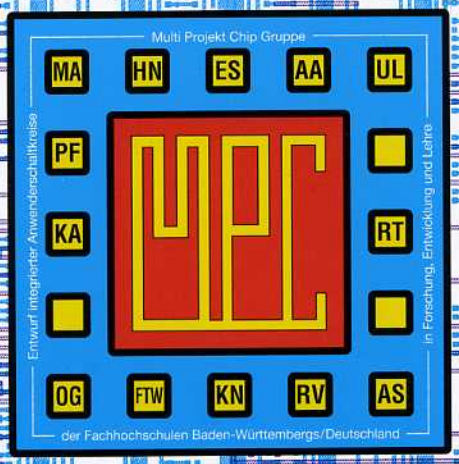
MULTIPROJEKTCHIP GRUPPE

BADEN-WÜRTTEMBERG

MPC-Workshop Juli 2003

Konstanz



MULTIPROJEKTCHIP GRUPPE

BADEN-WÜRTTEMBERG

MPC-Workshop Juli 2003

Konstanz

Cooperating Organization
Solid-State Circuits Society Chapter
IEEE Germany Section

Herausgeber: Fachhochschule Ulm

© 2003 Fachhochschule Ulm

Das Werk und seine Teile sind urheberrechtlich geschützt. Jede Verwertung in anderen als den gesetzlich zugelassenen Fällen bedarf deshalb der vorherigen schriftlichen Einwilligung des Herausgebers Prof. A. Führer, Fachhochschule Ulm, Prittwitzstraße 10, 89075 Ulm.

Adressen der

MULTIPROJEKT-CHIP-GRUPPE (MPC-Gruppe)

BADEN - WÜRTTEMBERG

http://www.mpc.belwue.de

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

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

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

Fachhochschule Furtwangen
Prof. Dr. Rülling, Postfach 28, 78113 Furtwangen
Tel.: 07723/920-503, Fax: -610, Email: ruelling@fh-furtwangen.de

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

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

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

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

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

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

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

Fachhochschule Reutlingen
Prof. Dr. Kreutzer, Federnseestr. 4, 72764 Reutlingen
Tel.: 07121/341-108, Fax: -100, Email: hans.kreutzer@fh-reutlingen.de

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

Inhaltsverzeichnis

Workshop-Vorträge

		Seite
1.	Single-Chip Implementierung eines 5 GHz WLAN Modems E. Grass, IHP Frankfurt (Oder)	5
2.	Design Automation Conference, DAC 2003 in Anaheim, Kalifornien D. Jansen, FH Offenburg	7
3.	Erschließung einer 0,35 µm Technologie für die Bearbeitung mit Mentor Tools M. Striebel, D. Jansen, FH Offenburg	13
4.	Redesign eines USB-Controllers in VHDL und Emulation mit einem Mikroprozessorkern auf FPGA A. Kurz, D. Jansen, FH Offenburg	19
5.	Datenbankbasierte Modellverwaltung J. Siegl, FH Nürnberg	25
6.	Moderne Entwicklungen in Speichertechnologien G. Higelin, FH Furtwangen	29
7.	Messtechnische Ermittlung des Intercept-Punktes IP3 A. Defer, H. Sapotta, FH Karlsruhe	33
8.	Die Muller-Pipeline Simulation und Implementierung einer asynchronen Kontrollstruktur K. Fahrion, E. Grass, FH Aalen	39
9.	Compiler controlled hardware synthesis for resource aware implementations of microprocessors B. Hoppe, N. Lugowski, A. Fallberg, E. Steindl, H. Meuth, FH Darmstadt	59
10.	Implementation of Signal Processing Algorithms Using FPGAs R. Bartholomä, F. Kesel, FH Pforzheim	65

Single-Chip Implementierung eines 5 GHz WLAN Modems

Dr. Eckhard Grass, IHP, Im Technologiepark 25, 15236 Frankfurt (Oder)
e-mail: grass@ihp-microelectronics.com

Seit die Standardisierungen für 5 GHz WLANs nach IEEE802.11a abgeschlossen ist, arbeiten eine Reihe von Firmen mit Hochdruck an der Implementierung entsprechender Systeme. WLANs nach IEEE802.11a und verwandte Nachfolger werden eine Reihe neue Anwendungen ermöglichen. Dazu zählen drahtloser Internetzugang mit hoher Datenrate und mobile Echtzeit Videoanwedungen.

Die sich ergebenden Anforderungen an die Qualität der Analogschaltungen und die Leistungsfähigkeit der digitalen Basisband-verarbeitung sind äußerst hoch. Für mobile Anwendungen ist die Batterielebensdauer ein entscheidender Akzeptanzfaktor.

In diesem Vortrag werden Entwurfsverfahren, Schaltungstechniken und Herstellungs-technologien diskutiert, die Anforderungen für die mobile Breitbandkommunikation gerecht werden. Erste Ergebnisse der Single-Chip Implementierung eines 5 GHz WLAN Modems werden berichtet.

1. 5 GHz WLAN Spezifikation

Der Standard IEEE 802.11a spezifiziert ein WLAN mit Datenraten zwischen 6 und 54 Mbit/s. die Reichweite ist im out-door Bereich mit maximal 300 m angebeben. Als Modulationsverfahren wird OFDM verwendet, welches sehr hohe Anforderungen an die digitale Basisband-verarbeitung stellt. Die OFDM-Symboldauer beträgt 4 µs.

Ausgehend von der Spezifikation im Standard werden zunächst die wichtigsten Funktionsgruppen des digitalen Basisbandprozessors (BB) identifiziert. Eine Abschätzung der benötigten Rechenleistung ergibt, dass im Empfangsbetrieb der Viterbi-Decoder die weitaus größte Rechenleistung benötigt. Danach folgt die FFT. Für den Sendebetrieb ist die inverse Fouriertransformation (IFFT) am rechenintensivsten. Daher wurde entschieden, dass zumindest diese Funktionsblöcke in Hardware zu implementieren sind.

Für den Betrieb in mobilen Terminals ergibt sich die Forderung, die Energieaufnahme unter 1,5 W im Dauerbetrieb zu bringen. Aus Kostengründen ist wünschenswert, dass die Chipfläche unter 60 mm^2 (0.25 µm Technologie) liegt.

2. Modem Architektur

Eine Reihe von alternativen Realisierungs-möglichkeiten bietet sich an. So ist u.a. die Entscheidung zu treffen, ob das Analog Frontend (AFE) als ‚Zero-IF', ‚Low-IF' oder ‚Superheterodyn' Architektur verwirklicht werden soll. Um das Risiko für die geplante Single-Chip Implementierung zu minimieren, haben wir uns für die am besten bekannte und beherrschte ‚Superhet' Lösung entschieden. Damit werden auch die Forderungen an die A/D- und D/A-Wandler in Grenzen gehalten. Nachteilig ist jedoch, dass externe ‚Surface Acoustic Wave' (SAW) Filter bei der Zwischenfrequenz von 810 MHz benötigt werden.

Für die wesentlichen Funktionen des BB besteht u.a. die Möglichkeit, hochleistungsfähige DSP einzusetzen. Da eine solche Softwarelösung im Energieverbrauch immer ungünstiger als eine dedizierte Hardware ist, haben wir uns für eine Datenpfadarchitektur ohne speziellen Prozessor entschieden. Steuerungsfunktionen werden durch Zustandsautomaten in Hardware ausgeführt. Um dennoch eine gewisse Flexibilität und Konfigurierbarkeit des Systems zu erreichen, können eine Reihe von Parametern des BB durch den Protokollprozessor (MAC) eingestellt werden. Dies betrifft insbesondere Schwellwerte und Zeitkonstanten für die Synchronisationseinheit. Der Protokollprozessor (MAC) wurde als Hardware-Software-Kodesign implementiert. Eigene Unter-suchungen haben gezeigt, dass mit einer reinen Softwarelösung Echtzeitforderungen nur dann erfüllt werden können, wenn extrem leistungsfähige Prozessoren eingesetzt werden. Dies widerspricht jedoch der Forderung nach geringem Energieverbrauch. Daher wurde ein spezieller Hardware-Accelerator entwickelt, der gemeinsam mit einem MIPS Prozessor die Funktionen des MAC ausführt. Es ist so möglich, diesen Prozessor mit

nur 80 MHz zu takten und trotzdem alle Echtzeitforderungen (Datendurchsatz, Latenz, Antwortzeiten) mühelos zu erfüllen. Als Interface des MAC zu einem Host-System wurde ein PCMCIA (Card-Bus) Interface gewählt.

3. Designflow und ‚Challanges'

Das AFE wurde auf der Basis der Systemarchitektur spezifiziert und mit den üblichen Entwicklungswerkzeugen implementiert. Dabei wurden zunächst Komponenten wie PLL, LNA oder VGA diskret implementiert und getestet. Auf dieser Basis wurde eine hochintegrierte Version des AFE entworfen. Diese bildet die Grundlage für die Systemintegration als Single-Chip Lösung.

Der Basisbandprozessor (BB) wurde zunächst in MATLAB bzw. SPW (Cadence) modelliert. Dieses Systemmodell wurde umfangreichen Tests unterzogen. Anschließend wurde das gesamte System in VHDL modelliert und gegen das MATLAB-Modell verifiziert. Zur VHDL-Synthese wurden ‚Synopsys' und für das Layout Cadence's ‚First Encounter' Tools verwendet.

Für den Protokollprozessor (MAC) wurde zunächst ein Verhaltensmodell in der, auch vom IEEE zur Standardisierung verwendeten ‚Specification and Description Language' (SDL) erstellt. Dieses wurde mithilfe einer umfangreichen Testumgebung verifiziert. Aus diesem SDL Modell kann automatisch C-Code generiert werden. Anfängliche Versuche diesen C-Code in Echtzeit lauffähig zu machen haben gezeigt, dass die Effizienz durch einen dedizierten Hardware Accelerator erheblich erhöht werden muss.

Die wesentlichen Herausforderungen im Prozess der Single-Chip Implementierung des 5 GHz Modems waren:

Hohe Systemkomplexität: Diese wurde durch Anwendung eines token-flow Konzeptes, verbunden mit lokaler Steuerung gelöst.

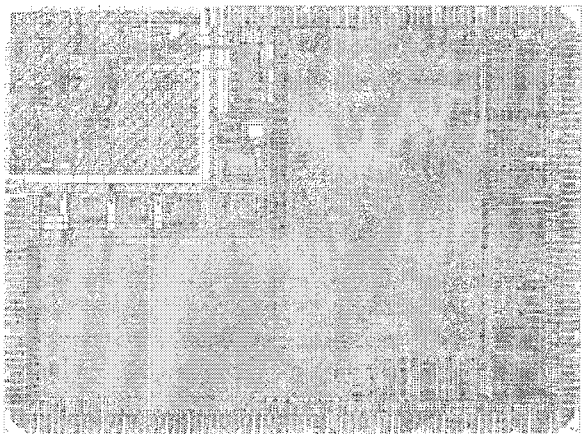
Energieverbrauch: Hier kommen zum einen eine ‚vertikale Optimierung', bei der mehrere Ebenen des OSI-Stacks gemeinsam betrachtet werden, als auch schaltungstechnische Maßnahmen zum Einsatz. Dazu zählen u.a. clock-gating und der Einsatz von asynchronen Schaltungen.

Elektromagnetische Interferenz: Die Interferenz des Digitalteils mit Komponenten des AFE stellt ein wesentliches Problem dar. Zur Verringerung dieser Interferenz können technologische Maßnahmen wie ‚buried layer shields', ‚n-well ringe' oder schaltungstechnische Maßnahmen wie etwa asynchrone Schaltungen eingesetzt werden.

Teststrategie: Es wurde ein Built-In Self-Test (BIST) implementiert, der alle wesentlichen Funktionen des BB testet, sowie den Test von AFE und MAC unterstützt.

4. Zusammenfassung

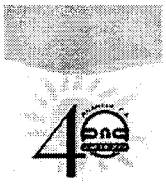
Ein 5 GHz WLAN Modem entsprechend dem Standard IEEE802.11a wurde auf einem einzigen Chip in der IHP-eigenen 0.25 µm SiGe:C BiCMOS Technologie integriert. Der Chip beinhaltet das 5 GHz Analog Frontend, A/D und D/A Wandler sowie den Basisbandprozessor (BB) und den MAC-Prozessor. Der Chip (Layout in Abb. 1) hat etwa 7 Mio. Transistoren von denen 500 bipolare SiGe-HBT's sind. Das tape-out für diesen Chip war Mitte Juli 2003 und erste Ergebnisse werden für Mitte Oktober 2003 erwartet. Die Chipfläche beträgt etwa 90 mm^2 und die maximale Leistungsaufnahme liegt bei etwa 2 W (nach Synthese).



Abbilding 1: Layout des Single-Chip 5 GHz WLAN Modems

Im Ergebnis der Implementierung wurden folgende wesentlichen Erfahrungen gemacht:

- Die Systemintegration ist ein relative aufwendiger und langwieriger Prozess und benötigt in der Regel mehr Zeit als geplant.
- Der Prozess des Physical Design (Floorplanning, Layout, Clock-tree Generierung, Routing, etc.) ist bei Schaltungen dieser Größe sehr komplex und benötigt erhebliche Ressourcen und Zeit.
- Test Strategien müssen zu einem sehr frühen Zeitpunkt in der Designphase berücksichtigt werden.
- Das Leiterplattenlayout muss schon frühzeitig bedacht werden und sollte gemeinsam mit dem I/O-Placement für den Chip erfolgen. Insbesondere die Platzierung der analogen HF-Signale muss mit dem Leiterplattenlayout abgestimmt werden.



Design Automation Conference, DAC 2003 in Anaheim, Kalifornien 02. – 06.06.2003

Prof. Dr. Dirk Jansen

1. Was ist die DAC?

Die DAC besteht seit dem Jahr 1963 und wurde als ein Treffen zwischen Universitäten und Industriepartnern gegründet. Es ging um das Thema: Programme und Software zum Entwurf von Elektronik wie z. B. Spice. Aus diesen Anfängen heraus hat sich die DAC zum Forum für „Electronic Design Automation" EDA entwickelt, das Anwender und Entwickler von EDA-Systemen zusammenbringt. Sie behandelt insbesondere Forschung auf dem Gebiet der EDA und gibt den Anbietern von EDA-Tools durch eine umfangreiche Ausstellung ein Forum zur Darstellung ihrer Produkte. Neuerdings werden auch so genannte IPs für den IC-Design und Bibliotheken sowie Entwurfssysteme für Embedded Systems, soweit sie IC-orientiert sind, thematisch behandelt.
Die Organisation der Konferenz, die etwa 4.000 Teilnehmer hatte, verlief in 5 parallelen Veranstaltungen und weiteren Workshops und Meetings, die gleichzeitig hierzu angeboten wurden. Die Ausstellung umfasste in diesem Jahr 200 Anbieter mit großer Demo-Suit. Umrahmt wurde die Konferenz von fachspezifischen Tutorials. Das Programm umfasste ferner Keynote Papers, in denen bedeutende Mitglieder der EDA-Szene Tendenzen der Entwicklung aufzeigten, Panneldiskussionen und nicht zuletzt Social Events, in diesem Jahr ein Abendbesuch von Disneyland mit Country Buffet und Gelegenheit zum intensiven Austausch der Konferenzteilnehmer.

Die Konferenz präsentierte in diesem Jahr 180 Original Papers, überwiegend aus dem Universitätsbereich, etwa ein Drittel auch aus der Industrie. Die Papiere sind etwa aus dem dreifachen Vorschlagsvolumen selektiert und von hoher Qualität. Im Folgenden sollen einige herausragende Beiträge dargestellt werden.

2. Zentrale Aussagen

2.1. Vortrag von Sir Robin Saxby, Direktor ARM

Der erste Keynote Speaker der Konferenz war SirSaxby von Advanced Risc Machines (ARM). Zunächst ging er auf die technologische Entwicklung ein und erklärte, dass das Moore`sche Gesetz, das eine Verdoppelung der Transistorzahlen pro Chip alle 1 ½ Jahre voraussagt und nun schon seit den 70`er Jahren des letzten Jahrhunderts gültig ist, auch noch in den nächsten 15 Jahren im Wesentlichen die Zukunft der Technologieentwicklung bestimmen wird. Eine Sättigung ist derzeit noch nicht erkennbar. Er stellte die Frage: „Was macht man mit den vielen möglichen Transistoren?"

ASIC`s mit mehr als 100 Mio. Transistoren werden in Zukunft realistisch. Mit den heutigen Entwurfswerkzeugen ist diese Komplexität nicht zu bewältigen. Die Design-Effizienz muss gesteigert werden. Derzeit steigen die Design-Kosten mit der 3. Potenz, die Maskenkosten quadratisch. Beide Kostensteigerungen sind nur noch bei High Volume Applications tragbar. Die Aufteilung der Industrie in FAB`s (Technologiehersteller) und IP-Provider wird sich verstärken.

Im Jahr 2007 wird es Chips mit 1 Mrd. Transistoren geben. Solche Designs sind nur entwickelbar, wenn in großem Umfang Soft- und Hardware – Re-use eingesetzt wird. Eine weitere Lösung können Plattformen sein sowie konfigurierbare Chips ähnlich den heutigen FPGA`s.

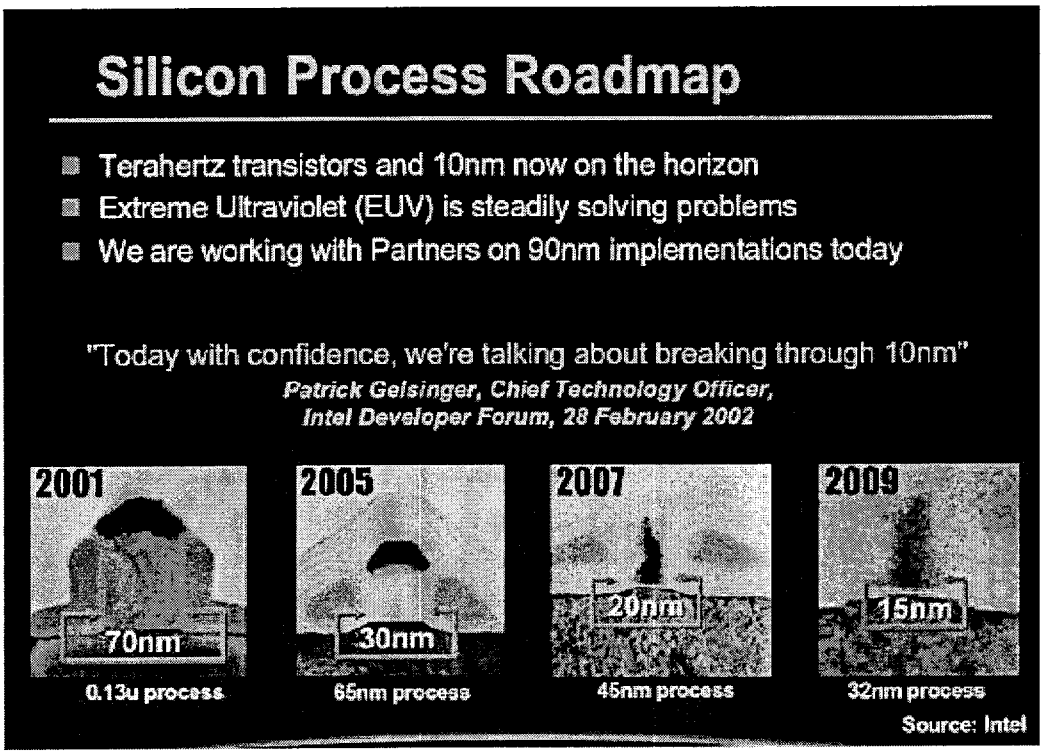


Abb. 2.1-1: Silicon Process Roadmap (Keynote Paper)

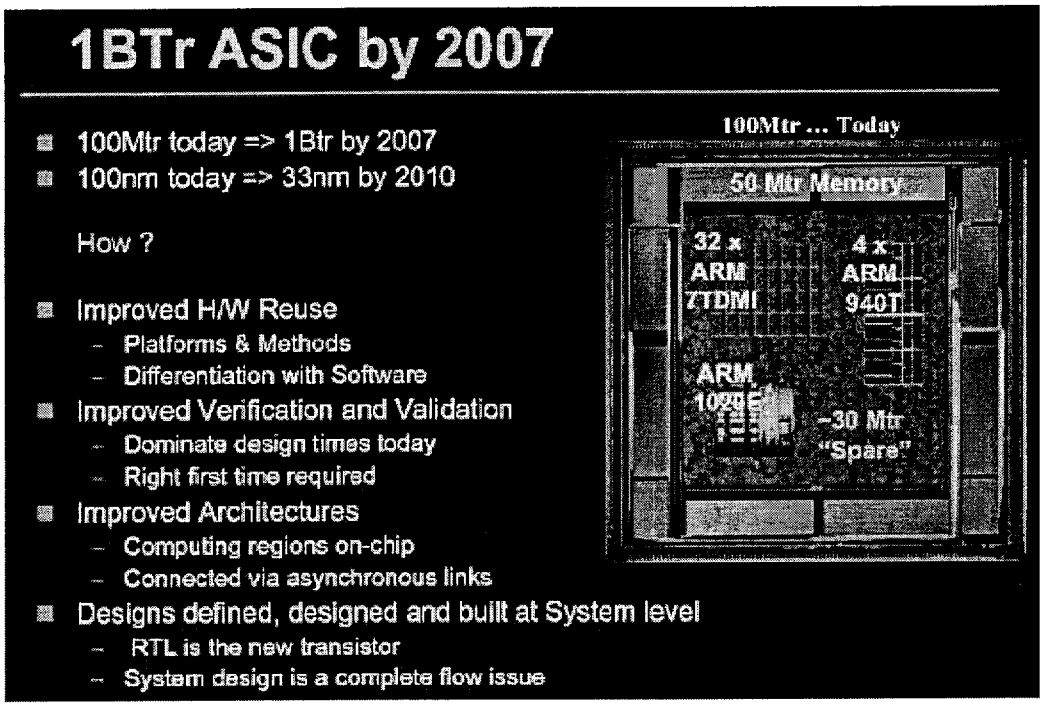


Abb. 2.1-2: 1 Billion Transistor ASIC by 2007 (Keynote Paper)

2.2. Vortrag von Prof. Alberto Sangiovanni-Vincentelli, UC Berkeley

Prof. Vincentelli zeigte anhand von eingängigen Bildern den Zyklus der EDA-Industrie auf. Das Pendel geht zwischen Verifikation und Synthese hin und her. Mit den immer steigenden Anforderungen und immer größeren Designs sind dringend neue Abstraktionsebenen erforderlich, um die Komplexität zu beherrschen. Der Design Gap, d. h. der Unterschied zwischen dem, was mit den bestehenden EDA-Tools in einer realistischen Zeit entworfen werden kann und dem, was von der Technologie her an Komplexität und Fläche möglich wäre, wird immer größer. Mit den extrem steigenden Einmalkosten (Maskensätze) wird es für kleine Firmen immer schwieriger, noch komplette Designs durchzubringen. Sie werden sich deshalb auf die Entwicklung von IP's, d. h. Substrukturen als Soft- oder auch Hardmakros konzentrieren müssen. Prof. Vincentelli sieht deshalb eine blendende Zukunft für programmierbare reguläre Strukturen jeder Art, insbesondere FPGA's. Er erhebt auch hier die Forderung, dass die EDA-Tools mit der Technologieentwicklung Schritt halten müssen:

EDA = Schlüssel für die Zukunft der Mikroelektronik

Fig 1. Increasing Design Size

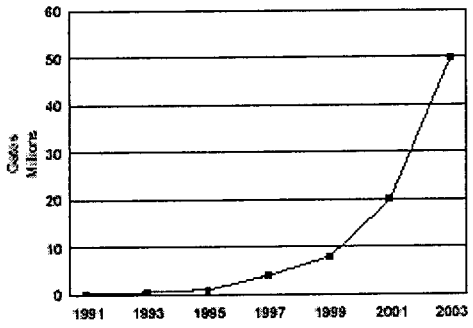


Abb. 2.2-1: Increasing Design Size

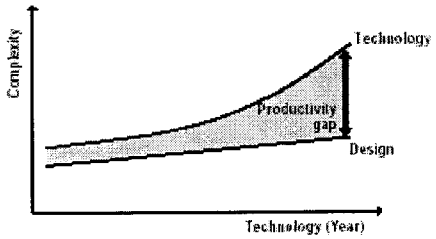


Fig. 1: Depiction of the design productivity gap.

Abb. 2.2-2: Productivity Gap

3. Übersicht über die wichtigsten Sessions und ausgewählten Vorträge

Die Konferenz bestand aus 55 Einzelsessions mit jeweils 3-4 Vorträgen in 5 parallelen Tracks. In diesem Jahre waren die Schwerpunkte

- Design for Power,
- Global Routing and Physical Design,
- Physical Synthesis,
- Simulation and Verification Technology
- Technology Modelling,
- High Level Synthesis,
- Embedded Systems on Chips.

Auf diese umfangreichen Programme kann naturgemäß nicht vollständig eingegangen werden. Es sollen hier einige Highlights herausgegriffen werden, die besondere Fortschritte kennzeichnen. Die Auswahl ist nicht repräsentativ.

3.1. Fortschritte auf dem Gebiet der Formalen Verifikation

Hier gab es einen Vortrag der Firma Intel über die Formale Verifikation des Pentium 4 Prozessors. Diese Aktivität war von Intel vor 5 Jahren in großem Stil begonnen worden als Reaktion auf die so genannten Pentium Bugs. In einem Team von etwa 20 Mitarbeitern wurde wichtige Grundarbeit geleistet, die sich nun als sehr erfolgreich und als Schlüssel für weitere Prozessorentwicklungen darstellt. Mit diesen Methoden konnten im

- Pentium 1: ca. 800 Bugs
- Pentium Pro: 2240 Bugs
- Pentium 4: 7855 Bugs, davon 8 schwer wiegende wie Deadlocks, falsche Ergebnisse, Blockaden usw.

gefunden und beseitigt werden, wobei es bisher immer noch nicht möglich ist, den Prozessor als

Ganzes zu verifizieren. Dies ist ein Forschungsgebiet für die nächsten Jahre und wird noch umfangreiche Arbeit erfordern. Es wird als Schlüssel für eine weitere Steigerung der Komplexität angesehen, andernfalls wird das Risiko für unentdeckte Fehler einfach zu hoch. Der Sprecher betonte auch die kommerzielle Seite von spät erkanten Fehlern, die, wenn sie erst beim Kunden auftreten, für eine Firma ruinöse Folgen haben kann. Für sicherheitsrelevante Anwendungen, z. B. im Automobil wird FV eine Notwendigkeit.

3.2 Fortschritte im Bereich von Hochfrequenz und Mixed Signal Schaltungen

In Vorträgen von Philips, IMAC und anderen Herstellern bzw. Universitäten wurden komplette RF Frontends für Mobilfunk und Wireless LANs vorgestellt. Hierbei handelt es sich um komplette Systeme, die in nur einem Chip integriert sind. Die besonderen Herausforderungen bei diesen Designs sind niedriger Leistungsverbrauch, die extreme Komplexität durch Mischung von Analog- und Digital Design sowie die Verifikation solcher heterogener Systeme- Es wurden komplette RF-SOC für Blue Tooth, W-LAN, GPS und andere für Massenanwendungen vorgestellt. Diese Designs enthalten Prozessoren mit 32 Bit und verbrauchen nur einige Dutzend mW im aktiven Betrieb. Kennzeichnend ist, dass der 2,4 GHz-Bereich heute komplett mit Silizium abgedeckt wird. Im Verifikationsbereich dominieren immer noch Spice-Programme in jeder Form, VHDL-AMS und DSP-Programme.

3.3 Fortschritte im Bereich Place und Route

Es gab zahlreiche Papers zu Place und Route. Hier gibt es unterschiedliche Neuentwicklungen. IBM stellte Designs vor, in denen Standardzellen bis zu Millionen-Transistor-Chips eingesetzt werden. Die Bibliotheken sind angepasst mit Timing Control und konfigurierbarer Treiberleistung. Für den Deep Sub Micron – Bereich werden Multi Threshold – Transistoren verwendet. Durch die flexiblen Libraries hält sich die Zahl der Library-Komponenten noch in Grenzen. Eine alternative Entwicklung ist der vermehrte Einsatz von „Physical Synthesis", was wieder dem Platzieren von Einzeltransistoren, skaliert auf die jeweilige Treiberleistung entspricht. Hierdurch wird versucht, das Timing Closure, d. h. das Erfüllen von Geschwindigkeitsbedingungen für gegebene Logikstrukturen zu erleichtern. Ein großes Thema im Zusammenhang mit Deep Sub Micron und Nanometertechnologien ist das Power Grid Routing sowie die damit zusammenhängende Qualität der Stromversorgung. Hier setzen sich in zunehmendem Maße Gitterstrukturen in den höheren Metallebenen als ausgeprägte Power Plains durch. Auch das Thema Clock Distribution bleibt ein Dauerthema, neben die noch gar nicht so alten PLL-Lösungen treten wieder klassische Gitterverteilungen. Die Software zu diesen Themen versucht die komplexe Interaktion von Placement, Timing Constraint – Vorgabe und Routing zu erfassen. Routing wird ein immer komplexerer Prozess im Deep Sub Micron – Bereich und deshalb immer teurer.

3.4 Deep Sub Micron Technology

Weitere Verkleinerungen der Strukturen werden vorhergesagt. Über Erfahrungen mit der 90 nm – Technologie wurde berichtet. Mit gezeichneten 90 nm ergibt sich eine effektive Kanallänge von 45 nm. Aufgrund der großen Toleranzen in der Lithografie sind große Sicherheitsmargen erforderlich, um eine vernünftige Ausbeute zu erreichen. Die Prozesstoleranzen sind vergleichsweise groß. Mit bis zu 25 Masken sind die Maskenkosten extrem hoch und können bis zu 1 Mio. US $ betragen. Fortgeschrittene Prozesse verwenden fast ausschließlich Kupfer in bis zu 8 Metall-Lagen. Leakage der Transistoren bei niedrigen Schwellspannungen ist ein großes Problem bei schnellen Schaltungen. Deshalb werden Transistoren mit unterschiedlichen Schwellspannungen erforderlich, wobei nur dort die schnellsten Transistoren eingesetzt werden, wo sie unbedingt notwendig sind. Bei den niedrigen Spannungen um 1 V wird die Power Grid Quality ein großes Problem. Um eine gute Signalqualität zu erhalten, müssen Abschirmmaßnahmen integriert und Bussignale als Differenzsignale geführt werden. Oberhalb von 2 GHz ist die Leitungstheorie im Chip zu beachten. Allgemein ist der Signalstörabstand, die so genannte Signal Integrity, ein ernstes Problem für die Entwicklung.

3.5 Entwicklung im Bereich der Mikroprozessoren

IBM stellte den Power 4 Prozessor, ein 1,3 GHz, 64 Bit Prozessor in 180 nm SOI – Technologie vor. Er verwendet 7 Kupferlagen zur Verdrahtung und weist 174 Mio. Transistoren auf. Der Design verwendet Standardzellen (Custom made) und ein Verfahren der Semi Custom Physical Synthesis. Es werden zahlreiche neue Verfahren für den Entwurf komplexer Systeme einschließlich neuartige Konzepte zu Place und Route unter Timing Constrains angewendet.

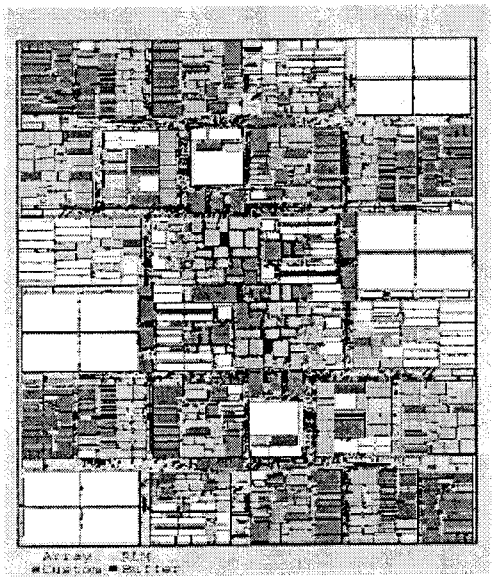


Abb. 3.5-1: Power 4 Prozessor

Fujitsu stellte einen Sparc 64 Prozessor vor. Er basiert auf einer 130 nm SOI – Technologie und nimmt 34,7 W bei 1,8 V Versorgungsspannung auf. Der Design umfasst 190 Mio. Transistoren, davon 19 Mio. Logik. Es handelt sich um eine 4 Isue out of order superscalar – Architektur. In großen Teilen handelt es sich um einen Static Design, Clock Gating wird in großem Umfang eingesetzt. Der Design wurde komplett in Verilog geschrieben, die Verifikation erforderte 100 Bill. Zyklen Logiksimulation.

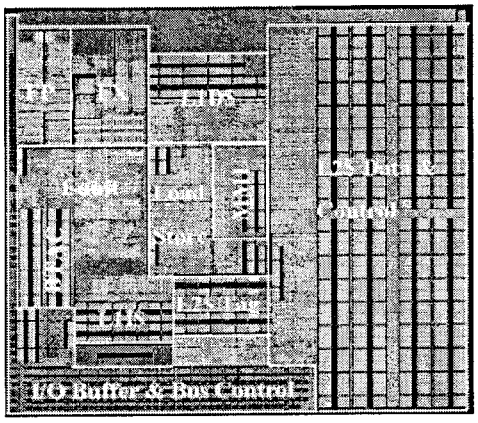


Abb. 3.5-2: Fujitsu Sparc 64

Intel stellte die 3. Generation seines 64 Bit Flaggschiffes Itanium 2 vor. Dieser 410 Mio. Transistoren große Design verwendet eine 130 nM Technologie. Bei 1,3 V Versorgung, 1,3 GHz Taktfrequenz nimmt der Chip im aktiven Zustand 130 W auf. Im Besonderen wurde auf die Taktverteilung im Chip eingegangen. In der Logik werden umfangreiche self timed und pulse circuits eingesetzt. Statische Buffer an vielen Stellen erlauben eine Reduktion des Leistungsverbrauchs. Für die Timing Verification wurden die vollständig extrahierten parasitics verwendet. Die Verifikation erforderte 30 % des Gesamtaufwandes.

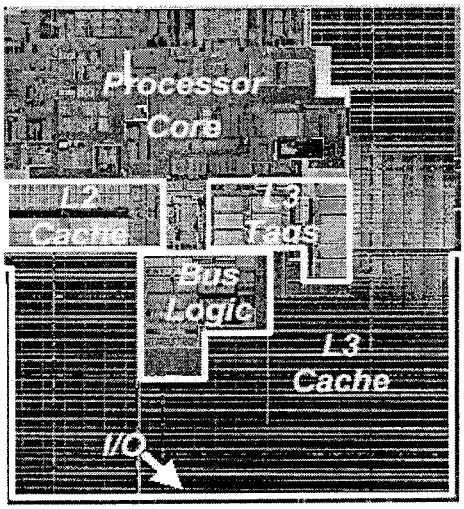


Abb. 3.5-3: Itanium 2 Prozessor Chip

3.6 Entwicklung im Bereich der FPGA`s

Den FPGA`s wird eine breite Zukunft eingeräumt. Bevorzugt werden reguläre, feinkörnige, programmierbare Strukturen. Alternativ dazu gibt es zahlreiche Vorschläge zu Fabrics (Gewebe) aus IP`s und Cores z. B. einzelner Prozessorkerne. Bussysteme werden als Verbindungsstrukturen vorgeschlagen. Der Abstand der FPGA`s zu ASIC`s nimmt zwar ab, ist aber immer noch groß in den Bereichen Kosten und Leistungsverbrauch. Der Fokus liegt heute neben der Verkleinerung der Strukturen auf Power und Timing. Eine größere Bedeutung werden auch FPGA`s mit integrierten Processor Cores gewinnen.

4. Zusammenfassung

Die DAC ist ein internationales Forum für Begegnungen auf dem Gebiet der EDA. Sie vermittelt eine Übersicht über den neuesten Stand der Algorithmenentwicklung und Software für den Entwurf integrierter Schaltungen. Die Ausstellung vermittelte eine Übersicht über das weltweite Angebot an kommerziellen Entwicklungswerkzeugen.

Erschließung einer 0.35µm Technologie für die Bearbeitung mit Mentor Tools

Dipl.-Ing. (FH) Markus Striebel, Prof. Dr.-Ing. Dirk Jansen, ASIC Design Center
Fachhochschule Offenburg, Badstraße 24, 77652 Offenburg
Tel. 0781/205-274, Fax 0781/205-174
markus.striebel@fh-offenburg.de

Im ASIC-Design-Center der FH-Offenburg wird ein Design Kit für die AMI 0.35µm Technologie erstellt. Der Design-Kit ist eine Zusammenstellung aller Dateien , die für ein Chipdesign unter Verwendung der Mentor Tools für den AMI 0.35µm Prozess benötigt werden.

1. Einführung

Der EKG-Datenlogger (Abb. 1), der derzeit in der 0.5µm Technologie der Firma AMI vorliegt, soll nun in der 0.35µm Technologie entworfen werden. Für den Entwurf des Digitalteils werden somit Standardzellen der Firma AMI verwendet. Für die 0.5µm Technologie stand ein kompletter Design-Kit zur Verfügung. Dieser Design-Kit (EP_MTC_CMOS05_v1.0) wurde von Europractice entwickelt. Für die 0.35µm Technologie ist derzeit kein kompletter Design-Kit verfügbar. Erhältlich sind GDS2 –Beschreibungen der Zellen, Technologie-Dateien für Synopsis, Vital-Codes für die Simulation, und Rules-Files für Verifikation der gerouteten Zelle mit Calibre. Wichtige Dinge wie z. B. Prozess-Dateien für das Routen oder auch Schaltplansymbole fehlen.

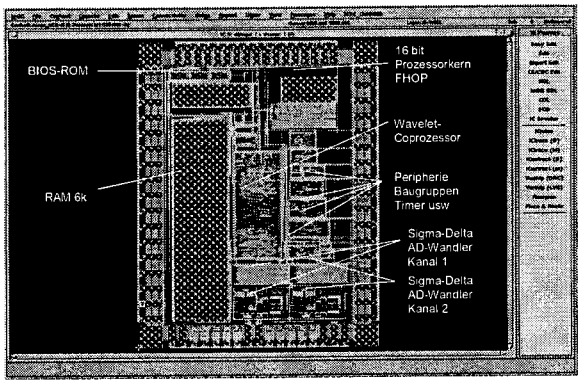


Abbildung 1: EKG-Datenlogger Chip mit Stanardzellenentwurf in der 0.5µm Technologie von AMI.

Der Design-Kit FHO_MTC_CMOS035_v1.0, der all diese Dateien enthält, wird derzeit an der Fachhochschule Offenburg entworfen.

2. Technologievergleich

Ein wesentlicher Vorteil der AMI 0.35µm Technologie ist, dass 5 Metallisierungslagen zur Verfügung stehen. Damit lässt sich ein sehr viel kompakteres Routen durchführen. In diesem Zusammenhang ist es nun auch von Vorteil, direkte Durchkontaktierungen (stacked vias) von einer tieferliegenden Metallschicht (z. B. Metall 1) in eine höherliegende Metallschicht (z. B. Metall 5) in der Prozess-Datei zu definieren. Somit lassen sich auch neue Konzepte für das Routen des „TOP-Sheets" erstellen. Auf dieser obersten Ebene kann nun Metall 5 und Metall 4 nur für das „Power-Routen" benutzt werden. Bemerkenswert ist auch, dass die internen Stromversorgungsleitungen der Zellen nur noch eine Breite von 1.4µm aufweisen. Bei der AMI 0.5µm Technologie betrug dieser Wert 4.7µm. Deshalb ist es nun wichtig, in regelmäßigen Abständen einen Metallisierungsstreifen („power-bar") für die Stromversorgung in das Design einzubauen.

	Mietec 0.5	Mietec 0.35
Anzahl der Metallisierungslagen	3	5
Einsatz von Stacked vias	Kein Vorteil Beim Routen	Routingergebnis Sehr viel besser
Breite der Powerleitungen Internal row	4.7 µm	1.4 µm
Minimale Leiterbahnenbreite Metal1	0.7 µm	0.5 µm
Abstand	0.8 µm	0.6 µm
Steilheit GMmax NMOS TYP (VDS=0.1V,Vbs=0V,27 degrees C)	W/L = 10/0.5 274 µA/V	W/L = 10/0.35 578 µA/V

Tabelle 1: Technologievergleich zwischen der 0.5µm und der 0.35µm von AMI [1], [2].

Auch die Analog-Zellen müssen im Zuge einer solchen Prozessumstellung modifiziert werden. Da sich die Steilheit GMmax der Transistoren ändert , müssen alle Transistoren, die für z. B. Stromquellenschaltungen oder Stromspiegel-schaltungen eingesetzt werden, neu dimensioniert werden. Die Leiterbahnenbreite in den verschiedenen Metallisierungsschichten oder Abstände zwischen den Metallisierungsschichten werden kleiner. In Tabelle 1 ist der Wert für Metall 1 aufgeführt.

3. Enwurfsablauf

Anhand des Enwurfsablaufs (Abb. 2) soll gezeigt werden, welche Programme zum Einsatz kommen und welche Dateien für die einzelnen Programme benötigt werden. Für die Synthese mit dem Design Analyzer von Synopsis werden die Technologiebeschreibungen der Logikgatter verwendet. Diese Dateien (Bibliotheken) sind auf der Europractice CD vorhanden, sowohl als ASCI Dateien, so z. B. die Datei MTC45000.lib, die die Beschreibung für alle Standardzellen enthält, sowie in compilierter Form (MTC45000.db). Entsprechend sind auch die Bibliotheken für die PAD-Zellen vorhanden. Für die Timing-Simulation nach erfolgter Synthese mit dem Programm Modelsim, werden die VHDL-Beschreibungen der Logikgatter benötigt, die sogenannten Vital-Codes. Auch diese sind auf der Europractice CD verfügbar.

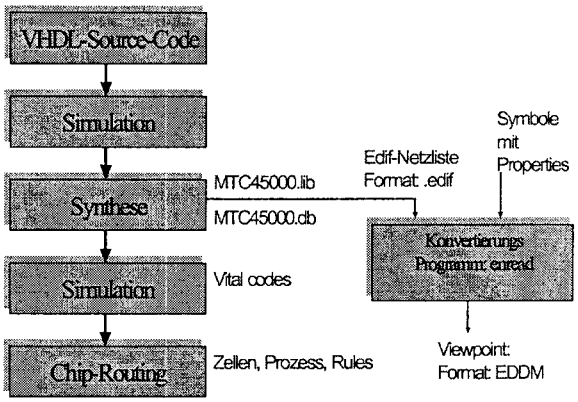


Abbildung 2: Entwurfsablauf beim Chipdesign.

Die Edif-Netzliste (edif-Format), die man als Syntheseergebnis erhält, kann nicht direkt zum Routen des Chips verwendet werden. Zunächst muss man mit dem Programm „enread" das edif-Format (ASCI-Datei) in ein EDDM Format (Erzeugen eines Viewpoints) konvertieren (Abb. 2, Abb. 3). Dieser Schritt ist keine reine Konvertierung. Benötigt wird hierfür ein Verzeichnis, indem sich sie Symbole und Schaltpläne der Logikgatter befinden. Diese Symbole dienen als Träger bestimmter Eigenschaften (Properties).

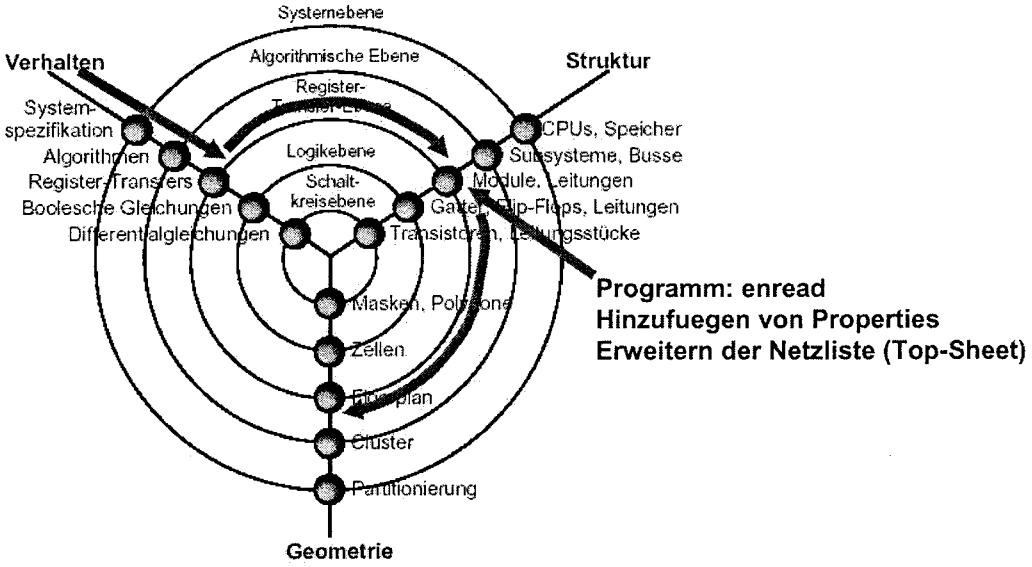


Abbildung 3: Entwurfsablauf beim Chipdesign mit Standardzellen im Gajski-Y verdeutlicht.

Konvertiert man also die Edif-Netzliste , so wird ein Viewpoint erzeugt (Binary Datei), der neben der reinen logischen Verknüpfung der Gatter (Informationsquelle ist die Edif-Netzliste) noch weitere Informationen enthält (Informationsquelle sind die Symbole mit den Properties).

Die Symbole und Schaltpläne der Logikgatter sowie der PAD-Zellen sind nicht erhältlich. Diese mussten mit Hilfe des Design-Architects von Mentor-Graphics entworfen werden.

Mit Hilfe der Viewpoint-Beschreibung seines Designs kann man mit dem Chip-Routen auf der IC-Station von Mentor-Graphics fortfahren. Für das Routen werden die Zellen im IC-Graph- Format benötigt. Vorhanden sind die Zellen jedoch nur im GDS2-Format. Die Zellen müssen mit Hilfe des Programms IC-Link von Mentor Graphics konvertiert werden. Dann müssen sie mit der IC-Station nachbearbeitet werden, da wichtige Dinge wie z. B. Portdefinitionen fehlen.

Selbst geschrieben werden mussten auch die Prozessdateien, die sämtliche Informationen für das Routen enthalten, wie z. B. verwendete Routing-Layer). Die Rules-Dateien (LVS, DRC) die für die Verifikation des gerouteten Designs mit dem Programm Calibre benötigt werden, waren vorhanden. Für einen analogen Designentwurf werden Schaltplansymbole für Transistoren, Widerstände und Kondensatoren benötigt. Auch diese müssen selbst entworfen werden. Die Properties für die Simulation mit dem Programm Eldo, und die Properties für die Device Generatoren, welche in der Prozess-Datei definiert werden, müssen eingefügt werden. Die Spicefiles der Transistoren (BSIM3) sind verfügbar.

4. FHO_MTC_CMOS035_v1.0, der Design Kit für die AMI 0.35µm Technologie

In Abb. 4 ist der Aufbau des Design-Kit dargestellt. Der Ordner *autocells* enthält die Dateien, die für das Routen mit dem Programm Autocells benötigt werden. Das ist zum einen die Autocells Technologie Datei **mtc45000** (ASCI-Format) und die Bibliothek-Datei mit den Zellen-Beschreibungen **mtc45000.L** (ASCI-Format). Außerdem die Datei **Iconvt.opt** (ASCI-Format) mit den Pfaden zu den oben erwähnten Dateien und die Datei **Autocells.PAR_user** (ASCI-Format), in der man die Möglichkeit hat, bestimmte Vorgaben für den Router anzugeben.

Der Ordner *GDS2* enthält die GDS2-Beschreibungen der Zellen.

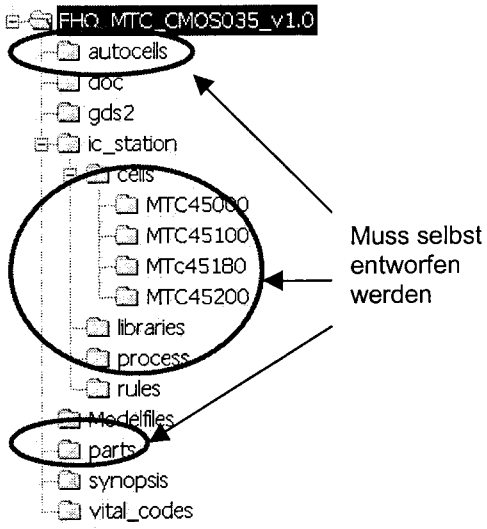


Abbildung 4: Bestandteile des Design-Kits FHO_MTC_CMOS035_v1.0.

Der Ordner *ic_station* enthält Unterordner und Dateien, die für das Routen mit der IC-Station benötigt werden. Der Ordner *cells* enthält den Unterordner MTC45000 mit den für die IC-Station aufbereiteten Standardzellen, den Ordner MTC45100 mit den MID-Profile-IO-Zellen, den Ordner MTC45180 mit den PAD-Limited-IO-Zellen, und den Ordner MTC45200 mit den Core-Limited-IO-Zellen. Der Ordner libraries enthält entsprechend die Bibliotheken **mtc45000, mtc45100, mtc45180, mtc45200.** Der Ordner *process* enthält die Prozesse **block_process, top_process** und **analog_process.** Der Ordner *Rules* enthält die Dateien für die Verifikation des Design mit dem Programm Calibre. Eine Datei für LVS und eine Datei für DRC.

Für die Analogsimulation ist der Ordner *Modelfiles* vorhanden, in dem sich die Spice-Beschreibungen für die verschiedenen Transistor-Modelle befinden. Das sind die Dateien **ntyp.dat, ptyp.dat, nslow.dat, pslow.dat. nfast.dat und pfast.dat**.

Der Ordner *parts* enthält die gleichen Unterordner wie der Ordner *cells*, jeder Ordner enthält die entsprechenden Symbole und Schaltpläne. Außerdem ist der Unterordner *devices* vorhanden, der die Symbole für die Analog-Devices enthält (Transistoren, Widerstand, Kondensator).

Der Ordner *Synopsis* enthält die Technologie-beschreibungen der Gatter für die Synthese mit Synopsis.

Der Ordner *vital_codes* enthält die vhdl-Beschreibungen der Logikgatter für die Simulation mit Modelsim.

5. Enwurf der Schaltplansymbole für den neuen Design Flow mit ADMS

Die Schaltplansymbole und die Schaltpläne der Logikgatter müssen mit dem Programm Design-Architect gezeichnet werden. Die Symbole müssen die richtigen Properties für die Simulation mit **ADMS** (Advanced Mixed Signal Simulator) und für das Routen mit der IC-Station erhalten. Für die Simulation sind die beiden Properties **View** und **Model** [2] von Bedeutung. Für jedes Symbol können mehrere Modelle im Design Architect registriert werden. Für ein Modell gibt es drei verschiedene Beschreibungsmöglichkeiten:

- eine compilierte HDL(VHDL, VHDL-AMS, Verilog),
- eine Spice-Netzliste für die Simulation mit dem ELDO-Kernel, der in das Programm ADMS integriert ist,
- oder ein Schaltplan.

Über das Property **Model** kann dann gewählt werden, welches Modell für die Simulaion verwendet werden soll. Mit Hilfe des Properties **View** kann man auswählen, welche der drei Beschreibungsmöglichkeiten bei einem Doppelklick auf das Symbol angezeigt werden soll. Für die Digitalsymbole wird also ein Modell mit dem compilierten Vital-code erzeugt. Die Analog-Devices müssen mit den richtigen Properties für die Simulation mit dem **Eldo**-Kernel versehen werden.

Der Vorteil des neuen Design-Flows mit ADMS ist, dass eine Schaltung simuliert werden kann, bestehend aus einem analogen Block, der im Spice-Format beschrieben ist, und einem digitalen Block , der in einer HDL-Beschreibung vorliegt. An das Programm ADMS wird dann eine Top-Spice-Netzliste übergeben. Für den Digitalteil wird dann der integrierte Modelsim-Kernel verwendet, für den Analogteil der integrierte Eldo-Kernel. Mit Hilfe des Viewers *Xelga* wird das Endergebnis angezeigt.

Entsprechend müssen alle Symbole noch mit den Properties für die IC-Station versehen werden.

6. Zellenaufbereitung für die IC-Station

Nach der Konvertierung der Zellen vom GDS2-Format in das IC-Graph-Format müssen die Zellen nachbearbeitet werden. Zunächst muss ein Prozess für die Zelle geladen werden. Dann müssen Ports definiert werden, Flächen, die keine Ports sind, müssen mit einem Blockage-Layer versehen und der Shape-Aspect aller Strukturen muss von *internal* auf *internal/external* umgestellt werden. Damit erst werden die Strukturen in einer höheren Hierarchie-Stufe sichtbar. Eine andere Möglichkeit wäre gewesen, die Zellen im LEF-Format (Cadence) zu importieren. Dies funktioniert, jedoch haben de Zellen andere Strukturen, wie Abb. 5 zeigt. Es wurde deshalb entschieden, die GDS2-Beschreibungen zu verwenden.

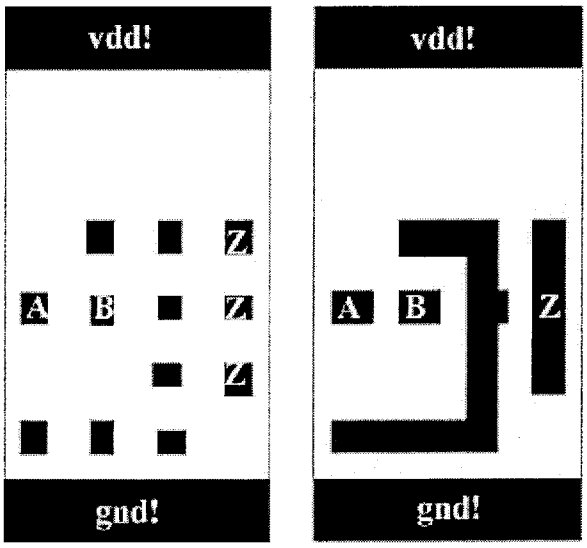


Abbildung 5: GDS2 konvertierte Zelle (AN2) und LEF-Zelle im Vergleich.

7. Entwurf der Prozess-Dateien

Für das Routen mit der IC-Station müssen die Prozess-Dateien geschrieben werden. Eine Datei für das analoge Routen (Abb. 6), eine Datei für das Block-Routen und eine Datei für das Routen der Top-Zelle. In der Prozess-Datei stehen alle wichtige Vorgaben für den Router. Hier wird definiert, welche Layer für das Routen verwendet werden oder wie groß der Abstand zwischen den Metall-Lagen sein muss. In der Prozess-Datei für das analoge Routen werden außerdem die Device-Generatoren definiert. Dadurch ist es möglich, sich in der IC-Station ein analoges Device automatisch aus dem Schaltplan-Symbol erzeugen zu lassen. Ein Transistor-Symbol verfügt über die Properties Gatelänge und Gatebreite (l_ic und w_ic), alle

anderen Spezifikationen für den Entwurf des Transistors sind in der Prozess-Datei enthalten. Diese Spezifikationen können aus den Design-Rules abgeleitet werden. Geschrieben wurden Device-Generatoren für PMOS,NMOS, Kapazität, Widerstand, Stacked-Vias und Guard-Band.

```
////////////////////////////////////////////////
// File:   analog_process.puf
//gekürzte Ausschnitte
////////////////////////////////////////////////
extern ic_process@@$process_name = "MTC45000";
extern ic_process@@$layer_spacing = [
["POLYSILICON", "POLYSILICON",      0.55],
["POLYSILICON", "polysilicon.blkg",      0.55],
["POLYSILICON", "ACTIVE_AREA",      0.3],      ];
extern ic_process@@$port_height = 0.7;
extern ic_process@@$routing_level = [
["POLYSILICON", @vertical, 0.5, "POLYSILICON",
"polysilicon.blkg"],
["METAL1", @horizontal, 0.5, "METAL1", "metal1.blkg"],
["METAL2", @vertical , 0.7, "METAL2", "metal2.blkg"],];
```

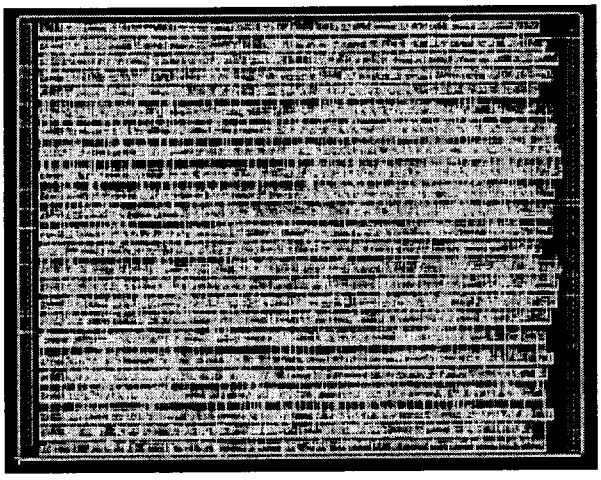
```
////////////////////////////////////////////////
//PMOS  Device Definitions, Device-Generator
//gekürzte Ausschnitte
////////////////////////////////////////////////
extern ic_process@@$dev_layer_mos_mp = [
["contact_layer",     "CONTACT"]
,["gate_layer",       "POLYSILICON"]          ];
extern ic_process@@$dev_rule_mos_mp = [
["contact_size",      [0.4,0.4]],
["gate_to_gate",      0.55]                   ];
extern ic_process@@$dev_info_mos_mp = [
["mos_flip",          @noflip]
["mos_rotate",        0]                      ];
```

Abbildung 6: Ausschnitt aus dem AMI 0.35 Prozess für analoges Routen

Verifikation des Design-Kits anhand der Enwicklung des Lottozahlengenerator-Chips

Die durchgeführten Arbeiten wurden am Entwurf eines Lottozahlengenerator-Chips verifiziert. Die Funktionsweise des Chips orientiert sich am Lotterie-Spiel 6 aus 49, wie es im deutschen Fernsehen zu sehen ist.

Bei Betätigung eines Tasters wird durch das Vorhandensein eines Zufallsgenerators eine Zahl ausgewählt, die durch das Aufleuchten einer LED angezeigt wird. Sind sieben Zahlen ausgewählt (sechs + Zufallszahl) so ertönt eine Melodie. Die 32kHz Frequenz für den Digitalteil (Abb. 7) wird durch den analogen Entwurf eines 32kHz Oscillators (Abb. 8) realisiert. Durch diesen Mixed-Signal-Chip (Abb. 9), der gefertigt wird, können alle geleisteten Arbeiten verifiziert werden.



Gesamtgrösse:
623 µm/493 µm

Stanardzellenentwurf

Zellenanzahl: 1134

Abbildung 7: Digitalteil des Lottozahlengenerator-Chip mit Stanardzellenentwurf der AMI 0.35µm Technologie.

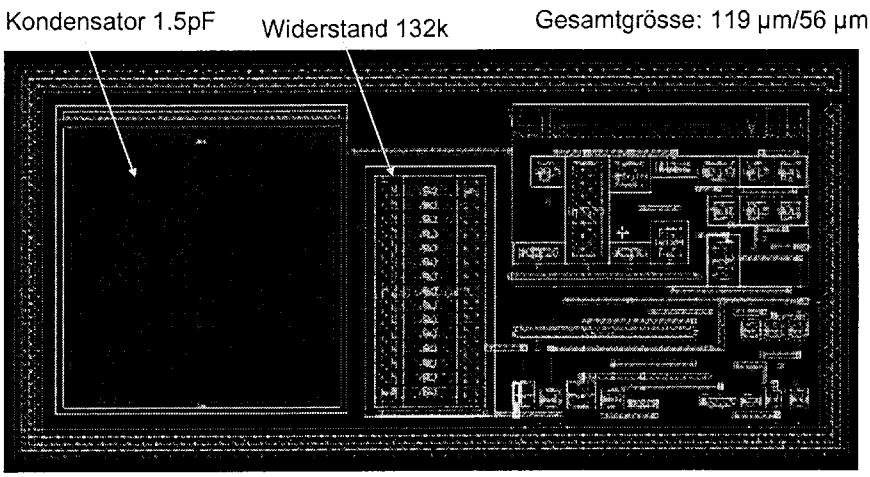


Abbildung 8: Analogteil des Lottozahlengenerator-Chip, 32kHz Oszillator.

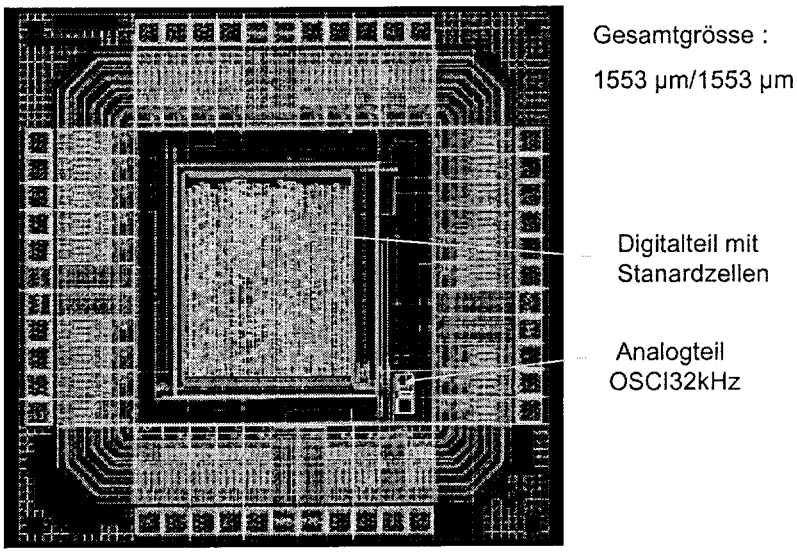


Abbildung 9: Der Lottozahlengenerator-Chip mit Digitalteil, 32kHz Oszillator und für die IC-Station aufbereiteten Padzellen.

8. Zusammenfassung

An der Fachhochschule Offenburg wird der Design-Kit **FHO_MTC_CMOS_035_v1.0** erstellt. Mit Hilfe dieses Kits lassen sich Designs in der AMI 0.35µm Technologie entwerfen. Alle durchgeführten Arbeiten werden durch den Entwurf eines Lottozahlengenerator-Chips verifiziert, der gefertigt wird. Damit sind alle wesentlichen Schritte bekannt, die für die Aufbereitung eines Design-Kits für beliebige Technologien für die Mentor-Tools erforderlich sind. Der Design-Kit wird für alle MPC-Mitglieder freigegen, die eine NDA für AMI bei Europractice unterzeichnet haben.

9. Referenzen

[1] AMI : C035M-A Design-Rule Manual, Document DS13337

[2] AMI : C035M-D Design Rule Manual, Document DS13330

[3] IC-Flow Bookcase

Redesign eines USB- Controllers in VHDL und Emulation mit einem Mikroprozessorkern auf FPGA

Artur Kurz, Prof. Dr. Dirk Jansen
FH-Offenburg, Badstrasse 24, 77652 Offenburg

Das Ziel dieses Projektes war, eine USB-Schnittstelle für das Mikrocontrollersystem FHOP zu realisieren. Der FHOP wurde an der Fachhochschule Offenburg entwickelt und wird in verschiedenen Anwendungen eingesetzt. Das USB-Modul ist in der VHDL-Sprache zu programmieren. Später soll es zusammen mit dem Mikroprozessor FHOP synthetisiert und auf einem FPGA-Board getestet werden. Für den Funktionsnachweis soll zusätzlich die Firmware des Mikrocontrollers um mehrere USB-Routinen erweitert sowie eine Applikation auf der PC-Seite in Delphi erstellt werden.

1. Einleitung

Vor etwa zehn Jahren wurde im ASIC Design Center an der Fachhochschule Offenburg der Mikroprozessor FHOP (First Homemade Operational Prozessor) entwickelt. Seitdem wurde der Prozessor in mehreren Studien- und Diplomarbeiten zu einem umfangreichen Mikrocontrollersystem weiterentwickelt, das in Anwendungen verschiedenster Art eingesetzt wird. Im aktuellen Stand besteht das System aus folgenden Komponenten: Serielle und parallele Schnittstellen, Chipselect-Einheit, Interrupt-Controller, Watchdog, Timer, Acoustic Human Interface. Für die Kommunikation mit der Außenwelt sind neben der seriellen RS232-Schnittstelle auch andere Schnittstellen realisiert worden. Dazu zählt PSK (Phase Shift Keying) sowie RFID (Radio Frequency Identification), wobei sich RFID noch in der Entwicklungsphase befindet.

Für die Kommunikation mit dem PC wird standardmäßig die serielle RS232-Schnittstelle benutzt. Sie ist nach mehreren Jahrzehnten gut dokumentiert und sehr einfach zu bedienen. Die RS232-Schnittstelle hat aber auch ihre Schwächen: Zum einen kann man ihr nur umständlich und sehr begrenzt Energie entnehmen, um die Applikationsschaltung damit zu versorgen, und zum anderen muss das Gerät vor dem Booten des PCs angeschlossen sein, also ist es keine Plug & Play-Anwendung realisierbar.

Diese Nachteile kennt eine USB-Schnittstelle nicht. Darüber hinaus hat Universal Serial Bus noch viele weitere Vorteile wie z. B. leichte Anwendbarkeit, schnelle und zuverlässige Datenübertragung, Flexibilität, geringe Kosten und niedriger Stromverbrauch.

Aus diesem Grund hat man sich entschieden, für den FHOP ein USB-Modul zu entwickeln, das später in das System integriert werden kann. In der Abbildung 1-1 ist das Blockbild vom FHOP mit der zugehörigen Peripherie dargestellt, so wie es später im ASIC realisiert werden kann.

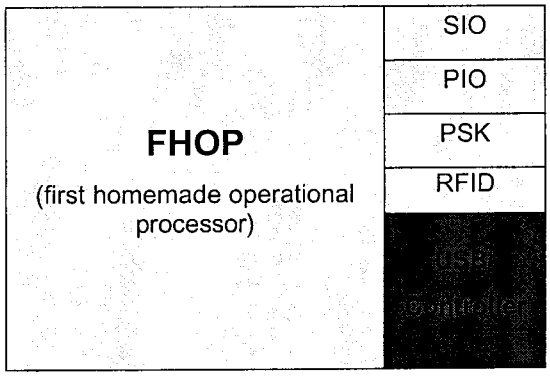


Abbildung 1-1: Blockbild für die Realisierung im ASIC

2. USB Grundlagen

Der USB ist ein serielles Bus-System (Universal Serial Bus), an das mehrere Geräte angeschlossen und adressiert werden können. USB ist wesentlich komplexer als RS232 und mit bis zu 1,5 MBit pro Sekunde (Low Speed - Geräte) oder 12 MBit pro Sekunde (Full Speed - Geräte) auch viel schneller. Dazu kommt, dass man bei der Entwicklung des USB großen Wert auf Einfachheit und Sicherheit der Bedienung gelegt hat. Erst mit dem USB hat Plug & Play seine eigentliche Bedeutung eingeholt. Denn ein USB-Gerät wird im Normalfall einfach nur

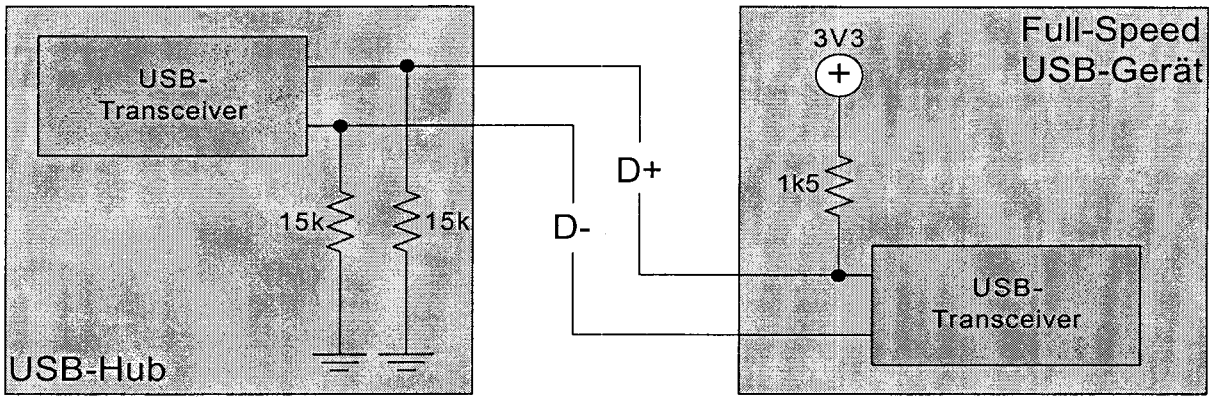


Abbildung 2-0: Gerätetyp Erkennung (Full-Speed)

bei abgeschaltetem oder laufendem PC angeschlossen und initialisiert sich dann automatisch. Das lästige Problem mit doppelt belegten Interrupts, falschen Adressen und fehlenden Treibern ist damit gelöst. Die Vereinfachung für den Anwender bedeutet aber auch einen wesentlich größeren Aufwand für den Entwickler.

USB Geräte werden mittels eines vieradrigen Kabels miteinander verbunden. Das Kabel hat folgende Belegung:

PIN #	1	2	3	4
Bezeichn.	+5V	Data –	Data +	GND

Tabelle 2-1

Der Anschluss stellt eine Betriebsspannung von 5 V zur Verfügung, die ohne weiteres bis zu 100 mA belastet werden kann. Bei einer besonderen Konfiguration kann das Gerät bis zu 500 mA entnehmen. Die Signale auf den beiden Leitungen D+ und D- sind Differenzsignale mit Spannungspegeln 0 V und 3,3 V.

Der USB ist ein Bus mit nur einem Master, alle Aktivitäten gehen vom PC aus. Daten werden in Paketen von 8 bis zu 256 Bytes versandt und empfangen. Der PC kann Daten von einem Gerät anfordern. Umgekehrt kann kein Gerät von sich aus Daten absenden. Der gesamte Datenverkehr hat einen Rahmen von exakt einer Millisekunde. Innerhalb eines Rahmens können nacheinander Datenpakete für mehrere Geräte verarbeitet werden. Dabei können Low-Speed- und Full-Speed- Pakete zusammen vorkommen. Wenn mehrere Geräte angesprochen werden, sorgt ein Bus-Verteiler (Hub) für die Verteilung.

Low-Speed-Geräte arbeiten mit einer Datenrate von 1,5 Mb/s, ein Bit ist also 666,7 ns lang. Full-Speed-Verbindungen verwenden 12 Mb/s entsprechend 83,33 ns. Die Geschwindigkeit wird allein vom Master vorgegeben. Die Slaves müssen sich auf den Datenstrom synchronisieren. Da kein getrenntes Taktsignal übertragen wird, muss der Takt aus dem Datensignal gewonnen werden. Man verwendet dazu das NRZI-Verfahren (Non Return to Zero Inverted). Daten-Nullen führen hierbei zu einem Pegelwechsel, Einsen lassen den Pegel unverändert. Zusätzlich wird beim USB Bit-stuffing benutzt. Bei diesem Verfahren wird nach sechs aufeinander folgenden Einsen eine Null eingefügt. Damit wird maximale Zeitdauer ohne einen Signalwechsel auf sechs Takte begrenzt.

Bei einem nicht verwendeten USB-Anschluss sind beide Signalleitungen low und haben einen Innenwiderstand von 15 kO. Jedes USB-Peripheriegerät besitzt einen Widerstand von 1,5 kO, der eine der beiden Signalleitungen mit +3,3 V verbindet. Bei einem Full-Speed-Gerät wird D+ mit 3,3 V verbunden (Abbildung 2-0), bei einem Low-Speed-Gerät D-. Der PC-Host erkennt daran den Typ des Geräts und kann eine Datenverbindung mit der entsprechenden Übertragungsrate aufbauen. Bei dem Enummerieren[1] eines Geräts wird automatisch der passende Treiber geladen. Umgekehrt erkennt das Betriebssystem auch, wenn ein Gerät vom Bus getrennt wird. In diesem Fall wird der Treiber wieder aus dem Speicher entfernt.

[1] Enumeration: Konfigurationsvorgang nach der Herstellung der Verbindung

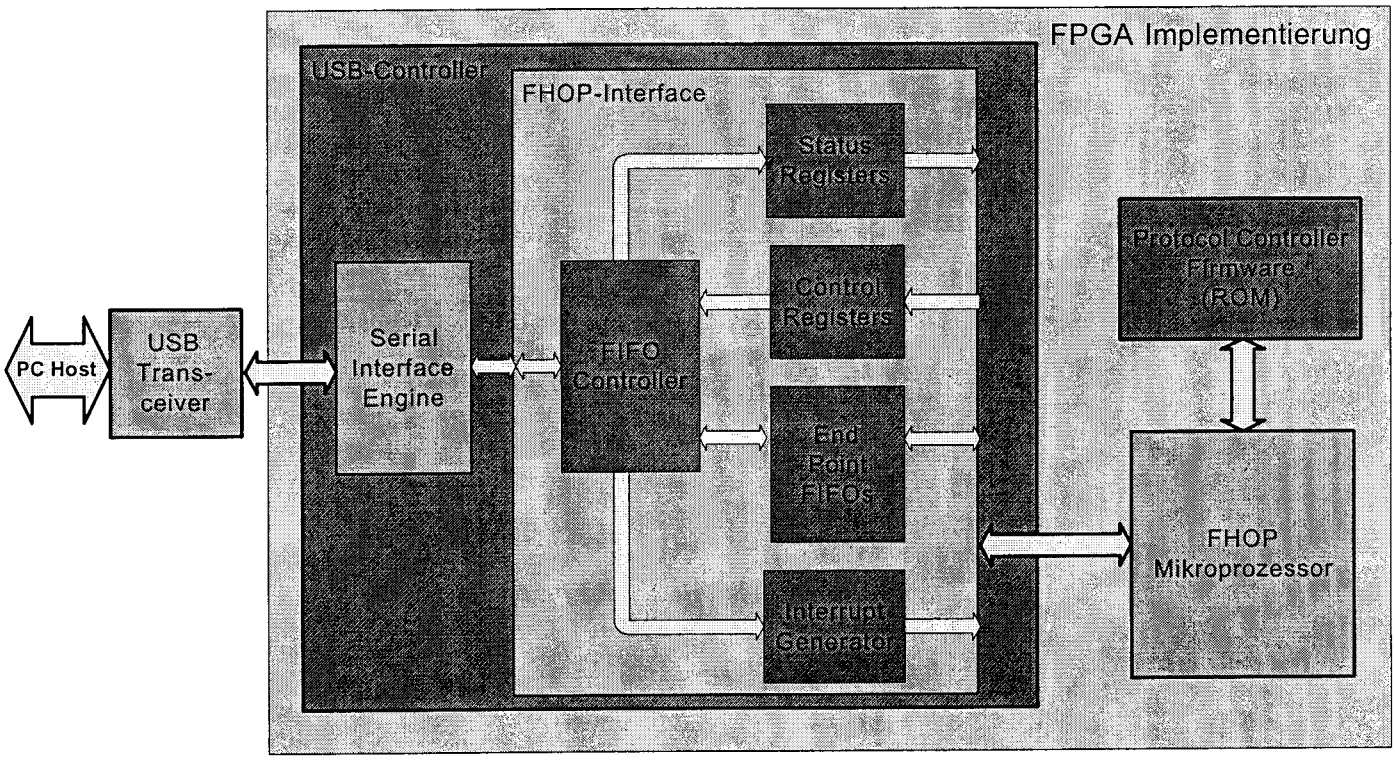


Abbildung 3-0: System-Übersicht

3. Realisierung des USB-Moduls

Eine Vorarbeit für dieses Projekt wurde von Dipl.-Ing. Gopal Pudasaini im Rahmen einer Master Thesis geleistet. Das Arbeitsergebnis war der VHDL-Code für den USB-Controller. Der Code besteht aus zwei Hauptblöcken:

- Serial Interface Engine
- FHOP-Interface.

Beide Teile wurden einzeln erstellt und simuliert. Die Simulation des gesamten Systems zusammen mit dem Mikroprozessor wurde damals aus Zeitgründen nicht mehr durchgeführt. Deshalb musste zuerst dieser Schritt ausgeführt werden. Die ersten Simulationsergebnisse des gesamten Systems haben gezeigt, dass die FHOP Interface Unit komplett überarbeitet werden musste. Serial Interface Engine war fehlerfrei und wurde nicht mehr geändert.

In der Abbildung 3-0 ist das gesamte System in einem Blockschaltbild dargestellt. Die grün dargestellten Funktionsblöcke (USB-Transceiver, SIE und FHOP Mikroprozessor) waren fertige Komponenten und mussten nicht mehr modifiziert werden.

Über ein USB-Kabel ist der PC mit dem USB-Transceiver verbunden, der die elektrischen Anforderungen auf dem Bus erledigt. Der Transceiver ist mit einem externen Baustein MAX3340E realisiert. Alle anderen Komponenten sind in VHDL-Sprache formuliert worden und können somit alle zusammen nach der Synthese in einem FPGA-Chip emuliert werden.

3.1. Serial Interface Engine (SIE)

SIE ist einer der wichtigsten Funktionsblöcke in diesem Design. Dieser wurde freundlicherweise von der Firma Intel im VHDL-Format zur Verfügung gestellt. Die SIE erledigt alle paketbezogenen Aufgaben. Im Folgenden sind die wichtigsten Aufgaben der Serial Interface Engine aufgelistet:

- Paket Erkennung,
- Detektierung und Generierung von Start of Paket (SOP), End of Paket (EOP), und Reset Signalen,
- Synchronisierung des Takts,
- NRZI-Daten Codierung/Decodierung,
- CRC-Generierung/Prüfung,
- Paket-ID (PID) Generierung/Prüfung,

- Umwandlung seriell in parallel und umgekehrt.

Die Funktion der SIE ist abhängig von der Datenrichtung. Es werden serielle Daten vom Bus empfangen, decodiert und mittels CRC auf die Gültigkeit überprüft. Die gültigen Daten werden byteweise an die FHOP Interface Unit weitergeleitet. In umgekehrter Richtung muss SIE vom FHOP empfangene Bytes in Pakete packen, CRC16 bzw. CRC5 generieren und als seriellen Bitstrom im USB-Format (NRZI) auf dem Bus senden.

3.2. *FHOP Interface Unit*

Sowohl die SIE als auch der FHOP Prozessor besitzen einen 8 Bit breiten Bus. Die FHOP Interface Unit wird als Systemschnittstelle zwischen den beiden Komponenten eingesetzt und muss damit die Übertragungsspezifikation der beiden Seiten unterstützen. Die FHOP Interface Unit besteht intern aus folgenden Funktionsblöcken:

- Address Decoder,
- Status Register,
- Control Register,
- USB Address Register,
- FIFO-Controller,
- FIFOs (4x),
- Interrupt Generator.

Der Address Decoder hat als Eingang eine Chip-Select Leitung (CS_N) und vier Adressleitungen (A0 bis A3), die vom FHOP Prozessor kommen. Er hat die Aufgabe, die Adresse verschiedener Register und FIFOs eindeutig zu decodieren. Über die Status Register werden die Statussignale vom USB-Modul an den Prozessor weitergeleitet.

Die Control Register werden zur Steuerung des USB-Controllers benutzt. Hier kann der FHOP festlegen, wie sich das USB-Modul verhalten soll, wenn das nächste USB-Paket empfangen wird.

Im USB-Address Register wird die Geräteadresse gespeichert. Diese wird jedem Gerät nach dem Anschließen an den Bus vom PC zugewiesen.

Zur Zwischenspeicherung der Daten werden vier FIFOs benutzt. Zwei FIFOs sind für Control-Transfer zuständig und die anderen zwei für den Datentransfer. Sobald das Modul ein USB-Paket erfolgreich empfängt und ins FIFO speichert, wird vom Interrupt Generator ein Interrupt ausgelöst. Dieser signalisiert dem Prozessor, dass die Daten zur Weiterverarbeitung bereit stehen.

Alle diese Register, FIFOs und Interrupt Generator werden vom FIFO-Controller gesteuert. Dieser Block ist als eine Statemachine realisiert worden. Jedes Paket auf dem Bus wird von der SIE empfangen und an den FIFO-Controller weitergeleitet. Der FIFO-Controller hat die Aufgabe, zu überprüfen, ob das Paket für das Gerät bestimmt ist. Das geschieht, indem er die Adresse im Token Paket mit der Adresse im USB Address Register vergleicht. Stimmen beide Adressen überein, werden die Daten in das betreffende FIFO gespeichert. Anderenfalls wird das aktuelle Paket ignoriert. Hiermit wird der Mikroprozessor entlastet, denn er braucht nicht mehr zu überprüfen, ob das Paket gültig und für ihn bestimmt ist.

3.3. *Protocol Controller*

Der Protocol Controller ist in der Software realisiert worden und muss alle protokollspezifischen Aufgaben erledigen. Die Software ist in Assembler geschrieben und besteht aus drei Interrupt Routinen und USB-Descriptor Daten. Je nach Art des empfangenen Pakets (Control, Daten IN / OUT) löst die FHOP Interface Unit den entsprechenden Hardware Interrupt aus. Die Endpoint-0 Interrupt Routine ist für den Control Transfer zuständig und erledigt alle im USB spezifizierten Standard Requests. Sie ist zuständig für das richtige Enummerieren des Geräts beim Anschließen an den Bus.

Die beiden anderen Routinen steuern den Daten Transfer (Endpoint-1 IN- und Endpoint-2 OUT-Daten). Damit das Gerät richtig konfiguriert werden kann, fragt das Betriebssystem von der neu angeschlossenen Hardware bestimmte Informationen in Form von Descriptoren ab. Dabei handelt es sich um genau definierte Datenblöcke von wenigen Bytes, die im ROM abgelegt sind.

Für das vollständige Beschreiben des Geräts gibt es fünf Arten von Descriptoren:

- Device-,
- Configuration-,
- Interface-,
- Endpoint- und
- String- Descriptor.

Die Control Requests vom Host werden analysiert, um die Art der Anfrage zu erkennen. Der Mikroprozessor liest die entsprechenden Daten aus dem ROM und schreibt sie ins Ausgabe-FIFO. Von diesem werden sie an den Host zurückgesandt.

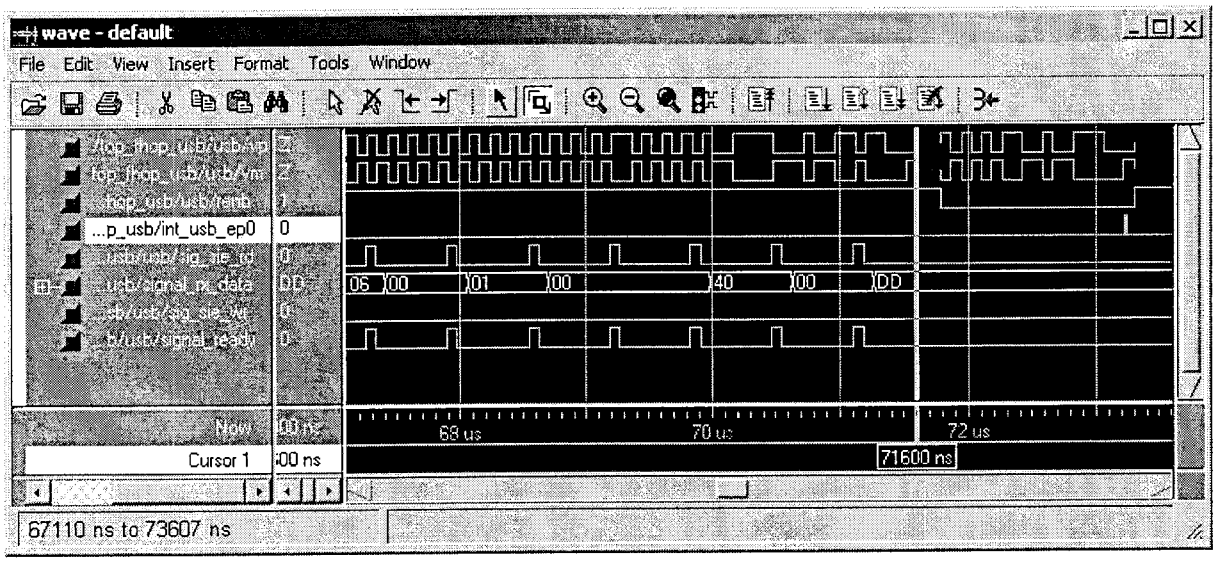


Abbildung 4-0: Ausschnitt aus der Simulation

4. Simulation und Synthese

Zum Simulieren des VHDL-Programms wurde die Software „ModelSim SE 5.7c" der Firma Model Technology Incorporated benutzt. Im ersten Schritt der Simulation wurden die Funktionsblöcke einzeln auf die korrekte Funktion getestet. Dabei wurden das Auslesen und Beschreiben der Register bzw. FIFOs simuliert. An dieser Stelle war es sehr wichtig, die Spezifikation des Zugriffs auf den FHOP-Datenbus streng einzuhalten. Denn die Fehler beim Buszugriff würden fatale Folgen verursachen.

Im nächsten Schritt wurde das komplette USB-Modul simuliert. Dazu wurden die USB-Pakete, so wie sie auf dem Bus benutzt werden, konstruiert. Die daraus entstandene serielle Bitfolge muss noch nach der USB-Spezifikation in NRZI codiert werden. Mittels einer Makro-Datei wird das Signal dann am Eingang des USB-Controllers simuliert. Hiermit kann man diverse Varianten des Datentransfers durchspielen und die Reaktion des Moduls am Ausgang überprüfen. Damit wurde unter anderem getestet, ob die Daten richtig decodiert werden, in das betreffende Endpunkt-FIFO geschrieben werden und der Interrupt an den Mikroprozessor ausgelöst wird. Die Abbildung 4-0 zeigt einen kleinen Ausschnitt aus der Simulation. Man sieht hier zunächst die beiden differenziellen Signale „vp" und „vm", die vom USB-Kabel kommen. Das Signal „signal_rx_data" ist der Ausgang von Serial Interface Engine (ein 8-Bit breiter Bus). Hier sieht man die Datenbytes in der Form, wie sie in das FIFO gespeichert werden. Zusätzlich ist hier dargestellt, wie das empfangene Paket (vor der Zeitmarkierung) durch ein ACK-Paket bestätigt wird. Nachdem das Paket erfolgreich übertragen wurde, wird ein Interrupt auf der „int_usb_ep0" ausgelöst.

Zuletzt wurde der Mikroprozessor FHOP mit der zugehörigen Peripherie in die Simulation eingebunden. Um diese Simulation zu ermöglichen, wurden auch die USB-Treiber-Routinen für den FHOP geschrieben. Die Treiber bestehen aus drei Interrupt-Routinen und übernehmen die Weiterverarbeitung der Daten auf der höheren Protokollschicht. Der kompilierte Assembler Code wird in das für Simulationsprogramme verständliche Format (.mif -Datei) konvertiert und in die Simulation eingebunden. Damit wurde das komplette Design getestet so wie es später auf FPGA emuliert wird.

Nach der erfolgreichen Simulation wurde der VHDL-Code mit dem Programm „Leonardo Spectrum" synthetisiert. Als Zieltechnologie wurde hier der FPGA-Chip der Firma Altera EP20K1500EBC652-3 eingesetzt. Mit 1,5 Mio. Gatter ist der Chip einer der größten dieser Art. Der Chip wurde zusammen mit einem Entwicklungsboard der Firma El Camino erworben. Das Board hat die Größe eines DIN A4 Blatts und beinhaltet alle notwendigen Komponenten für eine komfortable Entwicklung. In der nachfolgenden Tabelle 4-1 sind die Ergebnisse der Synthese zusammengestellt.

Device Utilization for EP20K1500EBC652			
Resource	Used	Avail	Utilization

IOs	30	488	6.15%
LCs	4220	51840	8.14%
Memory Bits	12800	442368	2.89%

Tabelle 4-1: Synthese Ergebnis

1,5 Mio. Gatter entsprechen ca. 50000 Altera spezifischen Logik Zellen. Davon sind 8,14% für dieses Design genutzt worden.

5. Funktionsnachweis

Beim Anschließen eines USB-Geräts an den Bus startet der Enummerierungsvorgang, bei dem der PC notwendige Daten von der neu angeschlossenen Hardware anfordert. Nach der erfolgreichen Enummerierung wird das Gerät im Geräte-Manager eingetragen. Hiermit kann der erste Funktionsnachweis durchgeführt werden. Abbildung 5-1 zeigt den Eintrag des FHOP-Controllers im Geräte-Manager nachdem er erfolgreich konfiguriert wurde. Hier sieht man auch, dass das Gerät als USB-HID (Human Interface Device) konfiguriert wurde. Das ermöglicht die Benutzung von Standard Windows Treibern bei der Applikationsprogrammierung.

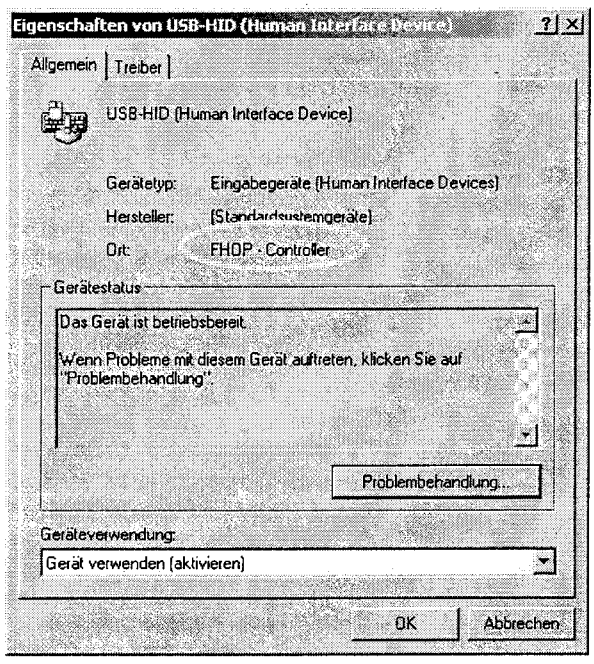


Abbildung 5-1: Eintrag im Geräte-Manager

Für den vollständigen Funktionsnachweis wurde auch eine Applikation programmiert. Die Applikation wurde in Delphi erstellt und realisiert eine Datenübertragung in beide Richtungen (PC → Gerät und Gerät → PC). Mittels 8 Schaltern, die an den parallelen Port des FHOPs angeschlossen sind, wird ein Byte definiert. Das Byte wird in periodischen Abständen (10 ms) über die USB Datenpipe an den PC übertragen und auf dem Monitor visuell dargestellt. Ähnlich funktioniert es auch in die andere Richtung. Mittels 8 Buttons in der Applikation kann man die an den FHOP angeschlossenen LEDs steuern.

Damit man die HID-Kommunikation in Delphi realisieren kann, muss ein Package zusätzlich installiert werden. Das Package beinhaltet „Interface for Microsoft Human Input Device (HID) API" und wurde von Robert Marquardt im Rahmen des Jedi Projekts erstellt. Das Package ist als Freeware auf der Homepage http://www.delphi-jedi.org verfügbar.

6. Zusammenfassung

In dieser Arbeit wurde eine USB-Schnittstelle für ein bestehendes Mikrocontroller System FHOP realisiert. Im aktuellen Stand funktioniert das Design zuverlässig in Low Speed Konfiguration. Im Full Speed gibt es noch einige Schwierigkeiten, denn die Kommunikation bricht nach einigen Paket-Transfers zusammen. Durch das Emulieren des Designs auf FPGA wurde die Funktion nachgewiesen. Die nächste Aufgabe wird sein, die Hardware zu optimieren, damit das USB-Modul auch im Full Speed zuverlässig funktioniert. Zusätzlich wird die Software auf der PC Seite optimiert, um höhere Übertragungsraten zu erzielen.

7. Literatur

[1] John Hyde, „USB Design by Example", A Practical Guide to Building I/O Device

[2] Jan Axelson, „USB Complete", Everything You Need to Develop Custom USB Peripherals

[3] Universal Serial Bus Specification, Revision 1.1 http://www.usb.org/developers/docs/

Datenbankbasierte Modellverwaltung

Prof. Dr. Johann Siegl
Georg-Simon-Ohm-Fachhochschule Nürnberg

Martin Bachhuber
Mentor Graphics, Nürnberg

Zur Verifikation von digitalen und gemischt analog/digitalen Systemen benötigen Simulationswerkzeuge unterschiedliche Modellbeschreibungen von der Systemebene bis herunter auf die physikalische Beschreibungsebene. Der Designer von Elektroniksystemen soll auf bemusterte Modelle zur Beschreibung von Systemmodulen (u.a. VHDL-, VHDL-AMS-, Spice- und IBIS-Modelle) über eine Wissensbasis schnell und netzwerkweit zugreifen können. Ein Hauptziel ist die Wiederverwendbarkeit von Designbeschreibungen (Design Reuse). Modellgeneratoren erzeugen anwendungsspezifische Modelle für bestimmte Funktionsmodule, ausgehend von bekannten Eigenschaften und Vorgaben. Eine objektorientierte, metadatenbasierte Modellverwaltung unter Zugrundelegung eines durchgängigen Datenmodells stellt die Konsistenz der bemusterten Modelle bzw. Modellgeneratoren sicher, mit Referenzen zu weiteren Designviews (u.a. Symbole und physikalische Views). Der Design-Flow soll durch die Bereitstellung von korrekten, erprobten Modellbeschreibungen und Designviews in allen Designphasen über geeignete Schnittstellen unterstützt werden.

1. Motivation für eine datenbankbasierte Modellverwaltung

Mit verbesserten Methoden gilt es, die Qualität eines Designs zu steigern und die dafür erforderliche Entwicklungszeit zu verkürzen. Der Entwickler benötigt eine geeignete Infrastruktur, die es ihm ermöglicht auf bereits verifizierte und bewährte Designkomponenten zurückzugreifen. Neben der Zeit- und Kostenersparnis bei der Entwicklung kann so durch „Design Reuse" auch die Qualität eines Produktes beträchtlich gesteigert werden. Es ist aber nicht genug, gebrauchsfertige Komponenten einfach zu speichern. Vielmehr müssen diese einem Standard entsprechen und hinreichend charakterisiert werden.

Eine datenbankbasierte Modellverwaltung ermöglicht dabei ein schnelles Auffinden verfügbarer Modell- und Designbeschreibungen und fördert damit die Wiederverwendbarkeit. Des Weiteren können alle Referenzen von Objekten durch ein Datenmodell abgebildet werden, die Vollständigkeit und Konsistenz erfasster und freigegebener Modellbeschreibungen wird sichergestellt, das Wissen über Modelle inklusive deren Testbenches wird in einer Wissensbasis zusammengefasst und eine netzwerkweite Zugriffskontrolle für autorisierte Mitglieder einer Design Community wird ermöglicht.

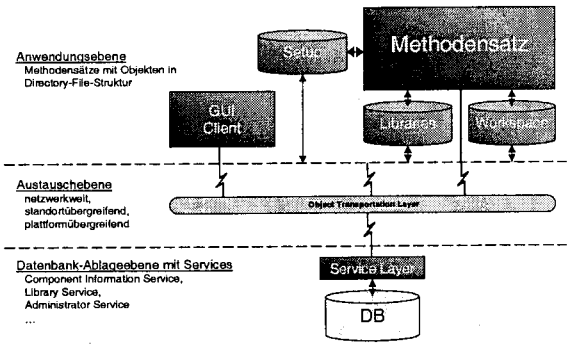


Abb. 1: Umgebung für Electronic System Design Automation

Abb. 1 zeigt prinzipiell Methodensätze z.B. für Designdefinition, Designverifikation oder für „Physical Design" mit ihrer Umgebung. Ein Methodensatz erstellt u.a. Designbeschreibungen und legt sie im Projekt-Workspace ab. Bei der Erstellung werden Library-Elemente (z.B. Part-Libs, Symbol-Libs, Footprint-Libs) benötigt, bei der Designverifikation darüber hinaus Model-Libs. Im Setup befinden sich Voreinstellungen für einen Methodensatz. Die Umgebung eines Methodensatzes (Setup, Libraries, Workspaces) ist in der Regel Directory/File-basiert organisiert. Es gilt diese Umgebung eines Methodensatzes durch Erweiterung der Infrastruktur um eine datenbankbasierte, netzwerkweite Verwaltung von Design-Objekten zu ergänzen. In der Anwendungsebene verbleibt, wie herkömmlich der Methodensatz mit all seinen benötigten Daten und Informationen, die in den Datenspeichern Setup, Libraries und Workspace zusammengefasst sind.

Grundsätzlich sollen alle von mehreren Entwicklern und verschiedenen Arbeitsgruppen zu verwenden-

den Design-Objekte, Designbeschreibungen und Designvoreinstellungen in einer Datenbank abgelegt werden. Den Zugriff und die Bereitstellung von dort abgelegten Objekten ermöglicht ein „Object Transportation Layer".

Warum es zweckmäßig ist, das Setup einer Design-Methode in die Datenbank aufzunehmen, ist erklärungsbedürftig. Schließlich handelt es sich hierbei um ein Objekt, das normalerweise für eine Arbeitsgruppe oder sogar nur für einen einzigen Arbeitsplatz individuell angelegt wird. Die darin gespeicherten Informationen, wie z.B. Pfadangaben zu bestimmten Libraries usw. lassen sich nicht ohne weiteres auf andere Arbeitsgruppen übertragen. Bei der Wiederverwendung eines Designs bietet das ursprüngliche Setup jedoch einen wertvollen Ausgangspunkt, da es die Einstellungen der Designumgebung dokumentiert und damit reproduzierbar macht. Es wird dadurch ermöglicht, das bestehende Setup auf eine andere Arbeitsgruppe anzupassen oder neu zu schreiben, wobei die Informationen über die notwendigen Bibliotheken und Grundeinstellungen, die sich in allen Methodensätzen wiederfinden, aus dem Originalsetup übernommen werden. Nicht zuletzt ist es möglich, das Setup mit Hilfe der Datenbank automatisch generieren zu lassen. Die Datenbank kann Informationen über benötigte Technologien, Libraries usw. speichern und daraus zusammen mit den spezifischen Informationen des jeweiligen Arbeitsplatzes ein anwendungsspezifisches Setup erzeugen.

Die Aufnahme der Libraries in die Datenbank bietet den Vorteil, dass diese dadurch netzwerkweit und plattformunabhängig für alle autorisierten Anwender einer Design Community zugänglich sind. Die Libraries können somit zentral verwaltet werden, wodurch die Konsistenz und Korrektheit sichergestellt wird. Neue Libraries oder Library-Elemente werden so erst dann in das System aufgenommen und damit allen Benutzern zugänglich gemacht, wenn sie die nötigen Voraussetzungen dafür erfüllen.

Der Workspace umfasst alle Objekte, die von einem Methodensatz erzeugt bzw. analysiert werden. Die Ergebnisse eines jeweiligen Designschritts dienen u.a. als Datenquelle für die im Designflow (siehe Abb. 2) nachfolgenden Methodensätze (Down-Stream Tools). Hierbei handelt es sich vorrangig um Viewpoints und Netzlisten aber auch extrahierte Timing-Informationen für die Backannotation.

Von besonderer Bedeutung ist die Charakterisierung von Designobjekten und Designkomponenten mit aussagekräftigen Attributen (u.a. Einführung eines Attribut-Dictionary), welche das Wiederfinden in der Datenbank erleichtern. Referenzen eines Objekts zu anderen im Designflow verwendeten Objekten ermöglichen eine bezüglich seiner Entwicklungsschritte durchgängige Abbildung eines Designs und bieten die Möglichkeit an verschiedenen Stellen des Designflows wieder einzusteigen und von dort aus mit z.B. einer anderen Technologie weiterzuarbeiten.

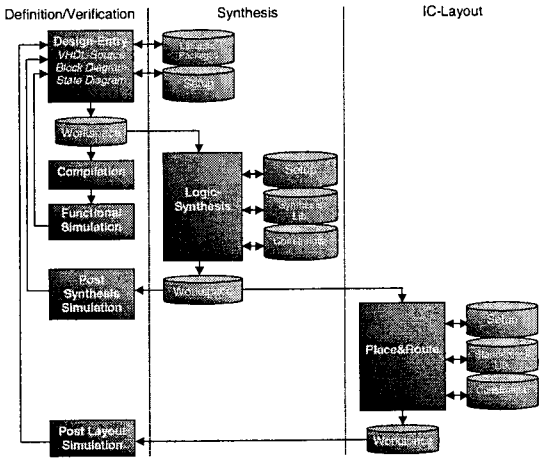


Abb. 2: Workflow für „High-level"-Design, Modellbeschreibung mit VHDL, Synthese und IC-Layouterstellung, Designverifikation

Die Austauschebene ist u.a. für die Bereitstellung erforderlicher Objekte und Libraries von der Datenbank-Ablageebene zur Anwendungsebene zuständig. Diese Schicht ermöglicht einen netzwerkweiten, standort- und plattformübergreifenden Austausch von Daten.

Der Zugriff auf die Datenbank wird in der untersten Ebene über Services gesteuert. Vorkonfigurierte Services (z.B. Component-Information-Service, Library-Service, Admin-Service) erleichtern das Suchen, die Aufnahme, die Bereitstellung und das Verwalten von Objekten.

2. Konzeptionelle Umsetzung

Nach der allgemeinen Einführung des in Abb. 1 skizzierten Konzepts geht es im besonderen um die Modellverwaltung. Ein VHDL-Modell besteht grundlegend aus den VHDL-Objekten Entity, Architecture und Configuration. Jedes Modell kann nur eine Entity haben, welche die Schnittstellen und die generischen Parameter beschreibt. In der Entity können u.a. Generic-Attribute, Funktionen bzw. passive Prozesse definiert werden, die für alle untergeordneten Architectures Gültigkeit haben. Einer Entity lassen sich mehrere Architectures zuordnen, die unterschiedliche Realisierungen, wie z.B. Struktur- oder Verhaltensbeschreibungen des Modells darstellen. Bei einer Strukturbeschreibung werden innerhalb der Architecture andere VHDL-Modelle durch „Component Declaration und Instantiation" eingesetzt und somit auf diese

verwiesen. Die Entity kann über Configurations mit ihren Architectures verbunden werden. Die Configuration verbindet zum einen eine Entity mit einer ihrer Architectures, zum anderen wird mit ihr die jeweilige Architecture konfiguriert und damit die Implementierungen der in ihr referenzierten Komponenten ausgewählt. In Packages und Libraries werden z.B. Funktionen und Konstanten gespeichert, die den Entities und Architectures mehrerer Designs zugänglich gemacht werden sollen. Die Vorgaben zur Umsetzung in ein Datenmodell zeigt Abb. 3 schematisch.

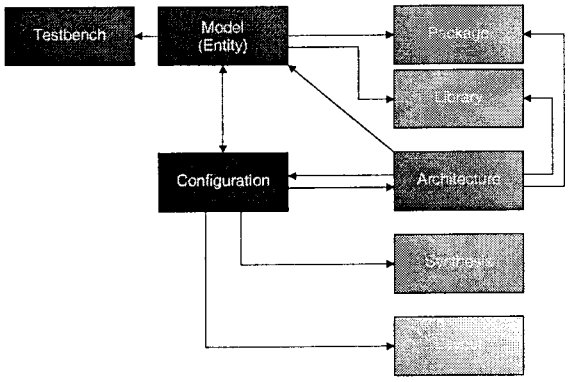


Abb. 3: Schematische Darstellung des VHDL-Datenmodells

Für jedes VHDL-Objekt wird eine eigene Objektklasse angelegt. Der Ausgangspunkt des Datenmodells ist die Klasse „Model". Sie stellt im VHDL-Modell die Entity dar und beinhaltet alle Charakteristiken, die eine Suche nach einem bestimmten Modell ermöglichen. Die Klasse „Model" hat eine bidirektionale Referenz auf die Klasse „Configuration". Im Sprachumfang von VHDL wird nur von der Configuration auf die Entity referenziert und nicht andersherum, jedoch ist dies in der Datenbank notwendig, da die Suche in der Klasse „Model" stattfindet und dann bekannt sein muss, welche Configurations für die gefundenen Entities verfügbar sind. Jede Configuration verweist auf eine Architecture, die zu der entsprechenden Entity gehört. Die Klasse Architecture referenziert gemäß VHDL-Vorschrift auf die Entity zu der sie gehört und auf die möglicherweise verwendeten Subkomponenten. Für letztere wird direkt auf die Klasse Configuration referenziert, da damit auch die jeweilige Implementierung der Subkomponenten festgelegt wird. Die Klassen Model und Architecture verweisen beide auf die Klassen Package und Library, um die entsprechenden Objekte in das Design einzubinden. Die Configuration verweist auch auf die Klassen „Synthesis" und „Layout", um die jeweiligen Synthese- und Layoutergebnisse eines Entwurfs zu referenzieren. Besonders hervorzuheben ist die Referenz auf „Testbench". Hierbei handelt es sich um die jeweilige Testbench zu einem Modell. Die Testbench ist aber nicht in einer eigenen Klasse gespeichert, vielmehr ist sie selbst ein Modell (eine Strukturbeschreibung von Stimuli und zu testendem Modell) und als solches in der Klasse „Model" gespeichert. VHDL-Source-Code, Synthese- und Layoutergebnisse werden in binärer Form als BLOB (Binary Large Object) gespeichert. In jeder Klasse können Informationen über Autor, Erstellungsdatum, Änderungsdatum usw. gespeichert werden. Des Weiteren kann eine ausführliche Dokumentation zu jedem Objekt in beliebiger Form (Text, Diagramm, Graphiken, Kurven...) abgelegt werden. Abb. 4 zeigt ausschnitthaft das Datenmodell.

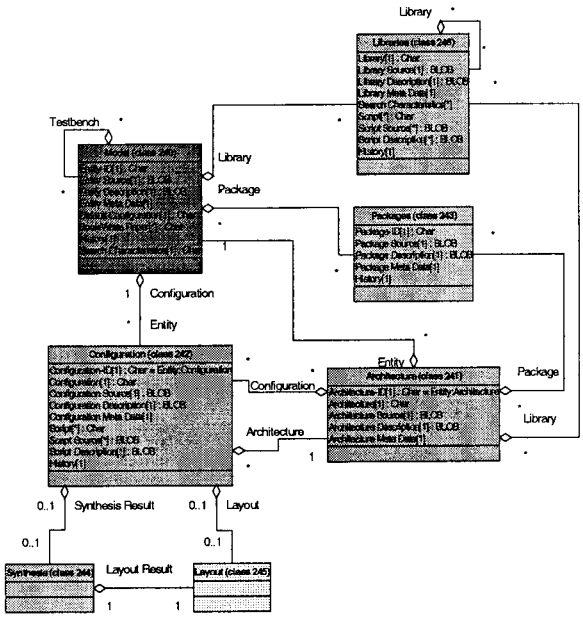


Abb. 4: Ausschnitt aus einem beispielhaften Datenmodell für VHDL-Modellverwaltung

3. Integration von Modellgeneratoren

Eine wesentliche Erweiterung bieten Modellgeneratoren. In Abb. 5 sind die grundlegenden Komponenten eines Modellgenerators dargestellt. Der Generator benötigt ein „Grundwissen" über das Modell, das er generieren soll. Dabei handelt es sich im einfachsten Fall um ein fertiges Modell mit Platzhaltern, die durch bestimmte Werte ersetzt werden. Es ist aber auch möglich komplexe Strukturen nach bestimmten Kriterien automatisch zu generieren, wenn eine entsprechende Regelmäßigkeit gegeben ist. Einfachere Strukturen lassen sich auch problemlos mit den VHDL Statements „IF...GENERATE" und „FOR...GENERATE" realisieren, jedoch stößt man bei komplexeren Modellen gelegentlich an Grenzen des Sprachumfangs von Standard VHDL. Bei der hier skizzierten Lösung wird die Modellierungssprache VHDL um die Programmiersprache Tcl erweitert, wodurch in VHDL jetzt auch der Funktionsumfang von Tcl zur Verfügung steht.

Nach Generierung eines anwendungsspezifischen Modells nach dem in Abb. 6 dargestellten Verfahren, bleibt das ursprüngliche Grundmodell in seiner allgemeinen Formulierung dem Anwender verborgen. Dies ist bei der Verwendung von den VHDL-Statements „IF ... GENERATE" und „FOR ... GENERATE" nicht möglich, da diese immer im VHDL-Code verbleiben. Die Ergebnisse des Modellgenerators eignen sich daher besser zur kommerziellen Nutzung, da die in der Entwicklung sehr aufwändigen allgemeinen Grundmodelle für den Anwender nicht sichtbar sind.

Für jedes Modell kann auch eine Testbench oder ein Stimuli-File generiert werden, um zu überprüfen, ob das Ergebnis die erwarteten Modelleigenschaften erfüllt.

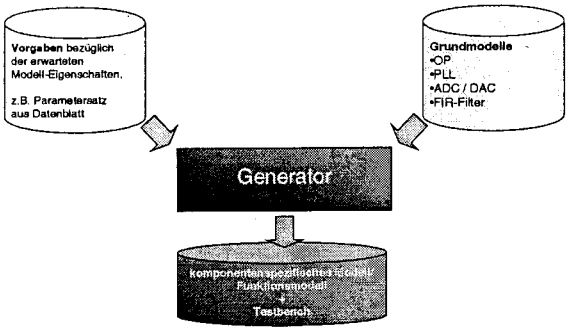
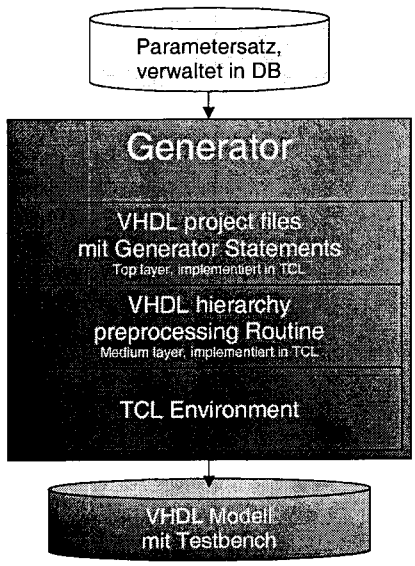


Abb. 5: Modellgenerator Grundstruktur

In Abb. 6 ist beispielhaft ein Modellgenerator skizziert. In der Datenbank werden die Parametersätze gespeichert, welche zur Erzeugung der Modelle notwendig sind. Der Generator selbst wird in drei Schichten unterteilt. Der Top Layer besteht aus den Basismodellen, also dem „Grundwissen" des Generators. Dabei handelt es sich um VHDL-Code, der um Anweisungen in der Sprache Tcl ergänzt wurde. Der Medium Layer arbeitet die Hierarchie des Modells ab und filtert den Tcl Programmcode aus dem Basismodell heraus. Die unterste Schicht ist das eingebettete Tcl-Environment, das sowohl die Preprozessor- als auch die Tcl-Statements in den Basismodellen auswertet. Das Ergebnis eines Generatordurchlaufs ist ein Modell inklusive Testbench bestehend aus reinen VHDL-Files, die keine Hinweise mehr auf die Implementierung des Basismodells liefern.



VHDL File mit Generator Statements

```
...
ARCHITECTURE str OF device IS
`if {$use_rom == 1} {
VH *
   COMPONENT rom
      PORT(in_adr  : IN  STD_LOGIC_VECTOR(1 DOWNTO 0);
           out_d   : OUT STD_LOGIC_VECTOR([expr $wordwidth -1] DOWNTO 0));
      END COMPONENT;

      SIGNAL opa, opb : STD_LOGIC_VECTOR([expr $wordwidth -1] DOWNTO 0);*
}`
...
    ### calculate maximum integer [wordwidth^2]-1
    for {set j 2} {$j <= $wordwidth} {incr j} {
       set weight [expr {$weight << 1}]
       incr weight
    }

    ### calculate coefficient and norm it to maximum integer
    set coeff [expr [round($weight * cos($pi * (2 * $i + 1)/(2 * $n)))]]
...
ARCHITECTURE str OF rom IS
BEGIN
  run: PROCESS(in_adr)
  BEGIN
    IF in_adr = "00" THEN
      out_d <= "`Bit_Gen $wordwidth 0 8`";
    ELSIF in_adr = "01" THEN
```

Abb. 6: Skizze zur Implementierung des Modellgenerators

4. Zusammenfassung

Eine datenbankgestützte Modellverwaltung bringt vielfältige Vorteile insbesondere in Bezug auf Sicherstellung der Korrektheit und der autorisierten Verteilung der Information u.a. über Web-Clients. Die durch die Modellverwaltung gegebene Wiederverwendbarkeit wird durch die Verwendung von flexiblen, anpassungsfähigen Modellgeneratoren um neue Möglichkeiten erweitert. Eine datenbankgestützte Modellverwaltung kann hochschulübergreifend die Zusammenarbeit fördern und den Wissensaustausch erleichtern.

Moderne Entwicklungen in Speichertechnologien

Prof. Dr. Gerald Higelin
FH Furtwangen
Gerwig Platz 1
78120 Furtwangen

ABSTRACT

Dynamische Halbleiterspeicher haben sich in den vergangenen drei Jahrzehnten kontinuierlich entwickelt und konnten somit stetig die Anforderungen hochintegrierter Digitalsysteme erfüllen. Der Integrationsgrad wuchs dabei um etwa 60% pro Jahr wodurch, dem Moore-Gesetz folgend, alle drei Jahre eine neue Generation mit dem vierfachen Speichervolumen eingeführt wurde.

Die Zugriffszeit konnte dagegen nur gering verbessert werden, sodass sich im gesamten Zeitraum etwa eine Steigerung um den Faktor 10 ergeben hat. Die dynamischen Speicher konnten damit der Entwicklung bei Prozessoren nicht mithalten. Statische Speicher sind gegenüber dynamischen Speichern um etwa einen Faktor 3 schneller im Datenzugriff, entwickeln sich in der Performance aber ebenso langsam.

Zunehmende Bedeutung erhielten in den vergangenen Jahren die nichtflüchtigen Halbleiter-Speicher, allen voran FLASH-Speicher für mobile Consumer-Anwendungen, Mobilfunk, Digital-Fotographie, MP3-Audiosysteme. Diese FLASH-Speicher erfordern jedoch eine Spannungs-Versorgung von bis zu 12V zur Programmierung und sind noch wesentlich langsamer im Schreib- bzw. Erase- mode gegenüber DRAMs.

Sowohl bei DRAMs wie bei FLASH können in Zukunft nur geringe Performance–Gewinne durch weitere Skalierung erreicht werden.

Mit den sog. magneto-resistiven Speichern (MRAM) stehen Bauelemente mit völlig neuen Fähigkeiten kurz vor ihrer kommerziellen Einführung. MRAMs werden gegenüber DRAMs und FLASH wesentlich schnelleren Datenzugriff haben, sie lassen sich sehr hoch integrieren, arbeiten mit 3V Spannungsversorgung und sind für eine nahezu unbegrenzte Zahl an Programmierzyklen nichtflüchtig.

1. Speicher-Organisation

Alle Typen von Halbleiter-Speichern sind matrixförmig organisiert. Die Speicherzellen sind in einem Array angeordnet und der Zugriff erfolgt über Wort- und Bitleitungen auf jede einzelne Zelle (random access) /1/. Die Adressierung erfolgt im Multiplex-Verfahren über ROW-Decoder und COLUMN-Decoder (Fig.1), sodass die Zugriffszeit sich im wesentlichen zusammensetzt aus t_{RAS} und t_{CAS}. In Fig.2 ist die 1-T Zelle eines dynamischen Speichers dargestellt. Die Information wird in Form elektrischer Ladungen im Kondensator der Zelle gespeichert. Der Speicherzugriff erfolgt durch Ansteuerung des Auswahltransistors über die Wortleitung. Das Signal wird über die Bitleitung (BL) geschrieben oder gelesen. Durch Unterschwell-Leckströme wird der Kondensator auch im gesperrten Zustand entladen, sodass beim DRAM nach ca. 10 ms ein Refresh-Zyklus notwendig ist. Vor dem Auslesen wird die Bitleitung auf ½ VDD vorgeladen. Beim Auslesen der Zellemuss die gesamte Bitleitung umgeladen werden, d.h. die Zellladung verteilt sich auf die Zell- (C) und die Kapazität der Bitleitung (C_{BL}), die etwa um den Faktor 5 größer als die Zellkapazität ist. Somit kommt am Leseverstärker nur ein Bruchteil der Zellspannung an.

Die Leseverstärker sind als Differenzverstärker realisiert. Es werden jeweils zwei BL miteinander verglichen. Als äußerst günstig hat sich dabei die sog, gefaltete Bitleitung erwiesen. Die Empfindlichkeit der Leseverstärker (Sense Amplifier) erfordert eine Mindestkapazität der Zelle von 25 µF.

$$V_{SEN} = \frac{1}{2(C + C_{BL})} \cdot \frac{\varepsilon_0 \cdot A \cdot V_s}{d} + \Delta V$$

V_{SEN} Eingangs-Signal am Leseverstärker

V_S Zellspannung

ΔV Rauschen

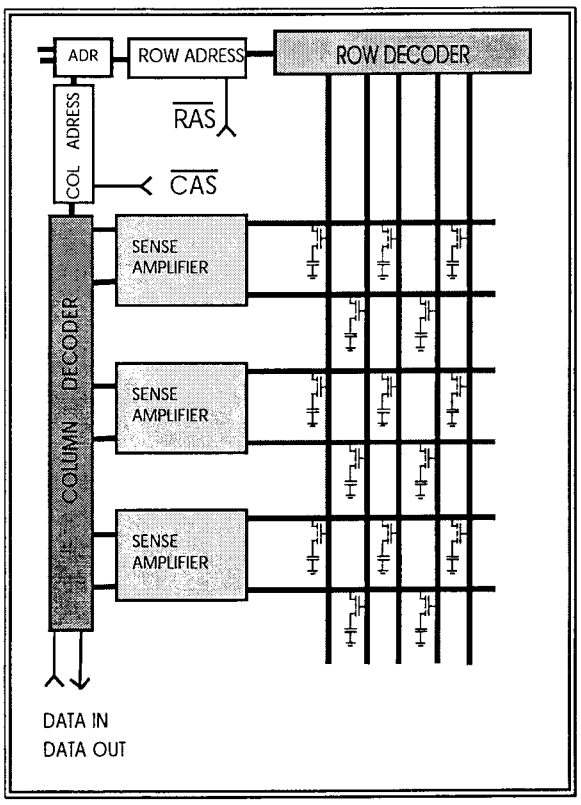


FIG.1 SPEICHER-ORGANISATION

2. DRAM

Die enorme Steigerung des Integrationsgrades konnte wurde erreicht werden durch Vergrösserung der Chipflächen und durch den kontinuierlichen Shrink der Dimensionen. Damit waren die DRAMSs in der Schrittmacherrolle für stets neue Technologie-Generationen. Für ein Gbit-DRAM ist heute eine Technologie mit 0.10 µm Lithographie notwendig. Die Beherrschung geringer Linienbreiten und verbesserter Justiergenauigkeit ermöglichte auch die Einführung verbesserter Prozess-Architekturen. Die Einführung mehrerer Poly-Silizium Schichten und Metallisierungs-Lagen aus Al, W und Cu sind dabei notwendige Voraussetzung. So wurden im Laufe der Entwicklungen unterschiedliche Konzepte realisiert (FIG.3), so die planare Zellen, Trench Zelle, Stack Zelle und in Zukunft voraussichtlich die COB (condensator over bitline). Durch Reduktion der Schichtdicke und Einsatz neuer Materialien mit hohem ε konnte die Zellkapazität trotzt Flächenshrink konstant gehalten werden. Heute kann bei einer Linienbreite von F= 0,13 µm eine Zellfläche von 0,11µm^2 erreicht werden. Damit hat man eine sog. 6 F^2 Zelle. Mit weiteren Architekturverbesserungen können auch noch 4 F^2 Zellen realisiert werden.

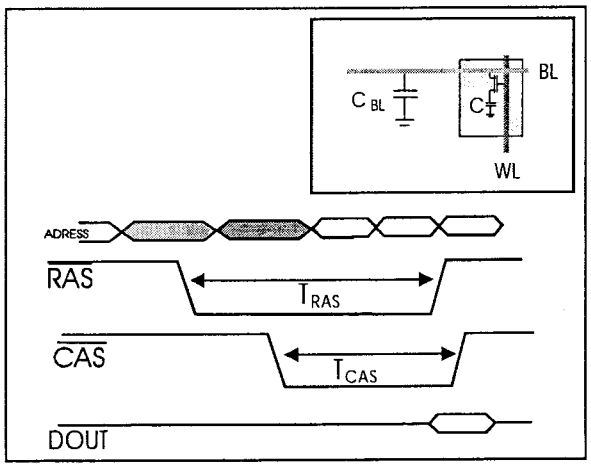


FIG. 2 DRAM-ZELLE, LESEZUGRIFF

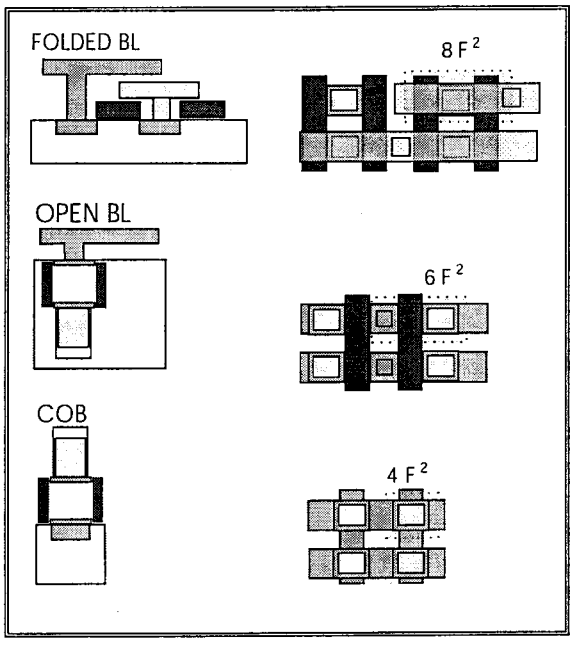


FIG.3 DRAM ZELL- KONZEPTE

Leistungsverbrauch und Zuverlässigkeits-Anforderungen machen eine kontinuierliche Spannungsreduktion notwendig. Eine Spannungsreduktion macht allerdings auch die Absenkung der Einsatzspannung V_T der Transistoren notwendig, wodurch das Leckstromproblem des Auswahltransistors kritischer wird.

Die prinzipiellen Grenzen von DRAM Zellen sind die notwendige Zellkapazität von 25 µF sowie die nach unten begrenzte Einsatzspannung des Auswahltransistors.

3. FLASH

Die heute realisierbaren Flash-Speicher haben eine Kapazität von 64 Mbit /4/. Eine Zelle in 0,13µm Technologie hat eine Fläche von 0,154µm^2 , dies entspricht einem 8 F^2 Layout. Die Prozess-Komplexität ist nur geringfügig höher im Vergleich zu einer Standard- CMOS Technologie. Voraussetzung ist die Beherrschung einer Floating Gate - Technik mit einem ultradünnen Tunneloxid. Mittels eines channel hot electron (CHE) Mechanismus kann der Speicher zellenweise programmiert werden, und mit dem Fowler-Nordheim (FN)Tunneln wieder gelöscht werde. Das FN-Tunneln funktioniert allerdings nur Blockweise, da die gesamte WL auf ca. -12V abgesenkt werden muss. Die Programmier - Spannungen erfordern einen on chip Spannungsgenerator, der als Dickson charge pump in CMOS realisiert werden.

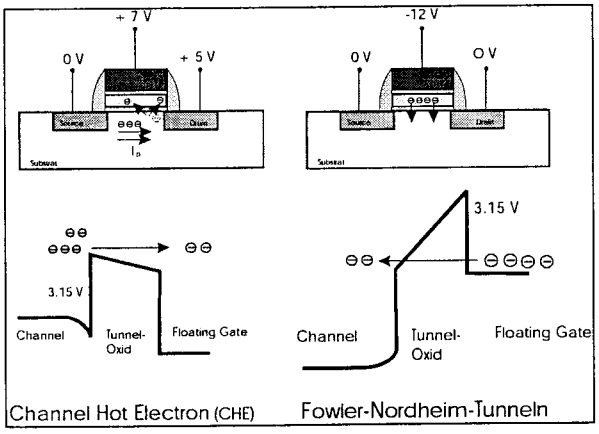


FIG.4 FLASH-ZELLE
PROGRAMMIERUNG UND ERASE

Der Lesezugriff erfolgt über die Bewertung der Einsatzspannung V_T des Floating Gate Transistors (FIG.5). Die Programmierzyklen von Flash Zellen liegen allerdings im Bereich von ms. Flash Speicher werden in Zukunft auch als 4 F^2 Zellen integrierbar sein. Eine weitere Integrationssteigerung wird möglich werden durch Dual-Bit Zellen. Dabei werden vier verschiedene Zustände der Einsatz-Spannung zu unterscheiden sein (FIG.5). Hierzu sind extrem enge Prozessfenster im CMOS Prozess notwendig.

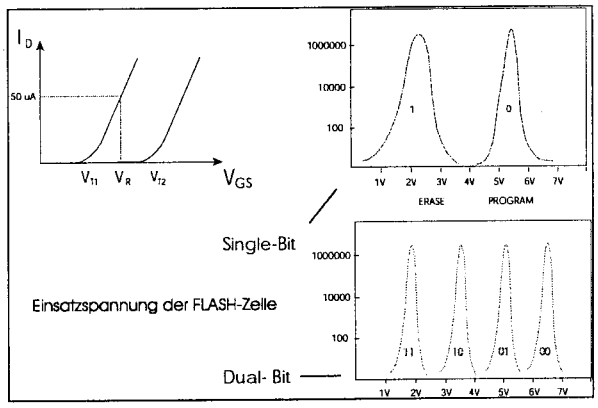


FIG.5 FLASH-ZELLE, SINGLE- DUAL-BIT

Die prinzipiellen Grenzen von FLASH Zellen sind die langsame Programmierzyklus sowie die durch V_T Drift begrenzte Anzahl an Programmierzyklen (ca. 10^5-10^6).

4. MRAM

Mit der Magnetic Random Access Memory (MRAM)-Technologie steht die modernste Speichertechnik kurz vor ihrer Marktreife. Mehrere grosse Halbleiterhersteller haben die Einführung für 2004/05 bereits angekündigt.

MRAMs speichern Informationen nicht in Form von elektrischen, sondern magnetischen Ladungseinheiten. Damit sind diese Speicher nichtflüchtig und die Zahl der Programmierzyklen ist unbegrenzt.

Grundlage dafür ist der magneto-resistive Effekt in dünnen Schichten /5/. Dieser Effekt ist schon seit über 30 Jahren erforscht, die notwendige Beherrschung dünner Tunnelbarrieren (3nm) stand bisher breiter Anwendung entgegen.

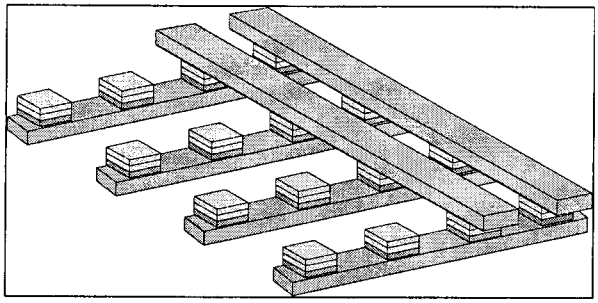


FIG.6 MRAM – ZELLENFELD

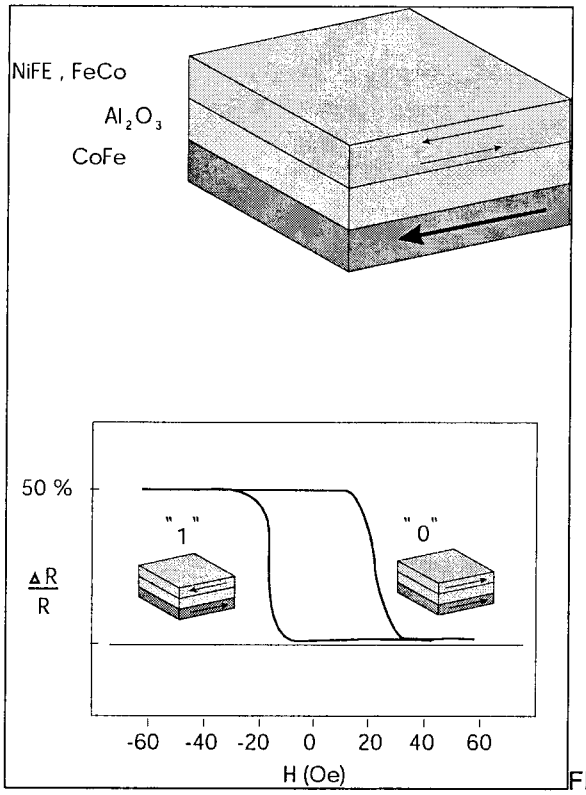


FIG.7 TMR – EFFEKT

Zwei magnetische, metallische Schichten sind durch eine ultradünne Isolationsschicht getrennt. Durch dieses Sandwich kann ein Tunnelstrom fließen. Der Übergangswiderstand hängt stark ab von der Magnetisierungsrichtung beider Schichten TMR (Tunnel-magnetic resistance). Die untere Schicht ist hartmagnetisch mit einer konstanten magnetischen Remanenz während die obere weichmagnetische Schicht durch einen Programmierstrom ummagnetisiert werden kann. Zwischen paralleler und antiparalleler Magnetisierung ergibt sich ein Widerstands-Differenz von etwa 50%, da beim Tunneln bevorzugt Elektronen mit parallel ausgerichtetem Spin durchgelassen werden (spin valve).

Zum Programmieren sind dabei Ströme von etwa 2mA notwendig (0,5 µm Technologie), während der Lesevorgang nur 20 µA erfordert. Bei geeigneter Orientierung der Magnetisierungsrichtungen, können die Rasterleitungen sowohl zum Lesen wie zum Programmieren genutzt werden. Damit ergibt sich ein optimiertes Layout für minimale 4 F^2 Zellflächen, F entspricht dabei den minimalen Breiten der Leitungen. MRAM versprechen extrem kurze Programmierzeiten im Bereich weniger ns.

Da in diesem Speicherprinzip der elektrische Widerstand gemessen wird ergeben sich durch Skalierung keine Probleme.

5. Zusammenfassung

Die bisherigen Speichertechnologien sind nur noch bedingt skalierbar, da die Prozessarchitekturen bei 4 F2 Layout ausgeschöpft sind. Bei DRAMs limitiert die erforderliche Zellkapazität die Shrinkfähigkeit, bei Flash sind die enorm langen Programmierzeit und die begrenzte Anzahl der Programmierzyklen limitierende Faktoren.

MRAMs vereinen die Vorteile von bisherigen Speichern- Technologien, einerseits die enorme Speicherkapazität und die niedrigen Kosten von DRAMs, andererseits die hohe Geschwindigkeit von SRAMs und die Nichtflüchtigkeit von Flash-Speichern.

5. Referenzen

/1/ B.Prince:Semiconductor Memories: A Handbook of Design, Manufacture and Application, J Wiley 1993

/2/ N.Lu, H.Chao, IEEE J.of Solid State Circuits, Aug 1984

/3/ S.Lai , IEEE IEDM Tech.Digest 1998

/4/ Al Fazio et al. Intel Technical Journal, Vol.6, 2002

/5/ M.Julliere, Phys.Lett. 54A, 1975

FH Karlsruhe
HOCHSCHULE FÜR TECHNIK

Messtechnische Ermittlung des Intercept-Punktes IP3

Alexey Defer, Prof. Hans Sapotta, FH Karlsruhe

ABSTRACT

Professional signal generators do not guarantee satisfying results when measuring the 3rd order intercept point. A simple, low cost dual tone generator based on crystals is being presented.

A professional spectrum analyzer need not be sufficient, too. A solution to improve the analyzer together with the above mentioned dual tone generator gives outstanding results for education of students at universities of applied sciences.

1. Einführung, Grundlagen

Der Intercept-Punkt IP3 stellt mittlerweile eine allgemein anerkannte Kenngröße für die Großsignalfestigkeit eines Verstärkers dar. Er hat die früher übliche Angabe des Klirrfaktors (außer bei Audio-Verstärkern) weitgehend abgelöst. Vorteile gegenüber dem Klirrfaktor sind zum einen die Gültigkeit des IP3 in Schmalbandsystemen und zum anderen die Angabemöglichkeit einer einzigen Zahl unabhängig von der weiteren Angabe der Messbedingungen. Im amerikanischen Kulturkreis und mittlerweile auch im deutschen Bereich ist somit der Intercept-Punkt eine häufig anzutreffende Größe.

Nach der so genannten Nichtlinearen Kleinsignaltheorie wird ein Zweitonsignal, das aus einer cosinusförmigen Schwingung mit identischer Amplitude jeweils auf der Frequenz f_1 und auf der Frequenz f_2 besteht, am Eingang des zu untersuchenden Objekts DUT (Device under Test, in der Regel ein Verstärker) angelegt. Aufgrund der nichtlinearen Verzerrungen, die an verschiedenen Stellen innerhalb der Schaltung des Verstärkers entstehen, kann man am Ausgang des Verstärkers mit einem Spectrum Analyzer eine Reihe von neuen Signalen, so genannte Intermodulationsprodukte, auf den unterschiedlichsten Frequenzen beobachten. Liegen die beiden Frequenzen f_1 und f_2 dicht beieinander, so sind die Intermodulationsprodukte auf den Frequenzen $2f_1-f_2$ und $2f_2-f_1$ von besonderem Interesse, da diese dann im Gegensatz zu allen anderen Intermodulationsprodukten nah an dem Originalsignal liegen und somit durch Filter und vergleichbare Maßnahmen nur schwer unterdrückt werden können. Bild 1 zeigt den Messaufbau und das Ergebnis.

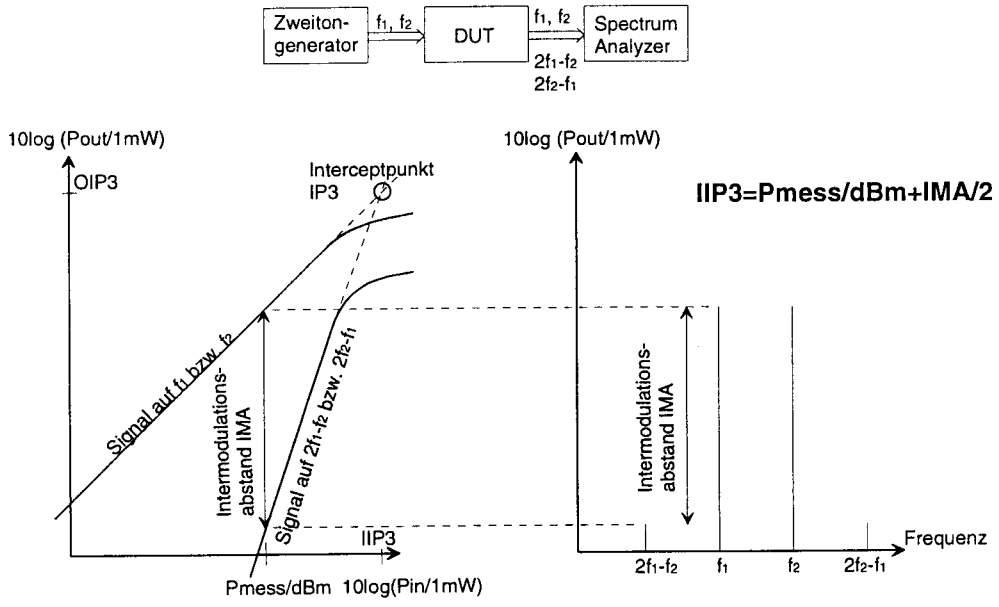


Bild 1 Zur Theorie des IP3

Trägt man nun den Eingangspegel (in dBm) gegenüber dem Ausgangspegel (in dBm) auf, so erhält man für kleine Eingangspegel gemäß der linearen Abhängigkeit des Ausgangsleistung von der verfügbaren Eingangsleistung auf den Frequenzen f_1 und f_2 eine Gerade mit der Steigung 1. Diese Kurve weicht bei größeren Eingangspegeln von der idealen Gerade ab, da Kompression (oder Expansion) auftritt.

Für das Signal auf den Frequenzen $2f_1$-f_2 und $2f_2$-f_1 hängt die Ausgangsspannung von der dritten Potenz der Eingangsspannung ab. Infolge der doppelt logarithmischen Darstellung ergibt sich für diese Signale eine Kurve, die bei kleinen Eingangssignalen eine Gerade mit der Steigung 3 darstellt. Diese Gerade weicht bei größeren Eingangssignalen aufgrund von Verzerrungen höherer Ordnung von der idealen Gerade ab.

Aus dieser Darstellung kann man sehr einfach den Intercept-Punkt dritter Ordnung IP3 als den Schnittpunkt dieser 2 Geraden definieren.

Die hierzu notwendige Messtechnik ist theoretisch betrachtet sehr einfach. Man benötigt nur einen Messpunkt bei einem Eingangspegel Pin/dBm, bei dem man den Intermodulationsabstand IMA als die Differenz der Ausgangspegel zwischen diesen beiden Geraden ermittelt. Der IP3 berechnet sich dann nach der Formel

$$IP3/_{dBm} = Pin/_{dBm} + IMA/_{2}.$$

Eine Messung reicht also zur Bestimmung des IP3 aus. In Bild 2 wird dies an einem Beispiel verdeutlicht. Man misst bei einem Eingangspegel von -10dBm einen Intermodulationsabstand von 53dB, was einem Intercept-Punkt von 16,5dBm entspricht.

Wie man an diesem Beispiel erkennt, ist der Pegel auf der Frequenz $2f_2$-f_1 größer als derjenige auf der Frequenz $2f_1$-f_2. Dies liegt an der Intermodulation höherer Ordnung, macht jedoch die reale Bestimmung eines IP3 immer zu einem gewissen Grade unsicher.

2. Der Messaufbau, Probleme

In Bild 3 ist ein typischer Messaufbau gezeigt. Man sieht 2 Messsender, die einfach mittels T-Stück verbunden sind und hier ohne DUT direkt den Spektrum Analyzer ansteuern. Wie man jedoch unschwer erkennen kann, sieht man am Analyzer genau das Bild, das aus Bild 2 bekannt ist. Der Messaufbau an sich hat also lediglich einen IP3 von 16,5dBm. Damit ist es kaum möglich, reale Intercept-Punkte von Verstärkern zu ermitteln. Deren IP3 kann Werte erreichen, die um 2 Zehnerpotenzen größer liegen.

Dieser geringe System-Intercept-Punkt hat im Wesentlichen 2 Ursachen. Zum einen geht die Leistung eines Messsenders hälftig auch in den 2. Messsender und intermoduliert in der Ausgangsstufe mit dessen Signal (und umgekehrt). Zum anderen ist das schwache Intermodulationsverhalten des Spectrum Analyzers selbst zu nennen. Zur Abhilfe kann man Dämpfungsglieder am Ausgang jedes Messsenders und am Eingang des Spectrum Analyzers vorsehen. In Bild 4 ist beispielhaft jeweils 20dB vorgesehen. Durch die Dämpfungsglieder am Ausgang der Messsender wird das Signal in den jeweils anderen Messsender um 40dB gedämpft, was sicher ausreichend ist. Am Eingang des Spectrum Analyzers hat der Hersteller in aller Regel in weiser Voraussicht bereits ein schaltbares Dämpfungsglied vorgesehen, von dem dann nicht automatisch, sondern manuell Gebrauch zu machen ist.

Diese Dämpfungsglieder führen zum Erfolg, jedoch wird durch die Dämpfung die verfügbare Leistung der Messsender entsprechend reduziert, was die Messung hoher Intercept-Punkte erschwert. Der Standard-Messsender hat einen maximalen Ausgangspegel von +13dBm, was an sich bereits knapp ist.

Besonders verheerend wirkt sich das Dämpfungsglied am Eingang des Spektrum Analyzers aus. Dessen eigene Rauschzahl liegt bei dem hier gezeigten Modell von Agilent bereits bei 20dB (ein sehr hoher Wert), der sich durch vorgeschaltete Dämpfungsglieder entsprechend erhöht. Unter diesen Umständen wird es sehr schwierig, im Rauschen nach Intermodulationsprodukten zu fischen.

Weitere Alternativen seien hier der Vollständigkeit halber noch genannt.

2 Messender können auch über einen so genannten 3dB-Koppler (Meinke-Gundlach: Taschenbuch der Hochfrequenztechnik, 5. Auflage 1992. Springer-Verlag, Berlin, Heidelberg, New York e.a, p. L32) verbunden werden. Dieser besteht aus 4 Leitungsstücken von $\lambda/4$ Länge und einem Wellenwiderstand von $Z_L/\sqrt{2}$. Diese Alternative funktioniert dann, wenn der Verstärker selbst einen geringen Eingangsreflexionsfaktor aufweist. Allerdings steht diese Forderung im Widerspruch zu einer anderen zentralen Forderung an einen Verstärker.

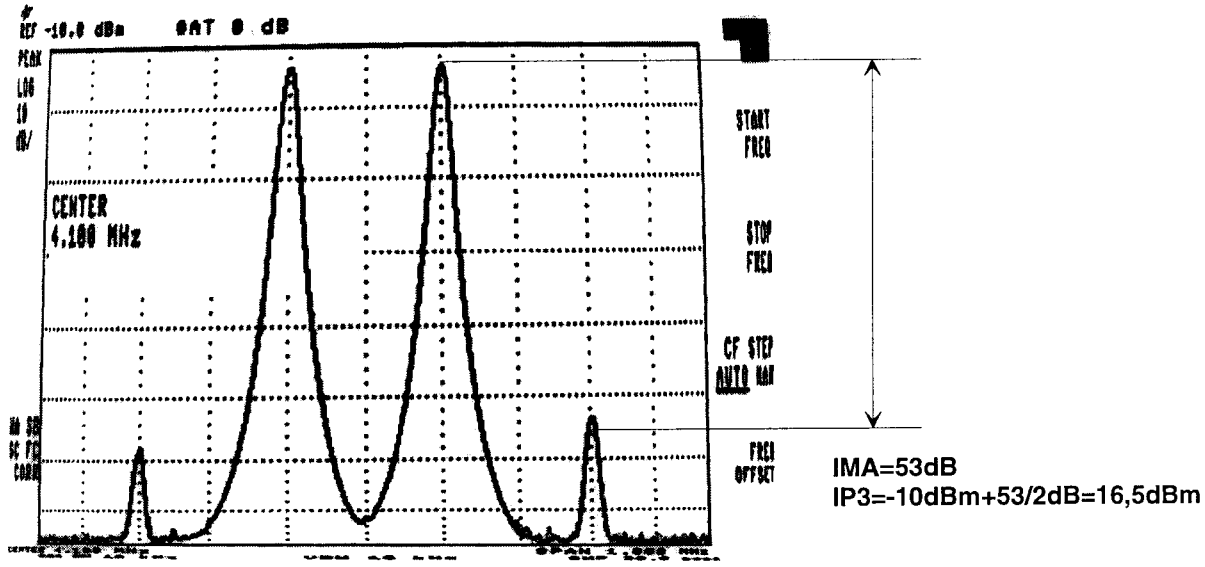


Bild 2: Typisches Messergebnis

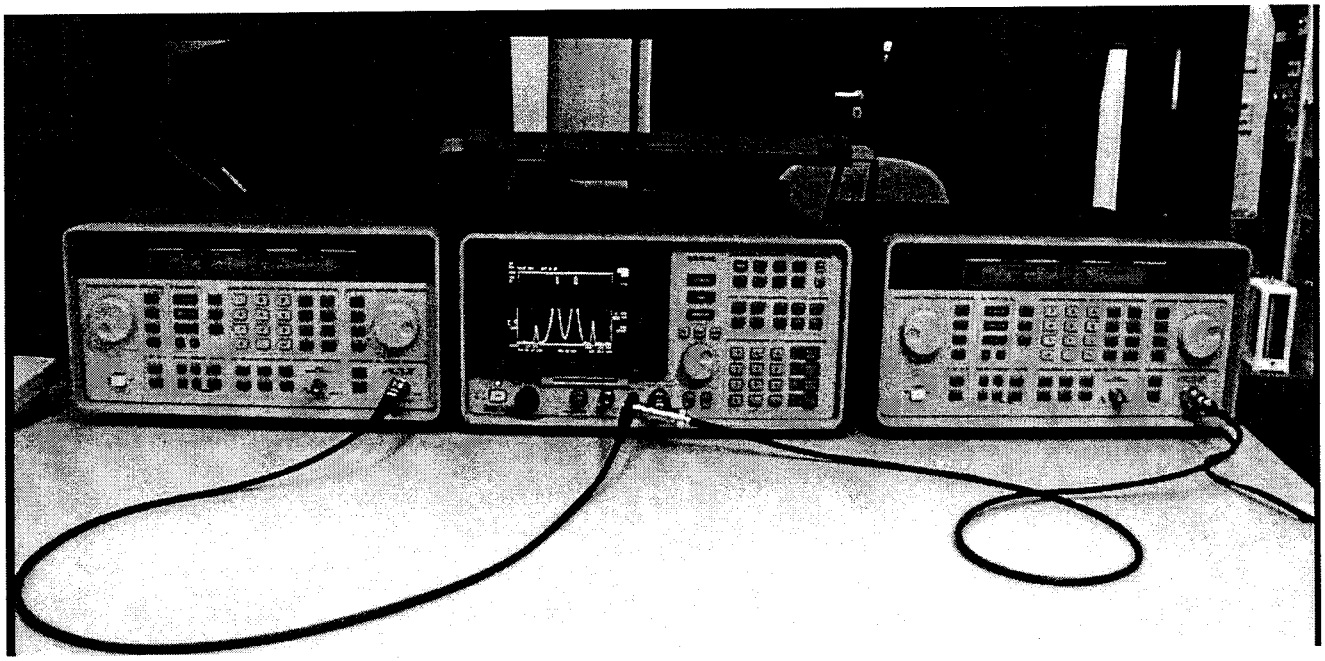


Bild 3: Messaufbau

Wenn die Eingangsimpedanz gleich der Generatorimpedanz ist, ist die Rauschzahl immer größer als 3dB. Kleinere Rauschzahlen sind nur mit Leistungsfehlanpassung erreichbar.

Eine andere alternative Messmethode besteht darin, den 1dB-Kompressionspunkt zu bestimmen. Nach der nichtlinearen Kleinsignaltheorie liegt dieser 9,7dB unterhalb des IP3. Dies gilt jedoch nur für komprimierende Systeme. In der Praxis sind die meisten System in der Tat komprimierend, der häufig verwendete Bipolartransistor in Emitterschaltung ist jedoch expandierend. Eine reale Schaltung hat nicht nur eine, sondern eine Vielzahl von Intermodulationsquellen. Man wird also immer eine Kombination von komprimierenden und expandierenden Signalen vorfinden, so dass die Bestimmung des IP3 über den 1dB-Kompressionspunkt nicht zuverlässig ist.

3. Problemanalyse

Angesichts dieser schwierigen Situation muss man sich fragen, ob der Griff in das Messgeräterepertoire von Agilent und Rohde & Schwarz eine angemessene Problemlösung darstellt. Natürlich gibt es auch intermodulationsfeste Messsender für einen entsprechenden Preis. Intermodulationsfeste Spectrum Analyzer werden z.B. von Rohde & Schwarz angeboten. Allerdings übersteigen diese den Etat von Hochfrequenzlaboren an Fachhochschulen in nicht geringem Maße.

Um Studierenden in einem Laborexperiment die nichtlineare Kleinsignaltheorie praktisch zu vermitteln, muss es auch preiswerter gehen.

Wenn man auf die freie Wahl der Messfrequenz verzichtet, bietet sich eine sehr einfache Problemlösung an. 2 Quarzoszillatoren werden zu einem Zweitongenerator verschaltet. Deren Frequenz ist stabil und exakt definiert. Wenn man einen Quarz auf einer Frequenz f_1 oder f_2 kauft, kann man auch einen zweiten kaufen und mit diesen ein Kerbfilter auf exakt dieser Frequenz aufbauen, um den Spectrum Analyzer auf diesen Frequenzen zu entlasten.

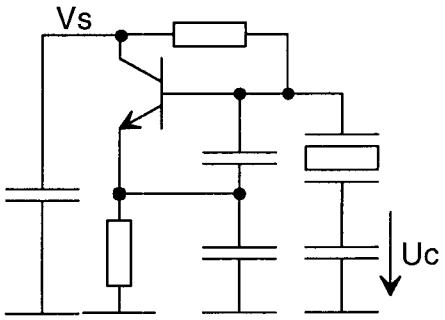


Bild 4: einfacher Quarzoszillator

4. Problemlösung

4.1 Der Zweitongenerator

Wir betrachten den Quarzoszillator nach Bild 4. Er besteht aus einem Transistor in Emitterschaltung, der über die Kapazitäten an Emitter und Basis in einem bestimmten Frequenzbereich einen negativen Realteil der Eingangsimpedanz realisiert. Somit wird der zwischen Basis und Masse angeschlossene Quarz entdämpft und beginnt die Schwingung. Üblicherweise wird das Oszillatorsignal am Emitter ausgekoppelt, da es dort niederohmig zur Verfügung steht. Hier soll aber die Aufmerksamkeit auf die Spannung Uc gerichtet werden, die an der Ziehkapazität in Serie zum Quarz ansteht. Uc hat auf der einen Seite eine hohe Amplitude, andererseits ist dieses Signal spektral äußerst rein. Oberschwingungen, die das nichtlineare Element Transistor erzeugt, werden durch die extremen Filtereigenschaften des Quarzes unterdrückt. Man erreicht mehr als 100dB Oberwellenabstand (wie misst man das eigentlich?).

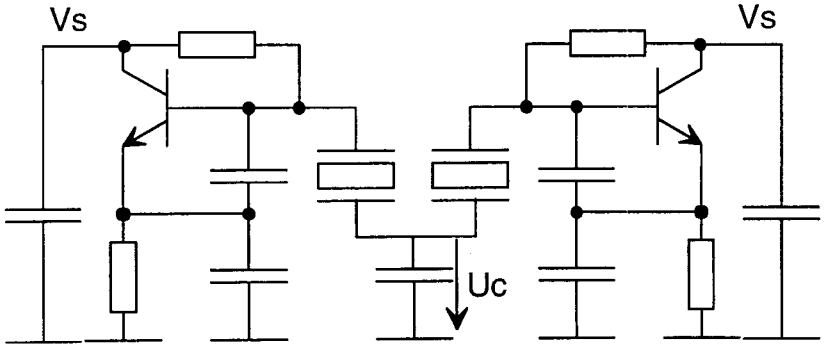


Bild 5: Zweiton-Quarzoszillator

Wenn ein Quarzoszillator ein derart reines Signal erzeugen kann, so kann man diese Schaltung um einen zweiten Quarzoszillator ergänzen, der auf die gleiche Ziehkapazität arbeitet. In Bild 5 ist das entsprechende Schaltbild gezeichnet.

Die beiden Quarze arbeiten auf verschiedenen Frequenzen. Verwendet man die handelsüblichen Frequenzen 5MHz und 5,2MHz, hat man einerseits einen hinreichend großen Frequenzabstand, um mit einem preiswerten Spectrum Analyzer arbeiten zu können, andererseits ist die Frequenz hinreichend niedrig, um spezielle HF-Effekte wie übermäßige Kabelverluste und Nichtlinearitäten durch spannungsabhängige Kapazitäten gering halten zu können.

4.2 Das Kerbfilter

Beim Einkauf der Quarze kann man jeweils 2 Exemplare dieser Cent-Bauteile vorsehen. Zusammen mit 2 Übertragern entsteht ein Kerbfilter, das vor dem Spectrum Analyzer die Signale auf f_1 und f_2 um jeweils 20dB dämpft. Der Analyzer wird von diesen Signalen entlastet und die Intermodulationsprodukte werden somit relativ hervorgehoben.

Der Serienwiderstand von Quarzen bewegt sich im Bereich von 10-30Ω. Vom Hersteller wird im Sinne einer hohen Ausbeute meist weniger als 100Ω garantiert. Ein einfaches Parallelschalten des Quarzes zum Signalweg ist daher nicht möglich, die Dämpfungswerte wären zu gering. Abgesehen davon stört der Parasit Parallelkapazität. Also wird das Impedanzniveau durch 2 Übertrager 1:3 am Ein- und Ausgang auf 450Ω angehoben. Durch die Streuinduktivität der Übertrager entsteht ein leicht induktiver Anteil, der durch die Parallelkapazität der Quarze etwa wegkompensiert wird. Die Parallelschaltung der 2 Quarze erzeugt nun die geforderte Dämpfung, wobei jeder Quarz durch eine Ziehkapazität auf die exakte Frequenz des Zweitongenerators gezogen werden muss. Um die Toleranz im Serienwiderstand des Quarzes auszugleichen, sind zusätzlich individuell einzumessende Serienwiderstände vorgesehen.

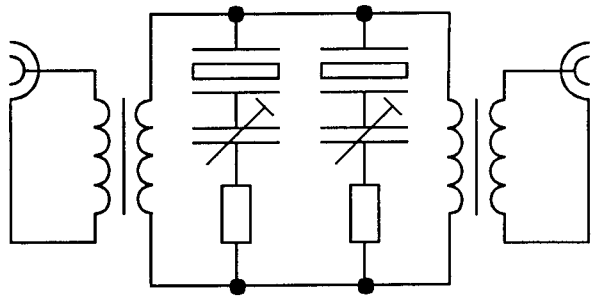


Bild 6: Schaltung des Kerbfilters

5. Messergebnisse

5.1 Der Zweitongenerator

Der Zweitongenerator erzeugt ein Signal, dessen spektrale Reinheit mit den hier zur Verfügung stehenden Messmitteln nicht messbar ist. Daher wurde das Kerbfilter bereits in Serie geschaltet. Gemäß Bild 7 erhält man bei einem Eingangspegel von -5dBm einen Intermodulationsabstand von 75dB, also 95dB ohne Kerbfilter. Diese Intermodulation ist wieder vom Spectrum Analyzer verursacht, der Zweitongenerator ist erheblich besser. Somit liegt der System-IP3 bei 42,5dBm, was eine erhebliche Steigerung bedeutet. Durch Einschaltung zusätzlicher Dämpfung vor dem Spectrum Analyzer kann man hier immer noch zulegen.

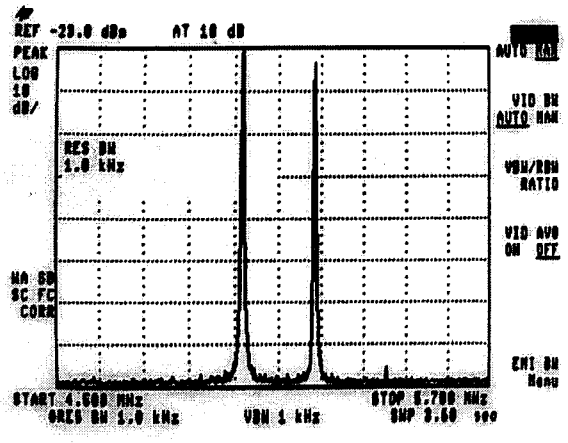


Bild 7: Spektrum des Zweitongenerators

5.2 Das Kerbfilter

Bild 9 zeigt die Übertragungsfunktion des Kerbfilters im Bereich einer Resonanzfrequenz. Man erkennt, dass abseits der Resonanzfrequenz gering (ca. 1dB) ist. Auf der Resonanzfrequenz ist das Signal um weitere 20dB gedämpft.

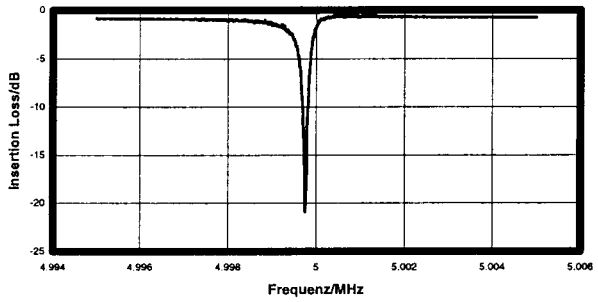


Bild 8: Übertragungsfunktion des Kerbfilters

5. Zusammenfassung

Es wurde gezeigt, dass professionelle Messsender bei der Bestimmung von Intercept-Punkten nicht zwangsläufig zum Ziel führen müssen. Alternativ wurde ein einfacher, preiswert aufzubauender Zweitongenerator vorgestellt.

Ein professioneller Spectrum Analyzer ist ebenfalls alles andere als ausreichend. Auch hier wurde eine Lösung zur Verbesserung des Analyzers vorgestellt, die zusammen mit dem Zweitongenerator zu einem kostengünstigen Messsystem für die studentische Ausbildung an Fachhochschulen führt.

Natürlich ist dort, wo Licht zu finden ist, auch Schatten. Problematisch ist vor allem die extreme Schmalbandigkeit des Kerbfilters. Bereits geringe Verschiebungen der Resonanzfrequenzen infolge Alterung und Temperaturtoleranzen führen zu undefinierten Dämpfungswerten auf f_1 und f_2. Dies ist aber problemlos mit dem Analysator durch einen einfachen Abgleichvorgang kontrollierbar.

Die Muller-Pipeline

Simulation und Implementierung einer asynchronen Kontrollstruktur

Kim Fahrion
Dr. Eckard Grass
Prof. Dr. Manfred Bartel

Fachhochschule Aalen, Fachbereich E/IN
Beethovenstrasse 1
73430 Aalen

Gliederung des Papiers

1 MOTIVATION 3

2 SYNCHRON VS. ASYNCHRON 3

3 GRUNDBEGRIFFE 4

4 FOUR-PHASE BUNDLED-DATA PROTOCOL 5

5 AUFBAU DER MULLER-PIPELINE 6

5.1 MULLER-PIPELINE UND DIE REGISTER-TRANSFER EBENE 7

6 DIE TESTBENCH DER SIMULIERTEN MULLER-PIPELINE 8

7 BETRIEBSARTEN DER MULLER-PIPELINE 9

7.1 BETRIEBSART 1: EINZELIMPULS 9
7.2 BETRIEBSART 2: DAUERLAUF 10
7.3 BETRIEBSART 3: FIFO 11

8 PETRI-NETZ DES EINFACHEN HSE 12

9 DAS TEILWEISE ENTKOPPELTE HANDSHAKE-ELEMENT 13

9.1 PETRI-NETZ DES ENTKOPPELTEN HANDSHAKE-ELEMENTS 13

10 DIE TEILWEISE ENTKOPPELTE MULLER-PIPELINE 14

10.1 BETRIEBSART 1: EINZELIMPULS 15
10.2 BETRIEBSART 3: DIE MULLER-PIPELINE ALS FIFO : FIFO FÜLLEN 15

11 DIE IMPLEMENTIERTE MULLER-PIPELINE 16

12 ZUSAMMENFASSUNG 19

LITERATURHINWEIS 20

1 Motivation

- Die asynchrone Idee ist so alt wie der moderne Computer selbst. Alan M. Touring war wohl in den frühen 50er Jahren der erste Wissenschaftler, der sich mit asynchronen Schaltungen beschäftigt hat.
- Heutzutage erlebt der asynchrone Gedanke eine Renaissance. Mit dem asynchronen Prozessor AMULET der Firma ARM haben englische Wissenschaftler gezeigt, dass große asynchrone Schaltungen funktionieren und Vorteile gegenüber synchronen Schaltungen besitzen. Die Firma Phillips entwickelte ein 8051 µC Derivat, das asynchron arbeitet.
- Die asynchrone Idee verspricht große Vorteile gegenüber synchronen Schaltungen. Allerdings setzt das die genaue Kenntnis und sehr sorgfältiges Vorgehen beim Entwurf voraus. Je mehr über das Verhalten asynchroner Schaltungen bekannt und erforscht wird, desto deutlicher überwiegen die Vorteile!

2 Synchron vs. Asynchron

Die Entwurfsparadigmen komplexer integrierter Schaltungen müssen permanent hinterfragt werden, um im Parameterspannungsfeld

Kosten ⇔ Leistungsfähigkeit ⇔ Energieverbrauch ⇔ Zuverlässigkeit

den best möglichen trade-off zu erreichen. Eine der wesentlichsten Fragen der letzten 40 Jahre ist, ob eine Schaltung synchron oder asynchron betrieben werden sollte. Die permanent zunehmenden Entwurfsschwierigkeiten beim synchronen und die Forschungsergebnisse des asynchronen Entwurfs lassen den Schluss zu, dass asynchron betriebenen Schaltungen eine Renaissance bevorsteht. Vorteile asynchroner Schaltungen sind:

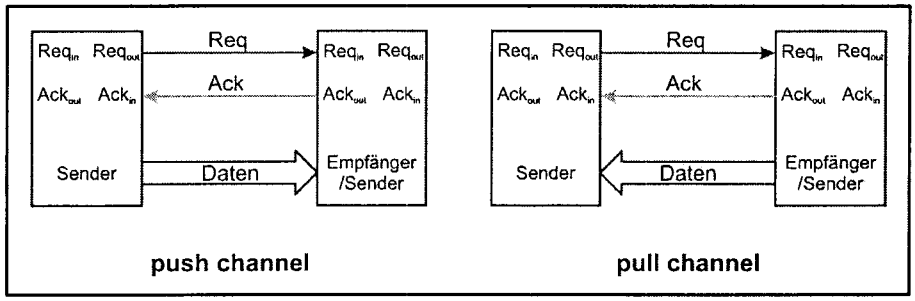
- **Schneller**: Bei synchronen Schaltungen richtet sich die Taktgeschwindigkeit nach dem langsamsten Schaltungsteil. Schnellere Schaltungsteile müssen auf die nächste aktive Taktflanke warten. Diese Wartezeit entsteht bei asynchronen Schaltungen nicht! Sobald ein Schaltungsteil sein Ergebnis berechnet hat, wird es per Handshake weitergereicht.
- **Sparsamer**: In einer großen Schaltung, wie z.B. ein Prozessor, sind nicht immer alle Schaltungsteile zugleich aktiv! In einer synchronen Schaltung werden aber alle Bereiche permanent mit dem globalen Taktsignal versorgt, so dass bei Schaltungen heute üblicher Größe ca. 30% des Energie für die Verteilung des Taktsignals verwendet wird. In einem asynchronen Entwurf können nicht benötigte Schaltungsteile einfach still gelegt werden. Damit verbrauchen sie nahezu keine Energie.
- **Störungsärmer**: Dadurch dass sich bei synchronen Schaltungen alle Aktivität auf den Zeitpunkt der aktiven Flanke des Taktsignals konzentriert, entstehen hochfrequente Störabstrahlungen mit beträchtlicher Energie. Bei asynchronen Schaltungen verteilt sich die Aktivität der Schaltung statistisch auf einen Zeitraum, so dass die durchschnittliche Stör-

abstrahlung und deren Energie geringer ist als bei synchronen Schaltungen. Die elektromagnetische Verträglichkeit ist deshalb besser.

3 Grundbegriffe

Im Zusammenhang asynchronen Entwürfen tauchen immer wieder bestimmte Grundbegriffe auf, die in diesem Abschnitt kurz erläutert werden sollen.

Abbildung 1: Grundbegriffe asynchroner Entwürfe



1. In einem asynchronen Entwurf werden die Daten immer zwischen zwei Modulen ausgetauscht.
2. Man spricht von einem von einem Sender und einem Empfänger.
3. Sender und Empfänger kommunizieren über einen Kanal. Ein Kanal stellt meist eine Punkt-zu-Punkt Verbindung her, die nur in eine Richtung arbeitet.
4. Man unterscheidet 2 Arten von Kanal: push channel und pull channel
5. Der Informationsaustausch zwischen Sender und Empfänger wird von einem Signalprotokoll gesteuert.
6. Es existieren zwei verschiedene Protokolle. Eines davon wird im folgenden Abschnitt beschrieben. Es handelt sich um das Protokoll, das in der Muller-Pipeline abläuft.
7. Prinzipiell wird jeder Datentransfer mit einem Request-Signal vom Sender angekündigt und mit einem Acknowledge-Signal vom Empfänger quittiert.
8. Die Datenflussrichtung relativ zu dem Request-Signal legt den Typ des Kanals fest:
   - push channel : die Daten bewegen sich in die gleiche Richtung wie die Request-Signale.
   - pull channel : die Daten bewegen sich in entgegen gesetzte Richtung wie die Request-Signale.

4 Four-phase bundled-data protocol

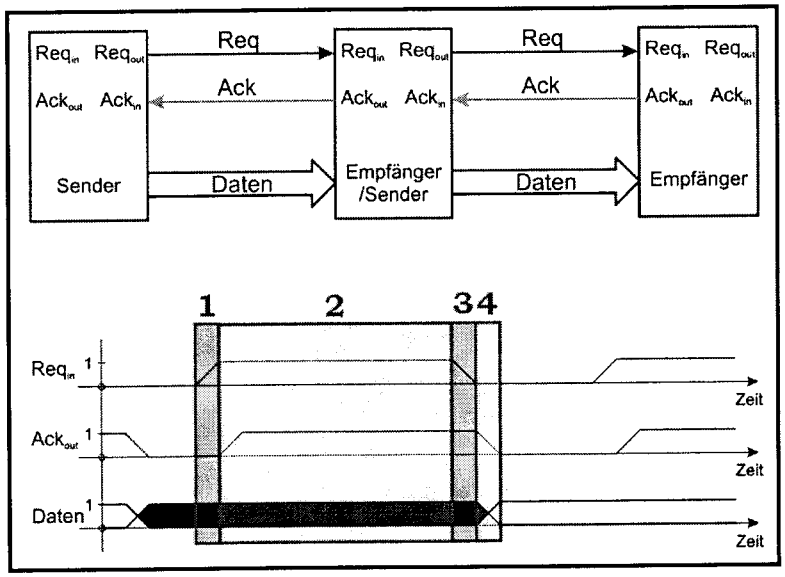
Bei diesem Protokoll werden die Signalpegel auf den beiden Steuerleitungen ausgewertet. Man könnte auch sagen die steigende Signalflanke ist die aktive Signalflanke. Das Signal muss immer zuerst auf log. 0-Pegel zurückgesetzt werden, bevor die Steuerleitung eine neue Information transportieren kann. Deshalb wird das Protokoll auch mit „return to zero" beschrieben. Auf diese Weise ergeben sich die vier Phasen des Protokolls.

Die Implementierung des four-phase bundled-data Protokolls mit einem einfachen Handshake-Element ist die einfachste Variante. Der große Nachteil dieser Struktur ist die Eigenschaft, dass nur jedes zweite Datenpfad-Register Daten speichern kann. Diese Variante ist also vergleichbar mit einem Master-Slave Flip-Flop aus der synchronen Welt.

Der prinzipielle Ablauf des four-phase bundled-data protocol sieht folgendermaßen aus:

1. In der ersten Phase signalisiert der Sender dem Empfänger neue Daten. Das geschieht mit dem Request-Signal.
2. In der zweiten Phase wird das Signal vom Empfänger quittiert durch ein Acknowledge-Signal an den Sender. In dieser Phase werden die Daten auch übertragen.
3. Bevor das Protokoll von vorne beginnen kann, müssen die Steuersignale auf Null zurück gesetzt werden. Das passiert in den nächsten zwei Phasen. Zuerst wird das Request-Signal vom Sender zurückgenommen.
4. Und daraufhin nimmt auch der Empfänger sein Acknowledge-Signal ebenfalls zurück.

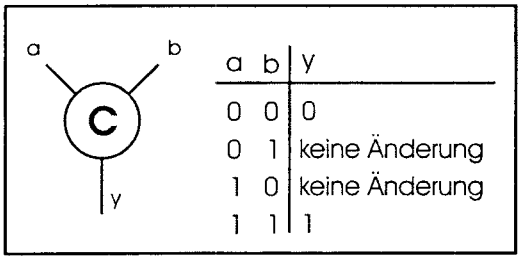
Abbildung 2: Signalverlauf des four-phase bundled-data protocol



5 Aufbau der Muller-Pipeline

Das C-Element ist einer der Grundbausteine der asynchronen Logik und der Hauptbestandteil der Muller-Pipeline. Es handelt sich um ein Speicherelement, wie man der Wahrheitstabelle in Abbildung 3 entnehmen kann. Erst wenn beide Eingänge auf log. 1-Pegel gesetzt werden wird die log. 1 im C-Element gespeichert und am Ausgang ausgegeben.

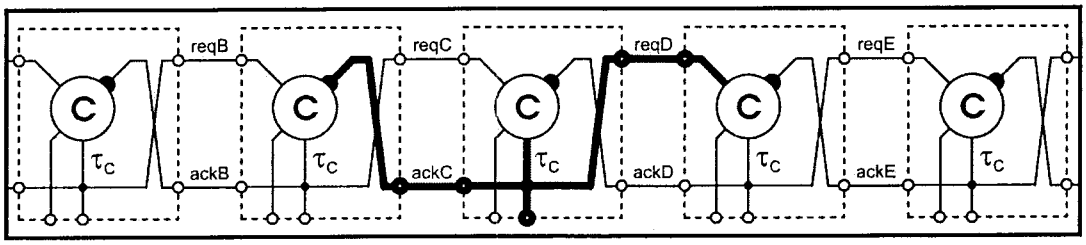
Abbildung 3: Symbol und Wahrheitstabelle des Muller-C Elements



a	b	y
0	0	0
0	1	keine Änderung
1	0	keine Änderung
1	1	1

In Abbildung 4 ist ein Ausschnitt aus einer Pipeline zu sehen. Sie setzt sich aus einzelnen gleichen Handshake-Elementen zusammen. Die fett gezeichnete Signalleitung soll eine Abhängigkeit zwischen den einzelnen Handshake-Elementen andeuten. Mithilfe Abbildung 5 wird der Aufbau der Handshake-Element erläutert.

Abbildung 4: fünf-stufiger Ausschnitt einer Muller-Pipeline

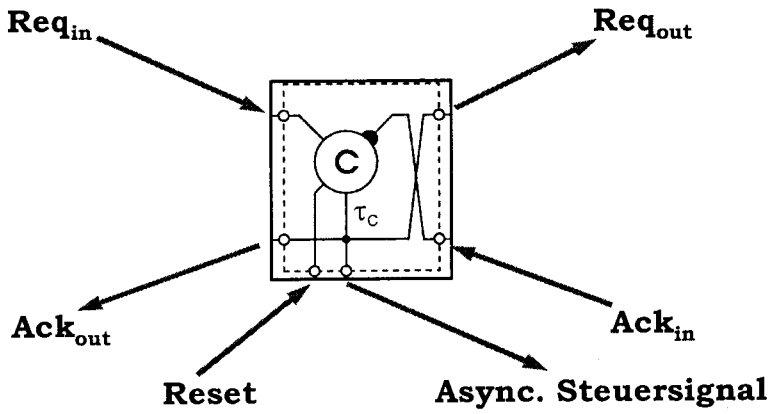


Jedes Handshake-Element besteht aus einem **C-Element** und einem **Inverter**, der mit Eingang b des C-Elements verbunden ist. Eingang a ist direkt mit dem Eingang $\mathbf{Req_{in}}$ des Handshake-Elements verbunden:

- Das Ausgangssignal y des C-Elements wird mit τ_C verzögert. In der Simulation beträgt τ_C = 1ns.
- Das empfangene Request-Signal wird bestätigt mit einem Acknowledge-Signal an den Vorgänger über den Ausgang $\mathbf{Ack_{out}}$.
- Zugleich wird das empfangene Request-Signal über den Ausgang $\mathbf{Req_{out}}$ weitergegeben.
- Dasselbe Signal dient auch als **asynchrones Steuersignal** für ein Datenpfad-Register.

- Wenn das nachfolgende Handshake-Element den Empfang des weitergeleiteten Request-Signals bestätigt, gelangt das Acknowledge-Signal über den Eingang **Ack_{in}** in das betrachtete Handshake-Element.
- Der zusätzliche Eingang **Reset** dient zum Rücksetzen des C-Elements.
- Die Signalkreuzung dient lediglich der Übersichtlichkeit und einer strukturierten Repräsentation.

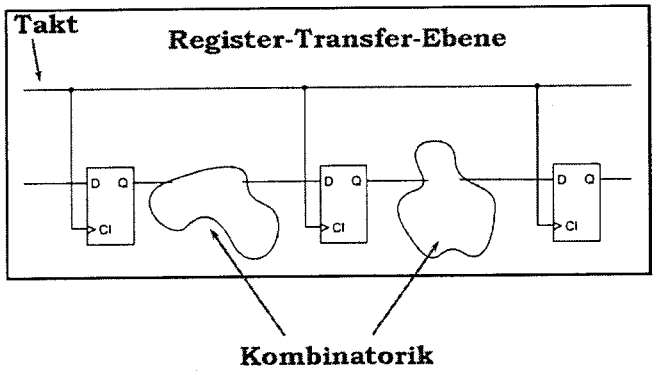
Abbildung 5: Ein Handshake-Element (HSE)



5.1 Muller-Pipeline und die Register-Transfer Ebene

Die Funktion, welche die Muller-Pipeline innerhalb einer asynchronen Schaltung übernimmt, soll mit Abbildung 6, Abbildung 7 und Abbildung 8 erläutert werden.

Abbildung 6: Die globale, synchrone Register-Transfer Ebene



Als asynchrone Kontrollstruktur des Datenflusses, übernimmt die Muller-Pipeline die Steuerung der Datenpfad-Register. Jede Stufe der steuert ein Register.

Abbildung 7: Austausch der Takt-Layer durch eine Muller-Pipeline

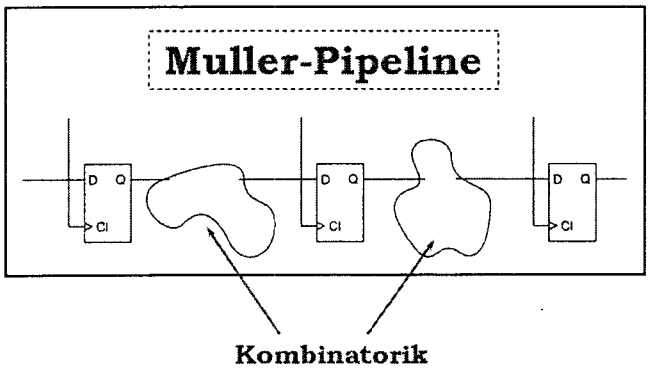
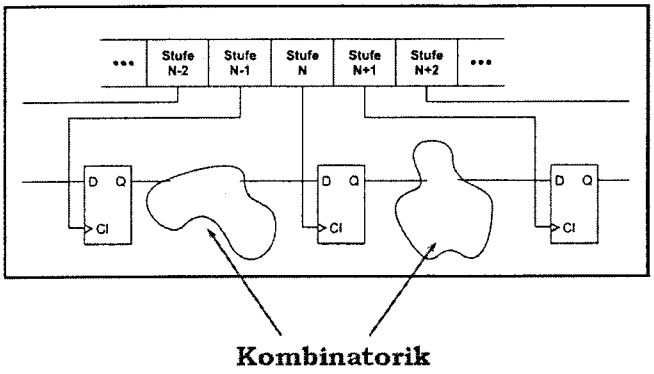


Abbildung 8 : Ersetzung der lokalen Takt-Layer durch eine Muller-Pipeline



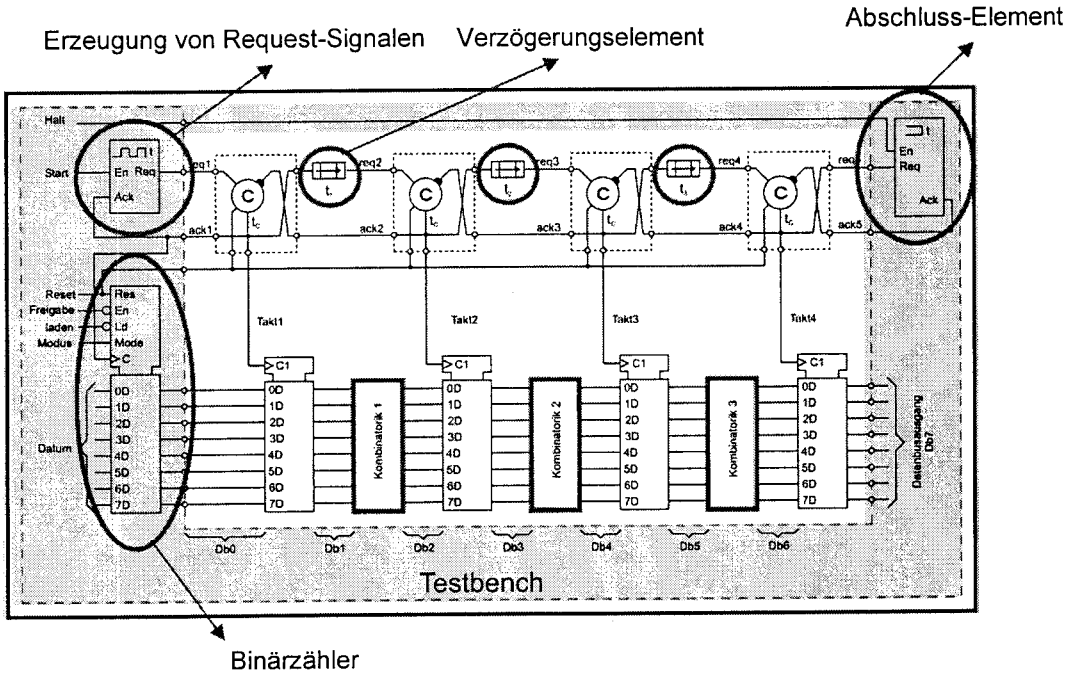
6 Die Testbench der simulierten Muller-Pipeline

Zwischen zwei Datenpfad-Register befindet sich im Normalfall eine kombinatorische Schaltung zur Datenverarbeitung im Datenpfad. Diese Schaltung benötigt eine gewisse Einschwingzeit der Ausgangssignale. Diese Verzögerungszeit muss in der Pipeline berücksichtigt werden. Zu diesem Zweck werden die Request-Impulse zwischen den Handshake-Elementen verzögert.

Die Verzögerungszeit ist individuell abgestimmt auf die maximale Einschwingzeit der Kombinatorik. Zur Simulation der Muller-Pipeline wird auf die Kombinatorik zur Datenverarbeitung verzichtet.

Eine Schaltung am Anfang der Pipeline kann Request-Impulse einzeln oder auch permanent erzeugen. Mit einem Abschluss-Element am Ende der Pipeline kann der Signalkreislauf unterbrochen oder geschlossen werden.

Abbildung 9: Die simulierte Testbench der Muller-Pipeline



Ein Zähler versorgt den Eingang des Datenpfads der Muller-Pipeline mit aufsteigenden 4-Bit Binärwerten. Jede Bestätigung eines Request-Signals erhöht den binären Zählerwert um eins.

Zu Beginn der Simulation wird der Zähler immer mit dem Reset-Signal auf Null zurückgesetzt. Die einzelnen Register des Datenpfads sind über einen Datenbus miteinander verbunden. Die Muller-Pipeline arbeitet somit als ein asynchrones Schieberegister. Der korrekte Ablauf des Handshake-Protokolls lässt sich durch Beobachtung der Fortsetzung der Zählerwerte im Datenpfad der Pipeline erkennen.

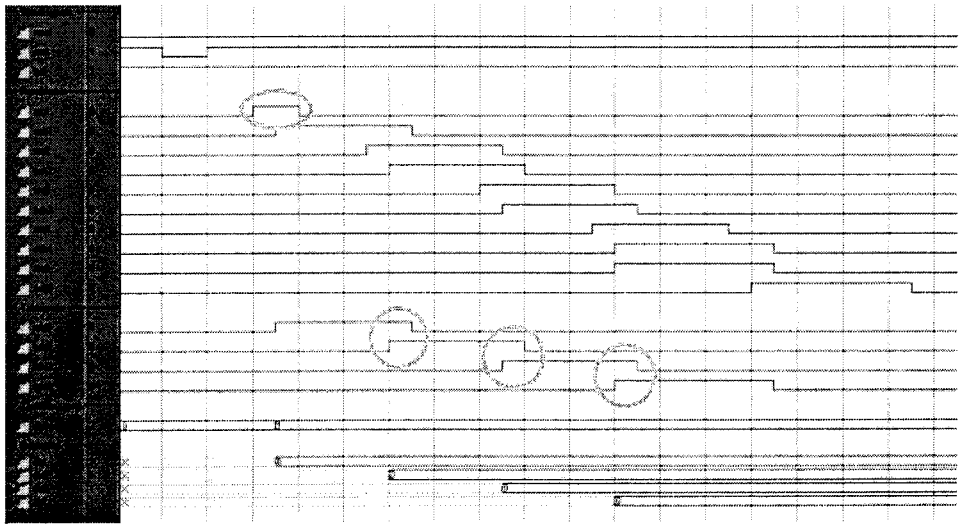
7 Betriebsarten der Muller-Pipeline

7.1 Betriebsart 1: Einzelimpuls

In dieser Betriebsart werden einzelne Request-Signale am Eingang der Muller-Pipeline erzeugt und hinein geschoben. Der Signalkreislauf ist am Ende der Pipeline geschlossen. Zur Bestätigung des Request-Signals sendet der Eingang der Pipeline ein Acknowledge-Signal zurück.

Im Diagramm ist der einzelne Request-Impuls markiert. Anhand den Signalverläufen der Handshake-Signale zwischen den einzelnen Handshake-Elementen erkennt man gut, wie sich der Impuls in der Pipeline fortsetzt.

Abbildung 10: Fortsetzung eines Einzelimpulses in der Muller-Pipeline



Als weitere Markierung ist in dem Diagramm eine Überlappung der asynchronen Steuersignale zu erkennen. Als Folge dieser Überlappung kann nur jedes zweite Datenpfad-Register einen neuen Wert des Zählers bzw. ein neues Datum speichern. Ursache dieses Verhaltens ist die Verkopplung der Ein- und Ausgangssignale der Handshake-Elemente.

7.2 Betriebsart 2: Dauerlauf

Bei dieser Simulation kann geprüft werden, ob die Daten vom Eingang des Datenpfads der Pipeline korrekt bis zum Ende durchwandern, wenn permanent neue Request-Signale in die Pipeline geschoben werden. Die Schaltung am Eingang der Pipeline erzeugt periodische Request-Signale a, während der Signalkreislauf am Ende der Pipeline über das Abschluss-Element geschlossen ist. Der Zähler wird wieder zu Beginn mit dem Anfangswert Null geladen. Mit jedem Acknowledge-Signal von der Muller-Pipeline erhöht sich der binäre Zählerwerte um eins.

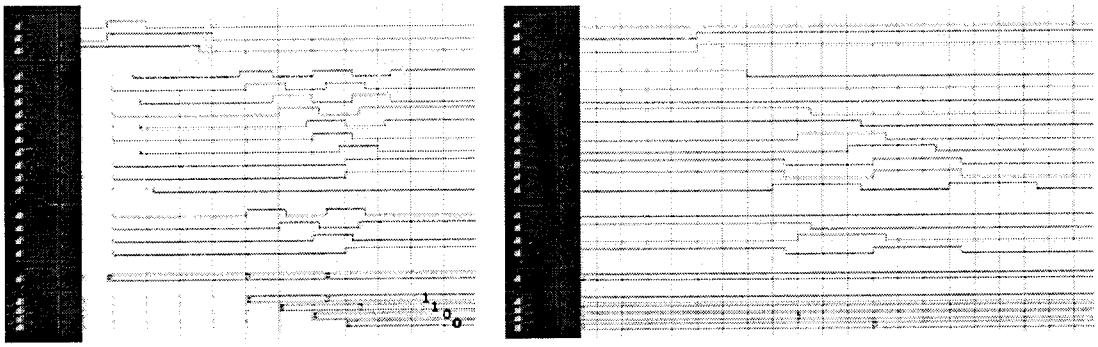
Bei korrekter Funktionsweise der Pipeline müssen sich die einzelnen Werte wellenartig durch die einzelnen Stufen der Pipeline fortsetzen. Eine Fehlfunktion im Ablauf des Handshake-Protokolls würde sofort auffallen, da in diesem Fall die aufsteigende binäre Zahlenfolge im Datenpfad unterbrochen wäre.

7.3 Betriebsart 3: FIFO

Eine Muller-Pipeline besitzt unter anderem die interessante Eigenschaft eines Schieberegisters (FIFO), wenn sie mit Daten gefüllt ist. Dieses besondere Schieberegister arbeitet jedoch ohne Taktsignal, denn es erzeugt sich seine asynchronen Steuersignale selbst!

Das bedeutet, die in der Pipeline gespeicherten Daten können am Ende per Handshake-Verfahren entnommen werden. Dabei rücken die in der Pipeline verbleibenden Daten automatisch um eine Pipelinestufe nach, solange bis die Pipeline leer ist oder solange neue Werte am Eingang der Pipeline nachgeschoben werden.

Abbildung 11: Die Muller-Pipeline als FIFO



Im linken Diagramm der Abbildung 11 wird die Muller-Pipeline gefüllt. Dazu muss der Signalkreislauf am Ende der Pipeline geöffnet werden. Anschließend werden permanent neue Request-Signale erzeugt, solange bis die Pipeline gefüllt ist. Die beiden notwendigen Steuersignale sind im Diagramm links oben markiert.

In gefülltem Zustand werden keine weiteren Request-Signale quittiert, wie die Markierung oben rechts zeigt. Während das Request-Signal innerhalb der Markierung ansteigt, bleibt das Acknowledge-Signal auf log.0 liege. Wie die gespeicherten Werte im unteren Bereich des Diagramms zeigen, wurden als Folge der Verkopplung von Ein- und Ausgangssignalen nur zwei unterschiedliche Werte aufgenommen.

Im rechten Diagramm ist die selbstgesteuerte Entleerung der Muller-Pipeline zu sehen, nachdem der Signalkreislauf am Ende der Pipeline wieder geschlossen und die Erzeugung der Request-Signale abgeschaltet wurde.

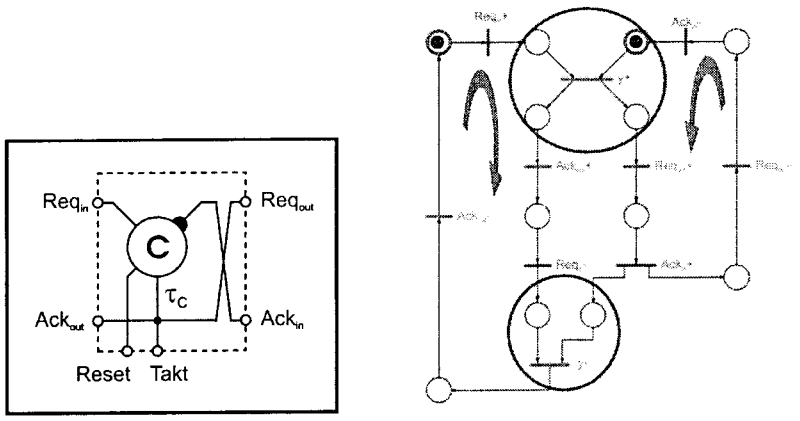
8 Petri-Netz des einfachen HSE

Die Petri-Netz-Repräsentation wurde eingeführt, weil so das dynamische Verhalten im Innern der Handshake-Elemente gut dargestellt werden kann. Die Verkopplung bzw. die teilweise Entkopplung der Ein- und Ausgangssignale sind auf diesem hohen Abstraktionsniveau besonders gut darstellbar.

Kurze Erläuterung zu Petri-Netzen:

- In einem Petri-Netz wandern eine oder mehrere Marken
- Eine Marke wandert von einem Platz zum nächsten. Dazu müssen die Transitionen zwischen den Plätzen schalten oder feuern.
- Wenn eine Transition schaltet, wird/werden die Marke(n) dem Vorplatz der Transition entnommen und auf den nächsten Platz gelegt.
- Eine Transition kann nur dann schalten, wenn alle Vorplätze besetzt sind.

Abbildung 12: Das Petri-Netz des einfachen Handshake-Elements

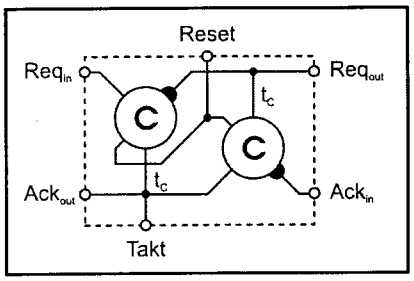


Die Verkopplung der Ein- und Ausgangssignale wird an den markierten Transitionen Petri-Netzes deutlich. Es müssen in beiden Fällen jeweils beide Vorplätze der Transitionen des besetzt sein, damit die betreffende Transition schalten kann. Somit kann die Marke im Eingangskreislauf nur dann einmal vollständig umlaufen, wenn die Marke im Ausgangskreislauf zu bestimmten Zeitpunkten bestimmte Vorplätze besetzt.

9 Das teilweise entkoppelte Handshake-Element

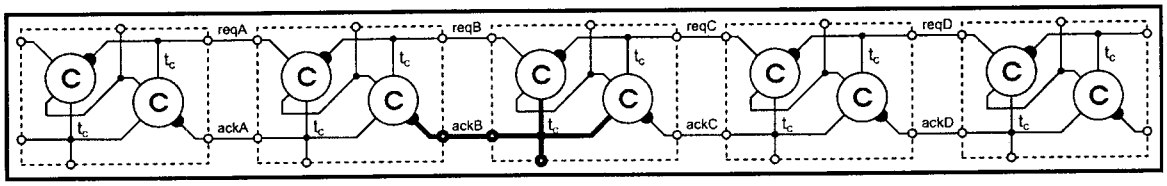
Die simulierte Muller-Pipeline Variante mit einem einfachen Handshake-Element hat, wie schon erläutert, den entscheidenden Nachteil, dass immer nur jedes zweite Datenpfad-Register einen neuen Wert speichern kann. Die folgende Variante, in Abbildung 13 dargestellt, besitzt diesen Nachteil nicht.

Abbildung 13: Das teilweise entkoppelte Handshake-Element



Durch die Verwendung von zwei C-Elementen und zwei Invertern pro Handshake-Element wird eine teilweise Entkopplung der Ein- und Ausgangs-Signale des Handshake-Elements erreicht. Das zweite C-Element sorgt im Prinzip dafür, das bei der Abwicklung des Handshake-Protokolls das nachfolgende Handshake-Element unbeteiligt bleibt, wie in Abbildung 14 durch die fett gedruckte Signalleitung angedeutet wird.

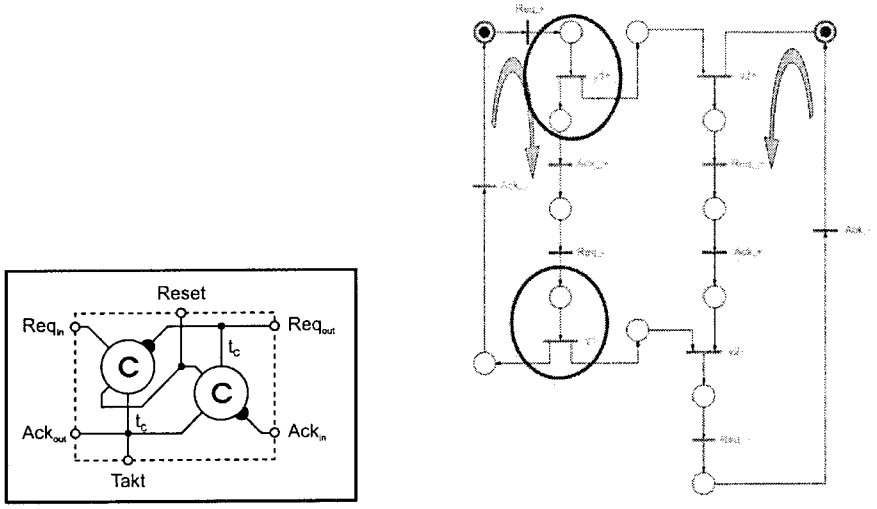
Abbildung 14: Die teilweise entkoppelte Muller-Pipeline



9.1 Petri-Netz des entkoppelten Handshake-Elements

Im Gegensatz zum Petri-Netz des einfachen Handshake-Elements besitzen die markierten Transitionen in Abbildung 15 nur jeweils einen Vorplatz. Das bedeutet, die Marke im Eingangskreislauf kann einmal bis zum Ausgangspunkt umlaufen, ohne von der gegenwärtigen Position der Marke im Ausgangskreislauf beeinflusst zu werden. Das dynamische Verhalten der Eingangssignale ist somit entkoppelt von dem der Ausgangssignale.

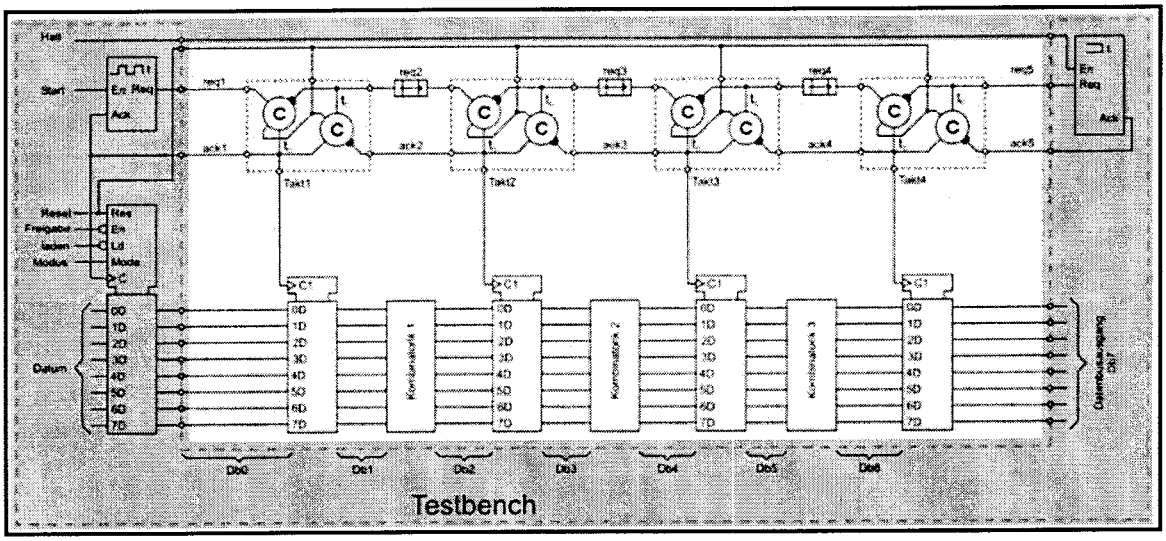
Abbildung 15: Das Petri-Netz des teilweise entkoppelten Handshake-Elements



10 Die teilweise entkoppelte Muller-Pipeline

Wie Abbildung 16 zeigt, ist der Aufbau der Testbench derselbe wie bei der vorherigen Version der Muller-Pipeline. Der folgende Abschnitt zeigt dieselben Simulationsläufe mit der verbesserten Variante der Muller-Pipelinie.

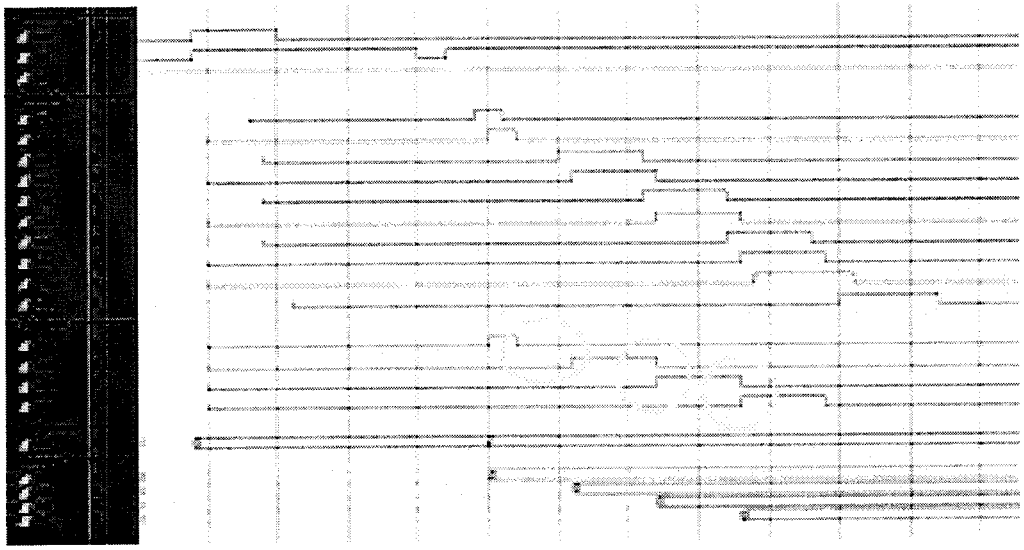
Abbildung 16: Die Testbench der teilweise entkoppelten Muller-Pipeline



10.1 Betriebsart 1: Einzelimpuls

Die markierten Stellen im Diagramm zeigen, dass die asynchronen Steuersignale sich nun nicht mehr überlappen. Demzufolge und nach vorherigen Erkenntnissen, kann somit jede einzelne Stufe der Muller-Pipeline einen neuen Wert speichern.

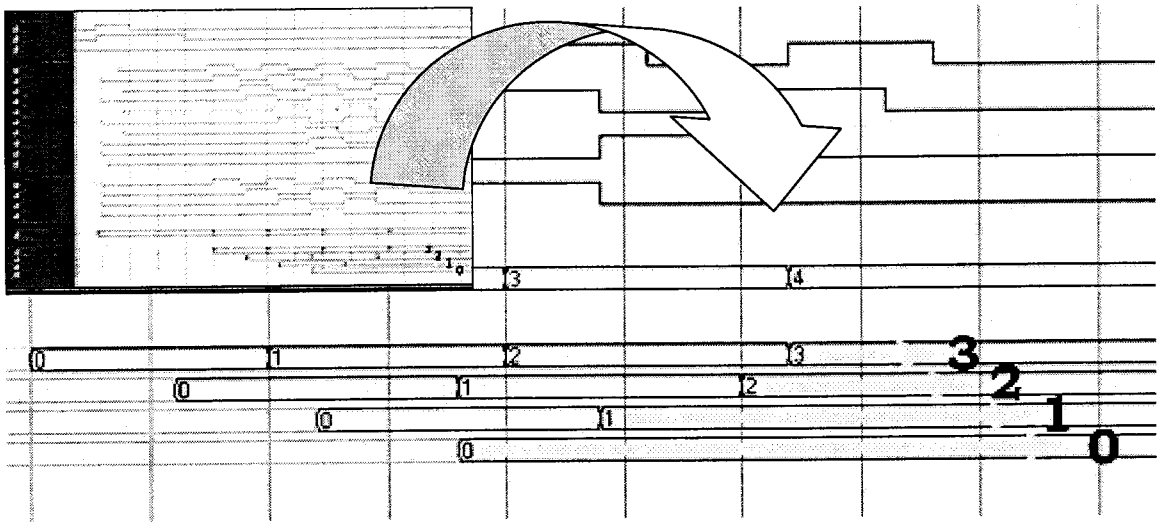
Abbildung 17: Einzelimpulsbetrieb der teilweise entkoppelten Muller-Pipeline



10.2 Betriebsart 3: Die Muller-Pipeline als FIFO : FIFO füllen

In dieser Betriebsart lässt sich die Wirkung der teilweisen Entkopplung der Ein- und Ausgangssignale am deutlichsten erkennen. Sowohl beim Füllen als auch beim Leeren der Muller-Pipeline ist anhand der gespeicherten Werte im Datenpfad die Wirkung zu erkennen.

Abbildung 18: Der verdoppelte Datendurchsatz der Muller-Pipeline

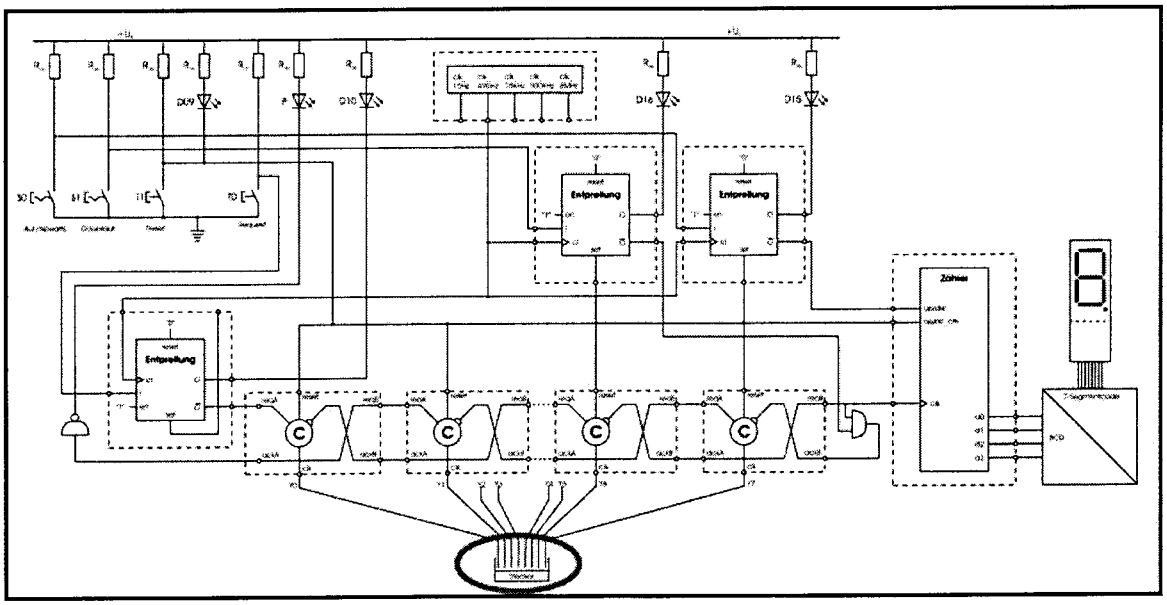


Nach dem die Muller-Pipeline gefüllt ist und sich alle Signale in einem stabilen Zustand befinden, enthält die Muller-Pipeline der einfachen Variante in 4 Stufen genau zwei verschieden Werte, wie in Abbildung 11 zu erkennen ist. Die Variante mit den teilweise entkoppelten Handshake-Elementen enthält dagegen, wie in Abbildung 18 zu sehen, ist die doppelte Anzahl an Werten innerhalb der 4 Stufen. Also besitzt dies Muller-Pipeline auch den doppelten Datendurchsatz.

11 Die implementierte Muller-Pipeline

Nun soll gezeigt werden, dass die erläuterten Eigenschaften einer Muller-Pipeline der Realität entsprechen. Die VHDL-Beschreibung der Muller-Pipeline wird zu diesem Zweck weiterverarbeitet mit dem Entwurfswerkzeug ISE 4.2 von Xilinx. Mit dem Entwurfswerkzeug für synchrone Schaltungen wird die beschriebene Muller-Pipeline in dem FPGA XC4010E von Xilinx implementiert. Die Implementierung einer asynchronen Schaltung in ein FPGA bringt zwar Probleme mit sich, hat jedoch den unbestrittenen Vorteil eines FPGAs, eine relativ kurze Entwicklungszeit zu benötigen.

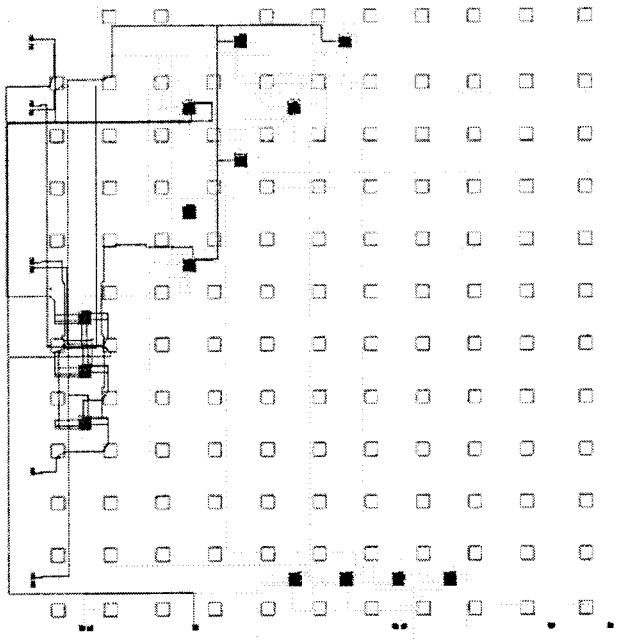
Abbildung 19: Die implementierte Testbench der Muller-Pipeline



Die Markierung in Abbildung 19 verweist auf einen Steckverbinder, über den die asynchronen Steuersignale der Muller-Pipeline aus dem FPGA herausgeführt werden können. Mit einem Flachbandkabel können die Signale in einem Logik-Analysator gemessen werden.

Nachfolgende Abbildung 20 zeigt das automatisch erzeugte Layout der gesamten Schaltung.

Abbildung 20: Das Layout der implementierten Schaltung



Die dunkel gefärbten Komponenten und Netze repräsentieren die eigentliche Muller-Pipeline, die hell gefärbten Bereiche stellen die Testbench dar, also Entprellschaltungen, BCD-zu-7-Segment Dekoder, Zähler und das Und-Gatter am Ende der Muller-Pipeline. Der onchip-Oszillator des FPGAs wird in dem Entwurf lediglich zu Entprellzwecken benötigt.

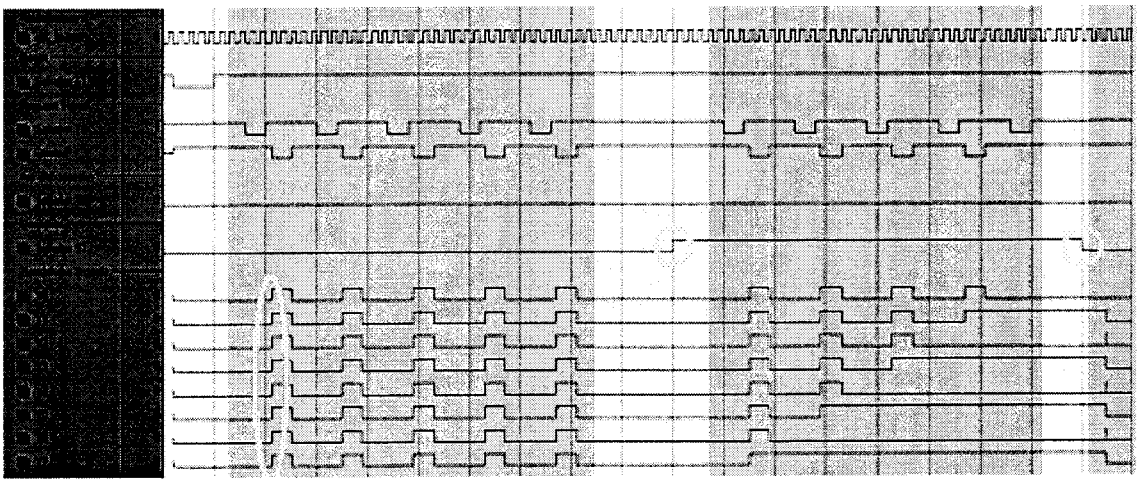
Das interessante an diesem Layout ist die Tatsache, das es mit einem Werkzeug automatisch erstellt wurde, das für synchrone Schaltungen entwickelt wurde. Es wurden also keinerlei asynchrone Aspekte beim Platzieren der Komponenten und bei der Verdrahtung berücksichtigt - die Schaltung arbeitete auf Anhieb korrekt!

Asynchrone Schaltungen lassen sich in verschiedene Kategorien unterteilen. Eine besondere Kategorie nennt sich delay insensitive circuit. Diese Kategorie zeichnet sich durch zwei Merkmale aus:

- Asynchrone Schaltungen der Kategorie „delay insensitive circuit“ arbeiten unabhängig von Signalverzögerungen auf den einzelnen Leitungen
- Die Gatterlaufzeit der Signale durch die einzelnen Komponenten spielen ebenfalls keine Rolle.

Betrachtet man sich nun noch einmal die wahllose Verdrahtung der Muller-Pipeline, erkennt man, dass diese asynchrone Schaltung in der Tat unabhängig von Signallaufzeiten und Gatterlaufzeiten korrekt arbeitet. Aus diesem Grund zählt man die Muller-Pipeline zur Kategorie der **delay-insensitive circuits (DI).**

Abbildung 21: Die Betriebsarten der implementierten Muller-Pipeline



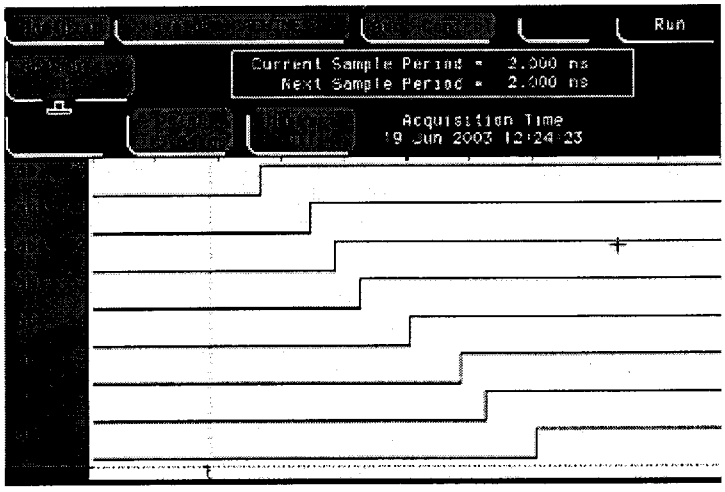
Die Simulationsverläufe, die in Abbildung 21 dargestellt sind, wurden mit dem Post Place & Route Simulation Model erstellt. Die Signalverläufe zeigen Betriebsart 1 und 3.

Mit der implementierten Testbench macht es keinen Sinn, den Dauerlauf zu prüfen, da die asynchronen Steuersignale zu schnell sind, um sie mit den vorhandenen Messinstrumenten zu erfassen.

Die Markierung am linken unteren Rand des Diagramms in Abbildung 21 zeigt die positive Taktflanke eines Request-Impulses und deren Fortsetzung durch die einzelnen Stufen der Muller-Pipeline. Diese Signalflanken wurden mit dem Logik-Analysator gemessen und sind in Abbildung 22 dargestellt.

An den gemessenen Signalverläufen erkennt man, dass der zeitliche Abstand zwischen den Signalflanken unterschiedlich ist. Diese Beobachtung entspricht der unregelmäßigen Verdrahtung der Muller-Pipeline, wie sie im Layout in Abbildung 20 zu sehen ist.

Abbildung 22: Die steigende Flanke eines einzelnen Request-Impulses



12 Zusammenfassung

- Die Muller-Pipeline ist eine asynchrone Schaltung der Kategorie „delay-insensitive circuit".
- Die Schaltung steuert den Datenfluss in einer asynchronen Schaltung.
- Durch Entkopplung der Ein- und Ausgangs-Signale kann der Datendurchsatz deutlich erhöht werden.
- Die asynchrone Schaltung lässt sich problemlos in ein FPGA implementieren.
- Erst bei der Implementierung der Kombinatorik im Datenpfad ergeben sich Probleme durch die Abstimmung der Verzögerungen der Request-Signale auf die zu implementierende Kombinatorik.
- Es existieren Entwürfe für die Struktur eines FPGAs, das sich für den Entwurf von asynchrone Schaltungen eignet.
- Der Entwurf von FPGA-Strukturen für asynchrone Schaltungen setzt eine Formalisierung der asynchronen Struktur voraus.

13 Literaturhinweis

[1] Stephen B. Furber and Paul Day, "Four-Phase Micropipeline Latch Control Circuits"

[2] Jens Sparso, Steve Furber, "Principles of Asynchronous Circuit Design", 2001

[3] Dr.-Ing. Lothar Quäck, "Petri-Netze in der Steuerungstechnik", 1988

[5] Kenneth Y. Yun, Peter A. Beerel, Julio Arceo, "High-Performance Asynchronous Pipeline Circuits"

[6] Ivan E. Sutherland, Jo Ebergen, "Computers without Clocks", 2002

Compiler controlled hardware synthesis for resource aware implementations of microprocessors

Bernhard Hoppe, Norbert Lugowski, Andreas Fallberg, Eduard Steindl und Hermann Meuth

FB Elektrotechnik und Informationstechnik, Forschungsgruppe ?SYST,
FH Darmstadt - University of Applied Science, Schöfferstr. 3, D-64295 Darmstadt
Tel.: +49 6151 168 322, FAX: +49 6151 168 934
e-mail: hoppe@fh-darmstadt.de

Abstract

Reconfigurable Computing faces two challenges: first to achieve drastic cost or performance benefits over conventional computers and second, solving the involved necessary task of the co-development of hard- and software. Both aspects are investigated using a hardware implementation of the JAVA virtual machine (JVM). It is shown that the synthesis of software specific hardware maybe controlled by data achieved during compilation of the JAVA bytecode to the appropriate machine instructions of the hardware core. Logic synthesis using this control information provides readily a design of an optimised computer, ready for layout generation or downloading to a particular FPGA device. Hardware and power savings achieved with this approach will be shown to be up to 60%.

Introduction

In this paper we will investigate resource aware microprocessor designs using reconfigurable computing [1,2] for JAVA applications. In particular we will investigate the significant area and power improvements for the required computer hardware may be achieved, if application (i.e. program specific) instructions sets are used instead of microprocessors with a fixed instruction set. We will point out, how the software/hardware programming problem, i.e. the great challenge of reconfigurable computers, may be facilitated by introducing a clear interface between software development and hardware implementation. To generate a processor with an specific instruction set, we will use a compilation step in which not only high level language constructs are translated into assembly language or into machine code respectively, but also control information for hardware synthesis is produced. We will exemplify this approach using a JAVA - processor, i.e. a hardware version of the JAVA virtual machine. The synthesis of hardware may be masked to a great extend from the view of the software developers, so that they are facing a fixed programmers model. The hardware design task maybe automated by scripts for the synthesis tool and the place and route software for a given FPGA-family.

1 From JAVA to a minimum set of machine instructions

JAVA gains more and more attraction for embedded control (real time) applications, which have to economize on computing resources, I/O-connections and memory to achieve cost, space and power efficient systems [3]. To achieve these goals a number of microprocessor cores (IP- blocks) were developed or are under development [4-6] for direct JAVA bytecode execution. These solutions achieve full compatibility with the JAVA language standard by implementing sufficient machine instructions and by using a software layer, in order to support some of the more complex JAVA - Bytecode instructions. Though such dedicated designs speed up the execution of JAVA code significantly, the main drawback of these components is their comparable huge gate need and hence their lacking power awareness. This is caused by the complexity of the JVM instruction set.

Our approach for designing a cost-effective JAVA - core is to utilize the software part in a more extended way, such that a customized hardware platform is generated for each specific JAVA application. Recent JAVA - cores run any JAVA program with the same hardware. If one analyses, how a core executes a given program, it may be noted that in general only a subset of the hardware functions are exercised by the given program. This overhead in hardware and hence in power consumption may be eliminated if an application specific hardware is used instead of a general purpose version. That means that for each new JAVA program to be exe cuted a new configuration of hardware blocks has to be synthesized, which is easily implemented in existing board designs using in-circuit programmable FPGAs.

1.1 The attoJAVA core design

In order to exploit the full capabilities of software specific core design, the portfolio from which different hardware-versions may be derived has also to be selected very carefully. A new hardware implementation of the JAVA virtual machine (attoJAVA core) using synthesizable VHDL was developed, in which the selection of hardware-blocks or the configuration of the in struction set may be controlled by VHDL generics. The attoJAVA supports with hardware a reduced, yet most frequently used bytecode instruction set. For JAVA compatibility

the remaining instructions may be mapped before run time onto the attoJAVA subset by a specifically developed compiler. In total, this 32-Bit stack microprocessor core supports 87 of the 226 standard JVM bytecode instructions directly by hardware including a floating point unit (FPU, IEEE- 754 Std.), and extended codes to facilitate memory or register access, subroutine calls etc. An Instruction Prefetch Queue was included, as for the picoJAVA cores, in order to speed up the loading of instructions. A Memory Interface Unit manages the data transfer between on-chip and off-chip memories. The stack is implemented with a dual port RAM for fast read/write cycles. The RAM is available on XILINX Virtex components or may be implemented via a layout-generator in any semicustom design flow, e.g. in the AMS CSD CMOS technology, available through EUROPRACTICE.

The bytecode-instructions implemented in the core were selected by their suitability i.e. hardware simplicity and by the effort required, to compile a given bytecode instruction by basic attoJAVA-instructions. Aside from the FPU, two versions for the ALU with fast (1 clock cycle operations) and slow (sequential) arithmetic operations are available.

1.2 Backend Compiler and program specific computer hardware

A special compiler (the backend compiler BC) was conceived to assemble, prior to runtime, standard JAVA bytecode in terms of the reduced attoJAVA instruction set. In order to control the application specific synthesis of computer hardware, an automatic analysis of the generated machine instructions is performed during the compilation. During this analysis all instructions used for a given application are noted and written out as table of labels in the format of a VHDL package. This package has to be loaded together with attoJAVA-VHDL-model to start the logic synthesis of an application specific implementation of the attoJAVA-core (Fig. 1). Unnecessary instructions of the underlying attoJAVA processor design are hence discarded, which reduces the complexity of the hardware design. The fast or slow ALU has to be selected by the user on the basis of the system specification. The length of the instruction prefetch queue and the size of the stack is adjusted by the system. The compiler calculates appropriate sizes on the basis of the number of instructions used, also as VHDL generics.

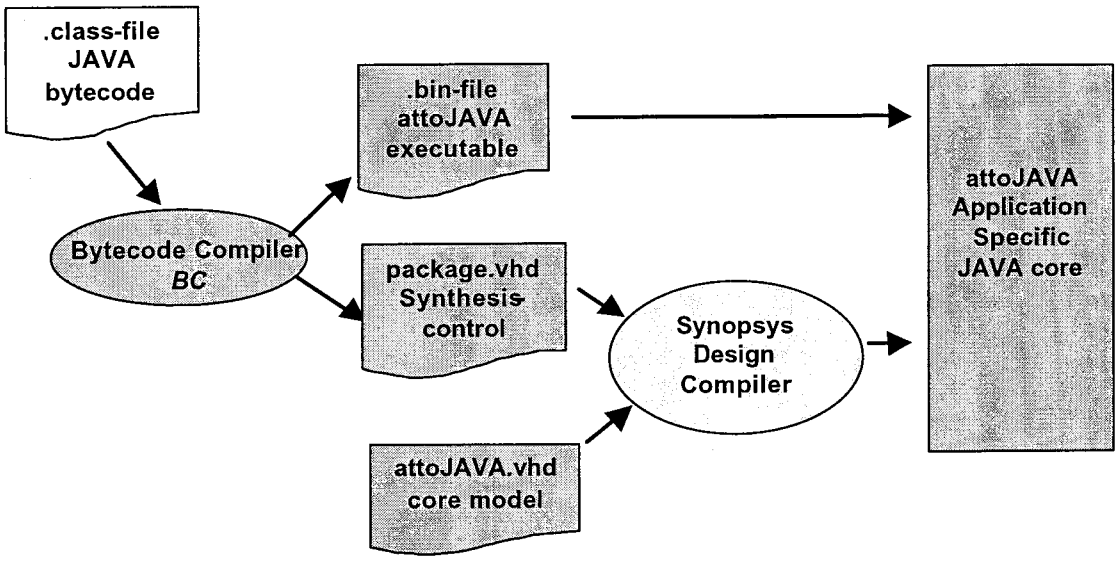


Figure 1: Flow for generation of a optimised hardware platform for executing a specific JAVA application. The attoJAVA components are shaded in grey. The class-file is the output of a standard JAVA compiler for the JVM. The Synopsys Design Compiler Tool is utilized for logic synthesis.

Note that in contrast to other solutions, the hardware generated is specific for a given JAVA- application. This is no drawback for applications, where the software to be executed is fixed, or changed only from time to time, a scenario resembling traditional microcontroller-systems using mask-programmed program ROMs. If the software has to be changed more often, then a system using in-circuit programmable FPGAs is appropriate. Any new software implies a new set of hardware elements to be implemented in the programmable logic. This maybe done by downloading a new JDEC file for reprogramming the FPGA generated by the place and route tool after logic synthesis of the appropriate version of the attoJAVA core.

1.3 JVM compatibility

There are some minor limitations with regard to full JAVA compatibility of this system: e. g. the compiler is not supporting complex inheritance- concepts of JAVA and allows only for fixed (multiple) array sizes. These limitations are mostly implied by the ahead of runtime compilation, which however is essential for real time control applications.

2. Instruction set efficiency

If one analyses the BC-compiled JAVA bytecode for different benchmark applications (Table 1) one notes that even for complex programs like the standard Dhrystone-benchmark [8] only 51 out of the 87 available instructtions are utilized, which is maximum for all programs tested so far. The minimum number of instructions is 21, occurring for Fibonacci-number calculation. On the average around 50 % of the instructions are needed. It is interesting to note that 11 instructions were used in each application and 26 instructions were not used in any of the JAVA benchmarks investigated so far. These bytecode instructions however are part of the JVM specification and hence will appear in other JAVA applications.

Integer algorithms	Fibonacci numbers (fib), bubble sort (bub), shell sort (she), prime numbers (era), Dhrystone (dry), digital filter (fil), object-oriented methods (inh)
Floating point algorithms	addition and subtraction (AS), multiplication (M), division (D), Square Root (sqr), bubble sort (bsf)

Table 1: Benchmark applications investigated. Dhrystone originally written in C was translated into JAVA.

The huge number of unused instructions implies, that there is optimization potential for the hardware part of the system. This potential may be exploited in two ways: either one trades in hardware instructions into software modules for the compiler in order to obtain an optimised general purpose system or one compiles a given application onto a dedicated and minimized application specific hardware, which implements only the required instructions and hence is then optimum in gate count and power resources. The second approach seems to be more promising, since one or the other instruction occurring rarely or never in the benchmarks investigated, may be critical for the performance of other JAVA-application not considered yet.

3. Core area and power consumption

In order to investigate the benefits of using an application specific instruction set, hardware was synthesized using 0.35µm CMOS standard cells as a target technology for different core architectures and benchmark applications (Fig. 2). In the figure results for area minimization with Synopsys Design Compiler are shown. There were no delay constraints set and operating conditions were typical commercial.

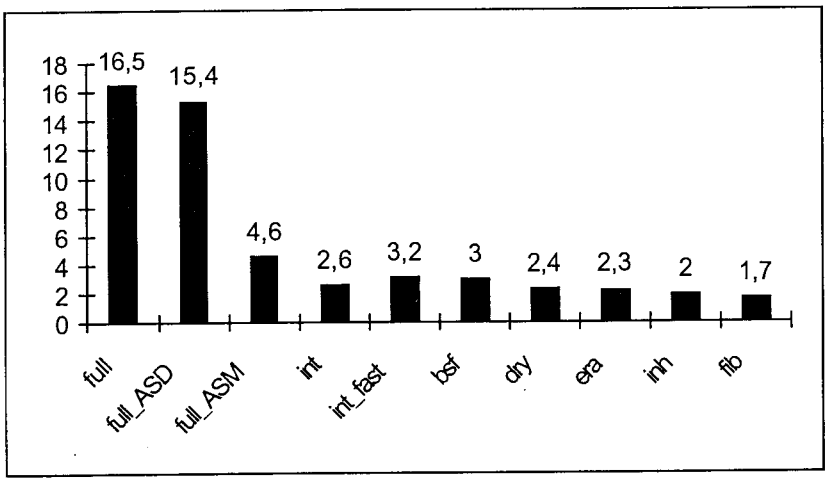


Figure 2: Core areas in mm²: From left to right: complete core with FPU (full), full core with fpu without multiplication (ASD) or division (ASM) respectively, core without fpu and slow ALU (int) or with fast ALU (int_fast). bsf to fib are representative application specific core versions. The labelling follows Table 1.

3.1 General Purpose hardware

The first five columns from left to right show implementations of the complete attoJAVA instruction set, with different degrees of hardware support for arithmetic operations: the full version including the FPU (full_fpu), versions with a reduced floating point instruction set and two versions with integer arithmetic units only. One is a fast version (int_fast) with one-cycle integer multiplication and division, whereas the other uses serial implementations of arithmetic operators (int). One may chose any of these versions as a general purpose hardware platform and use the corresponding version of the BC for hardware with all and without (some of the) floating point operations. The span of die sizes is from 2,6 mm² (int) to 16.5 mm² (full_fpu). However small cores without complex floating point instructions are significantly slower in core performance, if applications are considered which require the corresponding mathematical operations.

3.2 Applications specific hardware

If one chooses an application specific hardware platform, area maybe reduced without any performance losses: note that the floating point version of the bubble sort algorithm requires 3 mm² only, whereas one search stage is performed in one clock cycle! Significant area improvement maybe found also for applications with integer data. As compared with the fast integer core all algorithms maybe executed with same speed but with 20% (Dhrystone) to 50% (Fibonacci) less die size.

If one synthesizes algorithm specific cores under timing constraints (here 10 ns) the above findings are observed in a qualitatively similar fashion but even a more pronounced manner, see Figure 3. The Fibonnacci version, as the smallest application specific core, requires 59 % of the full integer core for the constrained area optimisation, whereas in the unconstrained case the corresponding percentage in area is 67%.

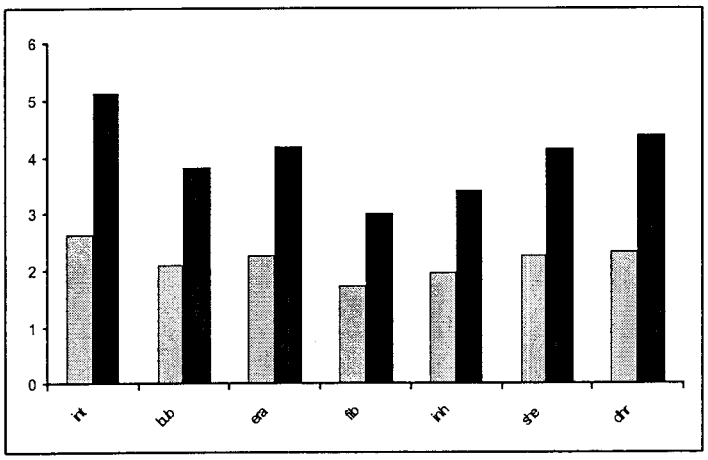


Figure 3: Comparison of core area for pure area optimisation (shaded light grey) and optimisation for a 10 ns timing constraint (dark grey). Optimised was the int-version of the core and cores with specific instruction sets for integer benchmarks applications.

3.3 Power reductions

The most important result is the significant drop of power consumption if one uses cores with application specific hardware (see Fig. 4). The range of values is 1549 mW for the full core, over 1012 mW for the full core without an FPU down to 399 mW for the core version specific for the inh- application. The power reduction of the floating point version of the bubble sort algorithm requires only 37% of the power consumption of the complete core including the FPU. The program exercising object oriented methods (inh) needs only 39% of the complete core with integer ALU only.

Target technology for testing systems performance was the 0.35 µm CMOS CSD ASIC technology. Some comparisons with XILINX Virtex FPGAs show, that the results for different realizations of the core/compiler-system obtained for the ASIC-technology may be transferred directly to FPGA -designs.

4. Conclusions

It was investigated how application specific instruction sets may be employed to improve the area and power consumption of microprocessor hardware. Moreover a methodology for generating application specific computer hardware with minimized instruction sets was presented, in which control information for core hardware synthesis is provided as a byproduct of the compilation of the code to the machine level. These investigations were performed for a dedicated processor for JAVA applications but the results maybe extended to other processors with complex instructions sets (CISC). Hardware implementations of the JVM are in particular excel-

lent candidates for this compiler controlled hardware design, since any hardware version of the virtual JVM requires some sort of software layer in order to implement complex bytecode instructions.

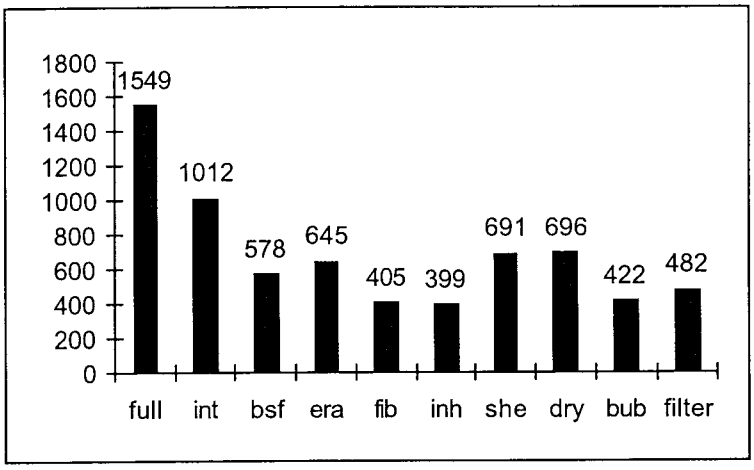


Figure 4: Power consumption in mW for different core implementations.

Area savings of up to 50 % and power reductions by 66 % clearly indicate that this approach for reconfigurable computing is very promising. These findings are in full accordance with the well accepted judgment that reconfigurable computers offer significant advantages over conventional processors. The known general drawback of reconfigurable systems, namely that their programming maybe cumbersome, since software developers also have to assume the role of hardware designers, is moderated by our approach. The compiler controlled synthesis flow is easily to be automated by appropriate scripting techniques available for up-to date design tools.

For practical applications of the new approach for using reconfigurable hardware it is noteworthy that there are many embedded applications, in which the software to be executed is not changed very often. In automotive applications for instance, the executed code, for instance for dash-board instrumentation or motor-management are not changed over the life - time of the automobile, since software changes in the field are hard to manage and affect safety considerations. Therefore a fixed minimized instruction set is no drawback, as its inflexibility resembles that in many embedded systems of the discussed kind mask- defined program- ROMs store the binary version of software code. In both cases a new software requires a new mask set and a new production run in the semiconductor manufacturing site. For FPGA based systems with in-circuit programmable devices a new software i.e. a new instruction set and hardware architecture may be readily downloaded after synthesis, without any changes of the printed circuit board. In these cases we will not exploit the reduced hardware cost, but we will benefit from the high power efficiency of application specific instruction set JAVA computers.

References

[1] A. DeHon and J. Wawrzynek: "Reconfigurable Computing: What, why and implications for Design Automation", in Proc. of the Design Automation Conference, New Orleans, June 1999, pp. 610 – 615

[2] R. Gonzales: "Xtensa: A configurable and extensible processor", IEEE Micro, vol. 20., pp. 60 – 70, Mar. 2000

[3] C. D. Locke and P. C. Dibble: "JAVA Technology comes to Real-Time Applications", Proc. of the IEEE, vol. 91, No. 7, pp. 1105-1013, Jul. 2003

[4] J. Michael O'Connor und Marc Tremblay: „PicoJAVA-I: The JAVA Virtual Machine in Hardware", IEEE Computer Magazine, S. 45 ff., March/April 1997

[5] PicoJAVAII, http://www.dacafe.com/DACafe/EDATools/picoJAVA-II

[6] Vulcan Machines' Moon Processor, An implementation of the JAVA Virtual Machine, http://www.vulcanmachines.com

[7] T. Lindholm und F. Yellin: „The Java™ Virtual Machine Specification", 1st Edition. Addison-Wesley Longman, Inc., 1996

[8] PDS: The performance database server dhrystone, http://www.netlib.org/performance/html

Implementation of Signal Processing Algorithms Using FPGAs

Ruben Bartholomä, Frank Kesel
University of Applied Sciences Pforzheim
Tiefenbronner Str. 65, D-75175 Pforzheim, Germany
Email: kesel@fh-pforzheim.de, Tel.: ++49/7231/286567

Abstract: *In this contribution we will show how high-level synthesis can be applied to solve the problem of mapping signal processing algorithms on FPGA hardware. The algorithms will be described in C++ and mapped onto a Xilinx FPGA using a high-level synthesis tool.*

1 Introduction

Today digital signal processing applications (DSP), like e.g. telecommunications or image processing, need considerable computing performance in order to work in real time. In some cases the performance of DSPs (Digital Signal Processor) is not sufficient to guarantee real time behavior. In these cases the problem is often solved by using dedicated hardware to do some pre-processing of the data or even implement the whole signal processing system. Since hard-wired ASICs (Application-Specific Integrated Circuits) are too expensive and not flexibel enough, FPGAs (Field-Programmable Gate-Arrays) have proven to be a viable alternative for implementing DSP algorithms in hardware in the last decade. FPGAs belong to the class of programmable logic devices (PLD) and are reprogrammable or reconfigurable. The reason why hardware implementations of DSP algorithms show a superior performance in many cases compared to a software implementation on a DSP is a different computing paradigm. Compared to a DSP, which implements basically the sequential "von Neumann" computing principle (computing-in-time), FPGAs can implement algorithms with much stronger parallelism, since the hardware is inherently parallel (computing-in-space). Unfortunately the design of FPGA hardware needs normally more effort than a software implementation on a DSP. Furthermore the development tools like hardware description languages (HDL, e.g. VHDL), simulators and synthesis and the way in which the implementation is described differs much from what the system or software engineers are used to, like e.g. describing the problem in tools like Matlab [6] or in languages like C/C++. On the other side tools have been developed in the last decade to describe hardware on a higher abstraction level, the so-called algorithm level, see e.g. [1], and to automatically map the algorithmic description onto a register-transfer-level-model and finally to a gate-level-implementation of an ASIC- or PLD-library. A commercial tool which implements this so-called "high-level-synthesis" (HLS) or "behavioral synthesis", see e.g. [2] or [3], is the "CoCentric SystemC Compiler" from Synopsys [10]. It accepts algorithmic descriptions in VHDL [4] or C++/SystemC [5]. According to the constraints given by the designer it tries to find an implementation for the given algorithmic description. In this contribution we will show how HLS can be used to implement DSP algorithms on FPGAs and we will evaluate the results with respect to computing performance and resource usage. In section 2 we will show how HLS basically works. Section 3 describes the FIR (Finite-Impulse Response) filter structures used for the evaluation. Section 4 presents the results and compares the HLS results with results from Matlab/Simulink [6].

2 High-Level-Synthesis

Digital hardware design today usually starts with a description of the circuit on register-transfer-level (RTL), using languages like VHDL. The RTL description models the circuit in terms of its registers and the combinational transfer functions, which are described with control structures similar to high-level programming languages, like "if", "case" or "for". The RTL-model is bit-correct and clock cycle accurate. An algorithmic description, according to the Gajski-Y-diagram [1], describes the circuit one abstraction level above the RTL. In this case the model is still bit-correct but not clock cycle accurate. This gives the freedom to implement the description with a variable number of clock cycles, which in turn enables the time-sharing of resources, like adders or multipliers. Therefore by using HLS one can trade-off the number of clock cycles (performance) against the number of adders or multipliers (resources) for different RTL-implementations of the same algorithmic description. This is not possible by using standard RTL synthesis tools.

This can be shown by the simple example in figure 1. Assuming that the variables are implemented with 16 bit, we need an internal bit width of 32 bit. If we synthesize this code with a standard RTL synthesis tool, like e.g. "Leonardo" from Mentor [7], for an Altera Flex10K20 FPGA [8], it will consume 95% of the resources, since we need 2 adders, 1 subtractor and 2 multipliers. The resulting logic is fully combinational, i.e. no registers are used.

```
int design(int a, int b, int c, int d, int e, int f)
{
        int y;

        y = a * b + c + ( d - e ) * f;

        return y;
}
```

Figure 1: Algorithm example

In contrast to standard RTL synthesis HLS can transform this description also into a sequential implementation which uses 1 adder/subtractor and 1 multiplier. Registers and a finite state machine are generated such that the circuit needs 3 clock cycles to calculate the result. We will describe in the following briefly how HLS achieves this transformation. In the first step a data flow graph (DFG) is extracted from the algorithmic description, which models the data dependencies. Each node represents an operation and two nodes are connected by an edge if there is a data dependency. Control structures like "if" can be represented in an extension, the Control DFG (CDFG) see [3]. In the following phase ("resource allocation") the type and number of resources or (arithmetic) components will be determined. First one needs the library for the ASIC or FPGA device and second the designer has to define the number of components to be used. This step basically determines if we will have a more parallel (more resources, faster) or a more sequential implementation (less resources, slower). The following "scheduling" assigns the operations to clock cycles with respect to the DFG dependencies and results in a scheduled DFG according to figure 2. Scheduling is the basic technology for HLS and is usually implemented by heuristic algorithms, see e.g. [3]. If an edge crosses a clock cycle boundary then a register is needed to hold the (intermediate) result ("register allocation"). Lifetime analysis tries to minimize the number of necessary registers by sharing them in different clock cycles, e.g. v1 and v4 in figure 2 could use the same register. Next the operations will be assigned to components ("binding"), which determines the number of necessary multiplexers, and finally a finite state machine (FSM) is constructed, which is necessary to control the datapath. Our simple example does not contain loops, like e.g. in the description of an FIR filter. In standard RTL synthesis loops in the code are always unrolled. This can result in considerable hardware, since it is a fully parallel solution. Therefore loops are normally used with care in RTL design. Since HLS constructs a FSM or sequencer it is possible to not unroll (fully sequential) or only partially unroll the loops.

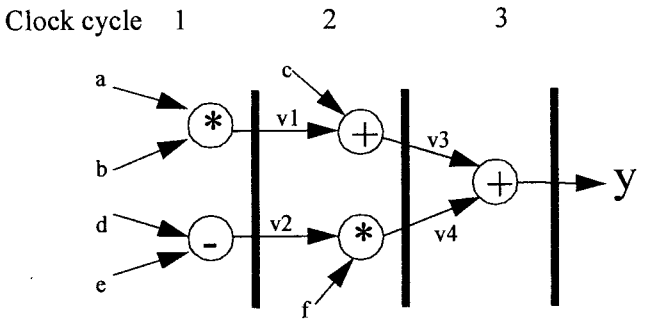


Figure 2: DFG with scheduling informations

3 FIR Filter Structures

FIR filters are one of the most important operations in digital signal processing systems. The output signal y of a FIR filter can be determined by the convolution of the input signal x with the impulse response h of the filter, as shown in equation (1). N denotes the number of taps used in the filter.

$$y(n) = \sum_{i=0}^{N-1} h(i) \cdot x(n-i) \tag{1}$$

The signal flow graph for the direct form of an FIR filter is shown in figure 3(a). It can be directly constructed from equation (1). The signal flow graph of the transposed direct form, as shown in figure 3(b), can be obtained by applying the transposition theorem to the signal flow graph of the direct form [9]. A fully parallel hardware implementation of both filter structures requires N multipliers, $N-1$ adders and $N-1$ registers. Due to the fact, that the direct form of the FIR filter merely needs to hold samples of the input signal in registers, there are only $(N-1)\cdot W_x$ flipflops necessary. W_x specifies the wordlength of the input samples x. However, the transposed direct form of the FIR filter requires more flipflops, because the registers hold the results of previous sums of products. An addition of two products may produce a carry bit, which has to be hold for preserving full precision. An upper bound for the total number of flipflops can be expressed as $(N-1)\cdot(W_x+W_h+\lceil ld(N)\rceil)$, where W_h denotes the wordlength of the filter coefficients h. Considering the critical path, which is the maximum delay between two registers, of both structures, we can see significant advantages of the transposed direct form. Let T be the delay of an adder and $3T$ the delay of a multiplication, then the total delay of the critical path of the transposed direct form is $4T$. In the case of the direct form the total delay of the critical path is $(N+2)\cdot T$, which reflects a linear dependency on the number of taps.

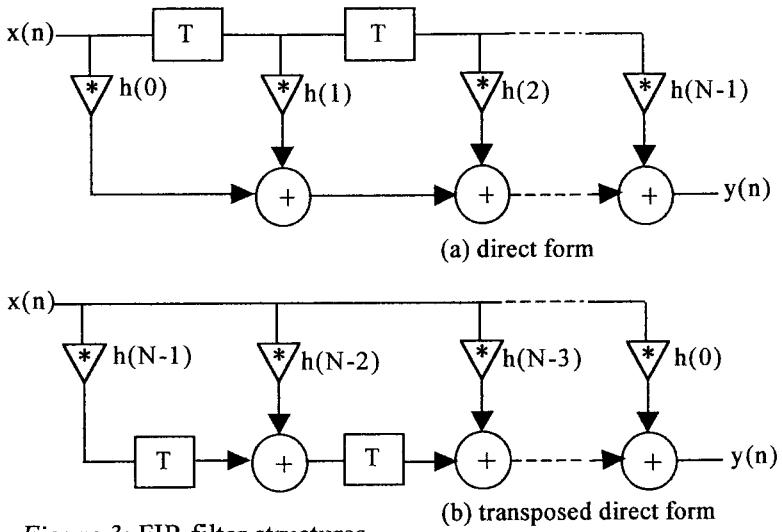


Figure 3: FIR filter structures

4 Results

This section presents the results of our study. We have generated different hardware architectures of a FIR filter, using the HLS tool "CoCentric SystemC Compiler" from Synopsys and targeting a Xilinx FPGA XC2V250 [11]. We have also created the same FIR filters with the "System Generator" from Xilinx, in order to compare the efficiencies of both approaches, according to figure 4, in terms of the tradeoff between resource usage (measured as equivalent gate count) and throughput (measured as maximum sampling rate). The "System Generator" can be used with Matlab/Simulink. It enables the designer to model a system, using Xilinx IP Cores and lower level elements like e.g. multipliers and registers. The system can then be simulated and mapped onto the Xilinx FPGA. It must be noted that the "System Generator" approach does not use HLS, but is based on a library approach using pre-desigined and optimized hardware blocks for a certain FPGA family. Therefore the approach is hardware dependent. The architectures generated with "CoCentric SystemC Compiler" on the other hand stem from one single C++ hardware independent algorithmic description of the filter. Figure 4 depicts the total equivalent gate count depending on the data throughput for different architectures of a 16 tap FIR filter, which has a data and coefficient wordlength of 16 Bit. All architectures make use of the embedded Xilinx Virtex multipliers. The results show that the tradeoff between area and speed can be approximated as a linear function in both cases. The fastest architectures in both approaches implement the tranposed direct structure with 16 multipliers and 15 adders, as mentioned in section 3. Slower architectures share adder and multiplier resources, which require additional logic for implementing control

logic (e.g. multiplexers) and additional registers. Figure 4 shows that there is nearly no difference in both approaches in the case of the fastest solution. In case of the slower solutions the "System Genera-

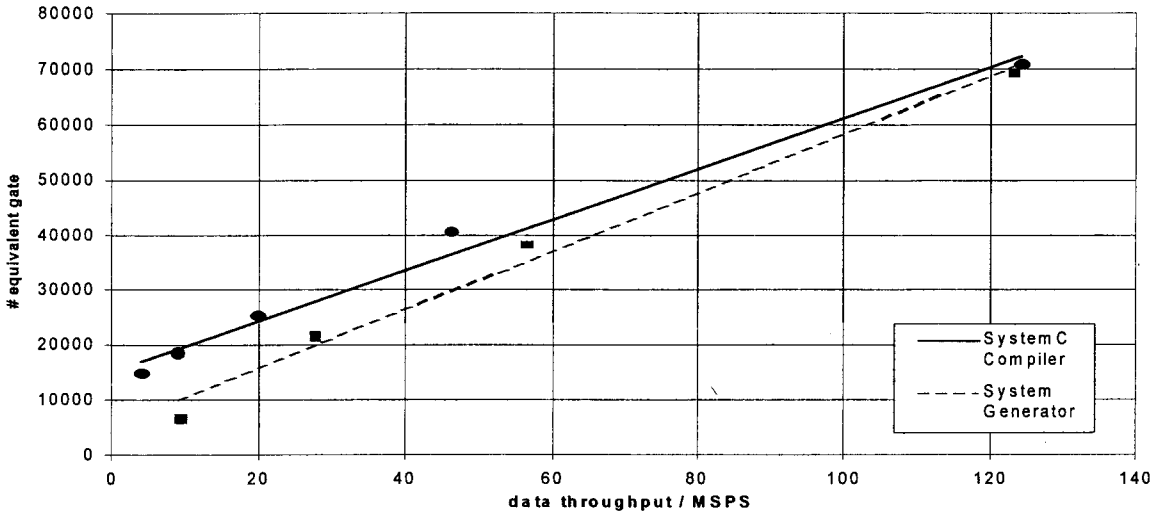


Figure 4: Efficiency of FIR filter implementations

tor" approach shows better efficiency, which can be attributed to a better control logic implementation. Further speedup of the throughput could be achieved in both approaches by pipelining the datapaths, consisting of multipliers and adders.

5 Conclusion

In this paper we have presented how HLS tools can be used for mapping signal processing algorithms onto FPGAs. We have implemented FIR filter architectures, using the HLS tool "CoCentric SystemC Compiler" and compared the results with similar architectures, designed with "System Generator". The comparison shows that both approaches are equally efficient with respect to the fastest or fully parallel solutions. With decreasing parallelism or increasing sequentialism the "System Generator" solutions are more efficient. One advantage of the "System Generator" is its possibility to model hardware architectures within the Matlab environment, which is often used for designing signal processing algorithms. However modelling hardware with the "System Generator" is similar to RTL modelling and thus time consuming and in this case also hardware dependent. In order to have a design path from a Matlab model to SystemC and HLS we are currently working on generating SystemC descriptions from Matlab models.

6 References

[1] Gajski, D.D., Dutt, N.D., Wu, A.C.; High-Level Synthesis: Introduction to Chip and System Design; Kluwer Academic Publishers, 1992
[2] Knapp, D.W.; Behavioral Synthesis - Digital System Design Using the Synopsys Behavioral Compiler; Prentice Hall, 1996
[3] Elliot; J.P.; Understanding Behavioral Synthesis; Kluwer Academic Pub., 2000
[4] Ashenden, P.J.; The Designer's Guide to VHDL; Morgan Kaufmann, 1996
[5] Grötker, T., Liao, S., Martin, G., Swan, S.; System Design with SystemC; Kluwer Academic Publishers, 2002
[6] Matlab-Homepage "www.mathworks.com"
[7] Mentor-Homepage "www.mentor.com"
[8] Altera-Homepage "www.altera.com"
[9] Wanhammar, L.;DSP Integrated Circuits; Academic Press, 1999
[10] Synopsys-Homepage "www.synopsys.com"
[11] Xilinx-Homepage "www.xilinx.com"