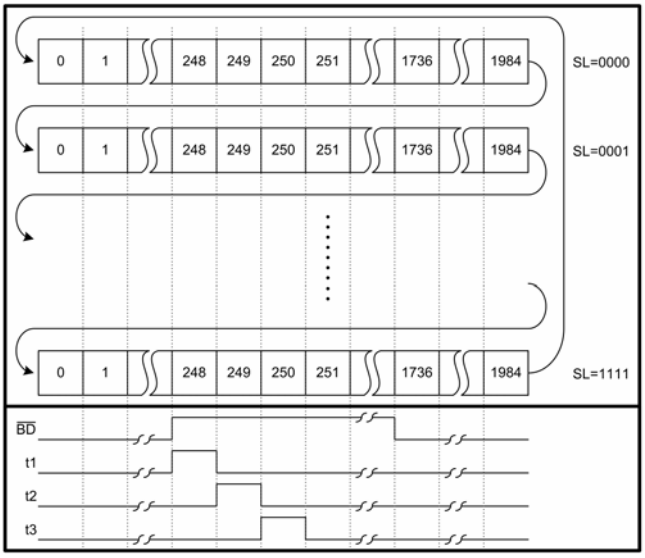


Abb. 4: Zeitlicher Ablauf der vom Timinggenerator erzeugten Signale

3.4. Der Eingabeteil

Im Eingabeteil werden die aktuellen Zustände der Tasten gespeichert und es erfolgt die Erkennung ob sich seit dem letzten Zyklus eine Taste verändert hat. Falls ein Tastendruck erkannt wird, wird ein IRQ ausgelöst, über den der Prozessor darüber in Kenntnis gesetzt wird. Der prinzipielle Ablauf ist in Abb. 5 skizziert.

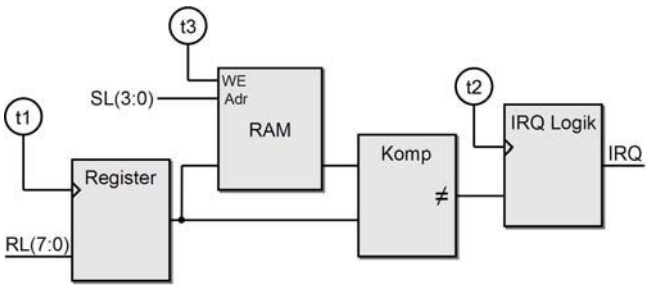


Abb. 5: Prinzipieller Aufbau des Eingabeteils

Die über die Scanlines ausgewählten Schalterstellungen werden zum Zeitpunkt t1 in einem Register zwischengespeichert. Zum Zeitpunkt t2 werden diese Daten mit den im RAM gespeicherten Daten verglichen. Sollten die Daten unterschiedlich sein ist ein Tastendruck erfolgt, und über die IRQ-Logik wird ein IRQ ausgegeben. Zum Zeitpunkt t3 werden dann die aktuellen Daten im RAM abgelegt. Anschließend werden die Scanlines inkrementiert und der Vorgang beginnt für die nächsten 8 Werte erneut.

Die Verbindung zwischen dem Businterface und dem Eingabeteil wurde hier aus Gründen der Übersichtlichkeit bewusst nicht eingezeichnet, der Prozessor kann aber jederzeit über das Businterface auf die Daten im RAM zugreifen.

4. Die verwendeten BlockRAM's

Auf den Spartan II FPGA's der Firma XILINX sind RAM's vorhanden die über eine entsprechende VHDL Beschreibung verwendet werden können. Diese RAM's sind in Organisation und Größe in gewissen Grenzen frei konfigurierbar. Ein weiterer Vorteil dieser RAM's ist, dass sie als echte dual-ported-RAM's ausgeführt sind. D.h. es gibt zwei getrennte Ports, auf denen gleichzeitig geschrieben und gelesen werden kann. Diese RAM's beherrschen allerdings keine Arbitrierung somit kann es zu Schreib-Schreib- wie auch zu Schreib-Lesekonflikten kommen. Ein Schreib-Schreibkonflikt ist leicht zu verstehen: Wird auf beiden Ports zur selben Zeit auf dieselbe Adresse geschrieben, sind die Daten die im RAM gespeichert werden ein Mix aus den beiden an den Port angelegten. Ein Schreib-Lesekonflikt ist dagegen nicht direkt ersichtlich: Wenn auf einem Port Daten in das RAM eingeschrieben werden und von derselben Adresse zur selben Zeit Daten von dem anderen Port gelesen werden wollen, können die erhaltenen Daten ebenfalls fehlerhaft sein. Die Daten am Ausgang des Ports auf den geschrieben wurde stimmen dagegen mit den geschriebenen überein. Nun kann dieser Fall aber durchaus vorkommen. Am Beispiel des Eingabeteils lässt sich dies verdeutlichen: Die Schalterstellungen werden zyklisch in das RAM geschrieben. Wenn nun der Prozessor zum gleichen Zeitpunkt Daten aus derselben Speicherzelle liest, tritt genau dieser Fall auf. Um zu verhindern, dass durch den Prozessor korrupte Daten gelesen werden lässt sich nun folgende Logik implementieren: Bei jeder Leseoperation des Prozessors aus dem RAM werden zuerst die Adressen verglichen. Sollten die beiden Adressen identisch sein, werden anstatt der möglicherweise fehlerhaften Daten einfach die Daten des momentan schreibenden Ports verwendet. Mit diesem Zusatz und der Tatsache, dass ein

synchrones Modell vorliegt, ist gewährleistet, dass die gelesenen Daten zu jeder Zeit verlässlich sind.

5. Die Adapterplatine

Ein weiterer Punkt dieser Arbeit war die Erstellung einer Testplatine mithilfe derer die erstellten Modelle im Feld getestet werden können. Ziel war es, eine Platine zu entwickeln, mithilfe derer die grundlegenden Funktionen des kompletten Digitalteils der Steuerung ersetzt werden können. Dazu zählt der Prozessor, der Multifunktionsbaustein SAB8256, die hier beschriebenen Tastatur- und Anzeigecontroller SAB8279, von denen 2 auf der Steuerung vorhanden sind, und das auf der Steuerung enthaltene RAM und ROM. Die Platine ist so ausgeführt, dass sie anstelle der Originalchips auf die Platine gesteckt werden kann. Abb. 6 zeigt den Digitalteil der Steuerung mit aufgesteckter Adapterplatine. Mit der Platine ist es nun möglich die Steuerung entweder mit den Originalbausteinen zu betreiben, indem diese einfach in die oben liegenden Sockel gesteckt werden, oder aber die Funktionalität der Bausteine über das vorhandene FPGA nachzubilden.

Die Nachbildung des RAM's erfolgt über ein auf der Rückseite verbautes SRAM. Damit das SRAM die Daten behält während die Steuerung ausgeschaltet ist, ist ein Nonvolatile RAM Controller verbaut, der das RAM während dieser Zeit über eine Batterie versorgt. Die Nachbildung des ROM's ist nur über einen Trick möglich: Der ROM-Inhalt wird zusätzlich zu dem Bitfile des FPGA's in das Konfigurations-PROM geschrieben. Nach dem Einschalten der Steuerung werden diese Daten dann in das erwähnte SRAM geladen, das die Steuerung dann wie das Original ROM ansprechen kann.

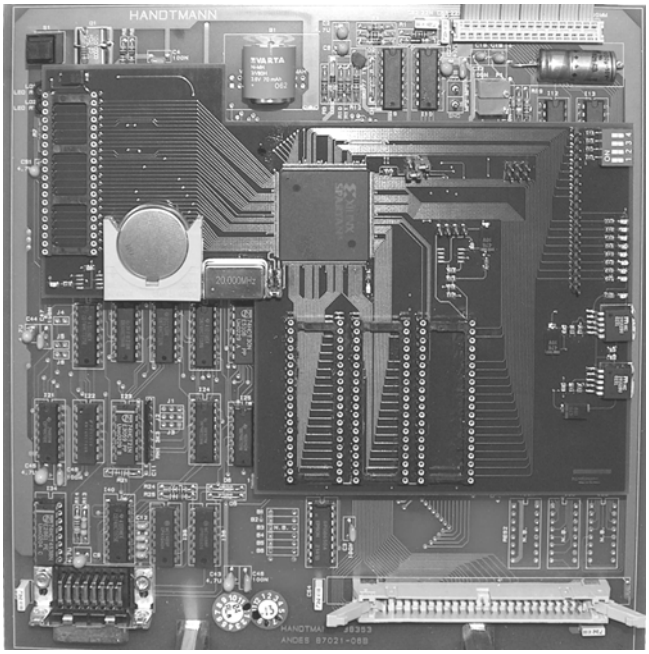


Abb. 6: Digitalteil der Steuerung mit aufgesteckter Testplatine

6. Ausblick

Sobald alle Modelle in VHDL realisiert sind, werden einige Steuerungen mit der Adapterplatine ausgestattet und im Feld auf Praxistauglichkeit geprüft. Sollten diese Tests erfolgreich verlaufen, wird mit dem Redesign des kompletten Digitalteils begonnen, auf dem möglichst nur noch Komponenten verbaut sind, die auch in absehbarer Zeit noch produziert werden.

Durch die ergriffenen Maßnahmen ist somit sichergestellt, dass die Steuerung auch noch in einigen Jahren als Ersatzteil geliefert werden kann.

7. Literatur

[1] 8279/8279-5 PROGRAMMABLE KEYBOARD / DISPLAY INTERFACE; intel; Sept. 1987

[2] XILINX TechXclusives; Get Your Priorities Right - Make your design up to 50% smaller; By Ken Chapman Senior Staff Engineer, Applications Specialist, Xilinx UK; Juli 2004

FachHochschule Offenburg
Hochschule für Technik, Wirtschaft und Medien

A Small Imprint RISC for Ubiquitous Systems for SOC Designs

Dirk Jansen

Hochschule Offenburg, ASIC Design Center

Tel:+49-781-205-267, -174, d.jansen@fh-offenburg.de

Für die Entwicklung komplexer SOC Bausteien ist ein Prozessorkern moderner Architektur unersetzlich. Als Ersatz für den schon 1993 entwickelten FHOP-Kern wird ein auf die Anforderungen der Hochsprache C optimierter SIRIUS- RISC – Kern mit 32 bit Struktur vorgestellt, der eine kompakte Bauweise mit hoher Leistung verbindet. Die verfügbare Peripherie soll weitgehend weiterverwendet werden können. Architekturkonzept, Entwurf des Instruktionssatzes sowie die Verbindung zum Compiler werden vorgestellt. Die Funktion des Prozessors wird durch eine Zyklusgenaue Simulation in einer angepassten Integrierten Entwicklungsumgebung sowie durch Simulation auf VHDL-Ebene mit ModelSim demonstriert.

(**S**mall **I**mprint **RI**SC for **U**biquitous Systems)

Abbildung 1: Logo des Prozessorkerns SIRIUS

1. Einführung: Warum ein neuer Prozessor – Kern?

Seit 1993 verfügt das ASIC Design Center der Hochschule Offenburg über den eigenentwickelten Prozessorkern FHOP, der mit 16 bit Datenwortbreite inzwischen in mehrere erfolgreiche ASIC Designs integriert wurde und auch derzeit noch durchaus für einfache Applikationen geeignet ist. Als wesentliches Problem dieses Prozessors ist die originale CISC Architektur anzusehen, die eine effektive Programmierung durch einen Hochsprachencompiler nicht zulässt. Hierzu trägt das Konzept des Akkumulator-Registers, die geringe Anzahl von verfügbaren Registern überhaupt sowie das Fehlen eines orthogonalen Befehlssatzes maßgeblich bei. Für die Befehlsverarbeitung werden im Durchschnitt beim FHOP etwa 4 Taktzyklen benötigt, da kein Pipelining vorgesehen ist. Die Steuerung erfolgt durch ein Mikrokode-ROM, was zwar sehr kompakt und flexibel ist, jedoch bei der Portierung auf neue Technologien Schwierigkeiten macht. Der erfolgreiche Ersatz des ROM durch ein synthetisiertes Gatternetz verbessert diese Situation zwar, jedoch ist dadurch die Komplexität erhöht und der Leistungsverbrauch auch nicht kleiner geworden.

Der Bedarf für einen modernen Nachfolger war also nicht länger abzuweisen. Untersuchungen an den im Internet angebotenen lizenzfreien Kernen für 8051, 8085 und als 32 bit Version vor allem des LEON 3 ergab keine befriedigende Lösung. Die frei verfügbaren 8 und 16 bit Kerne sind in der Leistung noch niedriger anzusetzen als der verfügbare FHOP-Kern. Der schon ziemlich perfekte LEON 3 ist als komplette SPARC – Architektur aber um 2 Größenordnungen zu groß und deshalb für die meisten Projekte ungeeignet. Der ARM 7 TDMI wäre zwar unter einer Europractice Lizenz für Forschung und Lehre verfügbar, eine Produktentwicklung mit diesem Kern aber auf Grund der Lizenzbedingungen ausgeschlossen. Zudem sollte die Tradition einer eigenen Prozessorentwicklung nicht zuletzt zur Unterstützung der Lehre fortgesetzt und die vom FHOP vorhandene und erprobten Peripherieeinheiten weiterverwendet werden.

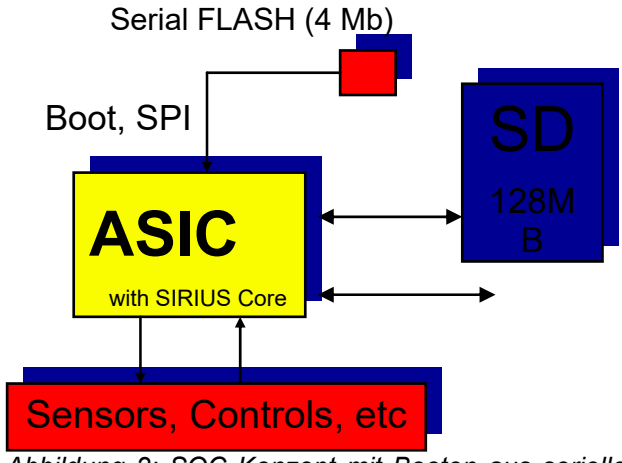


Abbildung 2: SOC Konzept mit Booten aus seriellem FLASH- Speicher

Die Forderungen an den Kern können schnell zusammengestellt werden:

- RISC Architektur mit internem Pipelining und einer 1 Instruktion/Zyklus Performance.
- 32 bit Architektur mit 32 bit breiten Daten und Adreßbussen.
- Orthogonaler Befehlssatz.
- Für die Hochsprache C optimierter Befehlssatz.
- Softcore in VHDL, d. h. als Macro wie als Netzliste in beliebige Technologien einfach zu portieren.
- Hardware Multiplier und Hardware-Shifter.
- Möglichst kompakt und Low-Power, einfaches Buskonzept und einfach mit der bestehenden Harware zu integrieren.
- 16 bit Instruktionsbreite.

Typische SOC Entwicklungen mit SIRIUS sollen nach dem in Abb. 2 dargestellten Konzept ein Booten aus einem seriellen Flash-Speicher ermöglichen, der z. B. auch eine SD-Karte sein kann. Damit können echte One-Chip Designs generiert werden.

2. Zuerst der Compiler

Die Vorgehensweise beim Entwurf muss also sein (Abb. 3):

- Erst der Compiler und die Entwicklungsumgebung,
- dann die Hardwareentwicklung in VHDL mit anschließender Synthese.

Von zentraler Bedeutung ist dabei der Entwurf des Befehlssatzes, der für eine effektive Umsetzung von C-Statements in Assembler Befehle maßgeblich ist. So bietet ein angepasster Befehlssatz die Chance, wichtige in der Hochsprache C beschriebene Algorithmen in kompakte Befehlsfolgen umzusetzen und damit nicht nur Speicherplatz zu sparen, sondern auch den Kode schneller zu verarbeiten. Hier gibt es große Unterschiede bei den bestehenden Prozessorgenerationen, wo sich großenteils aus Kompatibilitätsgründen umständliche Kodierungen finden lassen, die den Kode aufblähen und trotz hoher Taktfrequenz nur mäßige Bearbeitungsleistung bereitstellen.

Da der Kern auch weiterhin im Wesentlichen für Steuerungsaufgaben eingesetzt werden soll, steht die arithmetische Rechenleistung hinten an. Für Zwecke der Signalverarbeitung soll jedoch ein Hardware – Multiplizierer vorhanden sein, die Division wird mit Software erledigt.

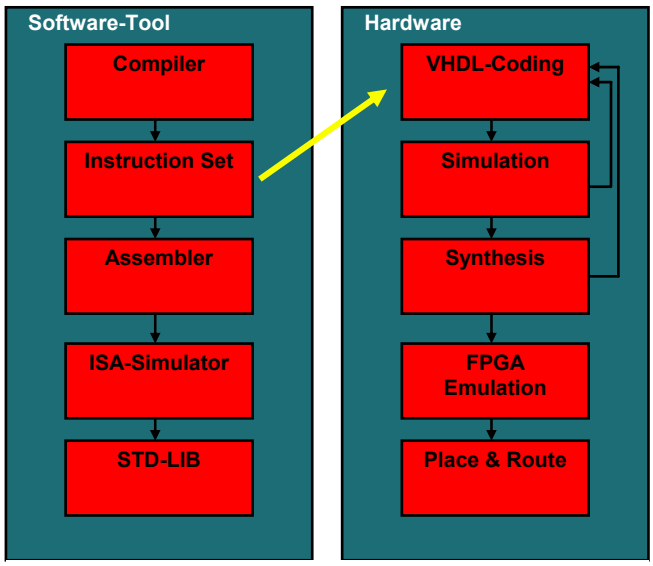


Abbildung 3 : Designflow bei der Entwicklung von SIRIUS

2.1. Forderungen der Hochsprache C an den Befehlssatz

Die Hochsprache C [1] ist die für “Embedded Systems” am weitesten verbreitete Sprache. Zudem sind praktisch alle Algorithmen, Programmsammlungen und Betriebssysteme in C verfügbar, nicht zuletzt ist C die Sprache von UNIX und LINUX. Für eine volle C Kompatibilität muss also das ANSI-C, so wie es standardisiert ist, sowie die zugehörigen Bibliotheken, insbesondere die C- Std-Lib umgesetzt werden können. Für volle LINUX- Kompatibilität muss zudem ein ausreichend großer Speicher (> 4 Megabyte) linear ansprechbar sein. Innerhalb eines Chips kann mit den uns zur Verfügung stehenden Technologien ein solcher Speicher derzeit nicht realisiert werden, er kann nur außerhalb des Chips angeordnet werden. Damit wäre auch eine Memory Management Unit und eventuell eine Ansteuerung für Dynamische Speicher zu entwickeln, was den Rahmen sprengt und auch aus Gründen des Leistungsverbrauchs für die Zielsysteme nicht in Frage kommt. Auf LINUX muss deshalb in absehbarer Zeit verzichtet werden, jedoch soll das Potential dafür in SIRIUS schon angelegt sein.

Die Sprache C verwendet für das Speichermodell folgende Elemente:

- Globale Variablen (initialisiert oder nicht initialisiert).
- Lokale Variablen (initialisiert oder nicht initialisiert).
- Dynamisch zuweisbarer Speicher auf dem „Heap“, einem von der Software über Pointer und Listen verwaltetem Speicherbereich.

Ferner beruht C auf dem Konzept der „Function“, d. h. alle Programme sind in Teilmodule zerlegt, die in hierarchischer Weise verknüpft alle Kind einer Hauptfunktion „Main“ sind. Das Funktionskonzept und auch das Konzept lokaler Variablen beruhen auf der Idee des Stack-Speichers, der über einen dynamischen Zeiger verwaltet, lokale Variablen, Rücksprungadressen sowie Parameter beim Funktionsaufruf aufnehmen kann. Ein Stack erlaubt zudem unbegrenzte Verschachtelungstiefe bei Unterfunktionen. Es ist für die Architektur von SIRIUS wichtig, das Konzept des Stacks als Zwischenspeicher und die Verwaltung lokaler Variablen konsequent hardwaremäßig durch geeignete Befehle und ein Stackpointer- Register zu unterstützen.

Die dynamische Speicherverwaltung des Heaps steht daneben als eigenes System, ebenso wie die damit verwandte Dateiverwaltung, und setzt als reine Software-Konzepte keine Hardwareunterstützung voraus.

2.2. Text Anpassung des LCC an SIRIUS

Als Basis für den Compiler wurde der von Frazer und Handson [2] entwickelte und relativ einfach zu portierende „Little C Compiler“ LCC verwendet, bei dem es sich um einen Cross-Compiler unter Windows handelt Abb. 6.

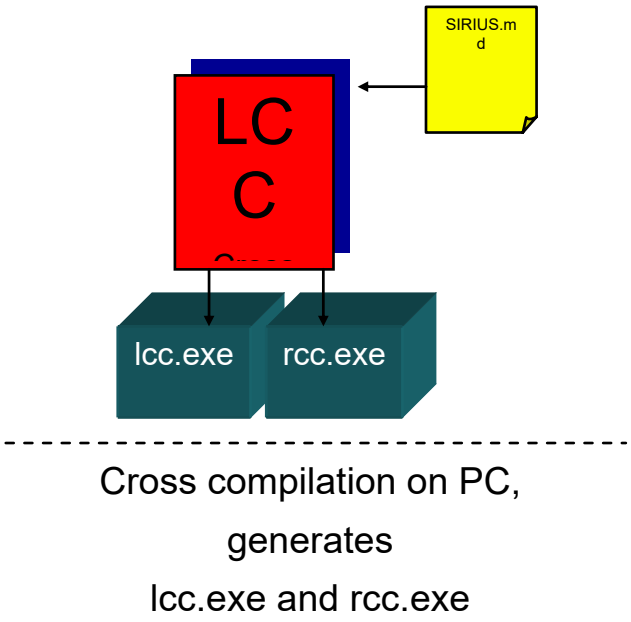


Abbildung 6: Cross Compiler LCC

Der LCC, bestehend aus den ausführbaren Programmen lcc.exe und rcc.exe, die in einem Compiliervorgang zusammen mit einem Machine-Description File sirius.md erzeugt werden. Der Compiler verfügt dabei über voll ANSI-C Kompatibilität und erzeugt direkt den Assemblerkode, der dann in einem 2. Schritt mit dem Crash – Assembler [2] noch zu lauffähigem Maschinenkode umgesetzt werden muss.

Load/store	Control	Arithm.	Logic
ASGN	JUMP	ADD	BAND
INDIR	CAL	SUB	BOR
ARG	RET	LSH	BXOR
LOAD	EQ	RSH	BCOM
VREGP	NE	MUL	
LABEL	GE	DIV	
CNST	GT	MOD	
	LE	NEG	
	LT	CV	

Abbildung 4: LCC-DAGs nach Frazer and Hanson

LCC verwendet einen C-Parser, der die Sprache des C-Kodes zunächst in eine Metasprache aus Directed Acyclic Graphs (DAG) zerlegt und daraus die Statements als Baumstruktur aufbaut, wobei in den Verknüpfungen die DAGs, in den Zweigen Zwischenspeicher (Register) angeordnet sind. Ein C-Statement bildet sich damit eindeutig ab auf einen DAG-Tree. Frazer und Handson benötigen nur exakt 29 unterschiedliche DAG Operatoren, um die gesamte C-Sprache abzubilden.

Type	name
U	unsigned
I	signed
F	float
P	pointer
B	Block
V	Void

Abbildung 5: Typen der DAG Operatoren

Die Operatoren können wiederum auf 6 unterschiedliche Datentypen arbeiten Abb. 5, wobei diese wiederum in den Wortbreiten 8 bit, 16 bit, 32 bit usw. variieren können. Zu den ersten Festlegungen beim Entwurf des Compilers gehört also die Entscheidung, welche Datentypen überhaupt und in welcher Auflösung bearbeitet werden sollen. Für Sirius wurde die Wortbreite auf 32 bit maximal definiert, d. h. „double float“ und „long long“ werden nicht realisiert.

Eine ähnliche Entscheidung ist für die Breite des Address-Pointers zu treffen. Für Sirius wurde in der Basisversion der Pointer auf 16 bit definiert, allerdings läßt der Befehlssatz auch 32 bit Pointer zu. Um dies zu nutzen, ist jedoch ein modifizierter Compiler erforderlich, der zunächst zurückgestellt wurde, da Speicher von größer 64kRAM vorerst nicht integriert werden sollen. Allerdings soll die Hardware hier nicht begrenzen, die entsprechenden Maßnahmen sind also bereits in den Kern integriert.

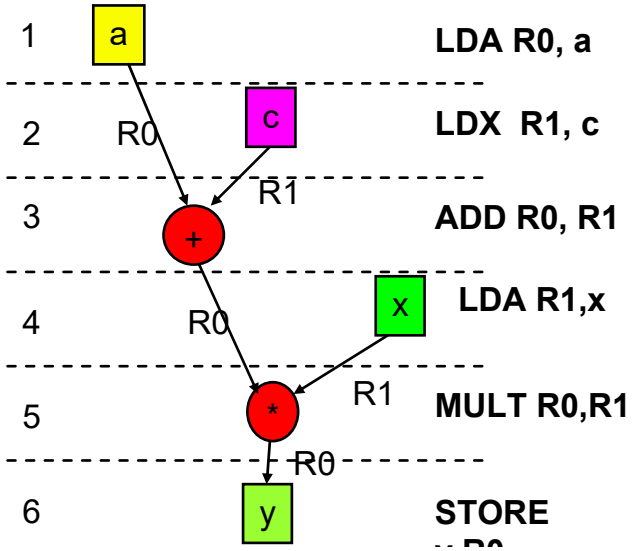
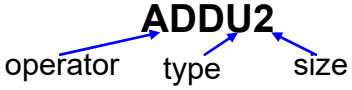


Abbildung 7: DAG Tree für ein einfaches C - Statement.

Abb. 7 zeigt den DAG-Tree für ein einfaches C-Statement. Wegen der Typabhängigkeit und der Wortbreite setzt sich ein DAG Operator aus drei Teilen zusammen:



Die „Rules“ für die Operatoren sind alle in dem MD – File zu finden, ebenso wie spezieller Kode zur Behandlung der Speicherbereiche wie auch zur Bildung der Funktionsaufrufe.

Die Funktionalität des Compilers konnte inzwischen durch erfolgreiche Übersetzung der C - STD- Lib sowie entsprechender Testfiles für den Compiler komplett verifiziert werden.

2.3. Definition des Befehlssatzes

Der Befehlssatz beschreibt letztlich die Befehle, die zur Realisierung der DAG-Operationen in der Hardware erforderlich sind. Es liegt nahe, sich hier an den Operatoren selbst zu orientieren, da dann das „Überdecken“ des Baums („Coverage“) mit Maschineninstruktionen am einfachsten fällt. Da die Zweige der Darstellung Zwischenspeichern bzw. Registern entsprechen, läßt sich auch hier die Anzahl und die Zuweisung der Register (Register- Allocation) organisieren. Auch das ist umso einfacher, je weniger Nebenbedingungen zu erfüllen sind. Hieraus folgt die Anforderung an die Gleichartigkeit der Register („orthogonaler Befehlssatz“), d. h. ein klassischer Akkumulator als bevorzugter Sammler von Ergebnissen widerspricht diesem Konzept.

Die Befehle orientieren sich deshalb direkt an den DAG-Operatoren, die Frage der Bearbeitungsbreite (16 oder 32 bit) wird durch ein zusätzliches in der Kodierung enthaltenes Attribut geregelt. Alle Befehle folgen einem 16 bit Format mit Ausnahme der Immediate Befehle, die weitere 16 bit als Konstante Extension mitführen. Der Kode ist durch das Vermeiden absoluter Adressen relocatable. Die komplette Darstellung aller Befehle sprengt den hier gesetzten Rahmen und kann bei uns angefragt werden.

2.4. Anpassung des konfigurierbaren CRASH – Assemblers an den Sirius Befehlssatz

Der LCC_Sirius-Compiler erzeugt Assemblerkode, der bereits geprüft und beurteilt werden kann. Für eine zyklengenaue Simulation ist jedoch Maschinenkode erforderlich (binär oder hex). Hierfür wird das vom Compiler erstellte File einem konfigurierbarem Assembler (CRASH) [3] zugeführt. Die Konfiguration des Assemblers erfolgt über einen Konfigurationsfile sirius.bef, der alle Informationen zur Umsetzung enthält. Als Ergebnis liefert er einen File im Intel-hex-Format, der zur Programmierung des Speichers, für die Simulation auf VHDL- Ebene mit MODELSIM wie auch für den integrierten SIRIUS- Simulator verwendet werden kann.

2.5. Aufbau einer integrierten Entwicklungsumgebung für Sirius

Alle Programme sind in einer integrierten Entwicklungsumgebung „IDE“ zusammengefasst, die

beginnend mit einem Editor die Erstellung der C-Programme unterstützt, den C-Compiler aufruft, den Assembler aktiviert und nicht zuletzt mit dem integrierten Simulator ein komfortables Debugging des Programms ermöglicht.

Im Simulationsmode Abb. 8 können gleichzeitig der Programmtext, der Inhalt der Register und Speicher sowie das Ergebnis der abgelaufenen Befehle angezeigt werden. Das Programm kann schrittweise durchgesteppt oder kontinuierlich ausgeführt werden. Als Besonderheit ist noch die DDE – Schnittstelle zu erwähnen, mit der externe Programme, z. B. zur Simulation von externen Komponenten, angeflanscht werden können. Damit können auch komplexe Abläufe mit Ein-/Ausgaben sowie Hardware-Interrupts nachgebildet werden.

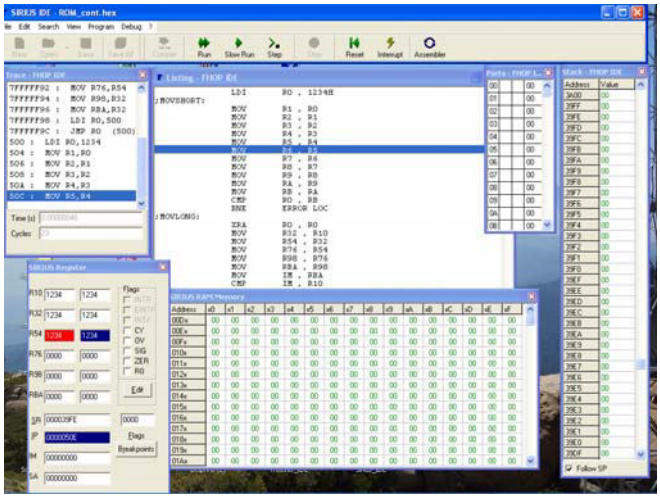


Abbildung 8: Sirius-IDE mit integriertem Zyklus-genauen Simulator

Die Entwicklungsumgebung, weiterentwickelt aus den Versionen für FHOP und FHOENIX, ist voll funktionsfähig für den Sirius und erlaubt, komplette C-Programme für SIRIUS zu übersetzten, zu simulieren und damit sowohl den Befehlssatz als auch die gesamte Verarbeitungskette einschließlich Compiler zu verifizieren. Mit der IDE wurde nicht zuletzt auch der Befehlssatz soweit ausgereift, dass er festgeschrieben werden konnte. Ein Testfile, der die richtige Bearbeitung aller Befehle nachweist und bei Fehlern zu einem Abbruch führt, wurde erstellt. Dieser File wurde auch zum Test des VHDL Kodes bei der Digitalsimulation verwendet und beweist die Übereinstimmung von Simulator und der durch VHDL beschriebenen Hardware.

3. Prozessor Architektur

Nachdem der Instruktion Set sowie die Registerarchitektur festgelegt worden war, konnte mit der Kodierung in VHDL begonnen werden.

Im Groben besteht die Architektur aus

- Dem Registerfile mit 12 general purpose 16 bit Registern R0 .. RB, die auch zu 32 bit Registern R10, R32 …RBA kombiniert werden können, und 4 special 32 bit Registern SP, IP, IM, SA,
- einer durchgehend 32 bit breiten ALU mit ADDER/SUBTRACTOR, Logischer Einheit und Barrel- Shifter,
- einem zur ALU gehörenden 16 x 16 bit Multiplizierer,
- einer Steuereinheit mit Befehlsdekodierung und Sequenzierung und
- einer Adresseinheit zur Berechnung der Adresse sowohl für die Befehls-Incrementierung wie auch für den Speicherzugriff.

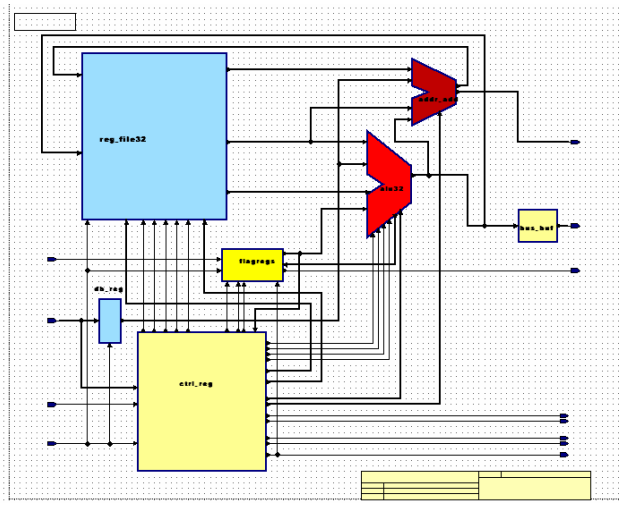


Abbildung 9: Blockbild des SIRIUS-Kerns

Der Zugriff zum Speicher erfolgt in der aktuellen Version über einen 16 bit Datenbus, eine Erweiterung auf 32 bit ist angedacht. Die Beschränkung auf 16 bit Busbandbreite erfolgte, um die Alignment- Anforderungen klein zu halten und den Speicher in 16 bit Breite realisieren zu können, was für die Integration von Vorteil ist. Sollte in Zukunft noch ein Cache verfügbar werden, kann die Busbreite auf 32 bit erhöht werden, ohne dass völlig neue Konzepte zu beschreiten sind. Nachteil der 16 bit Busbreite ist, dass Load/Store Operationen 3 Takte, immediate

Operationen 2 Takte benötigen. Alle übrigen Operationen werden in nur 1 Takt bearbeitet.

Die Pipeline verfügt über die Stufen

- Fetch (Instruktion vom Speicher holen),
- Decode (Befehl interpretieren),
- Execute (Befehl ausführen).

Die Phase „Write Back“ kann entfallen, da als Register DFFs verwendet werden, kein DP-RAM, was erstens schneller funktioniert und zweitens auch weniger Leistung aufnimmt. Allerdings benötigen Register-Fliflops mehr Platz, sodass mit den Bits gegeizt werden musste. Es wurde deshalb kein sonst üblicher 32 x 32 bit Registersatz integriert.

Ohne „Write Back“ gibt es auch die Problem mit Daten-Abhängigkeiten nicht mehr, es müssen also keine „NOP“ Befehle eingefügt und keine „Forwarding“ Maßnahmen in der ALU vorgesehen werden. Dies vereinfacht den Design signifikant. Auch die Interrupt- Fähigkeit wird dadurch maßgeblich verbessert und es müssen keine Sonderfälle abgefangen werden.

Das Ziel des Designs ist dabei eine kleine Zielfläche bei gleichzeitig hoher Performance (small imprint), Kompromisse sind immer erforderlich.

3.1. Kodierung in VHDL

Der Kern wurde komplett in VHDL kodiert. Hierfür wurde zunächst ein reiner „behavior“ – Stil verwendet, wobei die effektive Synthese zurückgestellt wurde. Auf Top-Ebene wurde der Kern mit Peripherie-Einheiten kombiniert, wie Sie schon vom FHOENIX [4] her verfügbar waren, sodass eine komplette SOC-Umgebung in der Hardware-Simulation aufgebaut werden konnte (Abb. 10). Auf dieser Umgebung konnte nun in den Speicher der Testfile im Hex-Format geladen werden, der bereits schon zuvor in der IDE entwickelt worden war. Der dazu verwendete RAM-Kode verfügt über einen passenden Hex-Loader.

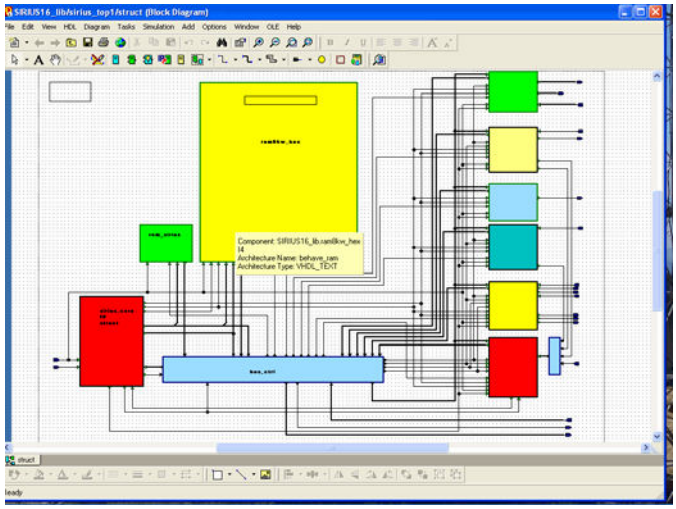


Abbildung 10: System-Testumgebung im SOC - Stil mit Peripherieeinheiten aus dem FHOENIX - Projekt.

In der Top-Umgebung sind auch Interrupt-Controller, Serial-IO, Parallel-IO usw. eingebunden, sodass auch die Kommunikation über die serielle Schnittstelle simuliert werden kann. Das verwendete Bussystem eignet sich direkt für eine FPGA-Emulation, die als einer der nächsten Schritte erfolgen soll.

3.2. Verifikation durch Simulation

Auf der VHDL Ebene wurden der Befehlsablauf, die Pipeline-Steuerung sowie der richtige Ablauf aller Befehle untersucht und nachgewiesen (Abb. 11).

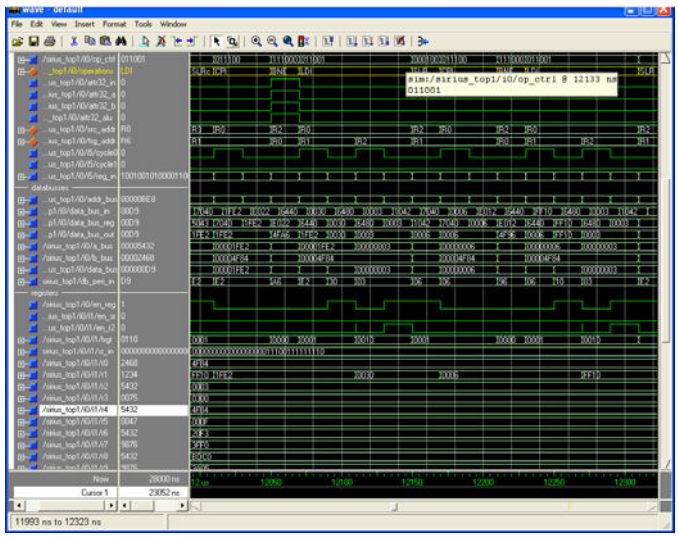


Abbildung 11: Simulation des Befehlsablaufs mit ModelSim

Dabei wurden alle Befehle einzeln und in Verbindung miteinander simuliert und die Abläufe optimiert. Erste Timing-Abschätzungen ergeben eine maximale Taktfrequenz von 50 MHz für FPGA (ALTERA-Cyclon), was im ASIC Kern mit Std-Zellen etwa 100

MHz erwarten lässt. Das würde einer Performance von knapp 100 MIPS entsprechen, was mehr als genug für die in naher Zukunft anstehenden Projekte sein wird.

In einem nächsten Schritt ist, nach Optimierung der VHDL – Kodes auf die verfügbaren Libraries und Synthesekonstrukte, eine Emulation auf FPGA durchzuführen, die zudem einen umfangreicheren Test der Software in Echtzeit ermöglicht. In Zusammenspiel mit Simulator, IDE und Compiler kann damit der Instruktionssatz weiter verfeinert und optimiert werden. Im letzten Schritt ist dann ein Standardzellen Design vorgesehen, um auch die Fläche in Silizium zu ermitteln. Als Zieltechnologie ist heute AMI 0.35 u geplant, morgen soll aber bereits eine Portierung auf UMC 1.8 u und darunter untersucht werden. Da der Kern keine Hard-Macros wie DP-Speicher enthält und auch sonst keine besonderen Forderungen an die Synthese stellt, ist eine Portierung auf beliebige Technologien relativ problemlos.

4. Ergebnisse

Die Entwicklung ist erst nach Vorliegen eines Testchips wirklich abgeschlossen. Die bereits vorliegenden Ergebnisse sind vielversprechend. Bis zur vollen Ausreifung auch unter Flächen- und Leistungsverbrauchsgesichtpunkten ist jedoch noch ein längerer Weg (Abb. 12). Es ist vorgesehen, den Prozessor wie auch zuvor den FHOP unter einer offenen Lizenz frei zugänglich zu machen.

Position	Dimen.	Value	Remarks
clock	MHz	100	Estimated on 0.35
D_BUS	bits	32	to internal/ext. RAM
ADDR_BUS	bits	32	
Periph_Bus	bits	8	To periphery modules
pipelining	stages	3	Including memory reg.
Reg./Reg.	cycles	1	ADD,SUB, INC, INV…
Load/Store	cycles	3	Data operations
Immediate	cycles	2	Local constant
interrupt	cycles	5	External event
Branch/jump	cycles	2	Relative to IP/ register
Address space	kBytes	64	Tiny mode, inside one segment
Address space	MBytes	3200	Full OS mode
Booting, reset	address	FFFFFF80	First instruction is fetched from address

Abbildung 12: Technische Daten des Sirius-Kerns

5. Referenzen

[1] American National Standard for Information Systems, Programming Language C ANSI X3.159-1989, American National Standards Institute, Inc., New York, 1990.

[2] Christopher W. Fraser, David R. Hanson, A Retargetable C Compiler: Design and Implementation, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, 1995.

[3] Daniel Vogel: CRASH – Assembler.

[4] Jansen, Dirk: Entwurf des RISC-Kerns FHOENIX zur Integration in SOC Designs. Vortrag auf dem MPC Workshop in Albstadt-Sigmaringen, Juli 2004.

Low Power Features in Design and Technology for Mobile Application Chips

Knut M. Just

Infineon Technologies AG, 81726 Munich, Germany

knut.just@infineon.com

Today's devices for mobile applications, such as GSM / UMTS baseband chips, demand high processing performance in terms of processor speed and low power (LP) dissipation as well. However, the performance demands vary strongly, depending on phone modes and activities e.g. stand-by mode vs. talk mode vs. high performance application modes such as video processing and gaming.

In recent years several low power features have been developed in logic design and silicon technology (of 90nm and below) to address both the static leakage power consumption and the dynamic active power consumption. These features, or a combination of them, can be tailored to dynamically varying performance needs of the chip in different modes meaning different use cases.

One challenge of now and the future is to optimize the timing, power, and area for different modes simultaneously with the EDA tools for RTL logic synthesis, timing driven layout, and static timing analysis. Another challenge is to verify the correct implementation of all low power features in simulation, structural netlist checks, and at the production tester.

1. General Trends in SoC Design

Systems-on-chip (SOC) designs in deep-sub-micron (DSM) technologies create big issues that cause the power influenza.

Modern chips are highly integrated systems since several formerly separated chips are integrated into one. This reduces the number of parts of the device and therefore cuts down the bill of material. Additionally each design block has an increased functionality due to the higher number of functions and higher number of cores.

The shrink of technologies further scales down feature sizes and physics of small dimensions reaches regions where single dopant atoms can be counted and cause statistical deviations from wafer-to-wafer and die-to-die. Within-die deviations cause two transistors to be different even if they lie next to each other.

Besides simple talk mode of mobile phones other applications, like fast data transfer, video streaming and gaming show enormous performance greed.

But high power requirements are firstly difficult to dispose and cause package issues since no cooling fan is available in hand-held devices. Secondly, the supply of power is limited due to battery capacities.

While Moore's law still shows a straight increase in area density doubling every 1.5 years, the trend in low power versions of technologies shows bent lines. Figure 1 shows qualitatively the deviation from a straight line for the design complexity increase, clock frequency boost, supply voltage reduction but also the leakage current explosion.

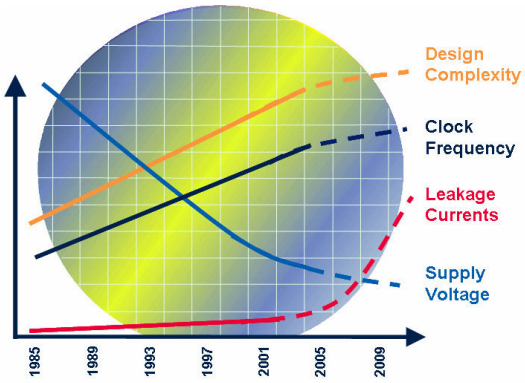


Figure 1 Do we see the end of Moore's law?

2. LP Features for Power Reduction

2.1. Everything you always wanted to know about "low power" but were afraid to ask

A) What targets will future applications have?

Today's and future's applications will have completely contradicting targets. On one hand data processing needs highest processor performance and on the

other hand long stand-by times require lowest power dissipation.

B) What is the key to a low power solution?

The key to the solution is concealed in those same contradicting targets of strongly varying performance demands in different application modes.

C) How does the solution look like?

The multi-fold solution is a combination of all low power features, tailored to the dynamically varying performance needs in different modes called use cases.

D) What challenges do we need to tackle?

The implementation challenge is to optimize timing, power, and area with EDA tools for logic synthesis, timing driven layout, and static timing analysis simultaneously at different modes.

The verification challenge now includes scrutinizing the correct implementation of all low power features at simulation, structural power/ground netlist checks, and production test.

2.2. Power equation parameters

Take a look at the power equation to find the parameters that can be influenced to reduce power. The equation for active and leakage power looks as follows:

$$P = \alpha \cdot C_L \cdot f \cdot V_{dd}^2 + I_{leak} \cdot V_{dd} \qquad \text{(EQ1)}$$

with switching activity α, load capacity C_L, operating frequency f, supply voltage Vdd, and leakage current I_{leak} (the different parameters are illustrated in Figure 2).

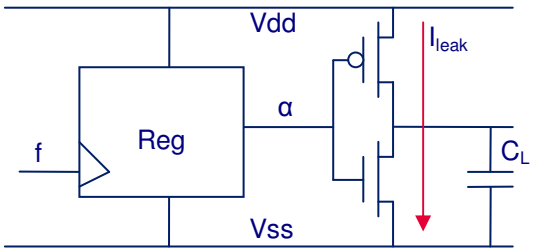


Figure 2 Power equation parameters

2.3. Power reduction features

In an active design block, glitch reduction can lower the activity of signal changes. While this feature is limited in leverage, clock gating is the most effective way to stop signal changes in an inactive block. Clock gating shall be done hierarchically, starting on clock inputs of top level super-blocks and going down to EDA tool supported gating of enable flip-flops.

The capacitive load is addressed by low-k metal stack design in technology and short wire lengths designed by timing-driven place and route in layout.

In all modes where the maximum performance is not required, frequency is scaled down under S/W control, thereby enabling dynamic voltage scaling (DVS), a voltage steering methodology, or even adaptive scaling (AVS) in a closed loop control of the supply voltage (see Chapter 2.4).

Leakage currents have to be addressed by optimum device design compromising transistor threshold voltage for an optimum combination of high on-currents for performance and low off-currents. Advanced libraries have both low-Vt cells for high-speed critical paths and high-Vt cells for low power in a mixed-Vt synthesis approach. Whenever a block is inactive over a longer time period power switches are used to turn-off leakage.

2.4. Frequency and voltage scaling

Dynamic voltage scaling (DVS) reads look-up-table (LUT) entries of pre-defined voltages for each combination of frequency, process, and temperature.

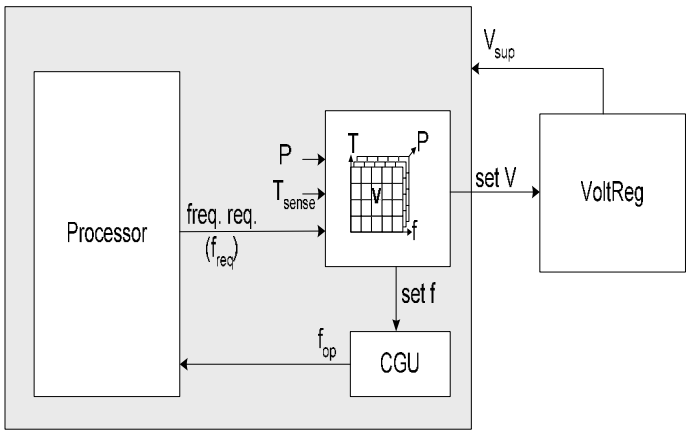


Figure 3 Open loop voltage setting in DVS

When the system requires a new frequency for a design block like a processor the correct voltage is read from the LUT subject to the die's process class (PC) and temperature. The process class might be determined with an on-chip process monitor being one or several oscillators which frequencies are sensitive to the silicon's speed.

If needed, the power control unit causes the voltage regulator to change the voltage, wait for the change and then causes the clock generation unit to change the frequency.

Adaptive voltage scaling (AVS) evaluates an on-chip performance monitor that is sensitive not only to process speed (P), supply voltage value and power grid voltage drops (V), and temperature (T) but also to the actually applied clock frequency.

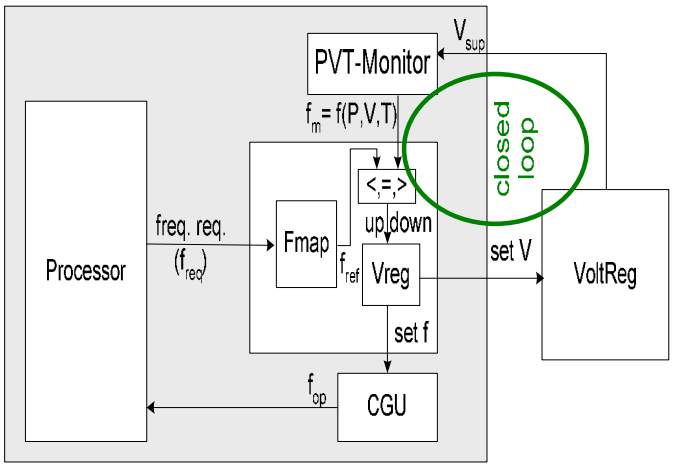


Figure 4 Closed loop voltage control in AVS

Figure 4 shows the closed loop AVS control where the control unit compares the output of the performance PVT-monitor with the required frequency to figure out whether the voltage must be raised or can be lowered. The command to the voltage regulator changes the voltage thus influencing the PVT-monitor's frequency.

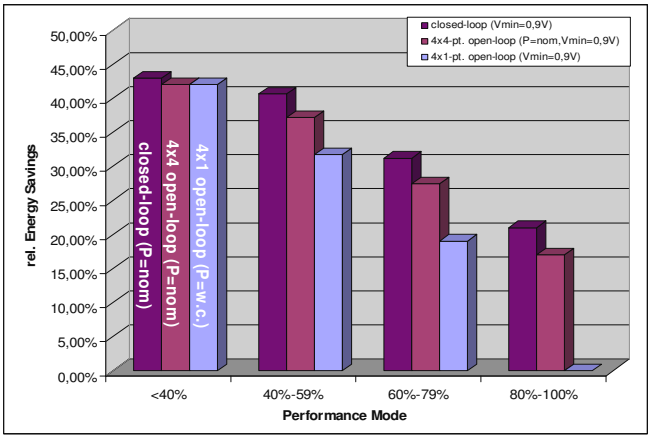


Figure 5 Average energy saving of AVS, 4-PC DVS and w.c. DVS over 4 performance modes for a nominal process (medium PC)

When the system requires a lower frequency f_{req} the PCU just needs to apply it. The correct voltage is found automatically by the closed loop control subject to the process speed, temperature and other influencing effects. The prerequisite is that all changes are slower than the control loop's time constant.

Before increasing the clock frequency the voltage is set to maximum then letting the control loop find the optimum voltage again.

Figure 5 shows the average energy saving of 3 different methodologies over 4 performance levels:

1. Worst case process DVS (4x1 LUT at w.c. PC) does not consider the process class so no saving can be achieved in the 100% performance mode.
2. 4-PC DVS (4x4 LUT) considers four steps of process class so even in the 100% performance mode savings of about 16% can be achieved with a nominal process.
3. AVS adds on average 4% points to 4-PC DVS.

At 40% performance and below, all methods show equivalent savings since a minimum voltage limit of 0.9V is assumed for the process and no further voltage reduction is allowed for correct functionality of logic and memories.

An example for the benefit is the extended video replay time of digital mobile TV (DVB-H) (assumed is 36% power saving at performance level of 40% to 60% and a power share of 55% of the chip in the system).

$\Delta t / t = 1 / (1 - (36\% * 55\%)) = 1.25$

This gives +25% extended time or 15 minutes extension per hour. So, if you watch the soccer championship this year you get headroom to watch the extra time if needed. We still work on featuring the penalties, too.

3. LP Requirements on EDA Tools

The complexity of the design flow increased tremendously from a 180nm design to a 65nm design. The largest contributing sub-flow is the timing verification by a factor of 17x. An increase of timing corners in the sign-off flow needs new methodologies in timing sign-off process. The number of technology design rules rose by a factor of 9x, and the physical implementation flow (with DSM effects) by 6x. The tools' run times contradict to the short development cycle times.

3.1. Tool issues for logic implementation

Constraint prioritization now also includes power. While timing is still the most important constraint that cannot be compromised, power reduction has become the second important optimization criterion before area reduction. And power features definitely need area!

Place & route during layout generation is no longer only timing-driven but power-driven, too.

The largest challenge, which still lacks an EDA solution, is multi-mode optimization for timing. The multiple modes of operation have different clock frequency constraints at different voltages. Unfortunately, there is not only one single worst case mode since the voltage in each mode is reduced as much as possible, creating new critical modes. Individual constraints for each mode with clock and PVT (process, voltage, temperature) settings need a simultaneous optimization of several modes to stop the ping-pong effect of endless iterations.

3.2. Content of EDA tool support

Requirements on tools comprise voltage awareness in library selection on PVT, place & route clock tree synthesis, and delay calculation. Voltage islands in designs build multi-voltage regions with block shut-off that needs level shifters and isolation logic insertion. Already available is multi-Vt logic synthesis.

Correct implementation needs analysis tools for static and dynamic power consumption as well as static and dynamic voltage drop analysis to evaluate maximum values and support power grid dimensioning.

Thorough verification of power shut-off must be handled during simulation for correct HDL coding and requires structural checking tools for power routing.

4. Applying LP Features in Software

An example is the behavior of a mobile phone in paging (standby) mode. About every second it has to decode 4 bursts from the base station to look for a call.

At first a hardware pre-wakeup trigger gets active and the ARM processor leaves wait-for-interrupt state (WFI). The software puts the system into idle mode. When the first burst is due, the software puts the system in normal mode. Then, all bursts are received, decoded and treated before ARM enters WFI state again. Finally, the software puts the system in deep sleep again and the paging period is over.

During this wake-up phase other inevitable actions are done, too, like monitoring of neighboring cells, battery and temperature measurements, display update for the watch, and a location update is transmitted every hour.

Some principles for improving power consumption e.g. in mobile phone standby mode with paging are to keep the wake-up periods as short as possible, to go into deep sleep between paging (at a low clock of 32 kHz), and avoid any unnecessary wake-up. The system operates at the lowest possible frequencies for cores and busses. Every design block, component, or bus, which is currently not in use, is switched-off.

5. Conclusion

Future designs will be larger, need more performance, and nevertheless have to consume less power. These given constraints force methodologies.

The key to a low power solution lies in the varying performance demands in different application modes. The solution itself must tailor the combination of low power features for each different application mode.

While low power features are available for use in designs, a lot of challenges remain; like optimizing timing, power, and area at different modes simultaneously. Also, the use of multi-voltage domains and power shut-off is not sufficiently supported in layout tools.

Finally, the verification of the correct implementation of all low power features in simulation, logic synthesis and layout currently needs proprietary solutions.

Investigations into Ethernet and Realtime-Ethernet Interfaces

Pei-Ching Wu
Institut für Mikroelektronik Stuttgart, Allmandring 30a, 70569 Stuttgart
Telefon 0 711 / 21 855 -200, Fax -222, E-Mail uwiep@yahoo.com

The Ethernet interface is investigated for use in custom devices. The Opencores Ethernet MAC core is researched, customized, and implemented on an FPGA platform. The open-source RTNet package is discussed as an extension to enable real-time Ethernet functionality. Power over Ethernet is also introduced as an alternative power source for Ethernet devices.

1. The Ethernet Interface

Due to the widespread adoption of Ethernet, building custom devices based around the Ethernet interface is quite appealing for industry. The Institute for Microelectronics Stuttgart wished to incorporate an Ethernet controller into some of its own ASIC designs, so the main focus of this project was to research a customizable Ethernet core, in this case, the Opencores Ethernet MAC. To continue on the Ethernet theme, the Real-Time protocol RTnet is also dicussed in this paper as a possible extension to offer Real-Time functionality to the Ethernet core. Power over Ethernet is also introduced as an alternative power source of for powering Ethernet device.

1.1. Overview of Ethernet MAC

The IEEE 802.3 Ethernet Protocol addresses the physical and datalink layers of the OSI reference model and provides standard techniques for implementing Local Area Network systems with data rates from 1 Mb/s to 1000 Mb/s. Of particular interest to this project is Ethernet Medium Access Control.

The Ethernet Frame is broken down below:

- 8 bytes Preamble
- 14 bytes Header
- 46-1500 bytes Data
- 4 bytes CRC

1.2. Ethernet Modes of Operation

There are two modes of operation specified by the IEEE 802.3 Ethernet Standard: Half-Duplex, and Full-Duplex.

1.2.1 Half-Duplex Operation

Half-Duplex Operation is the most common configuration for Ethernet networks. In Half-Duplex operation, multiple stations on the network must compete for use of the shared medium. This mode employs the CSMA/CD (Carrier Sense Multiple Access/Collision Detection) method to arbitrate the sharing of Ethernet medium. Stations must defer transmissions until the absence of the "Carrier Sense" signal so that the medium is free for use. When multiple stations try to transmit simultaneously, so called "Collisions" occur, and the involved stations must stop and try to retransmit based on an exponential back-off retransmission scheme.

1.2.2 Full-Duplex Operation

In Full-Duplex operation, two stations are connected directly to each other, and there are separate transmit and receive paths. Both stations can transmit freely, since the medium is not shared and there is no danger of collisions. CSMA/CD is not needed to arbitrate the sharing of the medium.

1.3. Opencores Ethernet MAC

Opencores.org is an online repository for open-source core designs. They offer numerous functioning cores, from processors to memory elements to communication controllers.

Although many vendors offer Ethernet MAC cores, the IMS chose to research the Opencores Ethernet MAC primarily because it is license-free and customizable.

The Opencores Ethernet MAC offers:

- IEEE 802.3 standards compli ance
- 10/100 Mbps data rates
- Standard MII signals
- Synthesizable Verilog core

1.3.1 Opencores MAC operation

On one end, the Ethernet MAC is connected to a 32-bit address and 32-bit data Wishbone bus, which interfaces with a processor/controller and a memory buffer element. The other end of the Ethernet MAC is connected to the Media Independent Interface of the PHY and Ethernet Network. **Figure 1.1** above demonstrates the normal layout of the Ethernet MAC core.

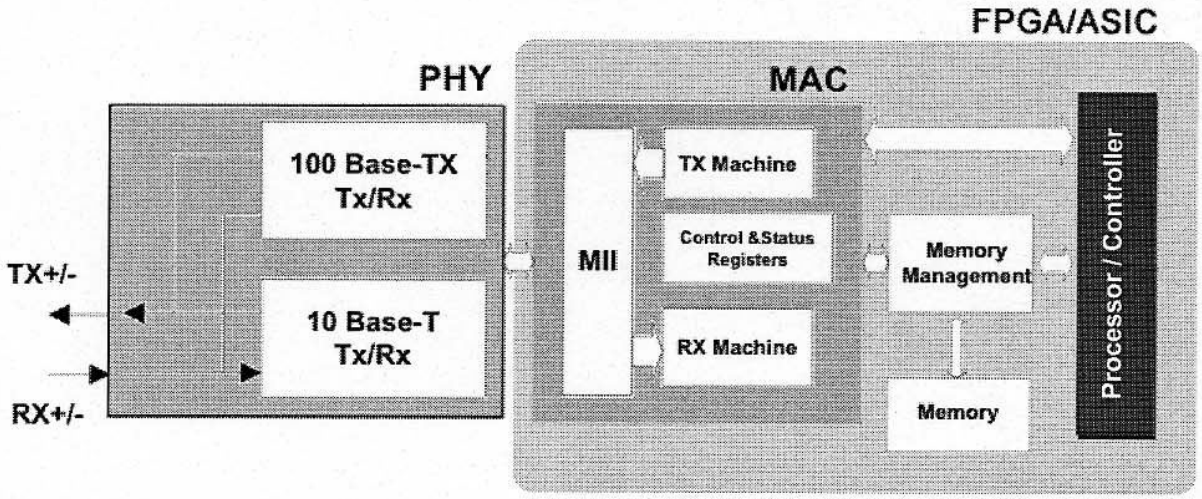


Figure 1.1 Layout of the Opencores Ethernet MAC.

The Ethernet MAC'sbehavior can be configured via a series of settings registers within the core. Transmitting and receiving are performed through a buffer descriptor and memory buffer scheme.

1.3.2 Implementing the Opencores MAC

The primary concern for the IMS was getting the MAC core to fit within the gate count constraints of the IMS Gate-Forest ASICs. Therefore most of the focus was placed upon customizing the core and reducing its size (possibly making the core non-compliant with IEEE802.3 standards).

The following steps were taken in an effort to reduce overhead and the size of the implemented core:

1. Constructed a simple state machine controller instead of using a full 32-bit processor.
2. Removed unused modules (stripped certain functionality).
3. Reduced depth of the internal FIFO of the Wishbone bus
4. Reduced size of Transmit/Receive buffer

Steps 1 and 4 offered the greatest reduction in size. The controller simply initializes the core and transmits a small packet intermittently, or the controller can be set to only transmit when polled with an incoming packet. This offered far less functionality than a full 32-bit processor, but it achieved the goal of reducing overhead. Shinkingthe size of the memory buffer also yielded a large gate count reduction, primarily because the original size of the memory buffer was 4x8192 bytes in order to accommodate Ethernet packets with a maximum size of 1.5 KB. But for the IMS, the intention was to perhaps use the MAC in a small remote sensor device which would only transmit a few bytes of data at the time, so a memory buffer on the order of kilobytes made no sense.

The operation of the core was simulated with a testbench in ModelSim to verify the successful transmission of packets and the proper behavior of the controller.

1.3.3 Synthesizing and FPGA testing

The core was synthesized with Xilinx ISE, and the FPGA used for testing was a Cesys XC2S200 board with a Xilinx Spartan II chip. The FPGA has approximately 2300 slices, which translates to 70000-200000 gates. The original unmodified core required roughly 180000 slices, largely because of the transmit and receive buffer. The final modified core required roughly 2000 slices. Due to a supplier issue, a PHY with integrated RJ45 port could not be obtained in time for testing with the FPGA. Instead various signals and the MII singals were routed to the centronics port available on the FPGA, and the signals were monitored on an oscilloscope and matched against the expected simulation results.

Two important points emerged from the synthesis and FPGA testing process:

Simulatable does not mean synthesizable!

And the corollary:

Synthesized does not mean functional or stable!

Although these points may seem obvious, the synthesis and FPGA testing process certainly reinforced these lessons.

2. Real-Time Ethernet

Real-Time systems require guaranteed and predictable responses – something not possible with standard CSMA/CD Ethernet. But there is a heavy interest in industry to use Ethernet to replace previous generation fieldbus systems. There are already an abundance of competing vendor-specific solutions that offer Real-Time functionality built around Ethernet.

RTnet is an open-source academic solution presented here as a candidate for providing Real-Time functionality on top of the Opencores Ethernet MAC.

2.1. RTnet

RTnet was developed by Ulrich Marx at the Institute for Systems Engineering, Real-Time Systems Group of the University of Hannover. Unlike most other competing real-time Ethernet solutions, RTnet has its open-source roots in academia, so it has no special biased interests other than the needs of the application and the end users. The goal of the protocol was to provide deterministic networking while staying relatively flexible and hardware independent. RTnet itself is "a purely software-based framework for exchanging arbitrary data under hard real-time constraints," and it implements a new Medium Access Control and higher layer protocols such as UDP/IP (User Datagram Protocol/Internet Protocol), ICMP (Internet Control Message Protocol), and ARP (Address Resolution Protocol) for real-time usage. [KIS05]

2.1.1 RTnet Specifications

The current requirements for running RTnet are as follows:

- Linux Kernel 2.4.x or 2.6.x
- RTAI Classic (2.4.1.9 or better, 3.x recommended), or RTAI/fusion (0.7.2 or better)
- x86, PowerPC, or ARM platform
- Standard Ethernet Components

The RTnet protocol is built upon the modularized struture of the Linux network subsystem in an effort to stay hardware independent. Applications that wish to use RTnet and the Real-Time network can interface to the RTnet communication services via a BSD socket interface. The application programs can use package-oriented services such as UDP/IP or raw Ethernet frames. [KIS04]

RTnet's Real-Time network must be closed in order to fulfill the Real-Time constraints, but nodes in the network can be connected to an outside non-Real-Time network. Non-critical data can also be tunneled through the Real-Time network.

2.1.2 RTnet Medium Access Control

RTnet provides an additional Medium Access Control layer to supplement the non-Real-Time capable Ethernet MAC layer. The Medium Access Control provided by RTnet is a Time Division Multiple Access (TDMA) scheme. Slave devices are synchronized to a Master device, and time slots are provided for each participant of the network. The configuration of the nodes and time slots are fairly flexible, and a back-up synchronization Master can also be used in case the Master node fails.

Further information about RTnet can be found at http://www.rtnet.org

3. Power Over Ethernet

Power over Ethernet offers the possibility of powering Ethernet devices over the Ethernet data wire Power over Ethernet was officially standardized and approved by the IEEE in June of 2003, and it is now referred to as the IEEE802.3af protocol.

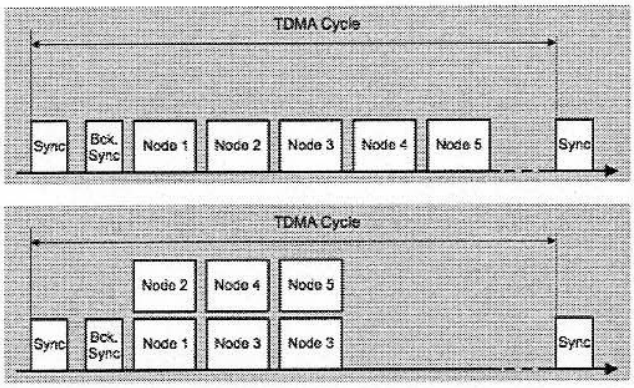


Figure 2.1 Possible time-slot configurations. [KIS05]

3.1. Power Over Ethernet Specifications

Power over Ethernet is media dependent, usually delivering power over standard Cat 5 twisted pair cables. In a Power over Ethernet setup, there must be at least two devices – the Power Sourcing Equipment (henceforth referred to as the PSE) and the Powered Device (henceforth referred to as the PD).

By default, the PSE can output 48 V DC with an output current of 350 mA, for a power load of approximately 16.8 watts. Typically about 13 watts of power is available to the Powered Device. The PSE can also output different power levels depending on the optional classification group of the connected Powered Devices.

3.2. Power Over Ethernet Operation

At the outset, the PSE must perform a probing to see whether valid Powered Devices are connected. The PSE applies a small current-limited voltage to the wire, and valid Powered Devices will provide an appropriate signature created by the presence of a 25k ohm resistor in the Powered Device. Only after a valid signature is detected will the PSE apply the full voltage to the wire, and in this way, devices on the wire that are not Power over Ethernet compliant will not receive any power and risk being damaged.

Once detection is complete, power can be delivered to the PD. Standard Cat 5 Ethernet cables have four pairs of wires, only two of which are used for transmitting data. The different pairings of the wires are detailed in the figures below. There are two modes of operation for delivering power.

3.2.1 Option A

In this option, the spare pairs of wires not used for data transmission are used for power transmission.

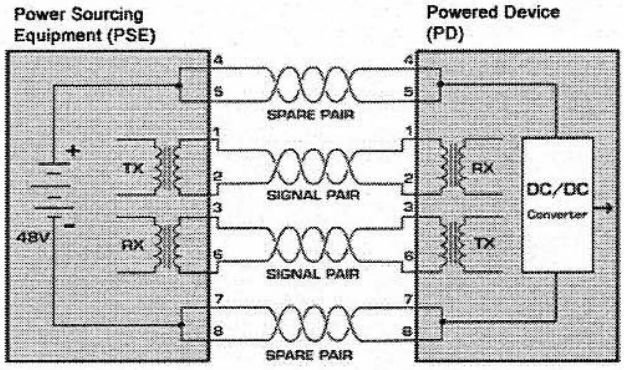


Figure 3.1 Powering via spare pairs. [POW03]

3.2.2 Option B

In this option, the data pairs are used for power transmission.

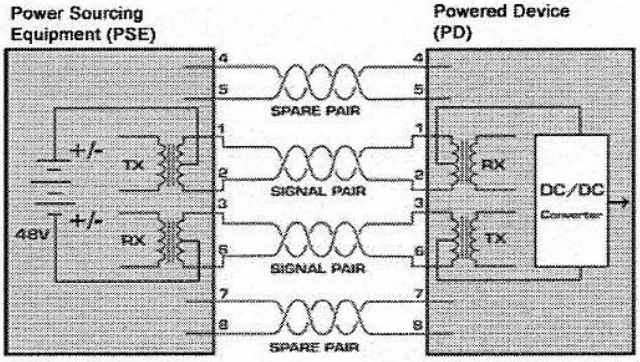


Figure 3.2 Powering via data pairs. [POW03]

To comply with the IEEE802.3af standard, Powered Devices must be able to support both modes of operation and be able to draw power from either pairs of wires. However, only one option may be used at a time; the PD cannot draw power from all four pairs of wires.

4. Conclusion

This project primarily focused upon the customization of the Opencores Ethernet MAC for use in Ethernet-based devices. The goal of reducing the Ethernet MAC core to fit within a restricted gate-count was successful, and further reductions could be made if so desired, of course with the caveat that certain funtionality would be removed from the Ethernet MAC core.

Additional studies presented in this paper also demonstrate the possibility of adding Real-Time functionality on top of the Opencores Ethernet MAC, of course with the requirement that the full Ethernet MAC and a 32-bit linux capableprocessor be implemented together. Power over Ethernet can also be used to power such a Real-Time capable Ethernet device.

About the Author

Pei-Ching Wu is U.S. citizen and earned a degree as Bachelor of Sciences at University of California, Berkeley. He joined the Master Students Program at FHTE, Esslingen, Germany. He wrote his Master's Thesis about Ethernet for FPGA and ASIC solutions at the Institute for Microelectronics Stuttgart.

Literature

[KIS04] J. Kiszka, R. Schwebel
Alternative: RTnet
A&D Newsletter, 2004

[KIS05] J. Kiszka, B. Wagner, Y. Zhang, J. Broenik
RTnet - A Fleible Hard Real-Time Networking Framework
The Institute of Electrical and Electronics Engineers, 2005

[POW03] PowerOverEthernet.com Staff
White Paper on Power Over Ethernet (IEEE802.3af) - A Radical New Technology
http://www.PowerOverEthernet.com, 2003

Funktionsberechnung mit hierarchischen Look-up Tabellen

Wolfgang Rülling
Hochschule Furtwangen

Look-Up-Tabellen ermöglichen die extrem schnelle Berechnung von Funktionen. Es wird gezeigt, wie man diese Technik unter Verwendung hierarchischer Tabellen platzsparend auf FPGAs durchführen kann.
In Abhängigkeit von der gewünschten Funktion und der erforderlichen Genauigkeit wird die VHDL-Beschreibung der optimalen Schaltung mit einem Generator erzeugt.

1 Einführung

In [1] wurde das Konzept für eine schnelle Spezialhardware zur Berechnung der Sinus-Funktion vorgestellt und es wurde vorgeschlagen, einen entsprechenden Spezialchip mit einer Genauigkeit $\epsilon_{\max} \leq 10^{-8}$ zu implementieren.
Prinzipiell ist das vorgestellte Verfahren auch für andere stetige Funktionen $y = f(x)$ anwendbar. Die Grundidee besteht darin, die gegebene Funktion f stückweise durch Geradensegmente zu approximieren und statt f die Approximationsfunktion $\widehat{f}$ in Hardware zu berechnen (siehe Abbildung 1).
Die Schaltung muss dann zu einem gegebenen Argument x das Intervall $[x_i, x_{i+1}]$ ermitteln, in dem x liegt und die entsprechenden Konstanten a_i und b_i in Look-up Tabellen nachschlagen. Danach kann die Näherung $\widehat{y} = a_i x + b_i$ mit nur einer Multiplikation berechnet werden.
Die hohe Verarbeitungsgeschwindigkeit dieses Ansatzes ergibt sich daraus, dass die Schaltung bis auf den Zugriff auf eine Look-up Tabelle rein kombinatorisch arbeitet. D.h. das Ergebnis $\widehat{y}$ wird in nur einem Takt berechnet. Die Geschwindigkeit der Schaltung wird im wesentlichen durch den Multiplizierer bestimmt (siehe Abbildung 2).
In [1] wurde auch hergeleitet, welche Speicherkapazität die Look-up Tabellen in Abhängigkeit von der gewünschten Genauigkeit $\epsilon_{\max}$ benötigen. Der Zusammenhang zwischen Speicherkapazität und Rechengenauigkeit ist in Abbildung 3 durch "•"-Zeichen dargestellt. Außerdem ist dort die bei VIRTEX4

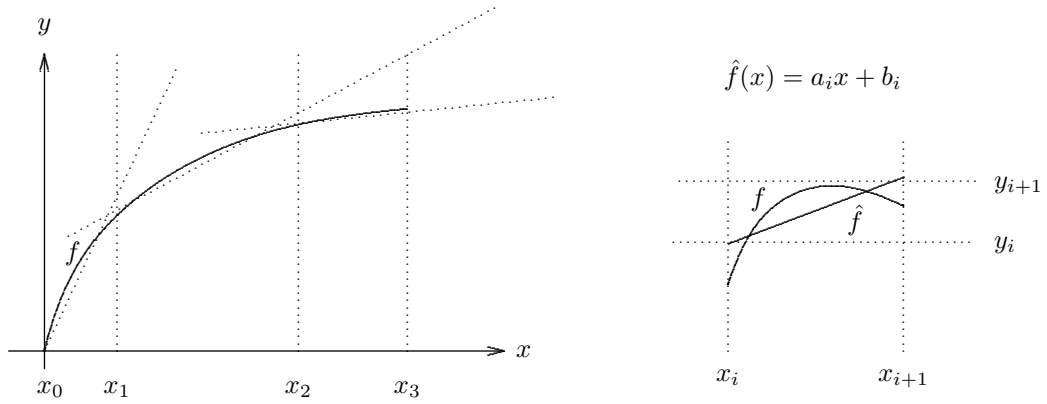


Abbildung 1: Lineare Approximation einer Funktion $y = f(x)$

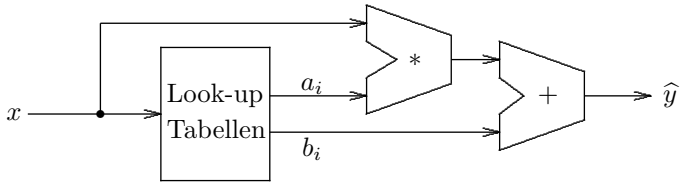


Abbildung 2: Skizze einer schnellen Schaltung zur Berechnung einer linearen Approximation

FPGAs der Firma XILINX verfügbare Speicherkapazität angegeben. Man erkennt, dass die Speicherkapazität dieser Bausteine für die vorgeschlagene Implementierung einer Schaltung mit $\epsilon_{\max} \leq 10^{-8}$ ausreichen müsste.
In der vorliegenden Arbeit soll nun eine Technik vorgestellt werden, mit der sich das skizzierte schnelle Verfahren mit weniger Speicherplatz realisieren lässt. In Abbildung 3 sind die erzielten Ergebnisse bereits dargestellt. Unter Beachtung des logarithmischen Maßstabs erkennt man im Diagramm, dass nur etwa ein Viertel des erwarteten Speicherplatzes benötigt wird.
Im Abschnitt 2 wird eine Modifikation des ursprünglichen Schaltungskonzepts vorgestellt, bei der die Look-up Tabellen durch eine baumförmige Speicherstruktur ersetzt werden.
Anschließend wird in den Abschnitten 3 und 4 auf die VHDL-Implementierung der Schaltung mit Hilfe eines Schaltungsgenerators und auf die Schaltungssynthese mit der ISE-Entwurfsumgebung der Firma XILINX eingegangen.
In Abschnitt 5 wird dann kurz erläutert, wie man entsprechende Schaltungen außer für die Sinus-Funktion auch für andere Funktionen generieren kann.

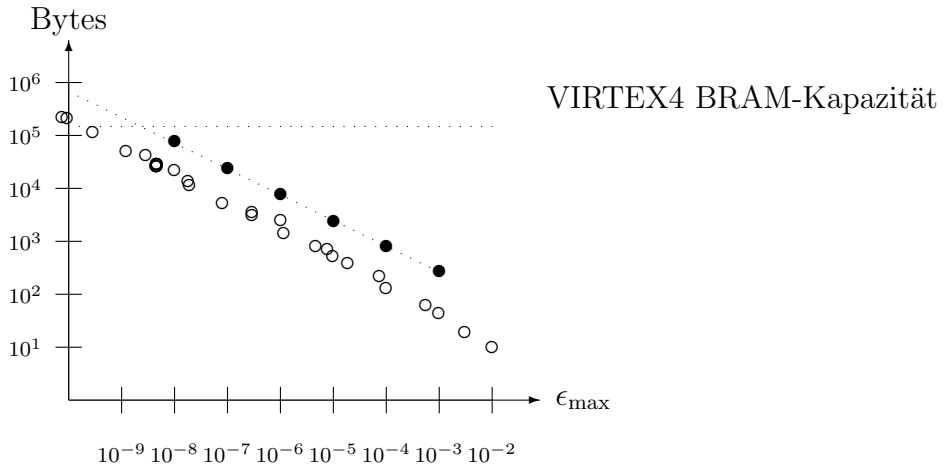


Abbildung 3: Die eingezeichnete Gerade durch die "•"-Punkte stellt den in [1] vorhergesagten Speicherplatzbedarf in Abhängigkeit von der Genauigkeit $\epsilon_{\max}$ dar. Mit "○" sind zum Vergleich die Speicherkapazitäten der in dieser Arbeit vorgestellten Schaltungen angegeben. Es zeigt sich, dass die verwendeten Baumstrukturen nur etwa ein Viertel des erwarteten Speicherplatzes benötigen.

2 Schaltungsentwurf

Das Hauptziel bei der Neuentwicklung der Approximationsschaltung besteht darin, den Speicherplatzbedarf zu reduzieren, ohne dabei die Entwurfsidee aufzugeben.

2.1 Verwendung von Dual-Port ROMs

Als erste Idee sollen statt der Geradenparameter a_i und b_i die Funktionswerte y_i an den Stützstellen gespeichert werden. Dabei setzen wir voraus, dass die gegebenen Funktion durch einen **Polygonzug** approximiert wird (siehe Abbildung 4). Eine solche Approximation ist etwas ungenauer, braucht aber nur halb so viel Speicher.
Beispielsweise kann man dann gemäß $a_i = \frac{y_{i+1}-y_i}{x_{i+1}-x_i}$ die Geradensteigung rekonstruieren. Sofern man für die Länge der Intervalle $[x_i, x_{i+1}]$ Zweierpotenzen 2^{-k} wählt, lässt sich die Division durch einfaches Shiften um k Binärstellen ersetzen.
Eine so arbeitende Schaltung ist in Abbildung 5 skizziert. Die führenden Binärstellen des Arguments x werden zur Adressierung der Look-up Tabelle benutzt und nur die niederwertigen Bits von x werden für die Arithmetik verwendet.

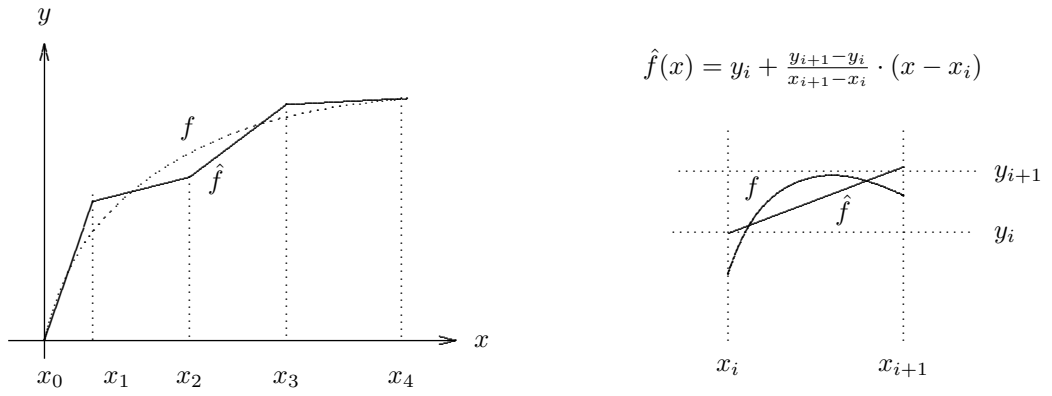


Abbildung 4: Approximation einer Funktion durch einen Polygonzug. Zur Verdeutlichung des Prinzips ist der Polygonzug nicht optimal gewählt.

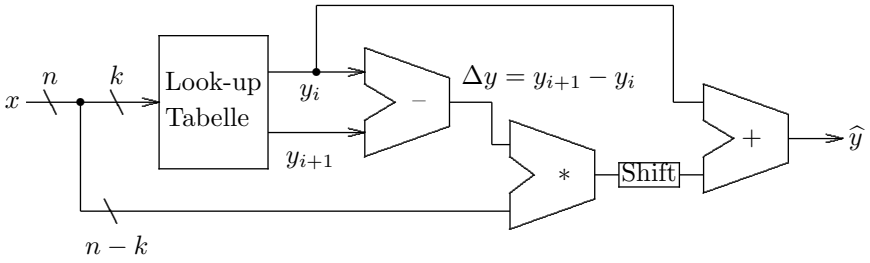


Abbildung 5: Skizze einer schnellen Schaltung zur Berechnung einer linearen Approximation mit nur einer Look-up Tabelle

Auf diese Weise kann ein bedeutend kleinerer und schnellerer Multiplizierer verwendet werden.
Weiter fällt auf, dass die Look-up Tabelle zwei Werte (y_i und y_{i+1}) liefern muss. Dazu bieten sich Dual-Port RAMs, bzw. Dual-Port ROMs an, mit denen man gleichzeitig auf zwei verschiedene Speicherplätze zugreifen kann. Da bei modernen FPGAs ohnehin Dual-Port Speicher statt Single-Port Speicher verfügbar sind, ist die Dual-Port-Technik ohne zusätzlichen Flächenbedarf verfügbar.

2.2 Verwendung einer Baumstruktur

Betrachtet man die Tabelle der y_i-Werte beispielsweise für $f(x) = \sin(x)$, so fällt auf, dass wegen der Stetigkeit der Funktion f die benachbarten Werte sehr ähnlich sind, sich also nur in den niederwertigen Bits unterscheiden. In Tabelle 1 ist beispielsweise der Vektor $(y_0, y_1, \ldots, y_{15})$ für eine grobe Approxi-

Index i	y_i	binär	$y_i \cdot 2^9$
0	0	0.000000000	0
1	0.125	0.001000000	64
2	0.248046875	0.001111111	127
3	0.3671875	0.010111100	188
4	0.48046875	0.011110110	246
5	0.5859375	0.100101100	300
6	0.681640625	0.101011101	349
7	0.767578125	0.110001001	393
8	0.841796875	0.110101111	431
9	0.904296875	0.111001111	463
10	0.94921875	0.111100100	486
11	0.982421875	0.111110111	503
12	0.998046875	0.111111111	511
13	1	1.000000000	512
14	1	1.000000000	512
15	1	1.000000000	512

Tabelle 1: Näherungsweise Darstellung der Sinus-Funktion (im Intervall $[0, 2)$) durch 16 Stützstellen y_i. Die Daten sind mit einer Genauigkeit von 9 Bit in den Nachkommastellen angegeben. Als kürzere ganzzahlige Darstellung wird auch der Wert $y_i \cdot 2^9$ benutzt

mation der Sinus-Funktion mit nur 16 Stützstellen im Intervall $[0, \frac{\pi}{2}]$ angegeben.
Um weiteren Speicherplatz einzusparen, sollen im folgenden statt der y-Werte nur noch Differenzen von y-Werten, also nur die niederwertigen Binärstellen gespeichert werden. Durch Summation solcher Differenzen lassen sich dann die ursprünglichen Funktionswerte wieder zurückgewinnen.
Um diese Vorgehensweise zu erläutern, sind in Abbildung 6 die 16 Funktionswerte der Sinus-Funktion in die unterste Baumstufe eines binären Baumes eingetragen. Diese Stufe repräsentiert die Aufteilung des Definitionsbereichs in 16 Teilintervalle.
Die darüberliegende Stufe entspricht einer groberen Intervalleinteilung mit nur 8 Intervallen, u.s.w.. Schließlich steht der Wurzelknoten für den gesamten Definitionsbereich.
Man beachte, dass ein solcher Baum mit n Blättern $n-1$ innere Knoten besitzt. Damit hat sich der benötigte Speicherplatz gegenüber der bisherigen Look-up Tabelle fast verdoppelt. Deshalb soll nun versucht werden, den Speicherplatz dadurch zu reduzieren, dass nur noch Differenzen zwischen y-Werten gespei-

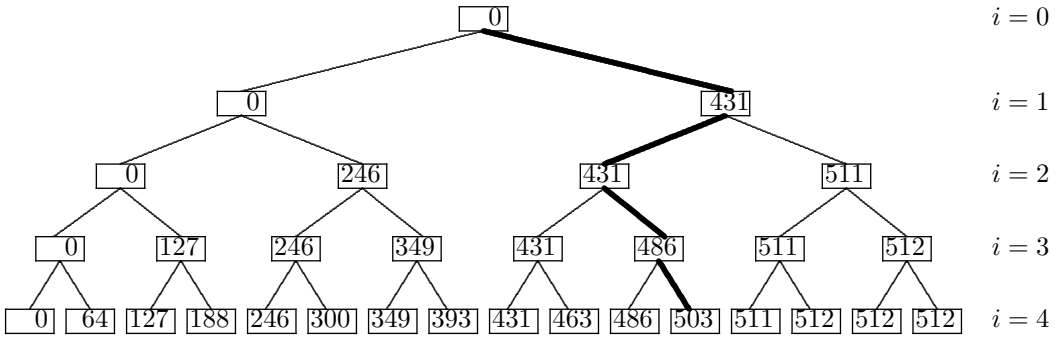


Abbildung 6: Baumstruktur zur Approximation der Sinus-Funktion. In Stufe i wird die Funktion mit Hilfe von 2^i Stützstellen approximiert. Der hervorgehobene Pfad führt beispielsweise zur Stützstelle $y_{11} = {}^{503}/_{2^9} = 0.982421875$.

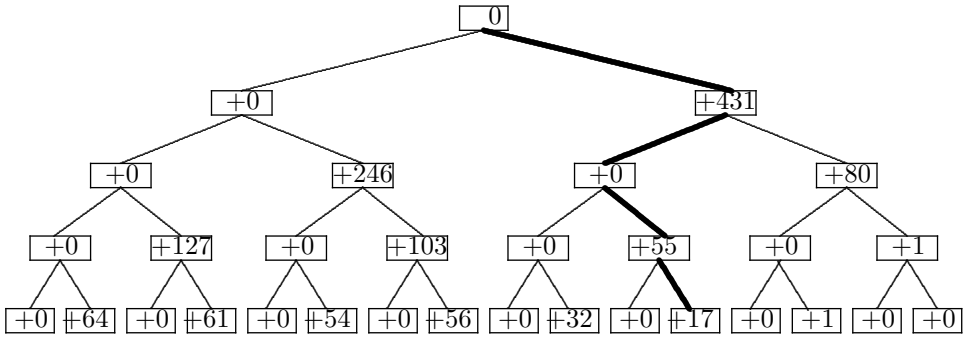


Abbildung 7: Baumstruktur zur Approximation der Sinus-Funktion unter Verwendung von Differenzen. Summiert man alle Werte auf dem Pfad von der Wurzel zu einem Blatt, so erhält man den Funktionswert der entsprechenden Stützstelle. Beispielsweise ergibt sich auf dem hervorgehobenen Pfad die Summe $0 + 431 + 0 + 55 + 17 = 503$, die y_{11} darstellt.

chert werden. Abbildung 7 zeigt den gleichen Baum, wenn man bei jedem Knoten nur noch die Differenz zum bisherigen Vaterknoten einträgt.

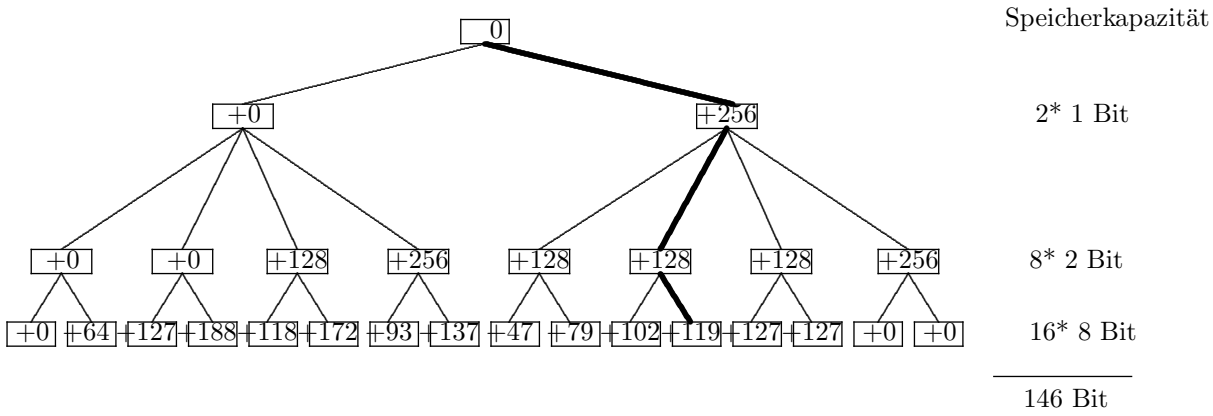


Abbildung 8: Optimierte Baumstruktur zur Approximation der Sinus-Funktion. Der hervorgehobene Pfad liefert die Summe $0+256+128+119 = 503$ für die Stützstelle y_{11}.

Dabei fällt auf, dass etwa die Hälfte der Knoten mit "+0" beschriftet sind. Bei monotion wachsenden Funktionen sind dies immer die linken Sohnknoten. Dieses Phänomen kann man ausnutzen, um den Speicherplatzbedarf des Baumes etwa zu halbieren. Damit braucht der Differenzen-Baum also etwa den gleichen Speicherplatz wie die ursprüngliche Liste der Funktionswerte. Außerdem verringert sich der Speicherplatzbedarf weiter, wenn man ausnutzt, dass die gespeicherten Daten kleiner geworden sind. Zudem lassen sich die niederwertigen Bits von den oberen Baumstufen nach unten verschieben, so dass pro Baumstufe nur noch wenige Datenbits gespeichert werden müssen. Lässt man außerdem noch irrelevante Baumstufen weg, entsteht der optimierte Baum in Abbildung 8.
Bei diesem kleinen Zahlenbeispiel braucht der optimierte Baum eine Speicherkapazität von 146 Bit gegenüber $16 \cdot 10 = 160$ Bit der ursprünglichen Liste von Stützstellen. Bei den in der Praxis verwendeten Bäumen mit wesentlich mehr Stufen, ergibt sich eine bedeutend größere Speicherersparnis. Beispielsweise braucht man bei 128 Stützstellen der Länge 10 Bit eine Speicherkapazität von 1280 Bit, während der entsprechende optimierte Baum nur 882 Bit benötigt.

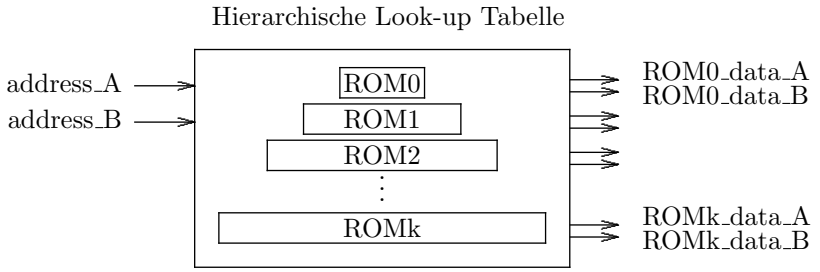


Abbildung 9: Jede Baumstufe wird durch ein separates ROM realisiert. Zu zwei Eingabeadressen werden die Daten zweier Baumpfade ausgegeben.

```
entity Sinus is
  port ( x: in unsigned(9 downto 0); y: out unsigned(9 downto 0));
end Sinus;

architecture structure of Sinus is
...
process
...
begin
-- Adressierung der Look-up Tabellen
address_A <= x(9 downto 6);                                  -- x_i
address_B <= address_A +1;                                   -- x_{i+1}

y_A := ROM0_data_A + ROM1_data_A + ROM2_data_A;              -- y_i
y_B := ROM0_data_B + ROM1_data_B + ROM2_data_B;              -- y_{i+1}

delta_y := y_B - y_A;                                        -- Δy = y_{i+1} - y_i
y <= y_A + (x(5 downto 0) * delta_y)/2**6;                   -- y = y_i + ((x-x_i)*Δy)/Δx
end process;
end structure;
```

Tabelle 2: Skizze der VHDL-Beschreibung für eine Approximationsschaltung mit 3 Baumstufen und Ein- und Ausgangsdaten der Länge 10 Bit

3 Implementierung

Wenn wir voraussetzen, dass die komplette Baumstruktur der Look-up Tabellen, wie in Abbildung 9 dargestellt, zu zwei angelegten Adressen die beiden Baumpfade zurückliefert, ergibt sich die in Tabelle 2 in VHDL skizzierte einfache Top-level Beschreibung der Schaltung.
Damit fehlt nur noch die Erzeugung der korrekten Daten für die Baumstruktur und die Verteilung der Daten auf ROMs. Dazu wurde ein C++ Programm erstellt, das zunächst die Baumstruktur berechnet und dann, wie in Abbildung 9 skizziert, jede Baumstufe einzeln auf ein ROM abbildet.

Auf XILINX-FPGAs gibt es verschiedene Techniken, ROMs zu implementieren:

a) kleine Look-up Tabellen realisiert man effizient durch CLBs (Configurable Logic Blocks), die ansonsten zur Implementierung der Schaltungslogik benutzt werden. Dazu benutzt man Anweisungen der Art
```
constant ROM_data:darray(0 to 7):=
            ("00", "00", "01", "10", "01", "01", "01", "10");
```

b) Look-up Tabellen mittlerer Größe lassen sich günstig mit RAMs realisieren. Dazu gibt es vordefinierte VHDL-Entities, die sich über generic-Parameter mit Daten initialisieren lassen (z.B. `RAMS16_S2`, `RAMS16_S1`). Diese Speicher sind ebenfalls mit Hilfe von CLBs implementiert.

c) Für sehr große Look-up Tabellen benutzt man (synchrone) **Block-RAMs** (BRAMs), die es in mehreren fest vorgegebenen Größen gibt. Diese Speicher nutzten nicht die CLBs, sondern sind auf dem FPGA mit separater Hardware realisiert. Je nach FPGA-Typ steht nur eine relativ kleine Anzahl dieser Block-RAMs zur Verfügung. Im Gegensatz zu den unter a) und b) genannten Techniken, benötigen die Block-RAMs ein zusätzliches Taktsignal, da die Ausgabedaten synchron mit einer steigenden Taktflanke geliefert werden.

Die Technik a) stellt die einfachste Möglichkeit dar, Look-up Tabellen beliebiger Größe zu beschreiben. Leider führt sie bei großen Tabellen zu extrem langen Synthesezeiten und einer schlechten Ausnutzung des FPGAs. Die Methoden b) und c) haben den Nachteil, dass nur Speicher fester Kapazität verfügbar sind. Größere Tabellen müssen deshalb aus kleineren Speicherblöcken zusammengesetzt werden und bei ungünstigen Tabellengrößen können die Blöcke nicht vollständig genutzt werden. Das C++ Programm entscheidet deshalb anhand der jeweiligen Block-Größe, welche Implementierung am besten geeignet ist und erzeugt entsprechenden Instanzierungen in VHDL.

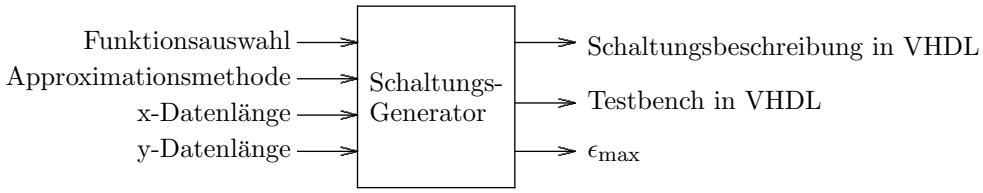


Abbildung 10: Der in C++ geschriebene Schaltungsgenerator erzeugt zu gegebenen Parametern ein entsprechende entsprechende Schaltungsbeschreibung in VHDL und eine Testbench der Schaltung. Außerdem ermittelt der Generator den maximal auftretenden Fehler bei der Approximation.

Neben der primären Aufgabe, Schaltungsbeschreibungen in VHDL zu generieren, kann das C++ Programm auch Beispielanwendungen der Schaltung durchrechnen und mit den dabei ermittelten Ergebnisdaten eine Testbench für die Schaltung erzeugen. Außerdem wird vom Programm der maximal auftretende Fehler $\epsilon_{\max}$ bestimmt (siehe Abbildung 10).
In Tabelle 3 wird anhand einer größeren Beispielschaltung gezeigt, wie das Programm Beispielrechnungen protokolliert. Dabei erkennt man deutlich, die in den Look-up Tabellen verwendeten Datenlängen und sieht, dass sich die Datenwörter aus verschiedenen Tabellen kaum überlappen. Für die Schaltung bedeutet dieses Phänomen, dass für die Summationen nur sehr kleine Addierer erforderlich sind.

4 Automatische Versuchsdurchführung

Bei der Schaltungsentwicklung waren umfangreiche Simulations- und Synthese-Experimente erforderlich, da unterschiedliche Datenlängen, Approximationsmethoden und Baumoptimierungen erprobt werden mussten. Deshalb wurde die Versuchsdurchführung frühzeitig automatisiert.
Für die **Schaltungssimulation** ist dies einfach, da der oben geschriebene C++ Generator bereits für jede Schaltungsvariante eine geeignete Testbench erzeugt und VHDL-Beschreibungen leicht im Batchbetrieb simuliert werden können.
Für die **Schaltungssynthese** wurde die Entwurfsumgebung ISE der Firma XILINX genutzt. Sie kann auch mit Hilfe einer Skriptsprache (z.B. Perl) automatisiert genutzt werden. Auf diese Weise kann man beispielsweise eine Schleife programmieren, in der nacheinander VHDL-Beschreibungen von Schaltungen mit unterschiedlichen Datenlängen generiert werden, die jeweils für das FPGA synthetisiert werden. Aus den dabei entstehenden Syntheseprotokollen

```
Beispielrechnung für x=0.436521
Binärdarstellung: x= 011011111011111[11101011100011011000000100]

Berechnung von y_A:
 0: ROM_data[    0] =  +_0___________________________________
 1: ROM_data[    3] =  +_101_________________________________
 2: ROM_data[    6] =  +___00________________________________
 3: ROM_data[   13] =  +____11_______________________________
 4: ROM_data[   27] =  +_____01______________________________
 5: ROM_data[   55] =  +______10_____________________________
 6: ROM_data[  111] =  +_______11____________________________
 7: ROM_data[  223] =  +________01____________________________
 8: ROM_data[  446] =  +_________01___________________________
 9: ROM_data[  893] =  +__________10__________________________
10: ROM_data[ 3575] =  +__________010_________________________
11: ROM_data[ 7151] =  +____________00________________________
12: ROM_data[14303] =  +_____________1001000001000111010101011
      Interpolation    +_____________0100101110101100011010 00
                      ------------------------------------------------
          Ergebnis y=   01100010000101011011000011101100010011
          Sollwert  =   01100010000101011011000011101100000100

Abweichung vom Sollwert : 5.456968211e-11
```

Tabelle 3: Auszug aus einem Protokoll des Schaltungsgenerators. Die Beispielrechnung verdeutlicht auch die Datenlängen in den Look-up Tabellen. Nicht benutzte Bitpositionen sind durch "_" gekennzeichnet.

lassen sich (mit einem weiteren C++ Programm) die wichtigen Daten, wie etwa die Schaltungsgröße und das Zeitverhalten extrahieren, um damit die weitere Schaltungsauswahl zu steuern. Dabei werden die Synthesedaten tabellarisch dargestellt und können direkt zu Dokumentationszwecken in Diagramme umgesetzt werden. Beispielsweise wurden die Diagramme 3 und 11 auf diese Weise generiert.
Manuell wären diese Experimente nicht durchführbar gewesen. Eine größere Versuchsreihe benötigte beispielsweise auf einem SPARC-Rechner länger als einen Monat und kleinere Versuchsreihen liefen auf einem schnellen PC (Pentium, 3 GHz) mehrere Tage. Der Grund für die großen Rechenzeiten ist, dass sehr viele Synthese-Experimente nahe der Kapazitätsgrenze des FPGAs mit sehr zeitaufwändiger Platzierung und Verdrahtung durchgeführt werden mus-

sten.
Unterdrückt man bei der Ergebnisdarstellung ineffiziente Schaltungsvarianten, erhält man beispielsweise das Diagramm aus Abbildung 11. Es zeigt für die optimalen Schaltungen den Zusammenhang zwischen der Schaltungsgröße A und der erreichten Genauigkeit $\epsilon_{\max}$. Dabei steht das Zeichen "$\circ$" für Schaltungen, bei denen die Sinus-Funktion durch Interpolation angenähert wurde und "$\bullet$" bezeichnet die Schaltungen, die eine Approximation verwendeten.

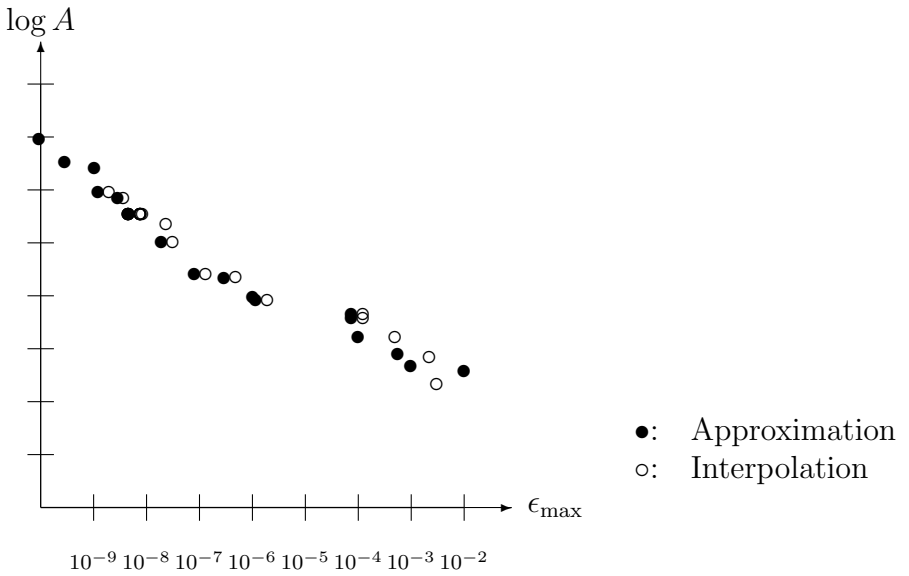


Abbildung 11: Flächenbedarf der Schaltung in Abhängigkeit von der Genauigkeit $\epsilon_{\max}$ des Ergebnisses bei Verwendung der jeweils besten Approximation bzw. besten Interpolation

Man erkennt, dass bei etwa gleichem Flächenbedarf die Approximation meist eine bessere Genauigkeit liefert als die Interpolation. Nur bei geringen Genauigkeiten ist manchmal die Interpolation besser.
Das wichtigste Ergebnis der Versuche besteht darin, dass die Speicherkapazität des verwendeten VIRTEX4 FPGA für Genauigkeiten bis zu $\epsilon_{\max} = 3 \cdot 10^{-10}$ ausreicht. Bei der genauesten Schaltung werden x-Eingaben der Länge 40 Bit verwendet, von denen die niederwertigen 25 Bit nur zur Interpolation zwischen den Stützstellen der Approximationsfunktion verwendet werden. Die höherwertigen 15 Bit dienen zur Adressierung der Baumstruktur. Die Stützstellen haben eine Datenlänge von 38 Bit und der Baum benötigt eine Speicherkapazität von 884730 Bit. Dabei wurden 75% der Block-RAMs ausgenutzt, aber nur 2% der verfügbaren Slices (CLBs). D.h. auf dem FPGA ist noch viel Platz für weitere

Logik. Für das Zeitverhalten der Schaltung wurde vom Synthesetool 20.4 ns ermittelt.

5 Anwendungen

Ein interessanter Aspekt des vorgestellten Konzepts besteht darin, dass es auf beliebige stetige Funktionen $y = f(x)$ anwendbar ist. Beispielsweise wurde der Schaltungsgenerator auch bereits erfolgreich für $y = \sqrt{x}$ und $y = \cos(x)$ angewendet. Dabei ergaben sich sehr ähnliche Schaltungsgrößen und Genauigkeiten wie bei der Sinus-Funktion.
Bei der aktuellen Implementierung des Schaltungsgenerators ist allerdings als Einschränkung zu beachten, dass sowohl der Definitionsbereich als auch der Wertebereich in $[0, 1)$ liegen müssen und die Funktionen monoton steigend sein müssen.
Für die Praxis bedeutet die Einschränkung des Zahlbereichs kein großes Problem, da man der eigentlichen Berechnung Transformationen vor- und nachschalten kann. Für die Sinus-Berechnung wird beispielsweise nach [1] nur der Definitionsbereich $[0, \pi/2]$ benötigt. Für die Experimente der vorliegenden Arbeit wurde dann statt $y = \sin(x)$ tatsächlich die folgende Funktion verwendet:

$$y = f(x) = \begin{cases} \frac{1}{2}\sin(2x) & \text{falls } x \leq \frac{\pi}{2} \\ 1/2 & \text{sonst} \end{cases}$$

Für die angegebenen Genauigkeiten wurden die Ergebnisse aber natürlich wieder auf den üblichen Wertebereich zurückgerechnet.
Bei der Funktion $y = \sqrt{x}$ funktioniert die erforderliche Transformation so, dass man beispielsweise nur den Definitionsbereich $[0.25, 1)$ nutzt, in dem sich die Wurzelfunktion gut approximieren lässt. Um damit beispielsweise den Wert $\sqrt{3}$ zu berechnen, würde man die Transformation $\sqrt{3} = 2 \cdot \sqrt{\frac{3}{4}} = 2 \cdot \sqrt{0.75}$ benutzen. Allgemein shiftet man das Argumente um $2k$ Stellen, um die führende 1 in die zweite Nachkommastelle zu bringen, berechnet dann die Wurzel durch Nachschlagen in Look-up Tabellen und shiftet das Ergebnis um k Stellen zurück.

6 Ausblick

Die vorgestellten Schaltungen sind sehr schnell, da die gesamte Berechnung in einem Takt erfolgt. Um sie mit Softwarelösungen zu vergleichen, soll ein XILINX-Evaluationsboard mit einem VIRTEX4 FPGA verwendet werden. Ins FPGA wird dann neben der neuen Spezialhardware auch ein Microblaze-Prozessor geladen, der in C programmierbar ist und die Spezialhardware als Co-Prozessor benutzt. Damit ist eine schnelle chipinterne Kommunikation zwischen Prozessor und Spezialhardware möglich. Da Prozessor und Spezialhardware mit dem gleichen Taktsignal betrieben werden, ist ein taktgenauer Vergleich zwischen Softwarelösungen und Hardwarelösungen möglich.
Erste Versuche haben bereits gezeigt, dass die in C verfügbare Bibliotheksfunktion zur Sinus-Berechnung viel langsamer ist. Allerdings muss noch untersucht werden, welcher Anteil der Laufzeit auf die notwendige Transformation vor der eigentlichen Berechnung entfällt.
Weiter müssen für einen fairen Vergleich auch Softwarelösungen mit Look-up Tabellen implementiert werden. Auch hier sollte die Hardwarelösung schneller sein.
Schließlich ist noch zu untersuchen, wie man die Transformationen vor und nach der eigentlichen Berechnung effizient in Hardware realisieren kann.

Literatur

[1] H. Spiro, K. Najmann, *Ein Verfahren zur schnellen Berechnung der Sinus- und/oder Kosinusfunktion mit Hilfe eines Spezial-Chips*, Tagungsband des MPC-FH Workshops im Januar 1998 in Albstadt-Sigmaringen, Herausgeber: FH Ulm, 1998.

Circuit Design Methodology

Oder: Die Physik hat uns wieder

Christoph Wandel
IBM Deutschland Entwicklung GmbH
Böblingen

Agenda

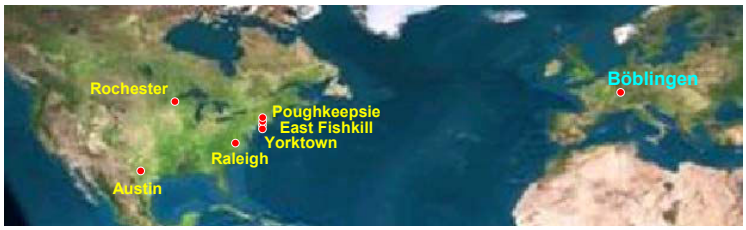
- Was wir tun, wer wir sind
- Motivation
- High-Level Design, Planung
- Macro Design
- Physical Design
- Release-Phase
- Wohin geht die Reise?

Wer wir sind - was wir tun

- Microprocessor Hardware Development
- Entwicklung von Prozessor-Cores für
  - Großrechner/Server
  - SONY PlayStation®, XBOX360®, Nintendo®
  - ASICs
  - Spezialanwendungen, z.B. weltgrößter Emulator für Logik (im Auftrag der Firma Cadence)
- Teil der IBM Systems&Technology Group

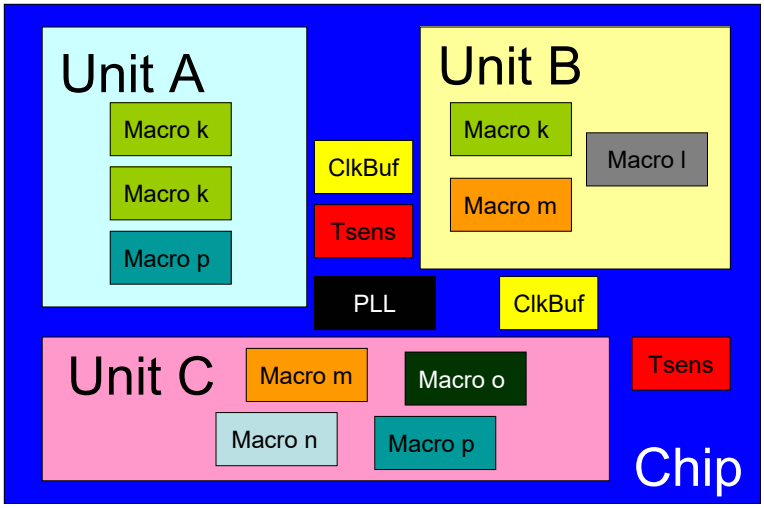
Motivation

- Transatlantische Zusammenarbeit
  Im aktuellen Projekt: 7 Standorte



→Ziel: Einheitlicher Ablauf in allen Lokationen

Überblick

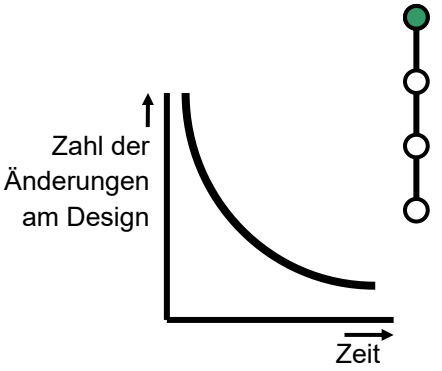


High-Level Design

- Logik-Design setzt vom Kunden geforderte μArchitektur um
- Verhandlungen:
  - Logik-Design <-> Circuit Design
  - Circuit Design <-> Fabrik
- Definition der Technologie, Anzahl und Art der Metalllagen, Spezialeigenschaften

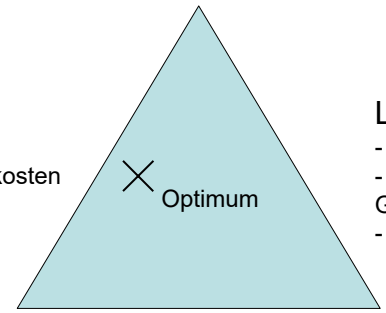
Arbeitsrichtung

- Logik-Design arbeitet Top→Down
  - Gesamtsystem-Modell
    - →Einzelchips
      - →Units
        - →Macros
- Circuit-Design arbeitet Bottom→Up
  - Building Blocks
    - →Macros
      - →Units
        - →Chip



Optimierungskriterien

Schaltungsgeschwindigkeit
- Taktfrequenz als Wettbewerbskriterium

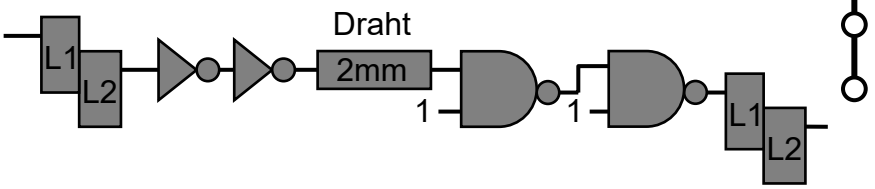


Fläche
- Produktionskosten
- Leckströme
- Ausbeute

Leistungsaufnahme
- Realisierbarkeit
- Kosten (Kühlung, Gehäuse)
- Zuverlässigkeit

High-Level Design…

Cross-Sections



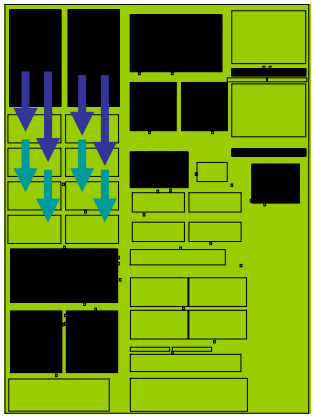
- Vereinfachte Modellierung des Zeitverhaltens
- Nur kritische Pfade enthalten
- Berechnung mit Schaltungssimulatoren

High-Level Design…

- Gesamtschaltung in Macros aufteilen
- Macro-Größe kritisch prüfen
- Erste Timing-Modelle
- Auswahl des Ansatzes je Macro:
  - Semi-Custom bzw. Random Logic Macro oder
  - Full-Custom

Unit Planung

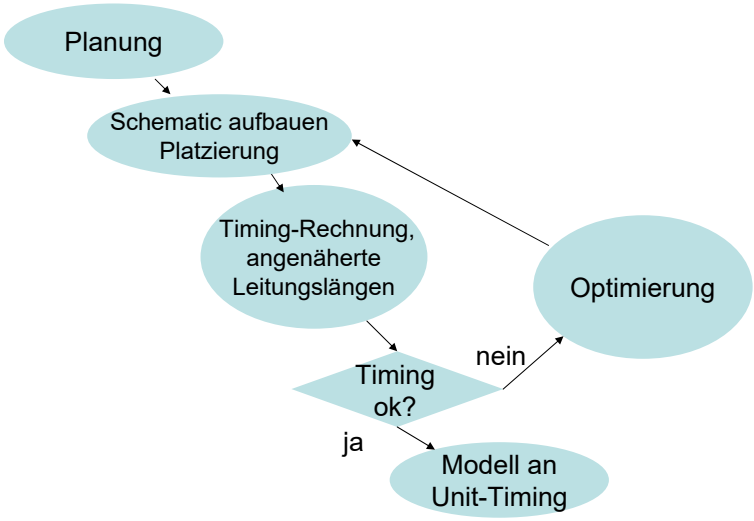
- Unit- und Chip-Integratoren:
  - Platzierung der Macros
  - Lage der Busse
  - Grobe Lage der Anschlüsse an den Makros



Design-Zyklus

- Rückmeldungen von Circuit Design an Logik-Design:
  - Repartitionierung?
  - Duplizierung?
  - Funktionsabsicherung und Fehlererkennung?
  - ➢Logik-Design ändert VHDL-Beschreibung…

Macro-Design



Schematic wird geprüft

- Einige Dutzend Regeln
- Prüfprogramm:
  - Topologie ok?
  - Belastung der Treiber?

- Review-Meeting:
  Schaltung wird am Bildschirm vorgestellt
- Checkliste

Physical Design

Abstract für Macro erstellen:

- Pin-Platzierung
- Aufteilung Wiring-Ressourcen
- Blockaden

… für Unit PD-Integrator

Layout aufbauen

- Hierarchie wie in Schematic
- Transistoren, Gatter endgültig platzieren
- Verdrahtung spielt immer größere Rolle
- Jede Ecke, jeder Versatz ist ein Risiko für die Ausbeute…

… jedes Einzel-Via ebenso

Layout …

- Abstände zwischen Leitungen maximieren
- Kompromiss: Dichte oder Geschwindigkeit Ergebnisse der Timinganalyse werden herangezogen
- Via-Widerstände werden durch Ver-doppelung und durch Überlappung mit Metall herabgesetzt

Auf Unit-Ebene…

- Unit-Integrator verdrahtet gesamte Unit
  - Symmetrische Verteilung der Taktsignale
  - Parasitäre Eigenschaften der Netze für Unit-Timinganalyse modellieren
  - Buffer werden eingeführt und platziert

Release-Phase

- Chip Integration
- Chip Checking:
  - Korrekte Verdrahtung
  - Einhaltung der Layout-Grundregeln
  - Signalintegrität der Nutzsignale
  - Stabilität auch bei Alterung
  - Testbarkeit aller Signale
  - Timing- und Poweranalyse des gesamten Chips

Ausblick: Wo geht die Reise hin?

- Es gibt immer mehr Effekte, die bedacht werden müssen:
  - Ausbeute
  - Parasitäre Elemente an der Verdrahtung
    - Timing
    - Verlustleistung
  - Relative Fertigungstoleranzen werden größer
    - Kanallänge
    - Dielektrizität des Isolationsmaterials
    - Widerstand der Vias…

Ausblick…

- Immer mehr Methodology-Checks werden in Prüfprogramme eingebracht
- Fehlermöglichkeiten werden so früh erkannt
- Audit-Levels geben der Projektleitung einen klaren Blick, an welcher Stelle noch Arbeit zu tun ist

Ausblick…

Aber:

Die Checks sind zeitintensiv

Vorteil jedoch:

Selten Fehlerrückmeldungen
von Fabrik und Bringup

Vielen Dank!

- Wenn Sie noch Fragen haben…

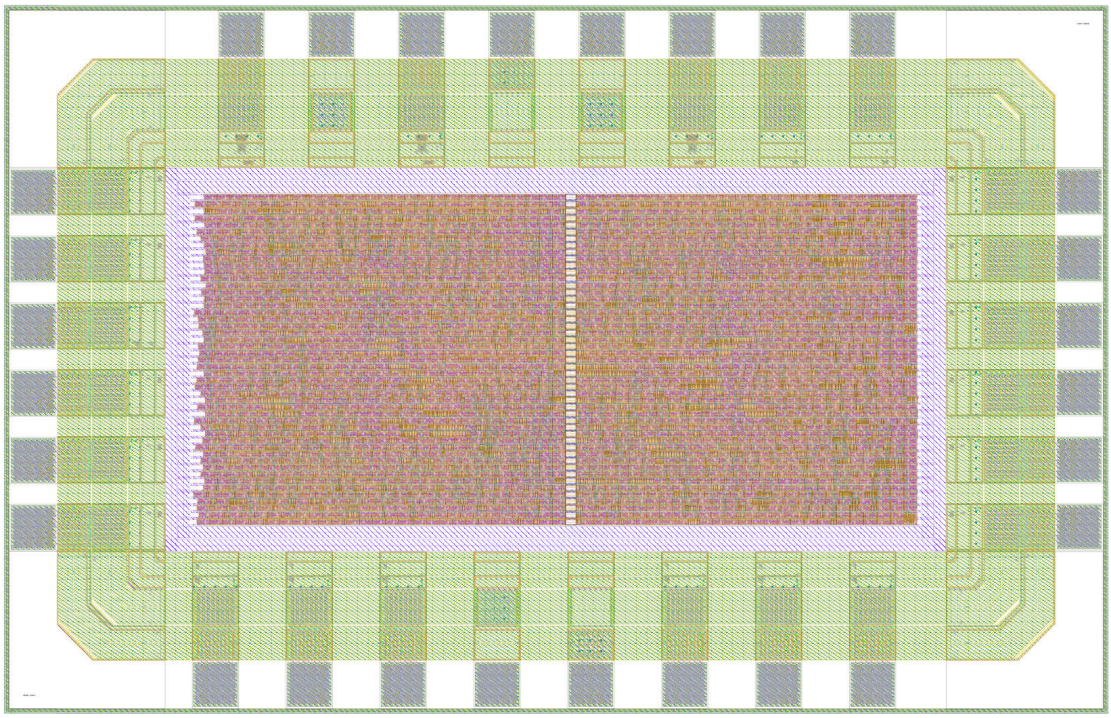
  …dann fragen Sie doch jetzt.

Oder:

wandel@de.ibm.com

Christoph Wandel
IBM Deutschland Entwicklung GmbH

Versatile Search Processor Array ASIC



Design: Fachhochschule Mannheim
Designer: Avi Epstein

Process: AMS 0.35µ CMOS, 4 metal layers (C35B4)
Waferfab: FHG IIS, Run Nr. 1198, 6. Jul. 2005
Dimensions: 2.4 x 1.5 mm
Frequency: 200 MHz
Array size: 32 x 32 cells
Package: SOIC28

Function: The ASIC implementation of a novel systolic array architecture for complex database searches. It is intended for the implementation of very fast complex motif searches in biological databases. It is, however, a promising tool for all fields requiring extreme high-throughput of data and flexible motif definition like comparative genetics or high throughput screening.

The ASIC can handle motif length of up to 32 and a character-set of up to 32. It performs up to 0.2 TeraOPS with a core size of 1 mm^2.

Further development into 65n CMOS will enable packing 256 arrays capable of handling full character searches (8-bit) and motif length of 64 each on a single 150 mm^2 die. Such a systolic array will be performing up to 1 ExaOPS.

LED-Flasher

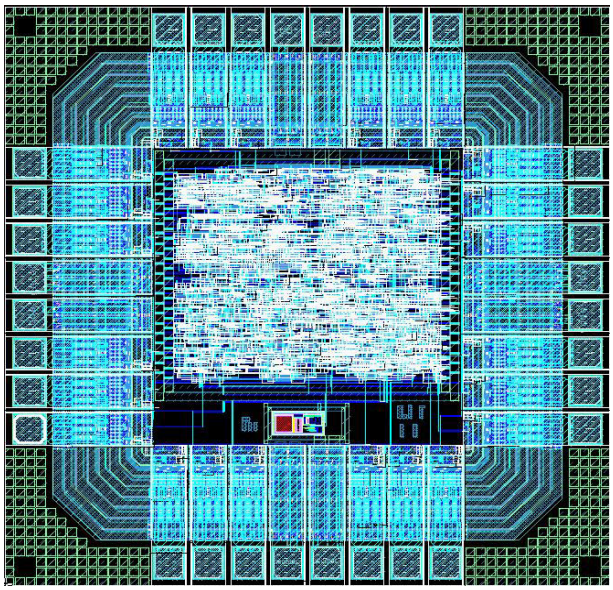


Abbildung 1: Layout auf IC-Station

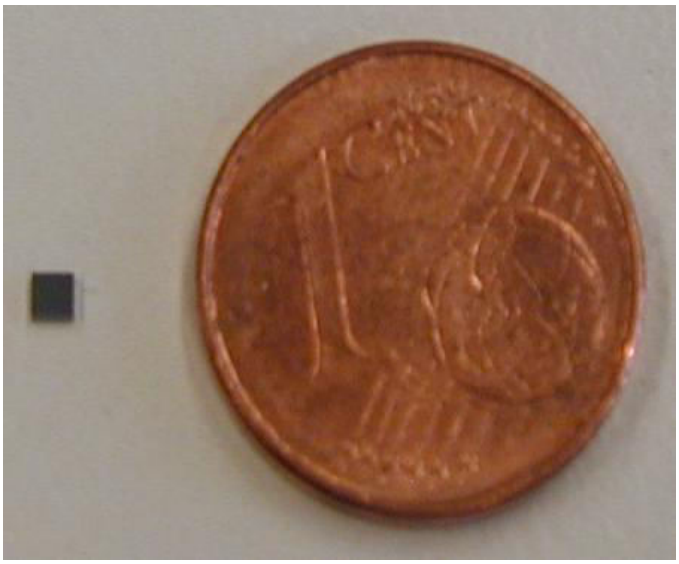


Abbildung 2: Fertiger Chip (1,5 mm²) im Centstück-Vergleich

Entwurf:	Fachhochschule Offenburg Bearbeiter: Daniel Bau Betreuer: Prof. Dr.-Ing. Dirk Jansen
Layouterstellung:	Fachhochschule Offenburg (Standardzellenentwurf), Prof. Dan Wang, Oszillator (VCO mit konstanter Frequenz)
Technologie:	AMIS 0.35 µm CMOS A
Chipfertigung:	Europractice, MPW Run 1213 (mini@sic)
Herstelldatum:	Juli 2005
Kostenträger:	MPC-Mittel FH-Verbund Baden-Württemberg
Chipdaten:	Chipgröße: 1500 x 1500 µm Gehäuse: JLCC 44 Komplexität: ca. 10000 Transistoren
Funktion:	Der LED-Flasher steuert bis zu acht LED´s über Zufallsgeneratoren an, so dass ein unregelmäßiges, nicht vorhersehbares Blinkmuster entsteht. Bei entsprechender Konfiguration ist dieses Muster geeignet, die Aufmerksamkeit des Betrachters anzuziehen, ohne dass der Betrachter sich des Blinkens bewusst ist.
Testergebnisse:	Das Design wurde zunächst als VHDL-Entwurf konzipiert, auf einem FPGA emuliert, als ASIC-Design mit Hilfe der Cadence- und Mentor-Software geroutet, über EUROPRACTICE im Miniasic-Programm gefertigt und erfolgreich in Betrieb genommen.