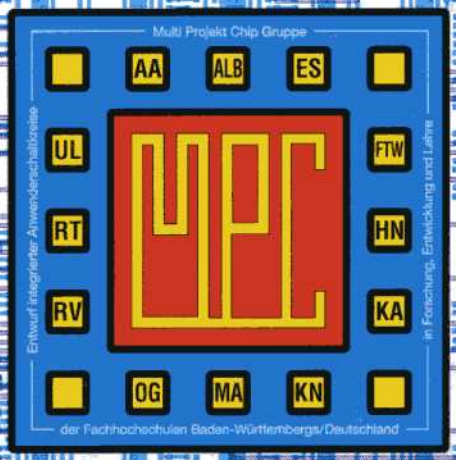
MULTIPROJEKT CHIP-GRUPPE

BADEN - WÜRTTEMBERG

MPC-Workshop Januar 2002

Göppingen



MULTIPROJEKT CHIP-GRUPPE

BADEN - WÜRTTEMBERG

MPC-Workshop Januar 2002

Göppingen

Cooperating Organization
Solid-State Circuits Society Chapter
IEEE Germany Section

Herausgeber: Fachhochschule Ulm

Adressen der

MULTIPROJEKT-CHIP-GRUPPE (MPC-Gruppe)

BADEN - WÜRTTEMBERG

http://www.mpc.belwue.de

Fachhochschule Aalen
Prof. Dr. Kohlhammer, Postfach 1728, 73428 Aalen
Tel.: 07361/576-296, Fax: -324, Email: bernd.kohlhammer@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.: 07131/504-400, Fax: /252-470, Email: clauss@fh-heilbronn.de

Fachhochschule Karlsruhe
Prof. Ritzert, Postfach 2440, 76012 Karlsruhe
Tel.: 0721/925-1512, Fax: -1513, Email: ritzert@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. Klotzbücher, Postfach 1261, 88241 Weingarten
Tel.: 0751/501-630, Fax: /49240, Email: klotzbuecher@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

Gefertigte Bausteine

Entwicklung eines Minimum Shift Keying ASIC zur drahtlosen Datenübertragung im Schmalbandamateurfunk

Volker Typke, Prof. Arnold Führer
Fachhochschule Ulm, Prittwitzstraße 10, 89075 Ulm

Im Rahmen einer Diplomarbeit wurde an der Fachhochschule Ulm ein ASIC entwickelt, der die drahtlose Datenübertragung im Schmalbandamateurfunk in der Betriebsart "Packet Radio" verbessert. Das neu entwickelte Modem erlaubt bei annähernd gleichem Bandbreiteverbrauch eine doppelt so hohe Datenrate im Vergleich zu den bisher verwendeten Modems. Der ASIC ersetzt alle vier bisher benötigten ICs und reduziert auch die externe Beschaltung erheblich. Damit ist es nun möglich, das gesamte Modem im Stecker für die serielle Schnittstelle des PC unterzubringen und so die Kosten zu minimieren.
Wo immer es möglich war, wurden Schaltungsfunktionen in digitaler Form realisiert, um die Zuverlässigkeit zu erhöhen und auf einen Abgleich des Modems verzichten zu können.

1 Motivation

Schon vor der großen Verbreitung des Internet hatte sich im Amateurfunk ein eigenes Verfahren zur Übertragung von Daten etabliert. Dabei handelt es sich um die Betriebsart "Packet Radio", deren Modem-Technologie auf der damals gängigen Technologie zur Übertragung von Daten über das Fernsprechnetz aufbaut. Die damals verfügbaren Modems benutzten einfache AFSK-Verfahren (AFSK = Audio Frequency Shift Keying), die in sogenannten Akustikkopplern zum Einsatz kamen, und die Datensignale in Frequenzen im Sprachband umwandelten. Bis heute hat sich diese Technologie im Amateurfunk gehalten, da sie auf billigen ICs aufbaute und die Technologie einfach zu beherrschen war. Ein Modem für den Schmalbandamateurfunk bestand aus gerade einmal 4 ICs: einem Modem-IC (TCM3105 von Texas Instruments), einer digitalen Rauschsperre bzw. Trägererkennung (DCD = Digital Carrier Detector, realisiert mit einem XR2211 von Exar) sowie einem Pegelwandler für die serielle Schnittstelle eines PC (MAX232 von Maxim) und einem Schmitttrigger (CD4093) für die Realisierung des Watchdogs für die Sendertastung.
Mit der Einführung neuer, besserer Verfahren zur Datenübertragung über das Fernsprechnetz wurde die Produktion der für die Packet Radio Modems wichtigen ICs eingestellt, und bis heute sind die Lager mit den für den Amateurfunk benötigten ICs leer. Der Amateurfunk muss nun also auf Alternativen zurückgreifen, was heißt, dass man entweder die Modems mit vielen Standard-ICs aufbaut oder einen teuren DSP verwendet.
Die Motivation der Diplomarbeit war, dem Amateurfunker wieder Modem-ICs zur Verfügung stellen zu können, wobei aber erstens eine Ein-Chip-Lösung, und zweitens eine höhere Datenübertragungsrate über dieselbe Übertragungsstrecke angestrebt wurde.
Die Zielgruppe für den entwickelten ASIC sind aber weniger die lizenzierten Funkamateure, sondern vielmehr nicht lizenzierte Amateurfunker (CB-Funker). Lizenzierte Funkamateure dürfen auf Frequenzen im UKW-Bereich mit großer Bandbreite senden, und auch die verwendete Trägerfrequenz mit dem Datenstrom direkt modulieren. Sie können damit die zur Verfügung stehende Bandbreite mit geeigneten Verfahren wie z. B. QAM besser ausnutzen und erreichen damit Datenraten bis zu 76 kBaud. CB-Funkern steht im Gegensatz dazu nur eine Kanalbandbreite von 10kHz auf Kanälen im 11m-Band zu, wobei hier der Träger nicht direkt moduliert werden darf. Vielmehr dürfen CB-Funker nur den Sprachkanal für die Datenübertragung nutzen, mit einer Bandbreite von gerade einmal 3,6kHz. Ein Eingriff in das Funkgerät ist einem CB-Funker nicht gestattet, daher führt also kein Weg daran vorbei, den Datenstrom ins Sprachband umzusetzen, und ihn über die Mikrofonbuchse des Funkgeräts einzuspeisen, wie das ursprünglich auch im lizenzierten Amateurfunk mittels oben genannter AFSK-Modems geschah. Dennoch ist das ASIC auch für den lizenzierten Amateurfunk geeignet, die Nutzung beschränkt sich dort aber auf den Kurzwellenbereich, insbesondere das 10m-Band. Die vom ASIC unterstützte Übertragungsgeschwindigkeit beträgt 2400 Baud, was immerhin dem doppelten der bisher möglichen Übertragungsgeschwindigkeit entspricht.
Das neu entwickelte Modem ist so entworfen worden, dass es problemlos zusammen mit herkömmlichen AFSK-Modems im CSMA-Verfahren auf dem gleichen Funkkanal betrieben werden kann.

2 Realisierung der Arbeit

2.1 Vorarbeiten

Zunächst wurde anhand von Fachliteratur und Simulation mit Matlab ein geeignetes Modulationsverfahren gewählt. Es sollte einfach zu realisieren sein, den Betrieb des Modems zusammen mit herkömmlichen AFSK-Modems auf dem selben Funkkanal ohne Probleme erlauben und eine höhere Übertragungsrate bei gleichbleibender Bandbreite des Sprachkanals ermöglichen.
Das Minimum Shift Keying Verfahren war hier die geeignete Wahl, da es sich leicht in eine Sonderform der AFSK überführen lässt.

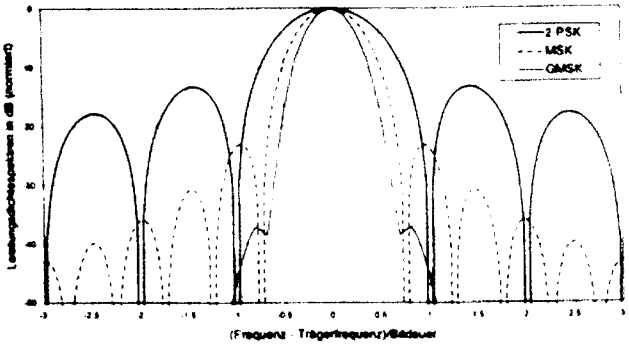


Abbildung 1: Vergleich Bandbreitebedarf 2-PSK , MSK, GMSK

Minimum Shift Keying kommt mit 75% der bei ASK, PSK und FSK benötigten Bandbreite aus und erlaubte damit (auch aufgrund bei AFSK nicht genutzter Bandbreite im Sprachkanal von Schmalbandfunkgeräten) die Verdopplung der Übertragungsrate.

2.2 Digitalentwurf

Der Digitalentwurf geschah mit der Hardware-Beschreibungssprache VHDL, als Entwurfswerkzeug diente Mentor Graphics FPGA-Advantage, ehemals "Renoir". Der Digitalteil wurde dabei hierarchisch aufgebaut. Der Modulator, der Demodulator mit integrierter digitaler Trägererkennung (DCD), der Watchdog und ein Timing-Generator (Systemtakt- und vor allem die Enable-Signalerzeugung) bilden dabei die Hauptkomponenten. Jede Komponente wurde nach DFT-Regeln mit einem Scanpath versehen, um später den Digitalteil des Chips mit Hilfe von Testvektoren testen zu können.
Jede Teilkomponente des Designs wurde mit Hilfe einer Testbench und dem Programm ModelSim simuliert und verifiziert, bevor damit eine neue Komponente aufgebaut wurde. Die Gesamtschaltung wurde ebenfalls simuliert.
Nachdem die Digitalschaltung verifiziert war, wurde sie mit Hilfe des Synopsys Design Analyzers und Design Compilers synthetisiert. Für den ASIC wurde auf die Mixed-Signal-Technologie CYE, einem 0,8μ-CMOS-Prozess von der Firma Austria Mikro Systeme (AMS) zurückgegriffen. Mittels Post-Synthesis-Simulation wurde das Zeitverhalten ebenfalls verifiziert.

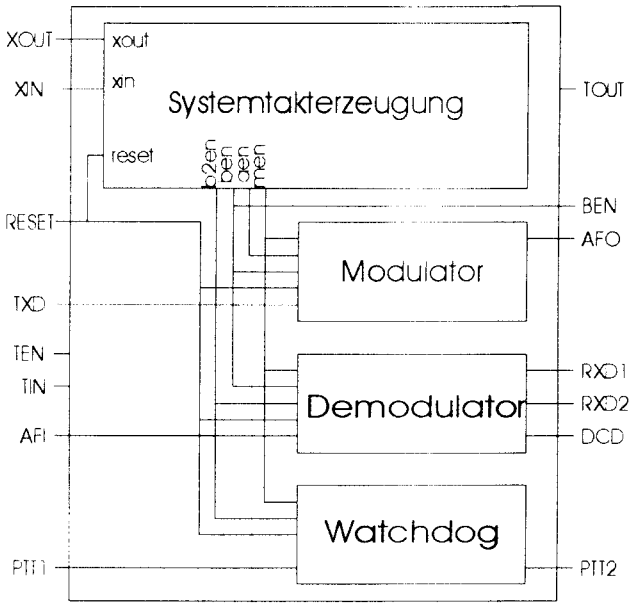


Abbildung 2: Blockschaltbild des Modems

2.3 Analogschaltungsentwurf

Der Analogschaltungsentwurf wurde mit Hilfe des Design Architects durchgeführt. Die dabei verwendeten Bauelemente waren ausnahmslos aus der Bibliothek des Halbleiterherstellers erhältlich.
Der Analogteil ist recht übersichtlich und enthält keine sonderlich anspruchsvollen Schaltungen. Daher war der Analogteil relativ schnell bis zum Stadium der Simulation gereift. Die Simulation wurde mit AccuSim durchgeführt.

2.4 Layout

Nachdem der Schaltplan fertig erstellt war, wurde als erstes der Digitalteil entflechtet. Zur Layouterstellung wurde mit dem Programm IC-Station von Mentor Graphics gearbeitet. Der Digitalteil konnte dabei weitestgehend automatisiert erstellt werden.
Beim Analogteil wurde eine Zelle selbst erstellt (eine Zweifachspannungsreferenz), die restlichen Zellen waren Standardzellen von AMS.
Vor dem Gesamtlayout wurde zunächst ein Topologieplan erstellt, bei dem "unruhige" Zellen möglichst weit entfernt von kritischen Zellen wie der Spannungsreferenz platziert werden konnten. Auch die Position der Peripheriezellen wurde bereits vor Beginn des Layouts festgelegt, und zwar nach den Kriterien "ruhiger" und "unruhiger" Peripheriezellen, sowie nach dem Kriterium der Trennung nach der Funktion. Beim Pinout sollten nach Möglichkeit später die Pins, die an die Computerschnittstelle angeschlossen werden sol-

len, auf der einen Gehäuseseite liegen, und die Pins, die später mit dem Funkgerät verbunden werden, auf der anderen Seite liegen. Als Gehäuseform für die Massenproduktion sollte ein SOIC-14 dienen, die Stromversorgungspins sollten dabei dem Standard entsprechen (GND auf Pin 7, VDD auf Pin 14).

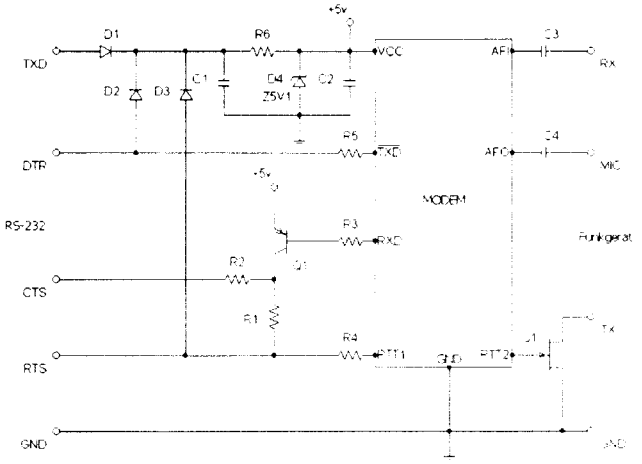


Abbildung 3: Minimalsystem

3 Zusammenfassung

Die zu Beginn der Diplomarbeit gestellten Anforderungen an den ASIC konnten eingehalten werden. Es entstand ein ASIC, der den Aufbau eines Modems mit nur einem IC und wenig externer Beschaltung erlaubt (siehe Abbildung 3). Die für das MSK-Verfahren gewählten Modulationsfrequenzen und die eingebaute digitale Trägererkennung erlauben den Betrieb des Modems auch auf Kanälen, die von herkömmlichen AFSK-Modems verwendet werden, ohne sich gegenseitig zu stören.

Die Übertragungsrate beträgt jetzt statt 1200 Baud 2400 Baud, die benötigte Bandbreite beträgt 3,4 kHz. Ein externer Abgleich ist nicht notwendig, digitale Trägererkennung und Watchdog zur Begrenzung der Sendedauer auch im Fall eines Programmabsturzes sind integriert. Das Modem kann mit der in Abbildung 3 gezeigten Minimalbeschaltung direkt an der seriellen Schnittstelle eines PC mit geeigneter Packet Radio Software betrieben werden.

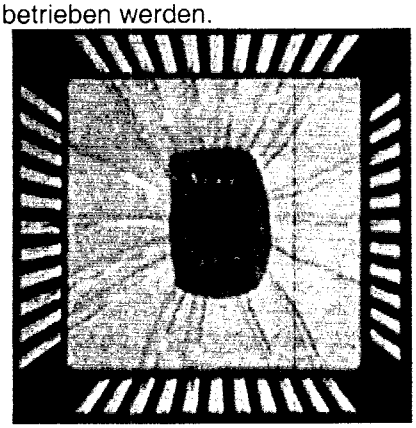


Abbildung 4: Die-Foto

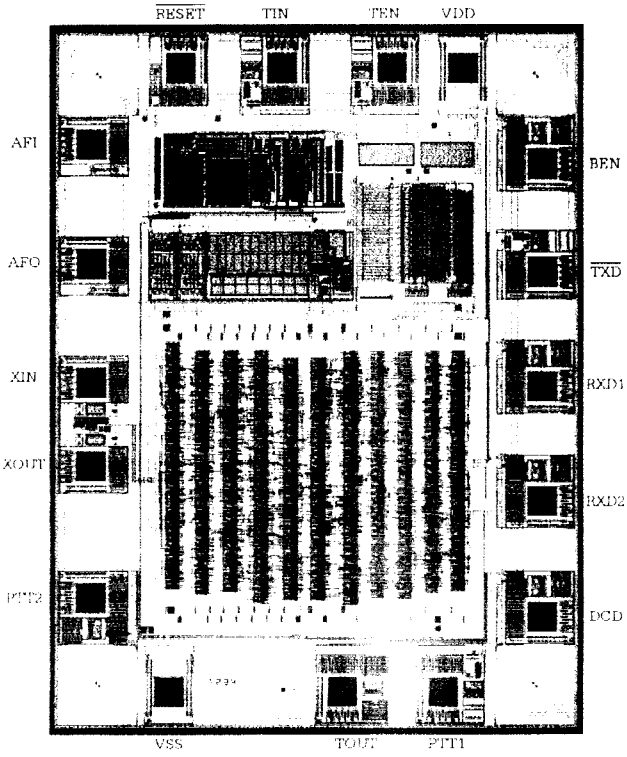


Abbildung 5: Layout

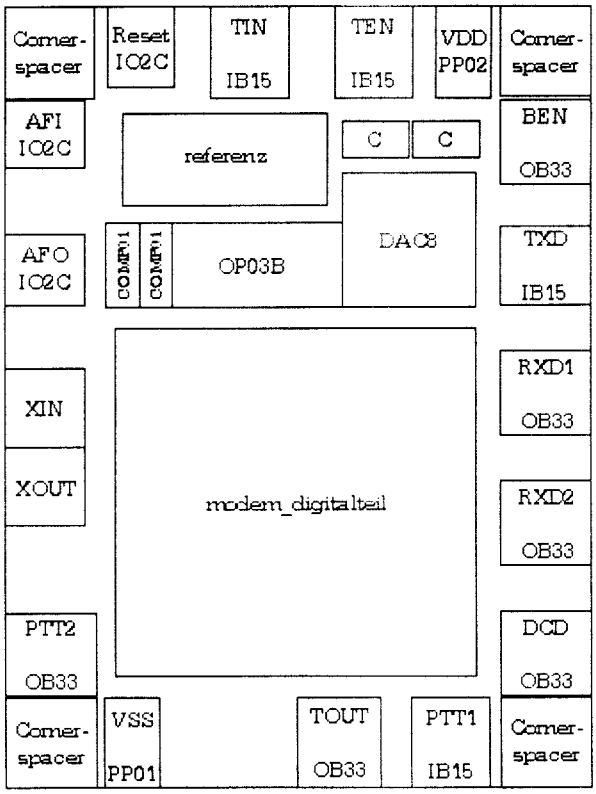


Abbildung 6: Chip-Topologieplan

FHOP-Prozessor-Core auf FPGA
Der neue Design-Kit für SOC-Entwicklung und Emulation

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 wurde ein neuer FHOP-Design-Kit entwickelt. Der Kit bietet die Möglichkeit, SOCs neu zu entwickeln sowie zu emulieren.

1. Einführung

Bei einem SOC-Design handelt es sich um einen komplexen ASIC, bei dem verschiedene Kerne miteinander kommunizieren. Ein Teil der Software ist auf dem Chip im ROM integriert. Es handelt sich hierbei um das BIOS, um Standardroutinen sowie Treibersoftware für die Kerne. Ein Beispiel für einen solchen SOC ist der an der FH-Offenburg entwickelte EKG-Datenlogger (*Abbildung1*).

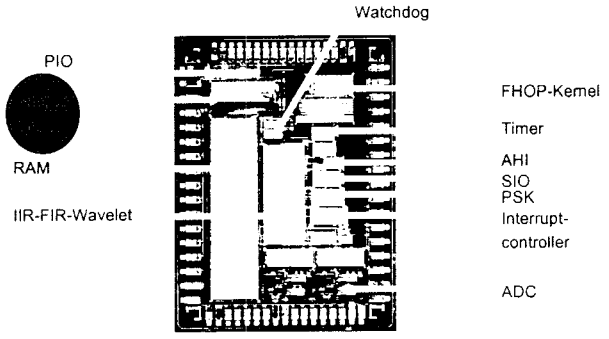


Abbildung 1 : EKG-ASIC [3]

Bei der Neuenwicklung eines solch komplexen ASIC's geht man so vor, dass man den Chip aus mehreren Kernen zusammenbaut. Parallel zur Hardware-Entwicklung wird die Software entwickelt, die im ROM enthalten ist. Es ist notwendig, vor der Entwicklung des ASIC's das entworfene Design unter realen Bedingungen zu testen, um das Risko zu senken. Ebenso muss bei der Neuentwicklung einer neuen Komponente das Gesamtsystem verifiziert werden, da die Komponente z.B. die IIC-Schnittstelle im Gesamtsystem in Interaktion mit anderen Komponenten wie z.B. einem Prozessor oder auch einer externen Komponente wie z.B. einem EEPROM tritt.

Zur Verifikation eines solchen SOC's bieten sich nun verschiedene Möglichkeiten an. Zum einen kann man die Software mit Hilfe des an der FH-Offenburg entwickelten SW-Entwicklungstools (PC-IDE) testen. Weiterhin kann man eine Digitalsimulation durchführen, und eine dritte Möglichkeit liegt darin, das komplette System mit Hilfe eines FPGA's zu emulieren (*Abbildung 2*).

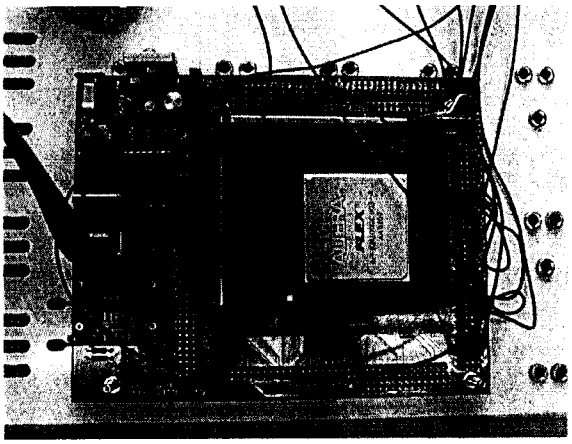


Abbildung 2: Emulation eines SOC in FPGA-Hardware

Alle drei Verifikationsarten besitzen ihre Vor- und Nachteile.

PC-IDE:

+ Ermöglicht ein einfaches und schnelles Testen der Hardware,

- Hardware wird dabei nicht getestet,

- Interaktion mit Schnittstellen ist schwierig,

- Keine Echtzeit.

VHDL-Simulation:

+ Hardware wird mit getestet,

+ exaktes Timing,

+ bitgenaue Darstellung (Traces),

+ Einbindung analoger Komponenten ist möglich,

- sehr langsam, längere Programmsequenzen können nicht getestet werden (BIOS, SIO).

Hardware:

+ Gesamtes System schnell testbar,

+ echtzeitfähig,

+ Interaktion mit Schnittstellen ist möglich, somit können auch externe Komponenten in die Emulation eingebunden werden,

- Fehlfunktionen sind schlecht überprüfbar (Oszilloskop erforderlich),

- das ASIC-Verhalten kann nicht exakt verifiziert werden, da der FPGA-Baustein andere Verzögerungszeiten aufweist als der ASIC.

Die Gegenüberstellung verdeutlicht, dass nur durch eine Emulation mit Hardware das real existierende Gesamtsystem mit allen externen Komponenten verfiziert werden kann. Nur dadurch ist es möglich, die Software (BIOS) sinnvoll zu testen.

2. Der neue FHOP-Design-Kit

Der neue FHOP-Design-Kit bietet die Möglichkeit, SOC's zu entwickeln und diese zu emulieren. Dabei sind alle drei Emulationsarten möglich. Die Zieltechnologie ist frei wählbar. Man hat damit die Möglichkeit, einen ASIC zu entwerfen und zuvor das System mit Hilfe eines FPGA's zu testen.

Der neue Kit enthält folgende Entwicklungstools:

- VHDL-Package,
- VHDL-Beschreibung der RAM/ROM Speicherbausteine für Digitalsimulation,
- Beispieldesigns, die die Verwendung des VHDL-Packages verdeutlichen,
- ROM-Software zu den Beispieldesigns,
- DO-Files für die Digitalsimulation der Beispieldesigns,
- IDE für SW-Entwicklung und Emulation,
- Anwendersoftware + BIOS,
- Hilfsprogramm prgtomif.exe.

Mit Hilfe der oben aufgeführten Tools kann man sein eigenes Design entwerfen, das folgende Komponenten enthalten kann:

- FHOP-Kernel, 16 bit,
- Interruptcontroller, 8 bit,
- Timer,
- Watchdog,
- Buscontroller,
- Chipselect,
- Serielle Schnittstelle (SIO),
- Parallele Schnittstelle (PIO),
- PSK-Modul,
- IIC-Schnittstelle,
- Acoustic-Human-Interface (AHI),
- Wrapper für die RAM/ROM-Speicherbausteine der Altera-Flex-10k-Technologie.

3. Anforderungen an den neuen FHOP-Design-Kit

Bei der Entwicklung des neuen Design-Kits waren folgende Gesichtspunkte zu beachten:

- Alle Komponenten sind in einer VHDL-Datei beschrieben.

- Die VHDL-Datei enthält nur synthetisierbares, strukturiertes VHDL.
- Die VHDL-Beschreibung ist technologieunabhängig.
- Durch die strukturierte Beschreibung ist ein Synthesetool wie z.B. Leonardo nicht unbedingt erforderlich. Man kann bei der Entwicklung auf eine FPGA-Zieltechnologie direkt die Entwicklungstools der Zieltechnologie einsetzen, wie z.B. MAX+PLUS2 für die ALTERA-Zieltechnologie oder XILINX-ISE für die XILINX-Zieltechnologie.
- Sowohl die Entwicklung auf eine ASIC-Zieltechnolgie ist möglich, als auch die Entwicklung eines FPGA-Bausteins.
- Der neue Design-Kit ist kompatibel zu der bereits bestehenden Dokumentation. Es wurde nichts an der Funktionsweise, wie an der Schnittstellenbeschreibung der einzelnen Komponenten geändert.
- Dadurch bleibt der neue Kit auch kompatibel zu der bereits bestehenden Software-Entwicklungsumgebung (SW-IDE)
- An der Busstruktur (16 bit Adressbus, 8 bit Datenbus wurde nichts geändert)

Anmerkung: Erstmals ist auch der FHOP-Kernel technologieunabhängig, da er ohne Microcode-ROM entworfen ist.

4. Anwendung des VHDL-Packages

Das Package besteht aus zwei Dateien. Die Datei *fhop_components.vhd* entält die Komponenten-Deklarationen (Abbildung 3), und die Datei *fhop_models.vhd* enthält die Entity- und die Architecture- Beschreibung der einzelnen Komponenten (Abbildung 4).

Diese beiden Dateien müssen in eine Library mit dem Namen *fhop_lib* kompiliert werden. Dann kann man sich sein Design durch Verknüpfen der einzelnen Blöcke zusammensetzen. Dies kann durch Verwendung des Tools *RENOIR* geschehen, oder man schreibt das Top-Sheet selbst. Hier benötigt man nur noch einfache Port-Map-Befehle mit denen man die Komponenten verbindet. Im Top-Sheet muss natürlich die *fhop_lib* eingebunden werden. Dadurch können die einzelnen Komponenten referenziert werden (Abbildung 5).

```
package fhop_components is
component
int_controller_8int_wrapper
  port (
        a0 : IN std_logic ;
        a1 : IN std_logic ;
     ...) ;
end component ;
component pio_wrapper
  port (
     ...) ;
end component ;
end ;
```

Abbildung 3 : Die Datei fhop_components.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
library fhop_lib;
entity int_controller_8int_wrapper is
  port (
        a0 : IN std_logic ;
     ...) ;
end int_controller_8int_wrapper ;

architecture struct of
int_controller_8int_wrapper is
...
begin
I0_notri_reg_int_pegel7 :
fhop_lib.fhop_components.DFF
 port map ( Q=>I0_int_pegel7, D=>nx1321,
            CLK=>clk, CLRN=>nx655,
            PRN=>nx656);
...
end struct ;
```

Abbildung 4 : Die Datei fhop_models.vhd

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
LIBRARY fhop_lib;

ENTITY microcontroller IS
        PORT
        (
         RESET_N :IN std_logic;
         ...
         );
END microcontroller;

ARCHITECTURE rtl OF microcontroller IS
for FHOP :
fhop_lib.fhop_components.fhop_kernel use
entity fhop_lib.fhop_kernel(structure);

FHOP :
fhop_lib.fhop_components.fhop_kernel
        PORT MAP(
                CLOCK=>CLK_IN,
                HOLD=>HOLD,
                ...
                    );
```

Abbildung 5 : Refenzieren der Komponeneten aus der fhop_lib im Top-Sheet

5. Testen des neuen Design-Kit an zwei Beispiel-Designs

Anhand von zwei Beispiel-Designs wurde die Funktion des neuen Design-Kits getestet. Das Design-A (Abbildung 6) besteht aus den Komponenten **FHOP-Kernel**, **Buscontroller**, **Chipselect**, **RAM-Wrapper**, **ROM-Wrapper**, **RAM** (2KByte), **ROM** (1KByte) und dem **Acoustic-Human-Interface** (AHI). Das ROM enthält eine Beispielsoftware. Diese erzeugt eine Melodie am AHI-Ausgang. Das Design-B ist etwas komplexer aufgebaut, es enthält zusätzlich zu den Komponenten des Designs-A noch die Komponenten **Timer**, **Interruptcontroller** und **Watchdog**. Die Beispiel-Software im ROM testet das Zusammenwirken aller Komponenten. Der Timer läuft ab und erzeugt einen Interrupt am Interruptcontroller, der Interruptcontroller führt das Handshake mit dem FHOP-Kernel durch, dieser springt in die Interruptroutine. In der Interruptroutine wird die Melodie am AHI-Ausgang erzeugt, anschließend läuft der Watchdog ab und erzeugt einen Reset am FHOP-Kernel. Die Software wird nun erneut abgearbeitet, so dass am AHI-Ausgang ständig die Melodie zu hören ist.

Beide Designs wurden auf den Baustein Altera-Flex-10k100GC503-4 umgesetzt und mit Hilfe eines Testboards getestet. Mit Hilfe der Simulationsmodelle der Speicherbausteine kann auch eine Digitalsimulation der Systeme durchgeführt werden.

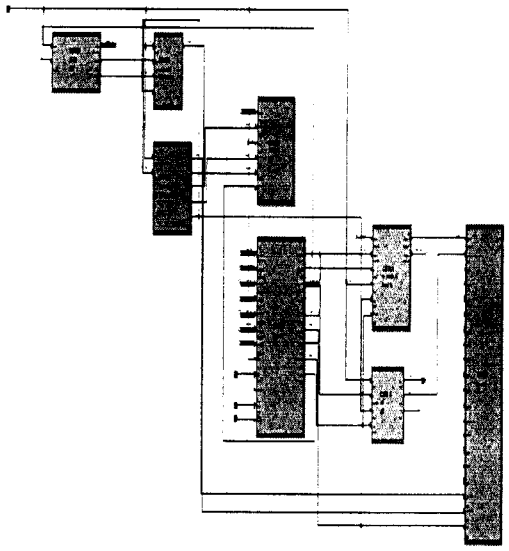


Abbildung 6 : Beispieldesign bestehend aus den Komponenten FHOP-Kernel, Chipselect, Buscontroller, RAM-Wrapper, ROM-Wrapper, RAM (2KByte), ROM (1KByte), Acoustic-Human-Interface (AHI)

	Memory Bits	Memory % Utilized	LCs	LCs % Utilized	Max. Frequency
Design A	24576	100 %	2564	51%	8.83 MHz
Design B	24576	100 %	3305	66 %	4.98 MHz

Tabelle 1 : Ergebnisse der Umsetzung der Beispieldesigns auf die Zieltechnologie Altera-Flex-10k100GC503-4.

Anhand der Tabelle 2 lässt sich erkennen, dass der gewählte Baustein im Falle des Designs-B beinahe voll ist, 66% der verfügbaren LC's wurden benötigt. Bei 70% ist der Baustein voll. Die Memorybits sind zu 100% belegt. Der Baustein verfügt insgesamt über 3KByte Speicher. Davon wurden 2KByte dem RAM zugeteilt und 1KByte dem ROM. Betrachtet man die Frequenzen so ist festzustellen, dass das Design-B eine deutlich geringere maximale Frequenz aufweist, was mit der größeren Ausnutzung des Bausteins zusammenhängt. Wählt man einen größeren Baustein, z.B. von Xilinx so müssten in beiden Fällen Frequenzen von ca. 11 MHz erreichbar sein. Damit hat man dann die Möglichkeit, auch noch das PSK-Modul mit in das Gesamtsystem zu integrieren, welches eine Systemfrequenz von 11,0592 MHz vorausetzt. Ebenfalls mit integrieren lässt sich dann die serielle Schnittstelle, so dass sich auch die UART-Betriebsart testen lässt.

6. Einschränkungen

Aufgrund der unterschiedlichen Schnittstellen der Speicherbausteine bei unterschiedlichen Technologien sind Wrapper für die Speicherbausteine notwendig, die speziell auf die Schnittstellen und das Timing der Speicherbausteine anzupassen sind. Entwirft man somit sein eigenes Design mit den Komponenten der *fhop_lib* und wählt z.B. einen Baustein von Xilinx, so lassen sich die Wrapper der *fhop_lib* nicht verwenden, da sie speziell auf die Speicherbausteine der Altera-Flex10k-Technologie ausgelegt sind.

Alle Komponenten sind über den Datenbus miteinander verbunden und greifen auf diesen über Tristate-Treiber zu. Die Tristate-Philosophie erweist sich beim Enwurf auf ASIC's als problemlos. Hier garantieren Buskeeper-Zellen, dass stets ein definierter Pegel am Bus anliegt, auch wenn gerade keine Komponente auf diesen zugreift. Setzt man das System auf eine FPGA-Technologie um, so hat man hier das Problem, dass keine Buskeeper-Zellen verfügbar sind. Dies stellt funktional gesehen kein Problem dar, führt aber unter Umständen zu einem erhöhten Leistungsverbauch.

Bedingt durch das Timing der Speicherbausteine enthält der Buscontroller noch asynchrone Teile. Ebenso sind in der Interruptverarbeitung des FHOP-Kernel noch asynchrone Teile vorzufinden. Diese asynchronen Teile werden in naher Zukunft beseitigt.

Die Umsetzung funktionierte nur unter Beibehaltung der Hierarchie. Ein Flatten des Designs war in der Synthese zwar möglich und brachte auch keine Fehlermeldungen, beim Test mit dem Testboard aber war die Funktionsfähigkeit nicht gegeben.

7. Verfügbarkeit und Nutzbarkeit

Der neue Design-Kit ist über Internet per FTP verfügbar, unter der Adresse

http://www.asic.fh-offenburg.de.

Die Vorgänger-Version des neuen Design-Kit ist auch weiterhin verfügbar [1].

Genutzt werden kann der neue Design-Kit für die Synthese mit Leonardo sowie zur Synthese mit Synopsis. Bei der Umsetzung auf eine FPGA-Technologie ist ein Synthese-Programm nicht unbedingt erforderlich. Es kann direkt ein Routingprogramm wie z.B. MAXPLUS2 oder XILINXSE verwendet werden.

8. Zusammenfassung

Der neue Design-Kit enthält ein VHDL-Package. Mit Hilfe dieses VHDL-Packages kann ein eigenes Design mit den Komponenten entworfen werden, die im Package enthalten sind.

Der neue Design-Kit eignet sich somit sowohl zur Emulation, als auch zur Neuentwicklung von SOCs. Das Risiko bei der ASIC-Entwicklung wird dadurch reduziert. Die Verwendbarkeit des neuen Desin-Kits wurde an Beispielen demonstriert.

9. Referenzen

- [1] Vollmer, W .: Dokumentation: „FHOP-Design-Kit". Fachhochschule Offenburg, 1997.
- [2] Vollmer, W.: Aufbau eines Mikrocontrollersystems auf der Basis des Mikroprozessorkernels FHOP und Entwurf der dazugehörigen Peripheriemodule als integrierter Anwenderschaltkreis (ASIC), Diplomarbeit, Fachhochschule Offenburg, 1996.
- [3] C. Störk, W. Vollmer : Dual Signal Wavelet Processing Controller, Projektbericht, Fachhochschule Offenburg, 1999.

Embedded system design with ARM7 processor cores

Christian Bechter
Atmel Germany GmbH, Lise-Meitner-Str. 15, 89081 Ulm
Fachhochschule Ulm, Prittwitzstraße 10, 89075 Ulm

Ziel dieser Arbeit war es, ein ARM7 basiertes Mikrokontrollersystem für eine Anwendung im Multimediabereich zu entwerfen. Dieses Mikrokontrollersystem sollte mittels entworfener Auswerteroutinen verifiziert werden. Die Verifikation sollte durch Simulation sowie durch ein „Hardware – Prototyping" erfolgen. Ausgehend von einem Referenzdesign wurde ein neues, überarbeitetes und verbessertes Mikrokontrollersystem entworfen. Um dieses Ziel zu erreichen konnte auf bereits vorhandene Systemkomponenten sowie auf einen ARM7 – Prozessor – Core zurückgegriffen werden. Um allerdings die volle Funktionalität zu erreichen, mussten noch diverse Systemkomponenten entworfen werden. Anschließend wurde das komplette ARM7 – System in einem FPGA abgebildet um es einem Hardware – Prototyping zu unterziehen.

1 Einleitung

1.1 Motivation

Die Firma Atmel bietet für den neuen Rundfunkstandard Digital Audio Broadcast (DAB) einen Chipsatz an, der vom Frontend bis hin zum Audioausgang alle relevanten Komponenten eines digitalen Rundfunkempfängers enthält. Bisher wurde ein externer Mikrokontroller für die Steuerung des DAB – Systems zu benötigen, sollte dieses ARM7 - basierte Mikrokontrollersystem entwickelt werden. Das ARM7 – System soll zukünftig zusammen mit den restlichen Komponenten des DAB – Empfängers in einen Chip integriert werden.
In dieser Arbeit stand die Entwicklung dieses ARM7 – Mikrokontrollersystems im Vordergrund. Hierbei musste eine Peripherie entworfen werden, in die ein ARM7 – Prozessor – Core eingebettet werden kann und somit ein lauffähiges Mikrokontrollersystem entsteht.

1.2 System Grundlagen

Der ARM7 – TDMI (Abbildung 1) bildet die Basis eines solchen Mikrokontrollersystems. Der Core ist nach den Prinzipien eines RISC (Reduced Instruction Set Computer) aufgebaut. Er besitzt 37 Register und eine dreistufige Instruction – Pipeline. Nur die Inhalte der 37 Register des Cores können durch Befehle bearbeitet werden. Beim ICEBreaker handelt es sich um einen Block, der aktiv das „On Chip Debugging" unterstützt. Dazu wird noch der TAP controller benötigt. Über diesen und die dargestellten Scan Chains wird der ICEbreaker angesprochen und programmiert, wenn das Debugging mit „Watchpoints" bzw. „Breakpoints" erfolgen soll.

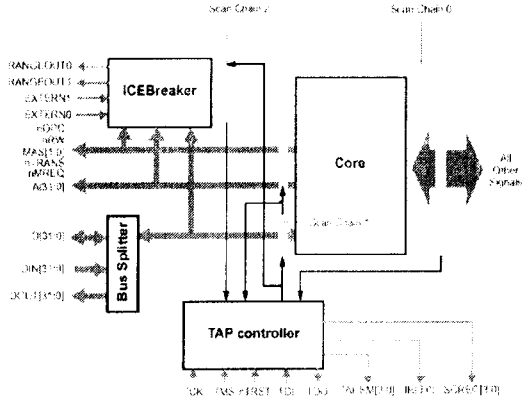


Abbildung 1: ARM7 - TDMI

Somit sind für den Benutzer alle systeminternen Register und Speicher jederzeit zugänglich und manipulierbar.
Ein weiterer wesentlicher Aspekt ist der grundlegende Aufbau einer ARM7 – Peripherie. Diese ist in Abbildung 2 dargestellt. Die beiden Systembusse, der APB (Advanced Peripheral Bus) und der ASB (Advanced System Bus), bilden hierbei das Rück-

grat des AMBA – Peripheriesystems (Advanced Microcontroller Bus Architecture). Über die beiden Busse läuft die komplette Kommunikation zwischen den Systemkomponenten ab. Der ASB ist ein Bus mit einem sehr hohen Datendurchsatz und eignet sich deshalb für den Anschluss von Speichern. Der APB ist im Gegensatz dazu ein Bus mit niedrigem Datendurchsatz. Er ist darauf ausgelegt, die Verlustleistung des Systems niedrig zu halten. An ihm sind vor allem Komponenten angeschlossen, die nur selten benötigt werden und auch dann nur einen mäßigen Durchsatz an Daten aufweisen.

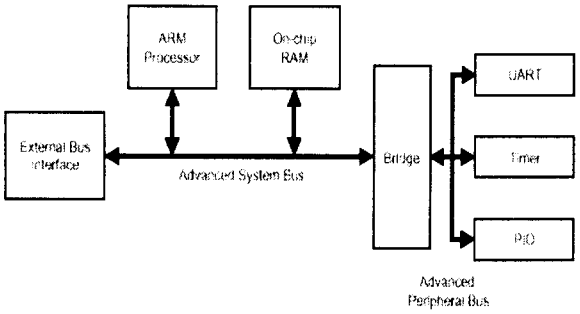


Abbildung 2: ARM7 Peripherie System

Die Bridge stellt das Bindeglied zwischen den beiden Bussystemen dar. Sie passt die beiden Busprotokolle aneinander an.

2. Schaltungsdesign

Abbildung 3 zeigt den Aufbau des ARM7 - basierten Mikrokontrollersystem. In dieser Abbildung bildet der ARM7 TDMI und sämtliche dargestellten Peripheriekomponenten das Mikrokontrollersystem.
Ein wesentlicher Aspekt war es, aus diesen zahlreichen Komponenten letztendlich das ARM7 – System zusammen zu fügen. Hierzu konnte auf schon vorhandene Komponenten zurückgegriffen werden. Es mussten aber auch noch einige der Komponenten neu entworfen werden. Der Entwurf des kompletten Design fand mit dem HDL – Designer statt, dem Nachfolgeprogramm von RENOIR. Nach dem Zusammenfügen sämtlicher Schaltungsteile konnte das System simuliert werden. Diese Simulation fand unter MODELSIM statt. Für die Simulation war es noch nötig, eine Software zu entwerfen, da hier nicht wie gewöhnlich mit einer Test – Bench simuliert wurde. Diese Vorgehensweise bietet sich deshalb an, weil es sich um ein Mikrokontrollersystem handelt, das für seine Funktion eine Software benötigt. Ein weiterer Vorteil ist, dass die entwickelte Software auch für das Hardware – Prototyping verwendet werden kann. Da es sich um die selbe wie in der Simulation handelt, können vorhandene Fehler schneller lokalisiert werden. Bei dem Design wurden zwei Simulationen durchgeführt, eine auf RTL (Register Transfer Level) und die andere auf Gate – Level – Ebene (Postsim). Danach erfolgte die Verifikation mittels des schon erwähnten Hardware – Prototyping.

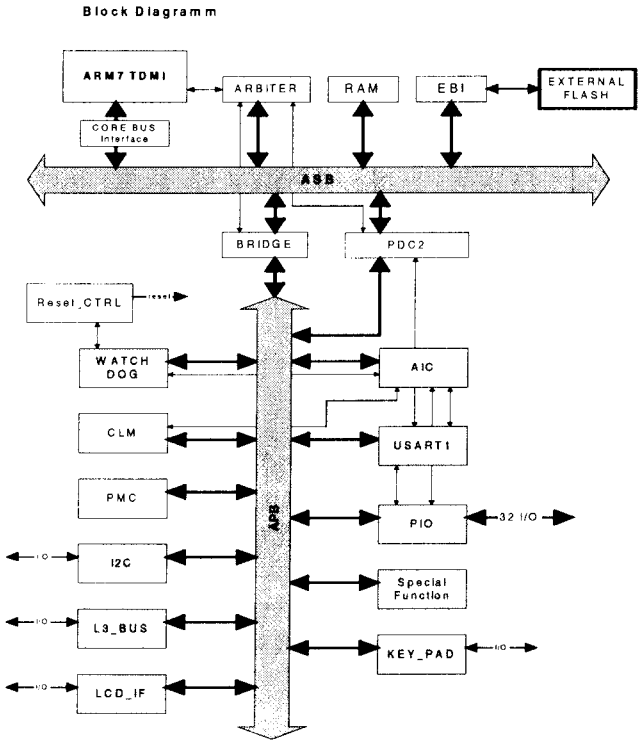


Abbildung 3: ARM7 - Mikrkontrollersystem

2.1 IP Design

Einige der benötigten Komponenten (IPs Intelectual Properties) waren noch zu entwerfen. Diese Komponenten waren das I2C_Interface, das L3_Interface, das LCD_Interface und das KEY_PAD_Interface. Abbildung 4 zeigt das I2C_Interface sowie seinen inneren Aufbau.

Das I2C_Interface kann von anderen I2C – Slaves Daten empfangen und an sie senden. In dieser

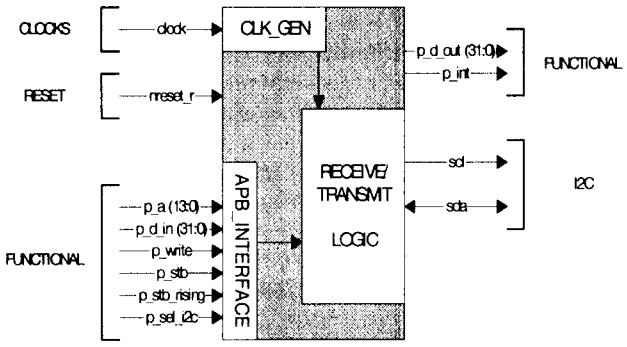


Abbildung 4: I2C_Interface

Implementierung ist das I2C_Interface ständiger und einziger I2C – Master am I2C – Bus. Grund für diese Art der Implementierung war, dass das ARM7 – System über diese Komponente mit dem Tuner des DAB – Empfängers kommuniziert. Da der ARM das gesamte DAB – System steuert, ist er zwingend immer der Master am I2C – Bus. Über das APB_INTERFACE ist diese Komponente an den APB angebunden. Der Block CLK_GEN erzeugt den Takt für die SCL – Leitung des I2C – Busses. Die Frequenz ist über ein Register in diesem Block durch die Anwender – Software zu programmieren.
Die SDA – Leitung wird durch den Block RECEIVE / TRANSMIT LOGIC bedient. Durch diese Leitung können Daten vom I2C – Bus gelesen und auf ihn geschrieben werden.
Die gesamte Komponente ist vollständig durch die Anwendersoftware konfigurierbar.
In Abbildung 5 ist die zweite der entworfenen Komponenten abgebildet, das L3_Interface. Vom Aufbau

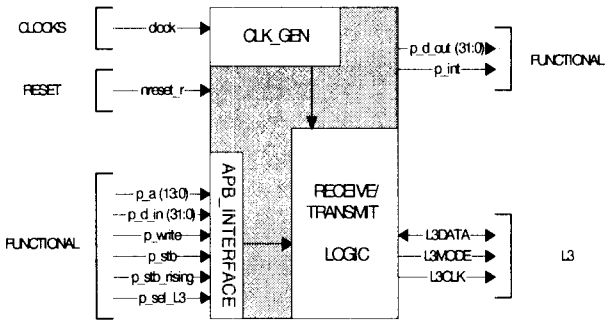


Abbildung 5: L3_Interface

her ist es dem I2C_Interface sehr ähnlich. Die internen Blöcke übernehmen die selben Aufgaben wie es auch beim I2C_Interface der Fall ist.
Diese Komponente erfüllt in vollem Umfang die Spezifikation des L3 – Busses. Auch diese Komponente ist vollständig über die Anwendersoftware konfigurierbar.
Zu beiden Komponenten ist noch zu sagen, dass es möglich ist, nach dem Empfangen oder Senden eines Datums einen Interrupt zu erzeugen. Zudem besteht die Möglichkeit, jederzeit den internen Zustand beider Komponenten über die Software abzufragen. Hierzu ist in jeder der Komponenten intern ein Status – Register implementiert.
Abbildung 6 zeigt eine weitere Komponente, die noch entworfen werden musste. Hierbei handelt es sich um ein LCD - Interface. Über dieses durch die Anwender – Software voll programmierbare Interface kann ein LCD – Display an das ARM7 – Mikrokontrollersystem angeschlossen werden. Das Interface erzeugt alle Steuersignale selbst. So muss der Anwender nur die Daten an das Interface übergeben und der Rest wird automatisch vom LCD – Interface erledigt. Das Interface kann zwei Arten von Displays bedienen: Graphische – und Character – Displays.
Zielsetzung beim Entwurf dieser Komponente war

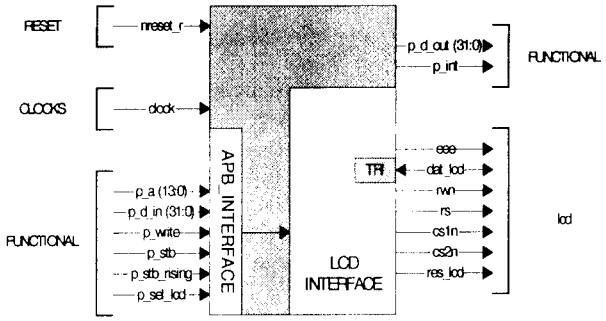


Abbildung 6: LCD_Interface

es, beim Anschluss der verschiedenen Displaytypen möglichst flexibel zu sein. Ferner sollte der ARM7 nicht an Rechenkapazität zu verlieren, wenn er mit dem LCD – Display kommuniziert.
Um Daten von Hand in das System eingeben zu können, musste noch eine vierte Komponente entworfen werden: Ein Interface für den Anschluss einer Matrix – Tastatur. Abbildung 7 zeigt das KEY – Interface.
Auch dieses Interface ist komplett über eine Software konfigurierbar. Die Tasten werden im Matrix_Interface decodiert. Neben den Zeichen (0,1,2..F) stehen auch Funktionstasten zur Verfügung. Eine DELETE -, ENTER - und SHIFT - Taste. Über die SHIFT Taste ist es möglich, dass insgesamt 30 verschiedene Zeichen verarbeitet werden können.

Um möglichst flexibel bei der Eingabe von Werten zu sein existieren zwei Betriebsarten. In der ersten wird nach jedem Tastendruck das vom Key – Interface ausdecodierte Datum in ein Ergebnisregister geschrieben. Um auch direkt im HEX – CODE Daten einzugeben, existiert noch ein weiterer Betriebsmodus. Hier werden immer erst nach zweimaligem Tastendruck Daten in das Ergebnisregister des Key – Interface geschrieben. Auf dieses Register kann der ARM7 zugreifen und so einlesen welche der Tasten gedrückt wurde.

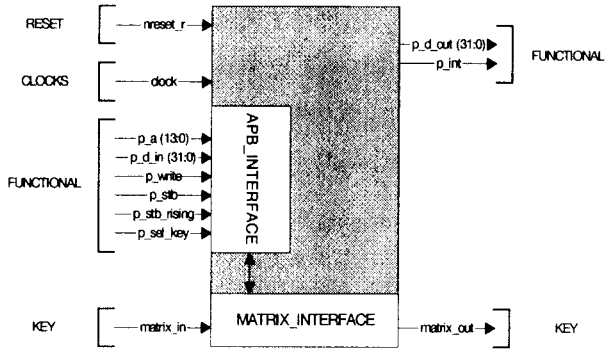


Abbildung 7: KEY_Interface

2.2 System Design

Um eine bessere Übersichtlichkeit des Designs zu erreichen, wurden mehrere Hierarchie – Ebenen eingeführt. In Abbildung 8 ist die Design – Hierarchie mit 5 Ebenen dargestellt.
Die Ebene ARM_SUBSYSTEM_LITE ist die synthetisierbare Peripherie für den ARM7 – Prozessor. Unter dieser Ebene sind sämtliche Systemkomponenten zu erkennen. Diese sind nach ihrer Position im System strukturiert. Unter APB_PERIPHERALS liegen z.B. alle Komponenten, die über diesen Systembus angesprochen werden. Für die Ebene ASB_PERIPHERALS gilt entsprechendes. Beide Ebenen werden durch das ASB_APB_TOP zusammengefasst. Unter dieser Ebene liegen noch diverse andere Komponenten. Die BRIDGE stellt das Bindeglied zwischen den ASB_PERIPHERALS und den APB_PERIPHERALS dar. Die Anbindung des ARM7 – Prozessors an das Peripheriesystem erfolgt über das CORE_BUS_INTERFACE. Um das System durch einen Reset rücksetzen zu können, wird der RESET_CONTROLLER benötigt. Die beiden Komponenten MEM_IF und FPGA_IF realisieren die Schnittstelle zwischen dem Peripheriesystem und den daran angeschlossenen Speichern (MEM_IF) und die Schnittstelle zwischen dem FPGA und dem ARM7 – Chip (FPGA_IF). Die Komponente CLK_GEN ist auch eine der selbst entworfenen Komponenten, sie hat die Aufgabe, das komplette Mikrokontrollersystem mit den nötigen Taktsignalen zu versorgen.
Die oberste Ebene TOP_ASSL dient nur zur Simulation des gesamten Systems. Die drei enthaltenen Simulationsmodelle (ARM7TDMI, INTEL28F320 und SRAM_BANK_0) sind nicht synthetisierbar. Auf diesen Punkt wird unter „Verifikation des Gesamt - Systems“ noch eingegangen.

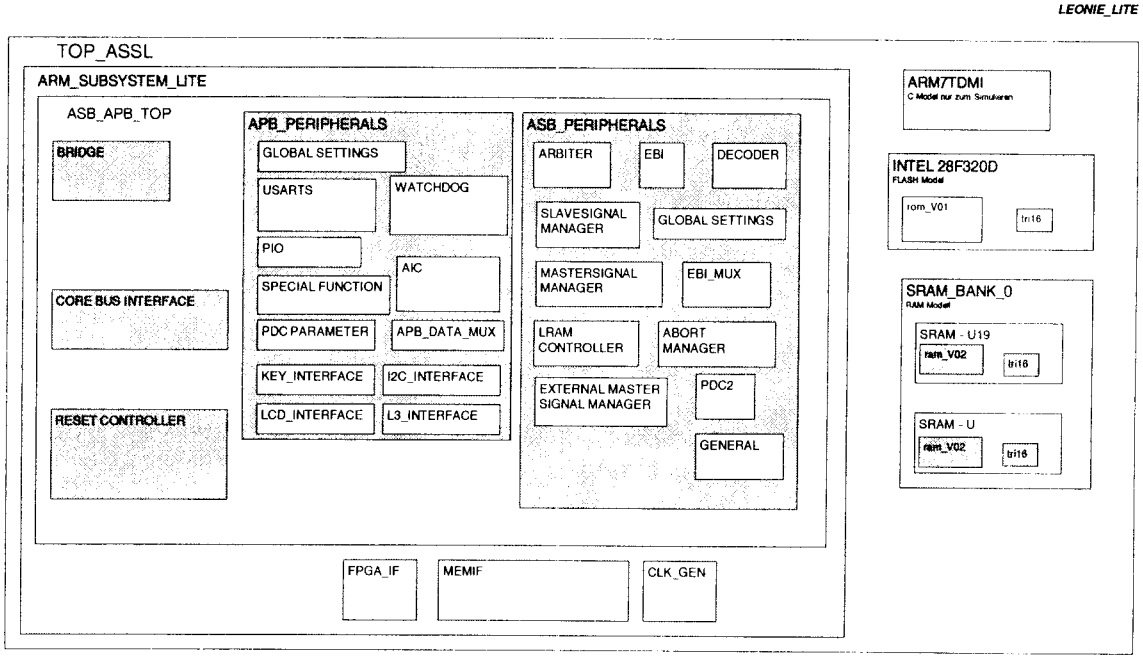


Abbildung 8: Design - Hierarchie

3. Design Flow

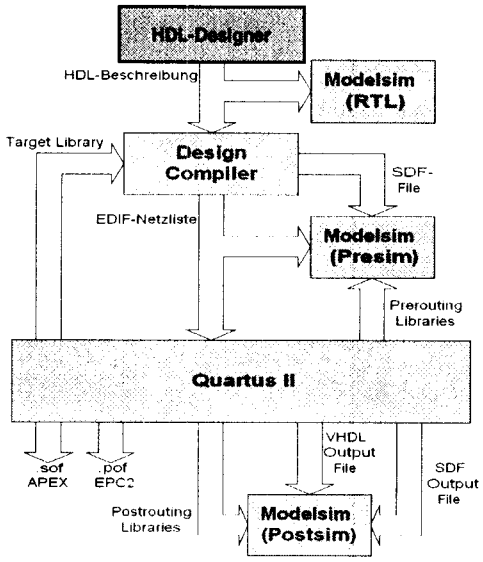


Abbildung 9: Design - Flow

Der durchgeführte Design Flow orientiert sich an dem zum Einsatz kommenden FPGA. Der hier verwendete FPGA weist eine Größe von 400K – Gattern auf. Diese Größe ist ausreichend zum Aufnehmen des kompletten Peripheriesystems (ca. 330K – Gatter). In Abbildung 9 wird der zur Anwendung kommende Design - Flow gezeigt.
Den Einstieg in den Design – Flow bildet der HDL – Designer. Mit diesem Programm werden VHDL – Beschreibungen entworfen, die mittels Modelsim kompiliert und anschließend simuliert werden. Diese Simulation findet zuerst auf RTL – Ebene statt. Zeigt das Design das gewünschte Verhalten, so kann es synthetisiert werden. Diese Synthese wird mit dem Design Compiler durchgeführt. Eine EDIF – Netzliste ist das Ergebnis der Synthese. Das Fitting in den FPGA wird durch Quartus II vorgenommen. Quartus II erzeugt aus der EDIF – Netzliste alle nötigen Dateien, um den FPGA zu konfigurieren (*.sof) und eine Postsim unter Modelsim durchführen zu können.

4. Verifikation des Systems

Ein Schwerpunkt der Arbeiten lag auf der Verifikation des Mikrokontrollersystems.
Zunächst wurde das Design einer gründlichen RTL – Simulation unterzogen. Hierbei kamen auch Simulationsmodelle eines ARM7, eines Flash und eines RAM zum Einsatz. Abbildung 8 zeigt den Aufbau des Designs. Hier sind auch die Simulationsmodelle zu erkennen (ARM7TDMI, INTEL28F320, SRAM_BANK_0).

Bei der Simulation wurde so vorgegangen, dass eine Software entwickelt wurde, die als Code im Flash hinterlegt wurde. Danach wurde das komplette Mikrokontrollersystem inklusive der Simulationsmodelle simuliert. Dieses Vorgehen bringt zwei entscheidende Vorteile: Der Schritt von der Simulation zum Hardware – Prototyping wird vereinfacht. Außerdem wäre eine Simulation mittels Test – Bench schwer realisierbar, da die Test – Bench bei der Vielzahl von Signalen gigantische Ausmaße annehmen würde.
Um das Design zu simulieren, ist eine Software nötig, mit der alle im Design enthaltenen Komponenten angesprochen werden. Nur wenn alle Register richtig beschrieben und gelesen werden können, wird die Komponente weiteren Tests unterzogen. In diesen Tests werden alle Möglichkeiten, die die jeweilige Komponente bietet, freigeschaltet und anschließend überprüft. Danach wird je nach Komponente weiterverfahren. Beim LCD – Interface wird z.B. ein Text auf einem Character – Display ausgegeben. Diese Ausgabe kann dann mittels Simulation und Hardware – Prototyping verifiziert werden. Haben alle Komponenten den Test erfolgreich durchlaufen, so wird dies über das USART1 – Interface auf einer seriellen Schnittstelle mitgeteilt.
Das Hardware – Prototyping wurde auf einer Plattform durchgeführt, die einen 400K – Gatter FPGA sowie einen ARM7 – Test – Chip enthält. Zudem befinden sich noch ein Flash und ein RAM auf dieser Platine. Abbildung 10 zeigt ein Bild dieser Hardware – Prototyping – Plattform. Um das Mikrokontrollersystem zu starten, müssen zunächst gültige Daten auf das Flash geschrieben werden. Hierzu musste noch eine Schaltung entworfen werden, die das Flash über ein Terminal – Programm, das auf einem PC läuft, konfigurieren zu können. Anschließend kann der FPGA mit dem entworfenen Design bespielt werden.
Ist das komplette Design, in das der ARM7 – CORE eingebettet ist (LEONIE_CORE), auf dem FPGA abgebildet und das Flash mit gültigen Daten bespielt, kann mit dem Hardware – Prototyping begonnen werden. Hierfür muss zwischen der Hardware – Plattform und einem PC eine Verbindung geschaffen werden. Dies geschieht über ein MultiICE – Interface. Dieses Interface erlaubt es dem Debugg – Tool, über ein JTAG – Interface mit dem Mikrokontrollersystem zu kommunizieren. Nach einem Systemreset beginnt der ARM mit der Abarbeitung des Codes, was eine Grundvorraussetzung für das Debugging ist. Beim Debugging der Hardware wird über das MultiICE – Interface das RAM mit der aktuellen Software bespielt (trotzdem müssen sich im Flash gültige Daten befinden). Der ARM beginnt dann mit dem Code ab Adresse 0. Mit dem Debug – Tool kann die entwickelte Software schrittweise durchlaufen werden. Es können aber

auch Breakpoints und Watchpoints gesetzt werden. Dadurch kann das komplette Design Stück für Stück geprüft werden.
Während des Debuggings können sämtliche Register des ARM und der ihn umgebenden Peripherie beobachtet und sogar modifiziert werden. Das selbe gilt für alle Speicher der Hardware – Prototyping – Plattform.

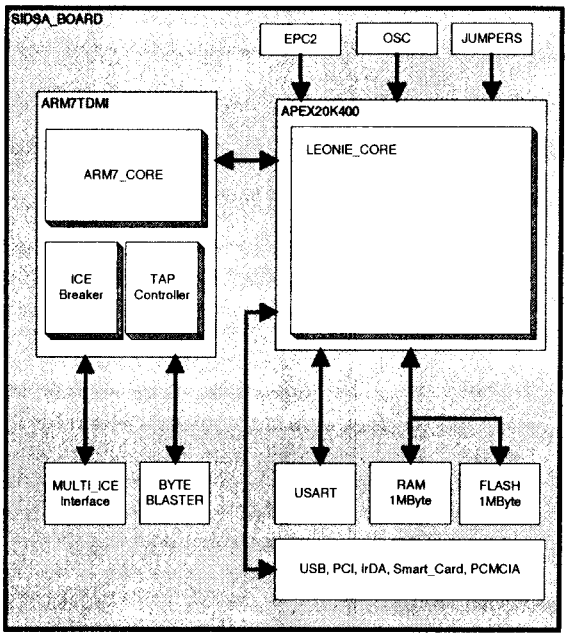


Abbildung 10: Hardware – Prototyping - Plattform

5. Zusammenfassung

Das Ergebnis dieser Diplomarbeit ist ein lauffähiges ARM7 basiertes Mikrokontrollersystem, das über Simulationen und Hardware – Prototyping verifiziert ist.
Probleme traten vor allem beim durchgeführten Design – Flow auf. Besonders Quartus II lieferte oft unbefriedigende Ergebnisse. Daher musste noch kurze vor Ende der Diplomarbeit die komplette Fitting – Strategie umgestellt werden, um wieder ein lauffähiges System zu bekommen. Bei dieser Strategie wurde der neu von Quartus II angebotene LOGIC – LOCK – Ansatz angewandt. Auch das Aufsetzten und durchführen einer Postsim (Gate – Level – Simulation) war problematisch.
Beim Hardware – Prototyping war es ein Problem, das zu testende Mikrokontroller – System auf der Hardware – Prototyping – Plattform soweit zu bringen, um ein „On – Chip – Debugging" durch zu führen.
Trotz vieler Probleme konnte das Ziel dieser Diplomarbeit erreicht werden, ein ARM7 basiertes Mikrokontrollersystem zu entwerfen und zu verifizieren.

6. Literaturverzeichnis

1.) Steve Furber: „ARM System-on-chip Archtecture", Addison-Wesley Verlag, London, 2001

2.) „Design Compiler Users Guide", Synopsis Online Documentation, Synopsis, 1999

3.) „Modelsim Reference Guide", Online Documentation, Model Technology, 1999

4.) „System Architecture: ARM7TDMI based Microcontrollers", Atmel, 2000

5.) „ARM7TDMI-S Technical Reference Manual", Advanced RISC Machines Ltd, 1999

6.) „HSDT200 Development Board Reference Guide and Users Guide version 3" Sidsa, 2000

7.) „Quartus II Online-Dokumentation", Altera Corp., San Jose, 2001

Modellierung mit VHDL-AMS

Prof. Dr. Gert Voland

FH Konstanz, Brauneggerstr. 55, 78462 Konstanz
Tel.: 07531 / 206 644, Email: voland@fh-konstanz.de

Es wird über ein Fortbildungssemester berichtet, was bei Infineon Technologies, München von März bis August 2001 durchgeführt wurde. Die Fragestellung, wie gut sich VHDL-AMS (IEEE 1076.1-1999) eignet, die Designaktivitäten speziell von Mixed-Signal-Schaltungen zu unterstützen, kann durchaus positiv beantwortet werden.
Es stellen sich die von Infineon entwickelten und betrieben Erweiterungen des Analog-simulators Titan als ausreichend gut und stabil heraus.
An Hand der Modelle eines Operations-verstärkers und einer PLL, die als Layout vorlag, konnte gezeigt werden, daß man mit der Sprache VHDL-AMS das nötige Verhalten beschreiben kann, und daß Titan diese Modelle auch ausreichend genau und schnell simulieren kann.
Zum Schluß wurde überprüft, ob die VHDL-AMS Implementierung von Titan soweit standard-konform ist, daß sich die Modelle auf andere, kommerziell verfügbare Simulatoren (z.B. ADVanceMS von Mentor Graphics) übertragen lassen.

1. Motivation

Da die CMOS-Technologie mit Strukturbreiten unter 0,1 µm in zunehmendem Maße die Integration der verschiedensten Schaltungstypen erlauben, steigt der Bedarf nach Designwerkzeugen, die digitale, analoge und HF-Schaltungen gleichzeitig bearbeiten können. Neben der erfolgreich betrieben Praxis, digitale Schaltungen mit VHDL auf RTL-Ebene zu beschreiben und dann auf die Zieltechnologie zu synthetisieren, werden analoge Schaltungen durch-aus noch per Hand entworfen und auf Transistor-ebene, also strukturell, verifiziert.
VHDL-AMS erhebt nun den Anspruch nicht nur analoge Schaltungen beschreiben und simulieren zu können, wie es mit SPICE bisher üblich ist, sondern auch die Schnittstelle zu digitalen Schaltungsteilen abzudecken. Dieses galt es zu überprüfen und herauszuarbeiten, in wie weit existierende Design-abläufe daraufhin angepaßt werden müssen.

2. Struktur und Positionierung von VHDL-AMS

Die gesamte Definition von VHDL-AMS stellt eine Übermenge dar, die Standard-VHDL (IEEE 1076–1993) um die AMS-Möglichkeit erweitert (**A**nalog **M**ixed **S**ignal). AMS erlaubt es, u.a. nichtlineare Differentialgleichungen zu beschreiben. Beide Teile benutzen die gleiche Syntax, müssen aber sehr verschieden interpretiert werden. Das dem VHDL-AMS-Compiler zugeführte Modell muß demnach in die ereignisgesteuerten Teile und in die zeitschrittgesteuerten Teile zerlegt werden.
Das nach außen als ein Programm erscheinende System besteht aus einem digitalen Simulator, der alle Standard-VHDL Teile bearbeiten muß und aus einem analogen Simulator (anlog solver), der die Differentialgleichungen zu lösen hat. Beide Simulatoren laufen eigentlich unabhängig von einander bis es zu einer gezielten Synchronisation kommt.

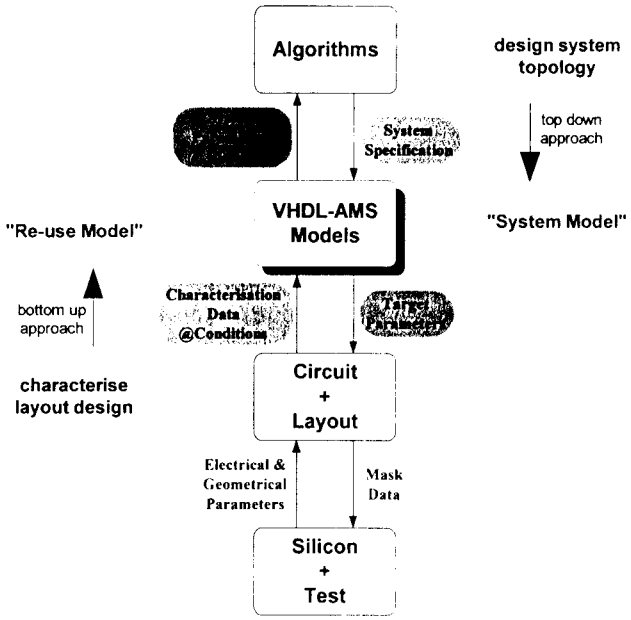


Abb. 1 Positionierung in Design-Flow

Bezüglich der Position im Designablauf will VHDL-AMS die Lücke zwischen algorithmischem Entwurf und dem strukturellen Design schließen (Abb. 1).

In dem "top-down"-Ansatz werden AMS-Modelle benutzt, die die prinzipiellen Funktionen bereitstellen und mit denen man dann das Systemverhalten verifiziert. Die zu übergebenden Parameter stellen systemspezifische Werte dar (z.B. Bitbreite, Verstärkung, Grenzfrequenzen usw.).
In dem "bottom-up"-Ansatz werden existierende Designs auf ein Modell abgebildet, was das technologiespezifische Verhalten möglichst genau nachbildet. Die Parameter stammen dann aus einem für das Design speziell zu erstellenden Charakterisierungsprogramm.

Einige der momentan zur Verfügung stehenden Simulationssysteme sind:

Titan (Infineon)
= Titan + AMS-Syntax + Event Simulation,
Viewer: Scope
+ ModelSim (externe Kopplung)
Viewer: ModelSim-View

ADvanceMS (Mentor)
= Eldo + ModelSim („single kernel")
Viewer: Xelga

VeriasHDL (Avant!)
= HSPICE(?) + VHDLSim
Viewer: Scope

3. Sprachkonstrukte

Mit Hilfe von LIBRARY- und USE- Anweisungen wird die elektrische "Nature" deklariert. Damit sind dann Spannung und Strom als die ACROSS und THROUGH QUANTITIES bei den konservativen Klemmen (mit TERMINAL deklariert) festgelegt.
Für Mixed-Signal-Designs ist es wichtig, daß man solche Klemmen beliebig mit den digitalen Typen SIGNAL, die ja eine Richtung aufweisen, mischen kann. Mit GENERIC-Konstrukten werden Parameter wie bei VHDL übergeben.
In Abb. 2 ist dargestellt, wie sich die AMS-spezifischen Anweisungen in die ENTITY-ARCHITECTURE-Struktur einpassen. In dem ARCHITECTURE-Teil können notwendige Berechnungen unter CONSTANT durchgeführt werden. Dies geschieht beim Kompilieren, da sich die Werte nicht mehr ändern können.
Die Anweisungen mit "==" sind keine Zuweisungen von Werten, sondern stellen Gleichungen dar, die dem Analogsimulator zugeführt werden. Mit dem Konstrukt q'DOT werden Ableitungen gekennzeichnet, so daß beliebige nichtlineare differential-algebraische Gleichungen formuliert werden können.
Die Verzweigung IF ... USE bewirkt, daß verschiedene Gleichungssysteme als Matrizen angelegt werden, zwischen denen je nach Bedingung des IF umgeschaltet wird.

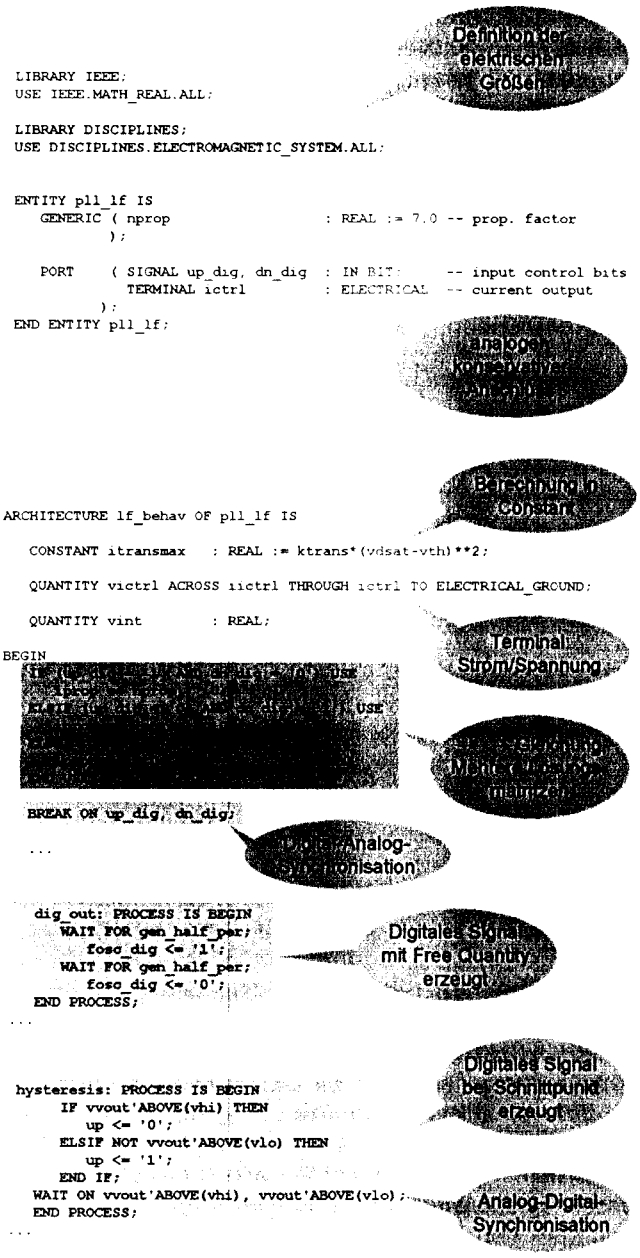


Abb. 2 AMS Konstrukte in ENTITY und ARCHITECTURE

4. Analog-Digital-Kopplung

Spezielle Aufmerksamkeit ist der Wandlung von digitalen Ereignissen in analoge Verläufe (D-A) und umgekehrt (A-D) zu schenken.
Abb. 3 beschreibt die Stellen, an denen die beiden, eigentlich unabhängig von einander laufenden Simulatoren, aufeinander synchronisiert werden können. Mit BREAK ON (s. Abb. 2) signalisiert man dem Analogsimulator, daß er "zurückgesetzt" werden muß. Ähnlich wie bei der Arbeitspunktberechnung wird die Zeitschrittsteuerung neu begonnen. Zu diesem Zeitpunkt können dann abrupte

analoge Verläufe berechnet werden (z.B. Dreiecksspannung).
Der umgekehrte Fall tritt dann ein, wenn eine analoge Variable einen Schwellwert kreuzt. Normalerweise wird der Kreuzungspunkt nicht genau auf einem berechneten Zeitpunkt liegen. Mit der a'ABOVE(b)-Anweisung wird der Schnittpunkt genau berechnet (genauer als die kleinste Schrittweite) und als Ereignis zu der Zeitscheibe hinzugefügt. Damit können Analogsignale in digitale umgewandelt werden.

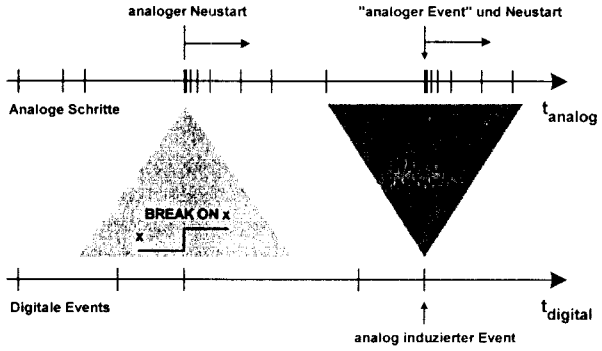


Abb. 3 Kopplung von Digital- und Analosimulator

5. Prinzipien der Modellierung

Besonders bei dem "bottom-up"-Ansatz der Modellierung ist es wichtig, die relevanten Eigenschaften und Parameter der zu modellierenden Schaltung zu erkennen. Gewöhnlich interessiert nicht nur das Verhalten im Arbeitspunkt, sondern auch in bestimmten Extremsituationen. Deshalb können Nichtlinearitäten von Kennlinien oder Begrenzungen durch Sättigungseffekte wichtig sein. Es läuft also auf eine genaue Spezifikation der gewünschten Eigenschaften hinaus.
Im Prinzip ist ein Modell immer ein "funktionales Skelett", bei dem die Funktionen durch R, C und gesteuerte Strom- und Spannungsquellen beschrieben werden (siehe /1/). Man muß dann jeweils entscheiden, ob man stückweise stetige Kurvenstücke mit Hilfe von IF ... USE erzeugen oder abrupte Änderungen zulassen will. Der Simulator verkraftet zwar die abrupten Übergänge wegen der BREAK-Resets, es kann aber wegen der Schrittweitenverkleinerung trotzdem zu größeren Laufzeiten kommen.

6. Beispiele

Als Hauptbeispiel für eine Modellierung nach dem "bottom-up"-Ansatz wird hier eine jitter-arme Standard-PLL beschrieben, die als Schematic- und Layout-Design mit etwa 2400 Transistoren vorlag.

Strukturell besteht die PLL aus vier Einzelblöcken
- Phase-Detector,
- Loop-Filter,
- Current-Controlled-Oscillator (CCO),
- Divider,

deren Simulation auf Transistorebene (BSIM4 Modelle) für einen Einschwingvorgang etwa 20 Stunden benötigt. Ziel der Modellierung war es, vier Blöcke so zu strukturieren, daß sie im Austausch, also "pin-kompatibel" zu den Layoutblöcken simuliert werden können. Zu diesem Zweck benötigten die Einzelmodelle also rein analoge Anschlußpins. D.h., interne digitale Signale sind noch intern D-A zu wandeln.
Bei dem Phase-Detector führt dieses Vorgehen sicher zu mehr Simulationsaufwand, da er eigentlich nur digitale Eingänge (fref und clkn) und digitale Ausgänge (up und down) hat.

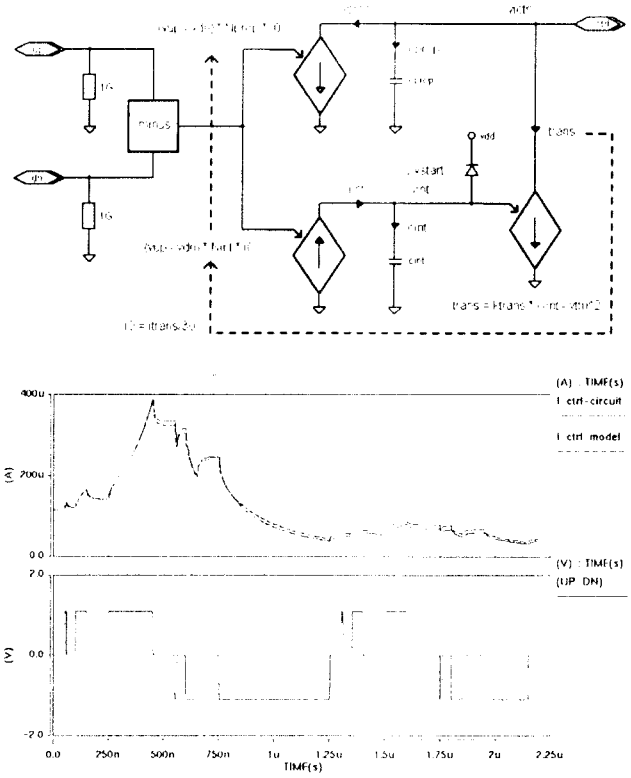


Abb. 4 Blockschalbild des Loop-Filter-Modells und Vergleich mit Transistorsimulation

Das Loop-Filter-Modell bewertet die Differenz der up- und down-Signale und steuert damit Stromquellen (positiv und negativ), die als Proportionalanteil und als Integralanteil aufsummiert werden. Die Transferkennlinie des Wandlers der Integralspannung zum Steuerstrom wird über zwei Parameter mit einer quadratischen Funktion beschrieben (entspricht Ids(Vgs)-Kennlinie). Die Anlage der Schaltung zur Selbstkompensation führt zu einer auf das Loop-Filter beschränkten

Mitkopplung. In Abb. 4 ist ein Blockschaltbild und die gute Übereinstimmung von Modell und Transistorsimulation dargestellt.
Der CCO wird durch eine einfache Sinusspannungsquelle realisiert, deren Frequenz vom Kontrollstrom aus dem Loop-Filter gesteuert wird. Die Nichtlinearität der Strom-Freuenz-Kennlinie wird durch ein Polynom 2. Grades abgebildet, wobei die Parameter direkt aus drei Wertepaaren berechnet werden. Ähnlich wird mit der Nichtlinearität der Zeitkonstante verfahren, mit der der Oszillator auf Stromänderungen reagiert. Die recht aufwendige Amplitudenstabilisierung des Oszillators wird im Modell nicht nachgebildet, da für die PLL-Funktion nur die Periodendauer, d.h. die Nulldurchgänge, benötigt werden. Diese Überlegung erlaubt auch eine Reduktion des CCO auf einen digitalen Pulsgenerator, dessen Periode über die Frequenz berechnet wird.

Der Divider ist eine rein digitale Schaltung, der die Oszillatorfrequenz mit einstellbarem Teiler reduziert.

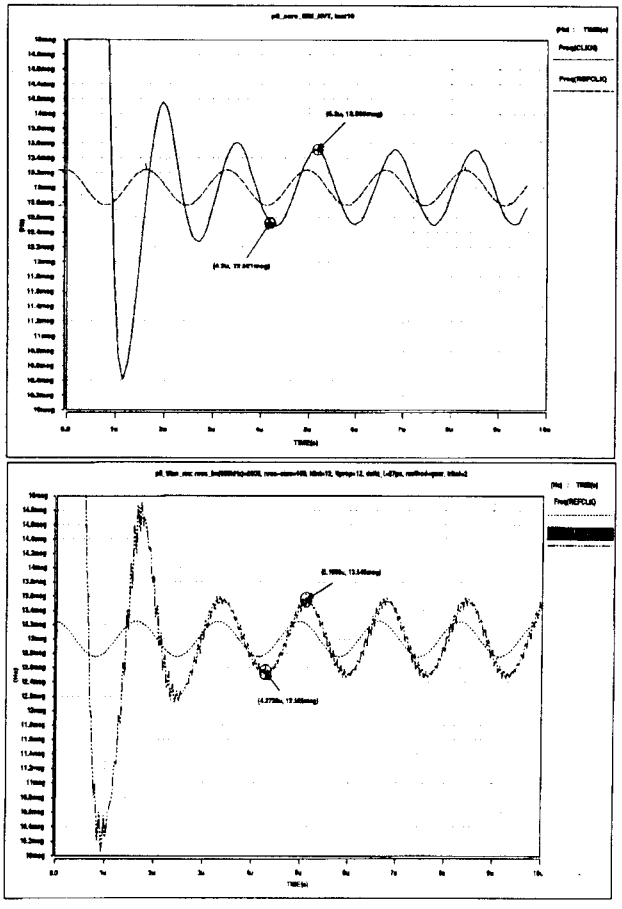


Abb. 5 Vergleich von Transitorsimulation (oben) und Modelsimulation (unten)

Die als GENERIC übergebenen Parameter stellen zum Teil Betriebsbedingungen dar (Vdd, Vss), feste Designgrößen (Kapazitäten der Filter) aber auch Charakterisierungsdaten, mit denen die Nichtlinearitäten erzeugt werden. Es stellte sich heraus, daß die gesamte PLL mit 8 Werten des letzteren Typs komplett modelliert werden kann.
Zum Beispiel spielen die digitalen Laufzeiten des Phase-Detectors und des Dividers für die Funktion nur eine untergeordnete Rolle.

In Abb. 5 ist ein Vergleich von Transistor- und Modellsimulation wiedergegeben. Die Referenzfrequenz von etwa 13 MHz ist mit einer Störung von 600 kHz beaufschlagt. Es kann der Zieh- und der Fangvorgang verfolgt werden. Die Störfrequenz ist offensichtlich zu hoch, so daß es zu einem konstanten Phasenachlauf kommt. Das "Rauschen der Simulationskurve ist lediglich durch die Meßmethode bei der ungeteilten Frequenz bedingt. Es wird von diesem Modell nicht erwartet, daß das Einschwingen des Oszillators und damit das Startverhalten der PLL genau reproduziert wird.

Die Portierung der Modelle auf ADVanceMS verlief völlig reibungslos. Die folgende Tabelle zeigt den Vergleich der Simulationszeiten (CPU) bei verschiedenen Simulatoren und Optimierungsgraden des Modells:

Typ	Pins	Oszill.	Simulator	CPU
Transistoren	analog		Titan	20 h
AMS-Modell	analog	Sinus	Titan	480 s
AMS-Modell	analog	digital	Titan	40 s
AMS-Modell	ana./dig.	digital	ADv.MS	2 s
Maximal Beschleunigung				* 36.000

7. Zusammenfassung

Das Ergebnis stellt sich folgendermaßen dar:

- VHDL-AMS eignet sich besonders gut für die Modellierung und Simulation von Mixed-Signal-Schaltungen in IC-Designs
- Die Titan-Implementierung von VHDL-AMS erlaubt nur Module mit konservativen Klemmen
- Parametrisierung erlaubt flexible Anpassung
- Die nötigen Nichtlinearitäten sind modellierbar
- Für kurze Rechenzeiten sollte es minimale analoge Anteile und eine minimale Anzahl von A-D- und D-A-Synchronisationspunkten geben
- Für einen "bottom-up"-Ansatz ist eine automatische Charakterisierung unabdingbar
- VHDL-AMS erscheint auch sehr gut geeignet für Wiederverwendung von Mixed-Signal-Design-Lösungen in "top-down"-Abläufen.

8. Literatur

/1/ Chua, L.O., Lin, P.-M. Computer-aided analysis of electronic cvircuits, Prentice-Hall 1975

Fachhochschule Aalen
Fachbereich Elektronik / Technische Informatik
Beethovenstrasse 1
73430 Aalen

Entwicklung von 8085 VHDL Modellen:

Interrupt Logik und Registerblock mit der Register-Steuerung

Studienarbeit von
Marco Jassmann
Betreuender Professor:
Prof. Dr.-Ing. Manfred Bartel

1 Einleitung

1.1 Stand zu Beginn der Studienarbeit

Im Rahmen der Diplomarbeit „Entwicklung eines Simulationsmodells des Prozessors 8085 in VHDL[1] mit modernen Entwicklungsmethoden“ (Bearbeiter Markus Fidelak [siehe Workshop Juli 2000, Ulm]) wurde die Grundlage für diese Studienarbeit gelegt. Bei diesem Projekt wurden die Blöcke „Timing and Control“ (Zeit- und Ablaufsteuerung), die „Arithmetic Logic“ Unit (ALU, arithmetisch-logische Einheit) sowie den Block „Instrucion Decoder And Machine Cycle Encoding“ entworfen (siehe Abbildung 1.1).

[1] VHDL = VHSIC HDL = very high speed integrated circuit hardware description language

Die Diplomarbeit von Herrn Fidelak wurde im EDA-Zentrum der Fachhochschule Aalen mit Hilfe von Mentor Graphics EDA-Software entwickelt. Für die Weiterentwicklung der 8085 Prozessorarchitektur war es daher sinnvoll, dieselben Werkzeuge einzusetzen. Hierbei handelt es sich um das Programmpaket FPGA Advantage.

1.2 Projektübersicht

Im folgenden wird aufgegliedert, welche Funktionsblöcke des 8085-Mikroprozessors bereits erstellt wurden und welche noch zu erstellen waren, beziehungsweise welche Funktionsblöcke im Rahmen meiner Studienarbeit erstellt wurden. Um dies besser darzustellen, wurden die entsprechenden Funktionsblöcke farblich markiert.

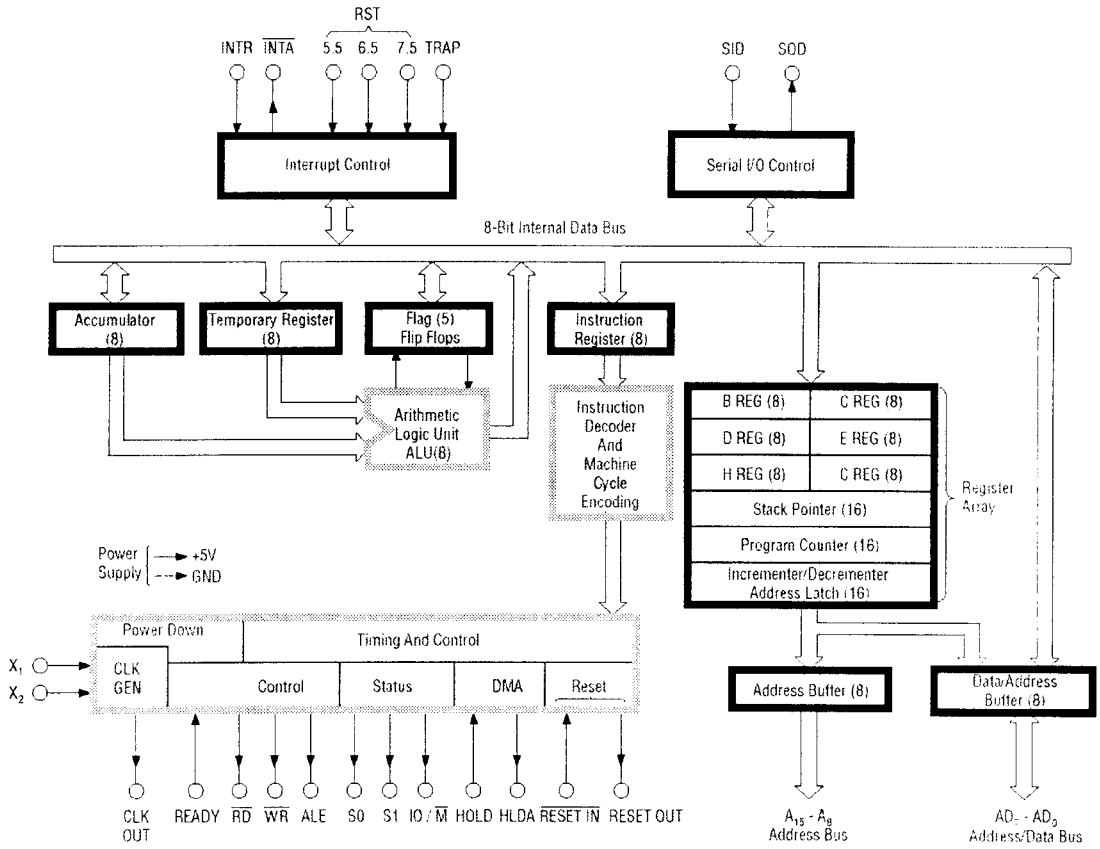


Abbildung 1.1: Blockschaltbild 8085 Mikroprozessor

Die in Abbildung 1.1 mit grau markierten Funktionsblöcke stammen aus der Diplomarbeit von Herrn Markus Fidelak, die schwarz hinterlegten Funktionsblöcke wurden im Rahmen dieser Studienarbeit erstellt.

Alle erstellten Blöcke müssen Busanschlüsse eine Breite von 8 Bit aufweisen, um der 8 Bit Architektur des 8085 zu entsprechen. Bei Komponenten mit bidirektionalem Buszugriff muss die Schreibelogik abschaltbar sein, um einen hochohmigen Ausgangszustand einschalten zu können.

Der gemultiplexte Adress- und Datenbus ist ebenfalls zu realisieren, so dass mit 8 Bit Daten und 16 Bit Adressen gearbeitet werden kann, ebenso die entsprechenden Steuerleitungen für die entworfenen Blöcke.

2 Vorgehensweise beim Entwurf der Funktionsblöcke

Nach der Feststellung, welche Funktionsblöcke noch zu entwerfen sind, mussten die bereits erstellten Funktionsblöcke analysiert werden. Hierzu war es notwendig die Blöcke bzw. die Bibliothek VHDL8085 von Herrn Fidelak in das Programm Renoir einzubinden und alle Blöcke intensiv zu analysieren.

Durch das Lesen zahlreicher Literatur und den Kenntnissen aus der Vorlesung Mikrorechnertechnik von Herrn Prof. Dr. Ing. Bartel konnte dann mit der intensiven Einarbeitung in die Bibliothek VHDL8085 begonnen werden. Die genaue Analyse des Prozessorkernes vermittelte die nötigen Kenntnisse wie man die Bibliothek erweitern kann. Als positiver Nebeneffekt lernt man auch das Programm Renoir und das Simulation-Tool ModelSIM besser kennen.

Die Simulationsergebnisse präsentierten einen vollkommenen isolierten Prozessorkern welcher für sich alleine nahezu perfekt funktioniert. Durch genauere Analyse folgte dann die Feststellung, dass die Anbindung anderer Schaltwerke sich etwas schwieriger gestalten wird, als bisher angenommen. Zum Beispiel werden die 8 Bit Befehle nicht von den bereits generierten Schaltwerken voll auskodiert, was aber zwingend notwendig ist um die anderen, noch zu entwerfenden Schaltwerke anzubinden.

Um eine Idee zu bekommen, wie ein Mikroprozessor intern aufgebaut werden kann, muss man wie bereits erwähnt, sehr viel Zeit in das Lesen entsprechender Fachliteratur verwenden. Eine große Hilfe war das Datenbuch der Firma Siemens „Mikrocomputer Bausteine – Mikroprozessor-System SAB8085" aus dem Jahre 1980/1981. In diesem Buch wird jeder mögliche Befehl des Mikroprozessors bis ins Detail Bit für Bit erklärt, woraus man sich gewisse Grundprinzipien ableiten konnte.

Mit diesen neu erworbenen Kenntnissen begann ich dann zuerst den Interrupt-Controller zu entwerfen. Doch hier tauchten bereits die ersten Probleme auf. In dem Datenbuch von Siemens taucht eine prinzipielle Beschreibung des internen Aufbaus dieses Schaltwerkes auf. Doch leider kann man diese Beschreibungsvariante nicht direkt benutzen, da es eben nur einen prinzipiellen Aufbau des Interrupt-Controllers widerspiegelt.

Die ersten Ansätze bestanden darin, alle möglichen Teilschaltwerke in irgendeiner Art und Weise als graphisches Eingabemodul zu programmieren. Zu meinem Bedauern funktionierte dies leider nicht, da manche Konstrukte z.B. als Flow Chart schlicht nicht in synthetisierbaren VHDL-Kode umsetzbar sind. Also musste eine andere Beschreibungsart eingesetzt werden, um diese Grundfunktionen zu beschreiben. Nach mehreren erfolglosen Versuchen den Interrupt-Controller zu beschreiben, bin ich nach intensiver Recherche im Internet auf ein ähnliches Projekt wie meines gestoßen. Hierbei handelt es sich um das GL85-Projekt der technischen Universität von Virginia aus den USA. Der freie Download kann bei der Universität Hamburg direkt erfolgen, da die amerikanische Adresse nicht mehr aktuell ist und mittlerweile das Projekt von deren Homepage verschwunden ist. Genauere Analysen dieses Projektes vermittelten mir die Kenntnisse, wie man die noch fehlenden Schaltwerke aufbauen kann. Es ist in Renoir auch möglich, direkte VHDL-Anweisungen in Embedded Block's[2] einzufügen und diese wiederum mit anderen Blöcken zu verbinden. Diese Funktion weist sehr viele Vorteile gegenüber den

[2] Embedded Block = Eingebetteter Block = Wird direkt in die Architecture mit eingebunden, d.h. es wird kein separater Quelltext erzeugt.

anderen Eingabewerkzeugen auf. Somit wird es möglich, logische Funktionsgleichungen direkt mit in die Blöcke einfließen zu lassen. Gewisse Grundbausteine, die benötigt werden um z.B. Speicher / Register zu realisieren wurden entweder direkt in VHDL beschrieben oder als Flow Charts entworfen und als eigenständige Bibliothekselemente abgespeichert.

Die stellenweise benutzen Logikgleichungen können in einigen Fällen direkt aus Zeitschnittdiagrammen von diversen Fachbüchern abgeleitet werden, wie es zum Beispiel in dem Schaltwerk für die Interruptsteuerung der Fall ist. Ebenso sind bereits vorgefertigte Logikgleichungen vorzufinden, wenn Bezug auf gewisse interne Funktionen genommen wird, sei es bei der Programmierung oder bei Versuchen, die einzelnen Schaltwerke zu erklären.

Um die Logikgleichungen auch zeitlich richtig darzustellen, mussten zahlreiche interne Steuerleitungen gebildet werden. Die Anzahl der internen Steuerleitungen und speichernden Knoten übersteigen die tatsächlich vorhandenen äußeren Anschlussleitungen um ein Vielfaches, da für nahezu jede Funktion die intern ausgeführt werden soll, eine speziell generierte Steuerleitung verantwortlich ist. Aus diesem Grund ist es zunächst nötig, alle möglichen Befehlskombinationen auszukodieren. Es ergibt sich ein 20 Bit breiter interner Datenbus aus welchem bis zu 2^{20}, das heißt 1.048.576 mögliche internen Steuerleitungen generiert werden können. Natürlich werden nicht alle genutzt, da eine gewisse Redundanz vorhanden ist und einige Kombinationen schlichtweg keinen Sinn ergeben, beziehungsweise für die Auskodierung von Pufferzonen benutzt werden. Doch dazu mehr bei dem entsprechenden Abschnitt Befehlsauskodierung).

Nach und nach hat sich dann ein Zustand eingestellt, welcher immer tiefere Einblicke in die Funktionsweise beziehungsweise in den internen Aufbau der Schaltwerke eines Mikroprozessors geboten hat. Die neu erworbene Denkstruktur zieht sich durch die gesamten entworfenen Schaltwerke dieser Studienarbeit, und nach und nach wurde ein immer größeres Problem bekannt : Die Anpassung an den bereits vorhandenen Prozessorkern wird sich sehr schwierig beziehungsweise trickreich gestalten, da die Schnittstelle des Prozessorkernes aus der Diplomarbeit sehr eingeschränkt ist.

3 Testbarkeit der entworfenen Prozessorteile

Die entworfenen Schaltwerke können mit dem Programm ModelSIM von Mentor Graphics auf ihre Funktion hin überprüft werden. Es handelt sich hierbei lediglich um einen theoretischen Test, d.h. das reale Verhalten der Schaltwerke in einem FPGA ist noch nicht bewiesen. Dort können Fehler auftreten, die man nicht voraussehen kann, da es zu ungewollten Zeitverzögerungen kommen kann. Zwar ist in LeonardoSpectrum ein entsprechendes „Place and Route"-Werkzeug integriert, aber die Möglichkeit besteht, dass die Schaltung trotz korrekter Simulation real nicht korrekt funktioniert.

Um einen schnellen und einfachen Test realisieren zu können, war ein Testboard mit einem entsprechenden FPGA von Nöten. Der Kauf eines Evalution-Boardes hätte wohl die preislichen Vorstellungen aller an dem Projekt beteiligten überschritten. Die im EDA-Zentrum der FH Aalen vorhanden Programmierboards sind für den schnellen Test eher nicht geeignet, da diese Board zu umfangreich aufgebaut sind. Natürlich kann es von Vorteil sein, wenn auf einem Experimentierboard ein zweiter FPGA integriert ist und diverse Taster und Anzeigen, doch genau dadurch wird die Arbeit erschwert. Man kommt nicht mehr so einfach an die einzelnen Ausgangspins des

FPGA heran. Hinzu kommt, dass auf den meisten Evalutionboards keine JTAG-Schnittstelle integriert ist, diese aber von Vorteil bezüglich der Programmierung ist.

Deshalb stand der Entschluss fest, sich selbst ein kleines Board zu produzieren. Nach einem kurzen Blick in die Datenblätter von XILINX-FPGA-Bausteinen, konnte man sofort ein kleines Board entwickeln. Ebenso ist ein paralleler Programmieradapter nach dem Vorbild einer XILINX-Bauanleitung entstanden.

Wenn man sich entschließt, die Schaltwerke unter realen Bedingungen zu testen, das heißt auf dem FPGA, müssen die einzelnen Schaltwerke synthetisiert werden und in den FPGA heruntergeladen werden. Dies geschieht mit dem Programmteil LeonardoSpectrum aus dem Programmpaket FPGA-Advantage.

Hierbei können weitere Probleme auftreten, da sich bestimmte Konstrukte aus VHDL partout nicht synthetisieren lassen, wie diverse Versuche an einem Testboard ergaben. Bei den in der Studienarbeit erstellten Schaltwerken wird dies sicherlich nicht der Fall sein, da auf derartige Konstrukte (unnötig komplizierte Flow Charts, State Diagramms und Truthtables) bewusst verzichtet wurde.

4 Steuerung der Funktionsblöcke untereinander

Um die einzelnen Register, welche sich in den Schaltwerken Registerarray, Instruktionsregister und Daten-Adress-Speicher (siehe Abbildung 1.1) befinden, anzusprechen, beziehungsweise ihre vorgesehenen Funktion auszuführen, benötigt man ein steuerndes Schaltwerk.

Wenn man sich das Übersichtsblockschaltbild (Abbildung 1.1) des 8085 genauer betrachtet, wird ein Schaltwerk dieser Art unter Timing und Control dargestellt. Ungeachtet dessen sind aber keinerlei Leitungen eingezeichnet. Erst nach intensiver Befehlssatzanalyse des 8085 Mikroprozessors wird es möglich, Kontrollschaltwerke zu entwickeln welche die gesamte Steuerung übernehmen,

Eine Übersicht über die Komplexität der Register-Kontroll-Logik kann der Abbildung 4.1 entnommen werden. Auf dieser Abbildung sind sämtliche Steuerschaltwerke für die einzelnen Funktionsblöcke zu sehen. Jeder Funktionsblock stellt jedoch nur einen Übergeordneten Block dar und ist in weitere Unterblöcke unterteilt.

Die Steueraufgabe, die dieses Schaltwerk auszuüben hat, ist sehr komplex, kann aber durch logisches überlegen und taktisches Vorgehen relativ schnell begriffen werden. Man muss sich, um Schaltwerke dieser Art entwerfen zu können, sehr genau an die Datenblätter des Mikroprozessors halten, speziell was die Programmierung betrifft. Durch die Programmierbefehle lassen sich sehr explizite Rückschlüsse über den internen Aufbau einzelner Teilschaltwerke ziehen. Dazu ist es notwendig, alle Befehle intern zu kodieren, was mit dem Instruktionsregister bereits erledigt wurde.

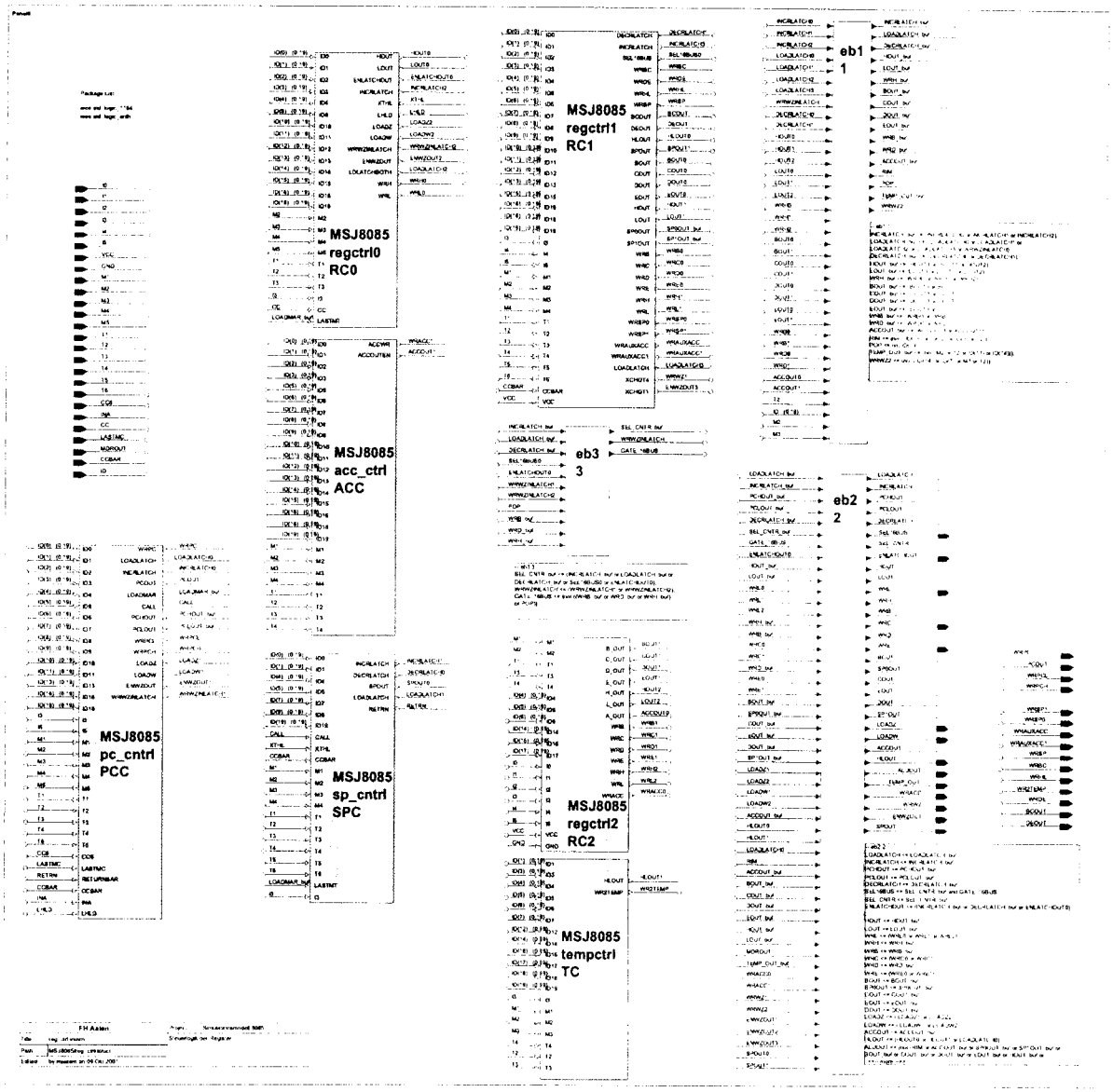


Abbildung 4.1: Interner Aufbau von reg_ctrl

Nebenbei sei erwähnt, dass sich die einzelnen Unterblöcke größtenteils aus Embedded Blöcken zusammensetzen. In diesen Embedded Blöcken sind die Steueraufgaben in reiner Logik-Beschreibung hinterlegt.

5 Befehlsauskodierung

Dadurch, dass der Befehlssatz des 8085 Mikroprozessors bereits nach Funktionen und Ausführbefehlen angeordnet ist, sind diese Merkmale unweigerlich auch in dem binären Kode eines Befehls enthalten. Dies war das Ziel bei der Entwicklung der Befehlssatz-Architektur durch die Firma Intel.

Untersucht man die Befehle nach ihrem Binärkode so stellt man fest, dass die oberen 2 Bit des 8 Bit Befehls bereits eine Voreinteilung in unterschiedliche Befehlsgruppen darstellen. Dies kann besonders gut einer Tabelle entnommen werden, welche in der Diplomarbeit von Herrn Fidelak verwendet wurde (Seite 99 in der Ausarbeitung seiner Diplomarbeit). Ebenso ist die Tabelle in dem Vorlesungsskript der Vorlesung Mikrorechnertechnik an der FH Aalen enthalten. Einen kleinen Auszug der Tabelle finden sie in der folgenden Tabelle 5.1 wieder.

	x0	x1	x2	x3	x4	x5	x6	x7	
0x	NOP	LXI B	STAX B	INX B	INR B	DCR B	MVI B	RLC	0x
1x		LXI D	STAX D	INX D	INR D	DCR D	MVI D	RAL	1x
2x	RIM	LXI H	SHLD	INX H	INR H	DCR H	MVI H	DAA	2x
3x	SIM	LXI SP	STA	INX SP	INR M	DCR M	MVI M	STC	3x
4x	MOV B,B	MOV B,C	MOV B,D	MOV B,E	MOV B,H	MOV B,L	MOV B,M	MOV B,A	4x
5x	MOV D,B	MOV D,C	MOV D,D	MOV D,E	MOV D,H	MOV D,L	MOV D,M	MOV D,A	5x
6x	MOV H,B	MOV H,C	MOV H,D	MOV H,E	MOV H,H	MOV H,L	MOV H,M	MOV H,A	6x
7x	MOV M,B	MOV M,C	MOV M,D	MOV M,E	MOV M,H	MOV M,L	HLT	MOV M,A	7x
8x	ADD B	ADDC	ADD D	ADD E	ADD H	ADD L	ADD M	ADDA	8x
9x	SUB B	SUB C	SUB D	SUB E	SUB H	SUB L	SUB M	SUB A	9x
Ax	ANA B	ANA C	ANA D	ANA E	ANA H	ANA L	ANA M	ANA A	Ax
Bx	ORA B	ORA C	ORA D	ORA E	ORA H	ORA L	ORA M	ORA A	Bx
Cx	RNZ	POP B	JNZ	JMP	CNZ	PUSH B	ADI	RST 0	Cx
Dx	RNC	POP D	JNC	OUT	CNC	PUSH D	SUI	RST 2	Dx
Ex	RPO	POP H	JPO	XTHL	CPO	PUSH H	ANI	RST 4	Ex
Fx	RP	POP PSW	JP	DI	CP	PUSH PSW	ORI	RST 6	Fx
	x0	x1	x2	x3	x4	x5	x6	x7	

Tabelle 5.1: Befehle x0-x7 (Auszug)

So ähnlich verhält es sich auch mit den restlichen 6 Bit des 8 Bit Befehls. Diese kennzeichnen zu einem gewissen Grad das beeinflusste Register in Abhängigkeit der obersten 2 Bit.

Somit ergibt sich zwangsläufig eine Busbreite von 20 Bit, denn mit 2 Bit können 4 unterschiedliche Signalzustände erzeugt werden, mit 3 Bit sind es 8 verschiedene. Somit ergibt sich bei einer Aufgliederung von 2:3:3 eine Möglichkeitsliste von 4:8:8 was letztendlich 4+8+8 = 20 Bit ergibt.

Ein Bus mit der Breite von 20 Bit würde exakt 2^{20} = 1.048.576 mögliche Kodierungszustände zulassen. Effektiv werden davon aber nur 256 Kombinationen genutzt. Dies hat mehrere Gründe. Zum einen wird hiermit vermieden, dass ähnliche Bit-Kombinationen, welche unterschiedliche Befehle ausführen, miteinander verwechselt werden. Somit hat man aus der Redundanz, welche hieraus resultiert, eine Art Sicherheitsabstand gewonnen, was auch als Hamming-Distanz bezeichnet wird. Zum anderen wird gewährleistet, dass bestimmte ID-Bits nur für bestimmte, vordefinierte Zwecke gültig sind, das heißt man hat eine eindeutige Zuordnung zu den Befehlsgruppen hergestellt.

Im folgenden wird die Dekodiertabelle für den ID-Bus auszugsweise dargestellt. Sinnvoller Weise ist die Tabelle des binären Kodes nach geordnet, da schon aus der Aufschlüsselung die entsprechenden Dekodiervorgänge ersichtlich sind.

Freie Felder in der Tabelle weisen darauf hin, dass diese Zustände noch frei belegbar sind. Die Bereiche werden als Hidden Instructions bezeichnet, und sind in diesem Projekt noch frei benutzbar. Bei einem realen 8085 Mikroprozessor sind die Bereiche entweder gesperrt oder schon durch den Assembler nicht zugänglich gemacht.

Befehl	Hex	ID-Bus																			
		ID 19	ID 18	ID 17	ID 16	ID 15	ID 14	ID 13	ID 12	ID 11	ID 10	ID 9	ID 8	ID 7	ID 6	ID 5	ID 4	ID 3	ID 2	ID 1	ID 0
NOP	00	1	1	1	0	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	0
LXI B	01	1	1	1	0	1	1	1	1	1	1	1	0	1	1	1	1	1	1	0	1
STAX B	02	1	1	1	0	1	1	1	1	1	1	1	0	1	1	1	1	1	0	1	1
--------	----	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
MOV E,B	58	1	1	0	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	0
MOV E,C	59	1	1	0	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	0	1
--------	----	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
ADC A	8F	1	0	1	1	1	1	1	1	1	1	0	1	0	1	1	1	1	1	1	1
SUB B	90	1	0	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	0
--------	----	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
XRA A	AF	1	0	1	1	1	1														
ORA B	B0	1	1	0	0	0	0														
--------	----	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
	FD	0	1	1	1	0	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1
CPI	FE	0	1	1	1	0	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1
RST 7	FF	0	1	1	1	0	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1

Tabelle 5.2: Kodierungstabelle für interne Befehlskodierung (Auszug)

6 Grundlegende Schaltwerke

Das gesamt Projekt besteht aus vielen, kleinen Einzelschaltwerken, welche immer wieder benutzt wurden. Deren Funktion und Synthetisierbarkeit ist einwandfrei.

Dadurch, dass man sich grundlegende Gedanken über den Aufbau eines Mikroprozessors gemacht hat, musste man zwar viel Zeit in den Entwurf kleiner Unterschaltwerke stecke, doch später bei deren Verwendung wurde enorm Zeit eingespart.

Um beispielsweise eine 16 Bit Registerbank aufzubauen, wurde ein 8 Bit Register erstellt. Dieses 8 Bit Register wiederum wurde aus einzelnen FlipFlops, (direkt in VHDL programmiert) zusammengefügt, welche sich ohne Probleme synthetisieren ließen.

Selbst die komplexesten Schaltwerke lassen sich auf eine Vielzahl einfacher Schaltwerke herunter brechen und mit einfachen grundlegenden Schaltwerken wie Multiplexer oder Dekodierer darstellen.

So ist es nicht verwunderlich, dass man die Schaltwerke in verschiedene Gruppen einteilen muss. In der folgenden Tabelle sind die am häufigsten verwendeten grundlegenden Schaltwerke aufgeführt:

Speicherbeschreibende Elemente	• D-Flip Flop • Registerbank 8 Bit und 16 Bit
Logikbeschreibende Elemente	• Multiplexer (2:1 und 4:1) • DeKoder (4 aus 2 und 8 aus 3)
Sonstige funktionale Elemente	• Busaufschaltelement • Zähler • Prioritätsencoder • Selektionselemente

Tabelle 6.1 : Grundlegende Elemente

7 Schlußwort

Die gesamten geforderten Schaltwerke wurden in der Studienarbeit erstellt. Leider fehlte die Zeit, den Prozessor komplett in einen FPGA downzuloaden. Die Funktion der einzelnen Schaltwerke konnte jedoch erfolgreich in einem kleinen FPGA-Board getestet werden, bis auf den Teil der Diplomarbeit. Hierbei handelt es sich in der Tat nur um ein Simulationsmodell, welches partout nicht synthetisierbar ist.

Die Arbeit wurde von einem weiteren Studienarbeiter übernommen, welcher nun die Aufgabe hat, die Implementation der Prozessorteile auf einen FPGA zu verwirklichen.

Embedded Java - Entwurf eines einfädigen Java-Prozessors in Renoir

Diplomarbeit von: Heiko Rathgeb
Erarbeitet im: SS2001
Betreuer: Prof. Dr.-Ing. Manfred Bartel

1 Einleitung

1.1 Was ist Java?

Java ist eine objektorientierte, C-ähnliche Programmiersprache, die auf den Vorteilen des WorldWideWebs aufbaut, jedoch völlig neue Fähigkeiten hinzufügt, wie z.B. Interaktivität... Das Erfolgskonzept von Java beruht dabei auf seiner Plattformunabhängigkeit.

In Abbildung 1 ist eine plattformabhängige Programmierung bzw. Kompilierung (wie z.B. unter C) dargestellt. Hierbei benötigt man für jede Rechnerarchitektur einen eigens für diese Architektur geschriebenen Compiler. Wir sehen hier ein Beispiel mit drei verschiedenen Rechnerarchitekturen.

Kompilieren wir unseren Code nun für einen Pentium-Rechner, ist dieses Binary File auf keinem der beiden anderen Architekturen (PowerPC, SPARC) lauffähig und umgekehrt.

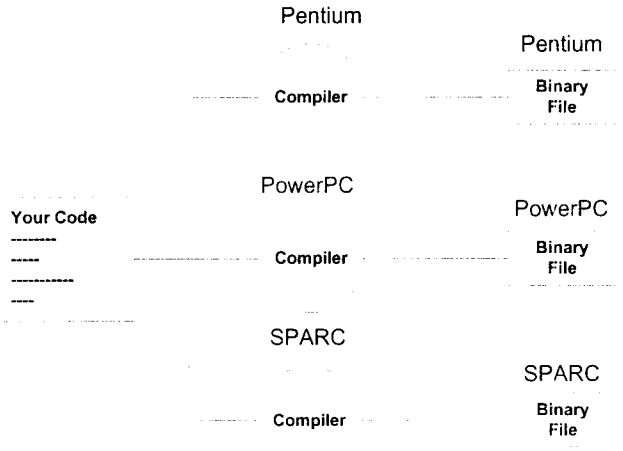


Abbildung 1: Plattformabhängige Compiler

Diese Plattformunabhängigkeit erreicht Java, indem der Compiler keinen echten Maschinencode erzeugt, sondern einen sogenannten Bytecode, dieser dient als "Pseudo-Maschinencode" für eine sogenannte "Virtual Machine". Dieser Bytecode kann sehr schnell in richtigen Maschinencode, den auch der Prozessor des Rechners versteht, umgesetzt werden. Damit ein Java-Programm auf einem realen Prozessor wie einem Intel-Pentium ablaufen kann, müssen die Befehle des virtuellen Prozessors in Befehle für den Pentium umgesetzt werden. Dies ist die Aufgabe des Java-Interpreters. Java-Programme werden also einmalig kompiliert, um den plattformunabhängigen Bytecode zu erzeugen, und bei jeder Ausführung interpretiert, um den Bytecode auf einem realen Computer auszuführen. Java-Programme können dadurch auf allen Plattformen ablaufen, für die ein Interpreter verfügbar ist.

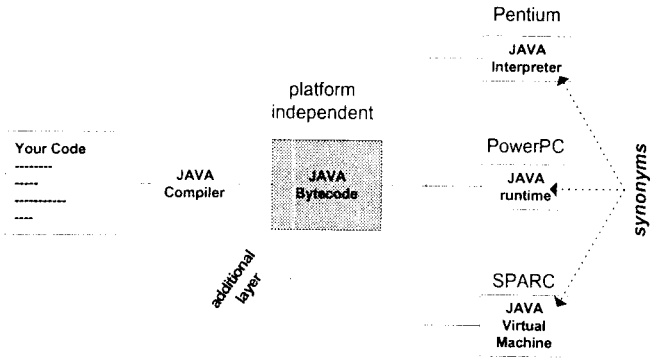


Abbildung 2: Plattformunabhängiger Compiler

1.2 Virtual Machine

Bei der virtuellen Maschine handelt es sich um eine weitere Software-Anwendung (Interpreter), die das bytecodierte Javadokument interpretiert und erst während des Programmablaufes den Code für den realen Prozessor erzeugt. Man kann sich die virtuelle Maschine als einen herkömmlichen Computer vorstellen, der in Software implementiert wurde.

1.3 Real Machine

Die Java Real Machine ist die Implementierung der Java Virtual Machine in Hardware. Vorteile sind dabei beispielsweiße eine Geschwindigkeits-Steigerung, da der Bytecode direkt, ohne "Softwarezwischenstufe" ausgeführt werden kann. Ein weiterer Vorteil ist das Einsparen von Ressourcen, wie Speicherplatz, da keine Zusatzsoftware (JVM) gespeichert werden muss. Dies ist vor allem in Kleingeräten wie Handys oder Sensoren von Vorteil.

2 Struktur des Prozessors

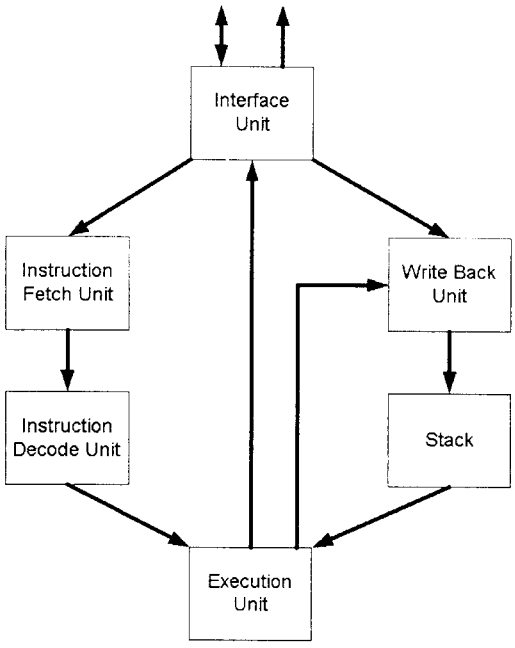


Abbildung 3: Struktur des Prozessors

In Abbildung 3 sind die Datenbusse innerhalb des Prozessors dargestellt.

2.1 Pipelining

Um die Arbeitsgeschwindigkeit eines Prozessors zu beschleunigen wird in einer Vielzahl von Prozessoren das Konzept des Pipelinings angewandt. Die grundlegende Idee ist die fast parallele Ausführung mehrerer Befehle. Beim Pipelining liegen mehrere Befehle wie auf einem Fließband hintereinander. Die auszuführenden Befehle werden in kleine Abschnitte zerlegt, in sogenannte Pipeline-Stufen. Dieses Konzept enthält vier Stufen, also eine vierstufige Pipeline.

Diese Stufen sind in einer festen Reihenfolge miteinander verbunden. Jeder Befehl tritt in der ersten Stufe in die Pipeline und verlässt sie bei der n-ten (in unserem Fall bei der vierten).

Die Abschnitte (Pipeline-Stufen) wurden in folgende Gruppen bzw. Units unterteilt:

- Befehlholstufe IF; **IFU** (Instruction Fetch Unit)
- Befehldecodierstufe ID; **IDU** (Instruction Decode Unit)
- Ausführungsstufe EXE; **EXE** (Execution Unit)
- Rückschreibstufe WB; **WBU** (Write Back Unit)

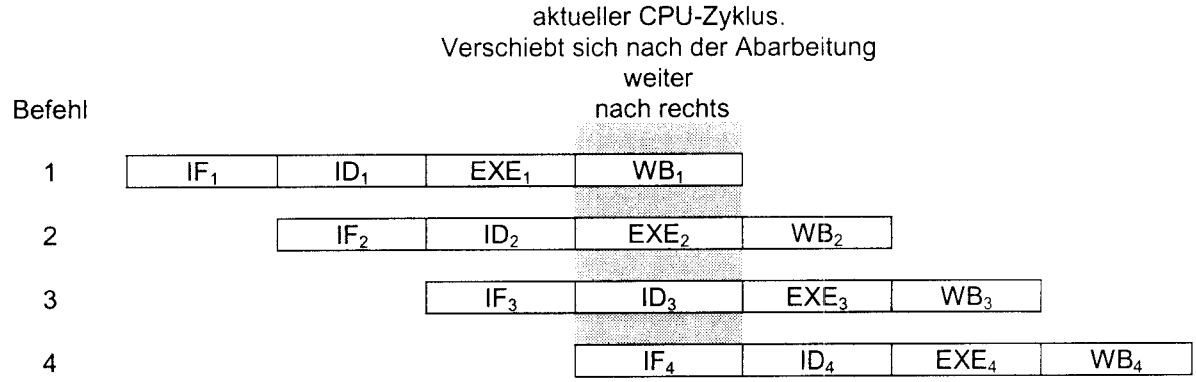


Abbildung 4: Pipelining

Ist beispielsweise die Abarbeitung der IF1 (ersten Befehl einholen) des ersten Befehls vollendet, wird beim nächsten CPU-Zyklus sofort der zweite Befehl eingeholt (IF2). Währenddessen wird bereits der erste Befehl decodiert (ID1) usw. usw.

3 Beschreibung der Units

Um die folgenden Flowcharts zu vereinfachen, wird für jede Unit die Abfrage des CLK'event und CLK="1" (Abfrage der positiven Taktflanke des Clock-Signals) als vorhanden angenommen. Diese Maßnahme gewährleistet, dass alle Schaltwerke Taktsynchron arbeiten.

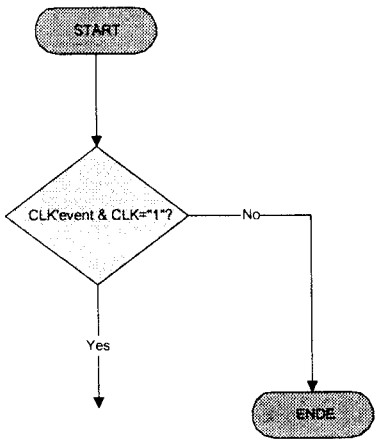


Abbildung 5: Taktabfrage

3.1 Interface Unit

3.1.1 Aufgaben der Schnittstelleneinheit

Die Interface Unit ist eine Schnittstelle zwischen externer Peripherie und internem Prozessor. Sie dient zur Ansteuerung, dem Auslesen und Beschreiben des externen Speichers. Weiterhin ist ein Clock-Eingang und ein Reset-Eingang nach außen geführt. Diese Signale und Busse sind die einzigen Signale, die später aus dem FPGA führen.

- Kommunikation mit externer Peripherie
  - Adressbus
  - Steuersignale (externer Speicher)
  - Datenbus
- Steuerung der Busprioritäten

3.1.2 Prioritäten

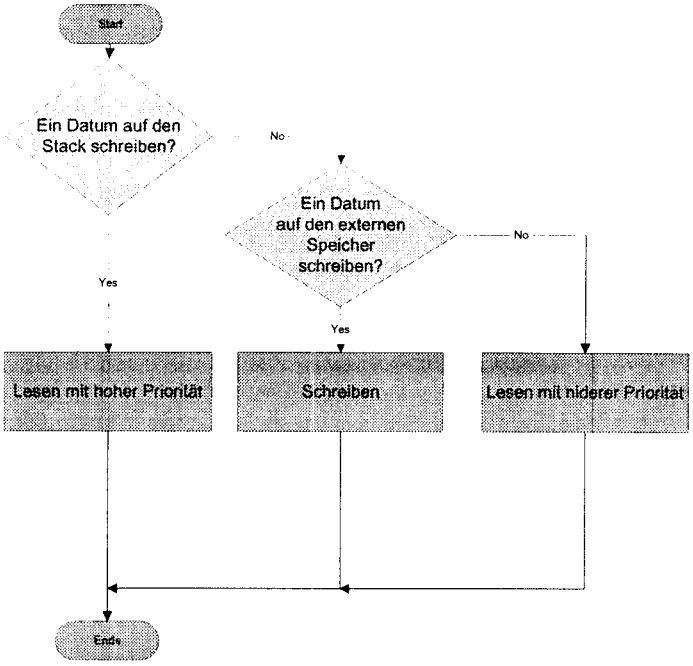


Abbildung 6: Prioritätensteuerung

Im ersten Zweig wird das Lesen, mit der höchsten Priorität ausgeführt. Dies geschieht z.B. wenn die EXE einen Befehl erhält, dass ein Datum auf den Stack geschrieben werden soll. Dieser Befehl wird bevorzugt behandelt. Eine neue "Instruction" von der IFU (Instruktion Fetch Unit) einzuholen, hat eine niedere Priorität und wird deshalb im Zweig "Lesen mit niederer Priorität" ausgeführt.

3.2 Instruction Fetch Unit

Aufgaben der Instruction Fetch Unit:

- Befehle aus dem externen Speicher holen
- Befehle in den Befehlspuffer schreiben
- Den PC (Program Counter) inkrementieren bzw.
- Nach einem Jump-Befehl dem PC (Program Counter) einen neuen Wert zuweisen

Zunächst wird überprüft, ob der Befehls-Stapelspeicher bereits "voll" ist. Ist dies der Fall, wird für diesen Takt kein neues Datum angefordert (da kein weiterer Befehl in den Befehlspuffer geschrieben werden kann).

Ist der Befehlspuffer nicht voll und das Datum am Eingang gültig, wird ein neues Datum in den Befehlspuffer übernommen und der PC (Programm Counter) wird um eins inkrementiert.

Bei einem Sprungbefehl wird die neue PC-Adresse von der EXE (Execution Unit) an die IFU (Instruction Fetch Unit) übergeben. Somit wird dem PC der neue Wert zugewiesen

3.3 Instruction Decode Unit

Aufgaben der Instruction Decode Unit:

- Bytecode identifizieren
- In Opcodes umwandeln
- Steuersignale ausgeben
- Operanden Ausgeben

Wird nun der erste Bytecode aus dem Befehlspuffer ausgelesen, wird er mit einer Tabelle verglichen und dem entsprechenden Befehl bzw. Opcode zugewiesen. In dieser Tabelle sind auch zusätzlich Informationen enthalten, die, je nach Befehl, den Ausgängen und internen Signalen Werte zuweisen.

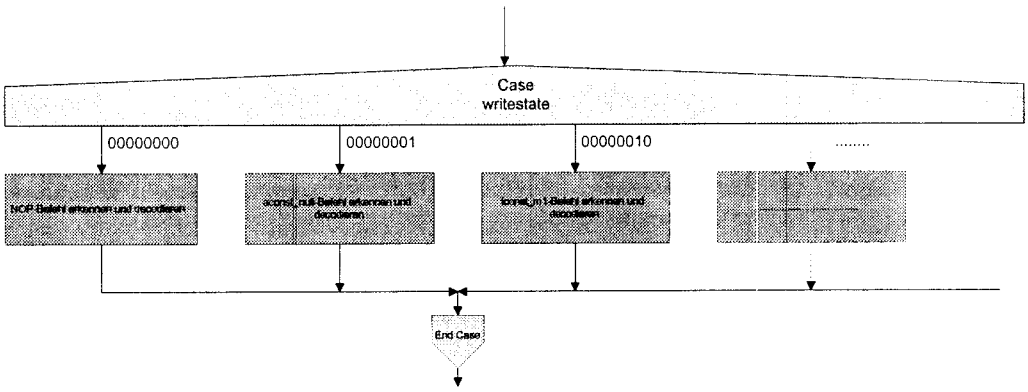


Abbildung 8: Tabelle zur Decodierung

Bei Nichterkennen eines Befehls wird nop (No Operation) ausgegeben

Kann ein Befehl (Bytecode) nicht direkt im Prozessor umgesetzt werden muss ein Microcode ausgeführt werden, der den Befehl in viele Unterbefehle teilt und diese sequentiell ausführt.

3.4 Execution Unit

Aufgaben der Execution Unit:

- Opcode identifizieren
- In Steuersignale umwandeln
- Operanden verarbeiten
- Operationen durchführen

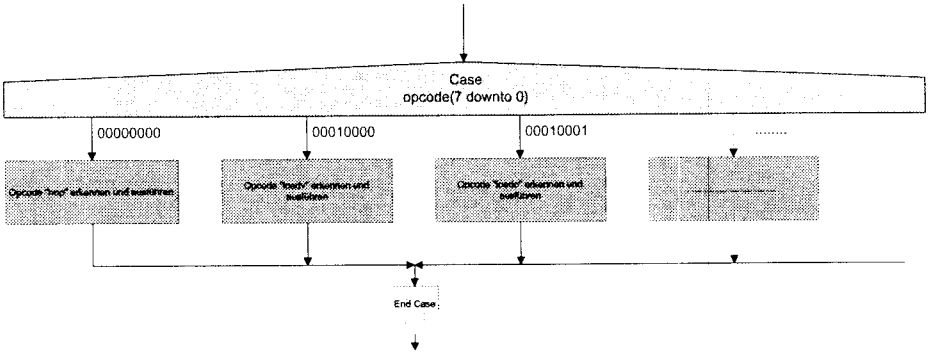


Abbildung 9: Tabelle zur Opcode-Identifikation

Hier ist die Architektur angedeutet, die in Renoir verwirklicht wurde. Es ist vergleichbar mit der Decodierung der Bytecodes in der IDU (Instruction Decode Unit).

Die EXE errechnet bei einem Jump-Befehl den neuen PC und übergibt diesen an die Instruction Fetch Unit.

3.4.1 Write Back Unit

Aufgaben der Write Back Unit:

- Daten von der EXE auf den Stack zu schreiben
- Daten vom externen Speicher auf den Stack zu schreiben

Die WBU besteht nur aus einem Prozess. Hier wird das Schreiben auf den Stapelspeicher koordiniert. Es wird gesteuert ob ein Datum zur SMU geleitet wird, welches Datum verwendet wird (vom externen Speicher oder von der EXE) und es werden Steuersignale für die Stack Unit übergeben.

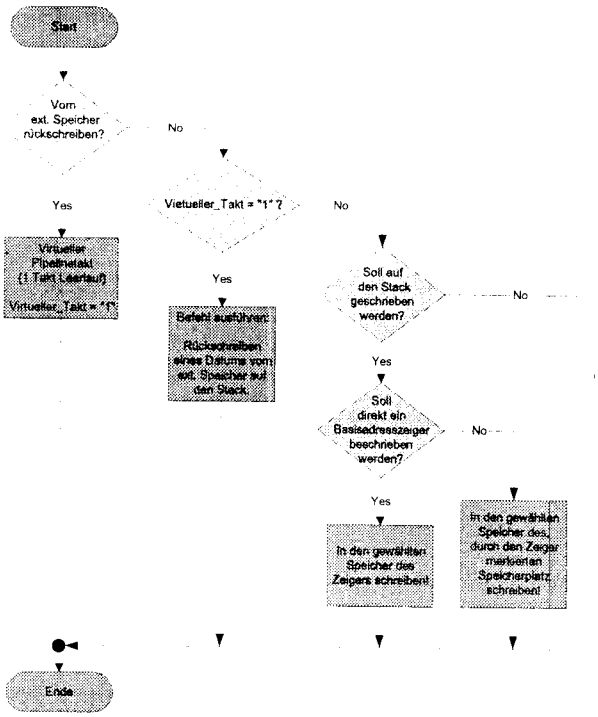


Abbildung 10: Flowchart WBU

Wenn die EXE einen Befehl erkennt, in der ein Datum vom externen Speicher auf den Stack geschrieben werden soll, durchläuft sie einen „Leerlauftakt", da das Einholen des Datums von einem externen Speicher diesen zusätzlichen Takt benötigt. In diesem Durchlauf wird ausschließlich das interne Signal Virtueller_Tak=1 gesetzt, um beim nächsten Takt den Befehl zu Ende zu führen.

Gleichzeitig wird in der EXE ein Stall-Befehl ausgegeben, der die Pipeline für einen Takt anhält.

Die nächste Abfrage überprüft nun, ob auf den Stack geschrieben werden soll und wenn ja, ob ein direktes oder indirektes Beschreiben erfolgt.

3.5 Stack Unit

Aufgaben der Stack Unit:

- Speichern von Daten
- Verwalten von Basisadresszeigern (z.B. VARS)
- Verwaltung des Speichers

Der Stack besteht wiederum aus zwei Untergruppen. Zum einen aus einem Speicher mit 16 Speicherstellen die jeweils 32 Bit breit sind. Zum anderen aus einer Ansteuerung, die den Speicher verwaltet und ihn nach außen als einen Stack erscheinen lässt.

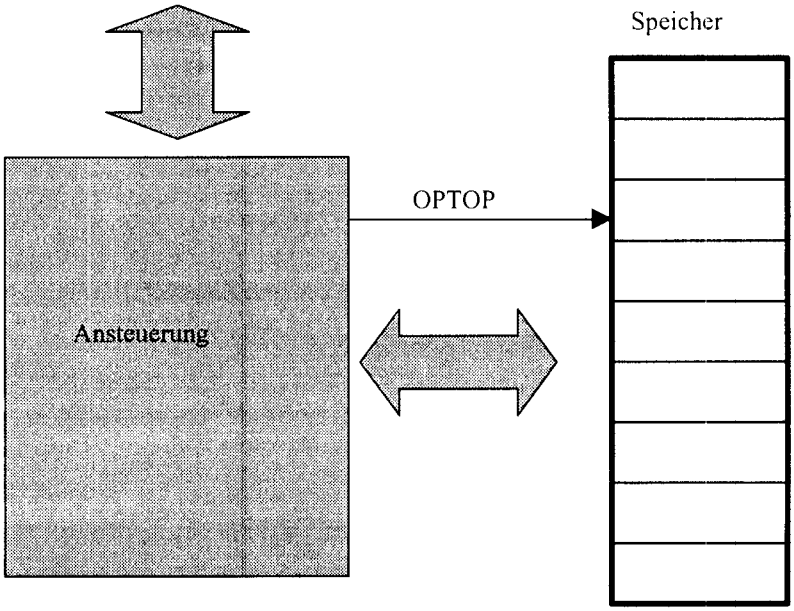


Abbildung 11: Aufbau des Stacks

4 Aussichten

- Schutz gegen Überlauf des Stacks (Stichwort: Unterlauf-, Überlaufmarke, Cache)
- Mehr Bytecode implementieren
- Mehrfädigkeit implementieren
- Instruction falding

Fachhochschule Aalen

Fachbereich Elektronik / Technische Informatik

Beethovenstrasse 1
73430 Aalen

Steuerung einer elektronischen ACDC-Last mittels FPGA

Entwicklung mit moderner EDA-Software in VHDL

Diplomarbeit von

Marco Jassmann

Betreuender Professor:

Prof. Dr.-Ing. Bernd Kohlhammer

1 Einführung, Motivation

Die Idee eines elektronisch geregelten Widerstandes, sprich einer elektronischen Last ist nichts neues. Es gibt bereits zahlreiche Hersteller von DC-Lasten, welche sehr professionell entwickelt wurden. Doch was führte dazu, einen Widerstand elektronisch nachzubilden, wird im folgenden versucht anhand von den herkömmlichen Widerstandsarten darzustellen.

1.1 Festwiderstände

Betrachtet man die Anzahl der verschiedensten Festwiderstände und deren Bauformen, so ist man zunächst ziemlich beeindruckt. Vom kleinen viertel Watt Widerstand bis hin zu den großen Leistungswiderständen gibt es zahlreiche, unterschiedlichste Modelle. Eine kleine Auswahl kann der Abbildung 1.1 entnommen werden.

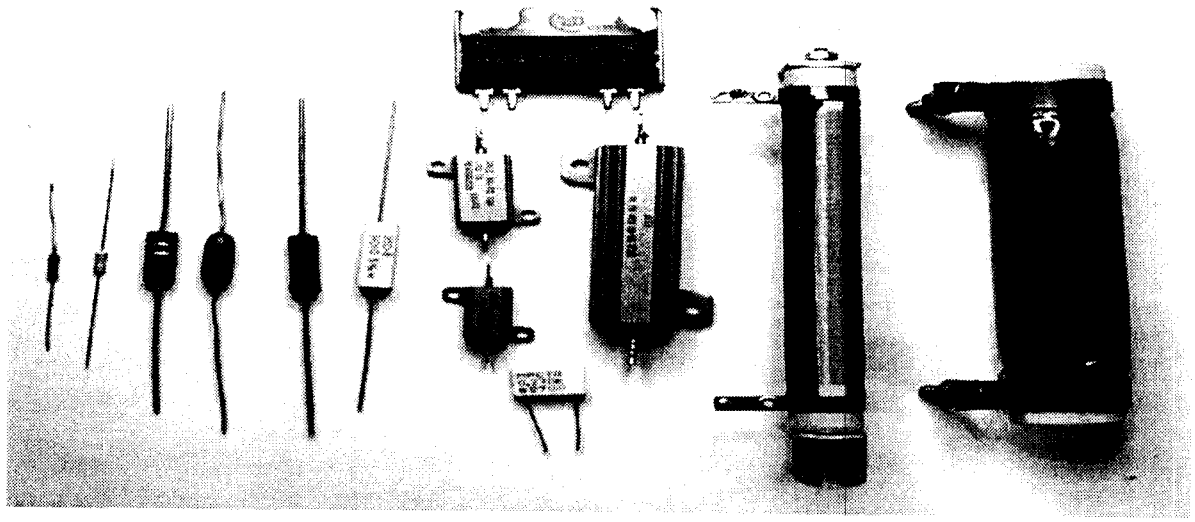


Abbildung 1.1 : Festwiderstände unterschiedlicher Bau- u. Leistungsart

Diese Widerstände finden Ihre Anwendung stets in fest berechneten Schaltungen oder Platinen. Auch kann, je nach Bauform, stellenweise eine hohe Leistung mit den Festwiderständen abgeführt werden. Es gibt sie ebenso als hochpräzise Messwiderstände oder einfach nur als Bauelement auf einer Platine um beispielsweise den Verstärkungsfaktor eines Operationsverstärkers einzustellen.

Trotzdem muss man sich mit diesen Widerständen nicht zufrieden geben. Wie der Name schon sagt, handelt es sich um Festwiderstände, d.h. deren Widerstandswert ist zum einen nicht veränderbar und zum andern nach einer genormten Widerstandsreihe (E-Reihe) fest gelegt.

Möchte man einen Leistungswiderstand in der Größenordnung von 100 Ohm in einer Schaltung oder einem Laboraufbau einsetzen, muss man zusätzlich noch mit einer relativ hohen Toleranz rechnen, welche meistens bei 10 % liegen dürfte. Da die Bauform des Widerstandes nicht linear zu der Leistung mit wächst, sondern eher quadratisch, kann man sich bei entsprechender Leistung schon mal einschlägige Gedanken machen, wo man den großen Widerstand unterbringt.

1.2 Potentiometer, Drehwiderstände

Bezüglich des Widerstandwertes sind Potentiometer und Drehwiderstände den Festwiderständen überlegen. Sie bieten unter Umständen eine wesentlich genauere Einstellmöglichkeit, abhängig davon, um welche Art eines variablen Widerstand es sich handelt.

Es gibt die hoch präzisen Spindeltrimmer, welche man zum Abgleich von beispielsweise genauen Operationsverstärkerschaltungen benutzt oder die handelsüblichen Drehscheibentrimmer für eine grobe Einstellmöglichkeit. Darüber hinaus existieren auch ohne Schraubendreher einstellbare Potentiometer, welche man hervorragend für kurzfristige Justierarbeiten benutzen kann, wie zum Beispiel zur manuellen Signalpegelanpassung.

Durch den Aufbau eines Schichtwiderstandes mit beweglichem Mittelabriff ist es sogar ohne weiteres möglich, einen linearen oder logarithmisch einstellbaren Drehwiderstand herzustellen.

Potentiometer und Drehwiderstände sind ebenso wie die Festwiderstände in allen möglichen Variationen und Bauformen zu finden, sogar als Leistungswiderstände. Einen kleine Einblick in die Welt der Potentiometer können Sie in der Abbildung 1.2 gewinnen.

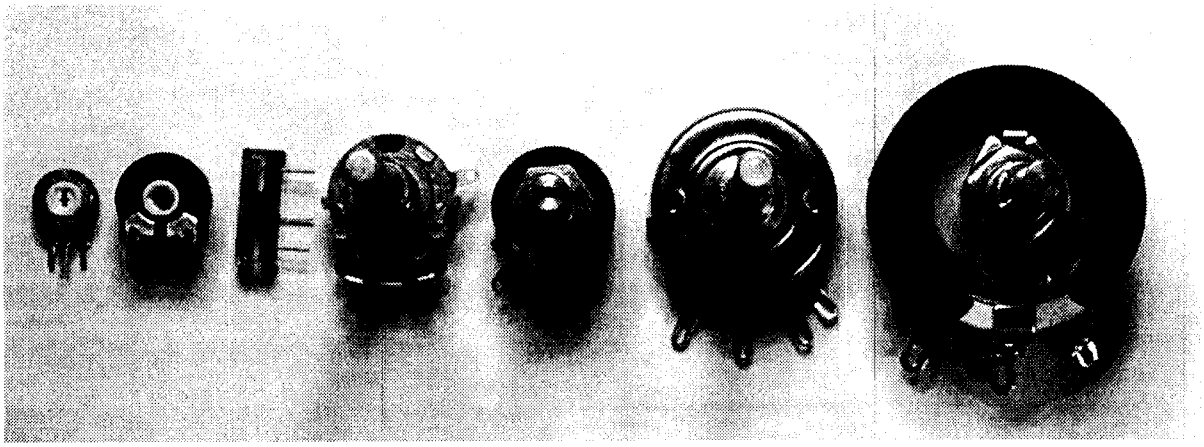


Abbildung 1.2 : Drehwiderstände mit Schicht- und Drahtaufbau

Trotz der Vielfalt dieser einstellbaren Widerstände müssen einige Abstriche gemacht werden.

Bei großen Leistungs-Drehwiderständen ist der Widerstand mittels eines Drahtes realisiert, welcher meistens um einen Keramikkern gewickelt ist. Auf einer der Stirnseiten kann dann ein Schleifkontakt immer wieder eine andere Windung kontaktieren und somit einen anderen Widerstandswert einstellen.

Leider verschmutzt die Kontaktfläche relativ oft, wobei es dann zu einem erhöhten Übergangswiderstand an der Kontaktfläche kommt. Diese wiederum wird dadurch sehr stark thermisch belastet und geht in der Regel kaputt. Außerdem kommt es vor, dass der eingestellte Widerstandswert springt, wenn sich der Schleifkontakt zwischen zwei Drahtwindungen befindet. Übrigens bestimmt die Drahtdicke und die entsprechende Kühlung mittels eventuellem Kühlkörper den Wert der Leistung.

Die Drehwiderstände mit Kohleschichtbahnen (gekapselt oder nicht) können relativ wenig Leistung, sind dafür aber genauer einzustellen, haben aber den Nachteil, dass sich bei häufigem Gebrauch die Kohleschicht abträgt und der Widerstand nutzlos wird.

Der Gebrauch im Labor ist schon eher interessant, als bei Festwiderständen, aber nicht wirklich ideal.

1.3 Leistungs-Schiebewiderstände

Mehr Flexibilität im Leistungsbereich bieten Schiebewiderstände, wie sie in Abbildung 1.3 dargestellt sind. Sie bieten wesentlich mehr Leistung als die Drehwiderstände und sind dennoch relativ erschwinglich.

Jedoch sind auch diese Widerstände für Leistungszwecke nicht das „non plus ultra". Die Widerstandswerte lassen sich selten genau einstellen, da sie sehr schwergängig sind. Vom Platz, den die Schiebewiderstände auf dem Labortisch einnehmen ganz zu schweigen. Auch hier gilt die Regel: Je mehr Leistung desto größer die Bauform.

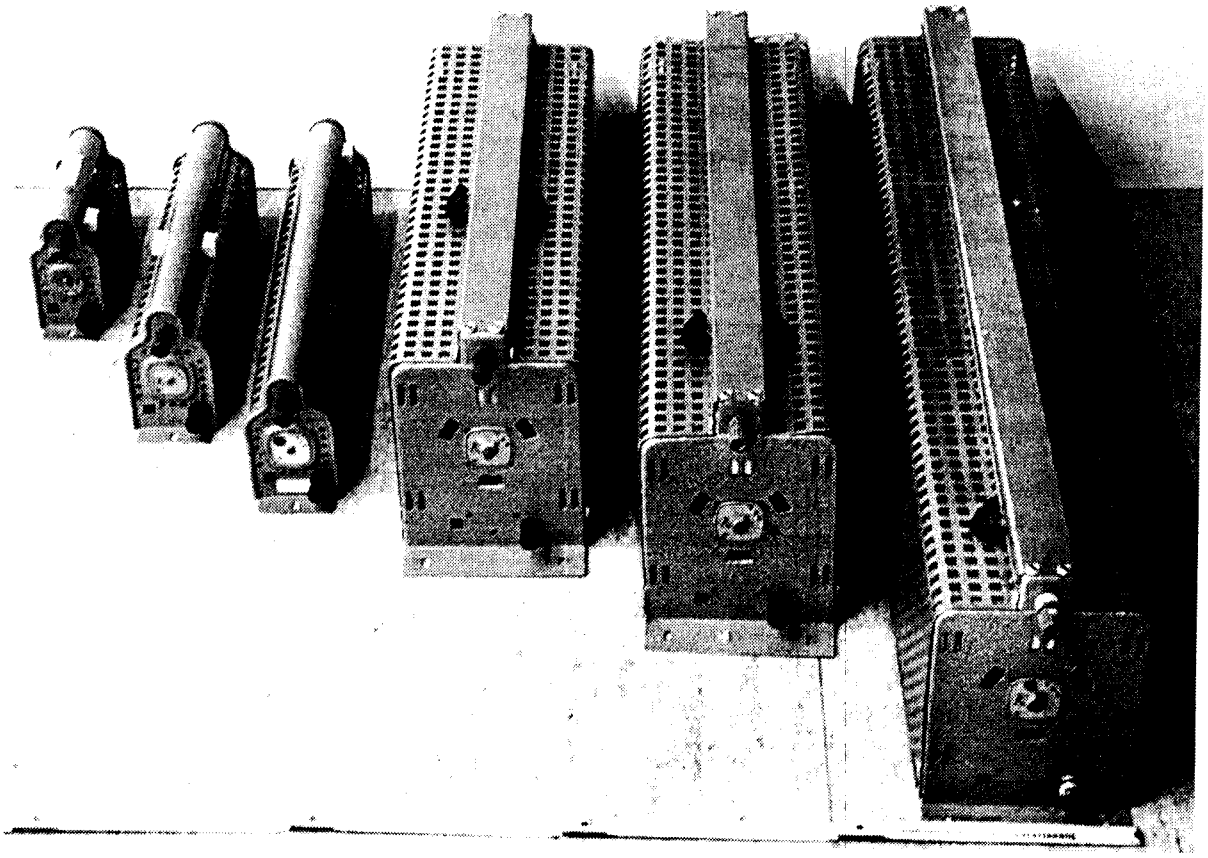


Abbildung 1.3 : Schiebewiderstände für Laboranwendung

Zudem ist eine Dauerbelastung der Widerstände ohne zusätzliche Kühlung sehr gewagt, da es vorkommen kann, dass sich an dem Übergangswiderstand vom Schleifer auf die Widerstandschicht eine unzulässig hohe Wärmeentwicklung einstellt. Eine mögliche Folge könnte beispielsweise ein Abbrennen oder Ausglühen des Leistungswiderstandes sein.

1.4 Elektronische DC-Last

Ein elektronisch geregelter Widerstand hingegen vereint alle Vorteile der herkömmlichen Widerstandsarten. Zudem ist eine hohe Genauigkeit bei nahezu unendlichen Widerstandwerten gegeben. Selbst Kurzschlüsse können mit einer elektronisch geregelten Last simuliert werden, um beispielsweise die Stromfestigkeit einer Endstufe zu testen.

Ein eindeutiger Vorteil ist jedoch der sehr große Leistungsbereich den die elektronische Last abdeckt. Es sind Geräte auf dem Markt, die eine Leistung von 10 KW mit Leichtigkeit in Wärme umwandeln ohne dabei Schaden zu nehmen.

Zudem können mit einer elektronischen Last auch noch andere Funktionen ausgeführt werden, wie zum Beispiel einer Konstantstromsenke oder ein Leistungsmessgerät. Eine Veranschaulichung des durchfliesenden Stromes auf einem Monitor ist genau so möglich wie die an die Last angelegte Spannung.

Jedoch haben elektronische DC-Lasten auch Nachteile wie zum Beispiel die relativ hohen Anschaffungskosten (im Normalfall vier- bis fünfstelliger Euro-Betrag). Sollte das Gerät durch unsachgemäße Behandlung einen Defekt aufweisen, werden sich

hohe Reparaturkosten nicht vermeiden lassen, da eine aufwendige und komplizierte Elektronik integriert ist.

Der größte Nachteil einer elektronischen DC-Last steckt jedoch schon im Namen: Sie sind nur für Gleichstromanwendungen brauchbar. Ein Betrieb an Wechselstrom ist zwar über einen Leistungsgleichrichter möglich, aber unrentabel, da man unweigerlich einen größeren prozentualen Fehler der Last in Kauf nehmen muss.

1.5 ACDC-Last mittels FPGA-Steuerung

Alle vorhergehenden Vor- und Nachteile sind eine Überlegung wert, ob man die klaren Vorteile einer DC-Last nicht auch mit Wechselströmen in Verbindung bringen kann.

Einem normalen Festwiderstand, Drehpotentiometer oder Schiebewiderstand ist es egal ob er mit Gleich- oder Wechselstrom durchflossen wird. Für die Widerstände gelten die normalen ohmschen Gesetzte. Dadurch dass man bei einer elektronischen Last aber die Natur abzubilden versucht, ist es für den Wechselstrombereich etwas schwieriger eine Realisierung zu finden.

Das Ziel einer elektronischen ACDC-Last ist wie es der Name bereits andeutet, einen elektronisch nachgestellten Widerstand zu realisieren, welcher sich bezüglich Gleich- und Wechselstrom wie ein realer Widerstand verhält.

Um dies bewerkstelligen zu können, benötigt man zunächst ein Bauelement, welches man relativ leicht programmieren und bei Bedarf beliebig ändern kann. Also kommt nur ein FPGA in Frage, da man mit einem FPGA bezüglich der zu entwerfenden Hardware absolute Entscheidungsfreiheit hat.

Dadurch wird es möglich, auf einfache Art und Weise auch komplizierte Zusammenhänge mit in die Schaltung zu aufzunehmen, die allerdings nur bei Wechselstromanwendungen wirklich Sinn machen.

So ist zum Beispiel eine Implementierung von ohmisch-induktivem und ohmisch-kapazitivem Verhalten (RL und RC) eines Wechselstromwiderstandes denkbar, ebenso eine Konstantstromsenke für Wechselgrößen. Ein Einsatz der ACDC-Last als Motorenersatz ist durchaus denkbar, da sich die große RL digital dynamisch anpassen lässt.

Somit kann man durchaus sagen, dass eine digitale Regelung mittels eines FPGA eine ungeahnte Flexibilität bietet und dennoch der Schutz von wichtigen Schaltungselementen gegenüber Dritten gewährleistet ist.

Ein weiterer, entscheidender Vorteil ist, dass die Leistungsanforderungen an die Last selbst, nur noch von den verwendeten MOSFET's, welche den Lastweg darstellen, abhängen. Dadurch dass man diesen Lastweg mit den MOSFET's komplementär aufbauen kann, ist man vollkommen frei in der Gestaltung, was die Parallelschaltung der MOSFET's zur Leistungssteigerung angeht. Denn alle Berechnungen und Ansteuerparameter werden in dem FPGA abgelegt, so dass man mit wenig Peripherie auskommt und trotzdem noch erweiterbar ist.

2 Realisierung der Steuerung

Grundsätzlich lässt sich der Leistungsumfang der Diplomarbeit in 3 Sparten einteilen:

- FPGA-Programmierung
- Messdatenerfassung und Wandlung
- Datenausgabe und Ansteuerung der MOSFET's

Die Programmierung des FPGA stellt die wohl größte Herausforderung des gesamten Projektes dar, denn es gilt das Herzstück der gesamten Steuerung direkt in Hardware zu realisieren.

2.1 Blockschaltbild

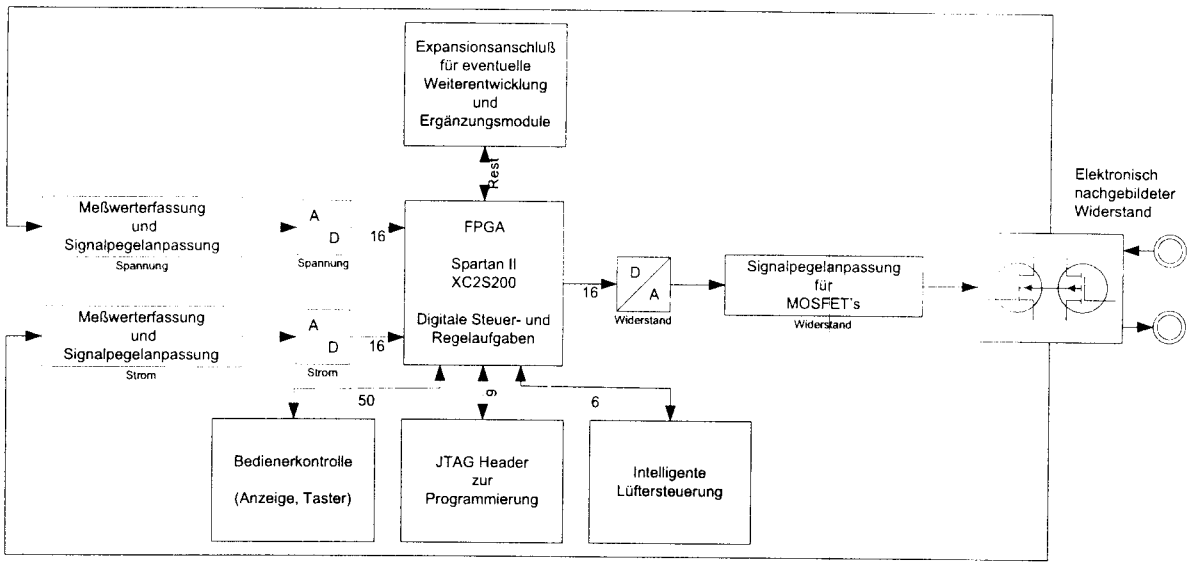


Abbildung 2.1 : Blockschaltbild der ACDC-Last, prizipieller Aufbau

Wie in der Abbildung 2.1 zu sehen ist, ist die Komplexität und der prinzipielle Aufbau einer ACDC-Last sehr einfach.

Von rechts ausgehend, den komplementären MOSFET's, beginnt der Regelkreis. Die MOSFET's bilden quasi den Widerstand nach, indem Sie je nach eingestelltem Wert, den der FPGA ausgibt, aufgesteuert werden. Da die MOSFET's ohne große Probleme parallel betrieben werden können (auch komplementäre) ist der eigentliche Leistungsteil beliebig erweiterbar. Man muss lediglich eine entsprechende Signalanpassung vornehmen.

Von den MOSFET's gelangt man dann zur Messwerterfassung und Signalpegelanpassung für den Strom und die Spannung. Die Strom- und Spannungsaufnahme darf man sich wie bei einem Multimeter vorstellen, nur mit dem Unterschied, dass für die Spannung keine extra Buchse zur Verfügung steht. Sehr wohl wird aber in zwei Buchsen unterteilt und zwar in eine für große Ströme (bis 20 A) und eine für kleine Ströme (bis 2 A). Hiermit wird garantiert, dass immer ein für den Einsatzzweck genügend genaue Auswertung des Stromes erfolgen kann. Zusätzlich wird noch eine Normierung der Ströme vorgenommen.

Die so gewonnenen Messspannungen werden jeweils eigenen A/D-Wandlern zugeführt. Einem A/D-Wandler für die gemessene Spannung und einem für den erfassten Strom. Beide A/D-Wandler sind in 16 Bit ausgeführt um eine hinreichende Genauigkeit des gewandelten Stromes zu erzielen. Hier werden sehr schnelle A/D-Wandler eingesetzt (10 MSP), um den Signalverlauf der Wechselspannung möglichst

genau zu erfassen (es sollten pro Signallänge mindestens 8 Abtastwerte vorliegen, sonst kommt es zu Quatisierungsfehlern.). Diese A/D-Wandler bestimmen primär die Frequenz des Eingangsignales.

Von den A/D-Wandlern gehen dann die gewandelten Messgrößen in den FPGA wo sie letztendlich verarbeitet werden. Wie bereits erwähnt, stellt der FPGA die zentrale Steuer- und Regeleinheit dar.

An ihn gekoppelt ist die Bedienerkontrolle, das heißt sämtliche Taster und Anzeige-LED's sowie das Anzeigedisplay. Dies wird auf einer separaten Platine realisiert und einfach über die I/O-Ports des FPGA verbunden. Die eigentliche Ansteuerung und Auswertung übernimmt komplett der FPGA.

Ebenso ist ein JTAG Header zur Programmierung des FPGA angebracht. Der FPGA kann somit im Master-Slave-Verfahren initialisiert und programmiert werden. Ebenso wird ein eventuelles Debuggen mit dem entsprechenden Kabel möglich.

Damit entstehende Wärme abgeführt werden kann, ist eine intelligente Lüftersteuerung an den FPGA gekoppelt. Intelligent bedeutet, dass die Lüftersteuerung über einen Bus mit dem FPGA kommuniziert und ihm ständig ihre Temperaturwerte mitteilt. Der FPGA entscheidet dann, welcher Lüfter durch die Lüftersteuerung in Betrieb gesetzt werden soll oder ob alle Lüfter anlaufen sollen, da das Gerät sonst zu überhitzen droht.

Des weiteren sind alle nicht benutzten I/O-Leitungen auf einen Expansionsport geführt, um eine möglichst hohe Flexibilität zu erreichen. Hiermit wird gewährleistet, dass eventuell Zusatzgeräte oder Zusatzfunktionen später noch implementiert werden können.

Nachdem der FPGA seine Regelung des Widerstandes vorgenommen hat, gibt er seinen Widerstandswert über einen 16 Bit breiten Bus an den D/A-Wandler weiter. Dieser erzeugt daraus eine entsprechende Spannung. Der D/A-Wandler muss ebenfalls relativ schnell sein, aber bei weitem nicht so schnell wie die A/D-Wandler.

Das von dem D/A-Wandler gewonnene analoge Signal wird zur Signalanpassung für die MOSFET's geführt. Hier wird das Signal für die Ansteuerung der MOSFET's konditioniert. Dies bedeutet, es findet eine dynamische Anpassung an die typische Kennlinie von MOSFET's im positiven und negativen Spannungsbetrieb statt.

Somit schließt sich der Kreis und die elektronische Last sollte fähig sein, auch Wechselströme verarbeiten zu können.

2.2 Implementierte Funktionen

Wie bereits erwähnt, sind bei einer ACDC-Last alle Funktionen einer reiner DC-Last mit implementiert, das heißt man kann das Gerät wie einen normalen Gleichstromwiderstand behandeln. Darüber hinaus ist für den Gleichstrombetrieb eine Konstantstromsenke mit enthalten.

Wäre es nun keine ACDC-Last, würden andere Funktionen kaum auffindbar sein. Durchdachte Algorithmen jedoch ermöglichen noch zusätzlich eine Fülle anderer Funktionen.

Dadurch wird es ermöglicht, mit dem Gerät einen reinen Wechselstromwiderstand abzubilden. Hierzu gehören ein ohmisch-kapazitiver Widerstand (RC) und ein ohmisch-induktiver Widerstand (RL). Das besondere hierbei ist, dass die benötigte Zeitkonstante (beinahe) stufenlos einstellbar ist. Somit eignet sich das Gerät zum

Beispiel hervorragend zur Messung von Endstufen, an welche verschiedene Motoren (RL-Lasten) angeschlossen werden, da man eine Vielzahl von Elektromotoren damit abbilden kann.

Ebenso ist eine Konstantstromsenke für den Wechselstrombetrieb enthalten, welche es einem ermöglicht (wie es der Name schon sagt) bei wechselnder Eingangsspannung einen konstanten Strom einzuprägen. Diese Funktion kann in sofern nützlich sein, wenn man eine Schaltung auf ihre Stromfestigkeit hin überprüfen möchte.

Sofern noch ausreichend Platz auf dem FPGA zur Verfügung steht, können noch weitere Funktionen dem Gerät hinzugefügt werden. Dazu wurde ein Expansionsport aus den restlichen I/O-Pins des FPGA gebildet. Denkbar wäre zum Beispiel eine Funktion zur Visualisierung der Messwerte an einem PC, einer Ansteuerung der Last über USB oder vielleicht auch eine erweiterte Fehlerdiagnostik. Die Wege stehen einem offen, vorausgesetzt die Funktion lässt sich noch in dem FPGA unterbringen.

2.3 FPGA-Module

Im Rahmen der Diplomarbeit wurden schon zahlreiche FPGA-Module erstellt. Alle hierbei entstandenen Module würden wohl den Rahmen des Dokumentes sprengen, weshalb nur ein paar herausgegriffen wurden.

Um Kosten zu senken, wurde sehr viel Aufwand in die Programmierung des FPGA's gesteckt. So ist es möglich, unentprellte Taster zu verwenden. Der Entprellvorgang selbst findet in dem FPGA statt. Nun ist es gewiss nicht sonderlich schwierig ein hardwarenahes Entprellen in einen FPGA zu programmieren, schwieriger ist es schon mit dem selben Modul zu erkennen, ob die Taste gedrückt bleibt oder nicht und man erneut einen Triggerimpuls für die internen Komponenten auslösen kann oder nicht.

Besonders interessant wird das Entprell-Modul wenn es darum geht, einen Wert zu inkrementieren oder dekrementieren, beispielsweise den einstellbaren Widerstandswert. Ein einzelner Tastendruck erhöht um eine Stelle, bleibt man auf der Taste, werden interne Impulse ausgelöst, mit deren Hilfe von alleine hoch- oder runter gezählt wird. Da das aber unter Umständen bis zu dem gewünschten Wert sehr lange dauern kann, wurde ein sich anpassendes Modul dazu entwickelt. Dieses Schaltwerk, bildet eine Rampenfunktion nach, das heißt der einzustellende Wert wird zunächst langsam, dann schneller in- oder dekrementiert.

Um die erfassten Strom- und Spannungswerte zu verarbeiten, beziehungsweise um daraus einen Widerstandswert zu bilden, mit dem man den eingestellten Wert des Gerätes vergleichen kann, benötigt man ein Dividierschaltwerk. Dieses muss eine 16-Bit Division innerhalb kürzester Zeit bewerkstelligen (maximal 20 interne FPGA-Taktzyklen), um die gesamte Performance des Gerätes nicht zu beeinträchtigen.

Das Endergebnis der Division wird einem internen 8stelligen BCD-Wert angepasst, wozu ebenfalls ein nicht ganz unkompliziertes Schaltwerk benötigt wird. Dieses Schaltwerk selbst bildet dann auch schon die Grundlage für ein neues Modul, dem der Anzeigesteuerung, denn es werden immer nur 3 Stellen nach außen angezeigt (inklusive Trennpunkt, Polaritäts- und Bereichsanzeige) auf vorerst 3 Siebensegment-Anzeigen und einfachen LED's.

Um die Daten letztendlich zu verarbeiten, benötigt man einen digitalen Regler. Dieser ist auch in einem separaten Modul untergebracht. Vom Prinzip her arbeitet dieser

Regler wie ein Regelalgorithmus auf einem Mikrocontroller oder Mikroprozessor. In der Fachliteratur ist er auch als PI-Regler des Typs II bekannt. Da sämtliche Algorithmen bezüglich digitaler Regler auf Mikrocontroller oder besser gesagt auf deren Programmierung zugeschnitten sind, musste man hierbei wiederum sehr viel Zeit investieren um den Algorithmus in Hardware abzubilden.

Es wurden aber auch noch andere Module entwickelt. Zum Beispiel eine sich selbstständig anpassende Messwerterfassung. Wird der durch den A/D-Wandler gewandelte Digitalwert zu klein oder zu groß, wird über einen Bus den PGA-Verstärkern in der Messwerterfassung mitgeteilt, wie sie Ihre Verstärkung zu verändern haben, damit man immer den bestmöglichen Messbereich selektiert hat und am besten auflösen kann.

2.4 Gerätespezifikation

Das in der Diplomarbeit erstellte Gerät wurde entsprechend folgender Spezifikation entwickelt:

Widerstand (DC [R] und AC [Z])	250 mΩ .. 100 KΩ
Strom (Spitzenwert) maximal	Buchse 1: 20 A Buchse 2: 1.25 A
Spannung (Spitzenwert) maximal	1 V .. 48 V
Leistung	Max. 120 W
Zeitkonstante τ, stufenlos	1 ns .. 20 ms
Frequenz, mindestens bis	150 KHz
Temperaturstabilität	-10 °C .. 100°C
Alle Parameter durch digitale Implementierung anpassbar, Einschränkungen rein durch PCB-Aufbau	

3 Problembehandlung

Das größte Problem bei der ganzen Diplomarbeit ist der digitale Regelalgorithmus. Diese Algorithmen sind meistens nur für Mikrocontroller und Mikroprozessoren vorgesehen. Natürlich wäre es denkbar gewesen, einen IP-Core eines Mikrocontrollers in den FPGA einzubringen und dann den FPGA so zu konfigurieren, als ob noch das entsprechende Programm implementiert ist. Dann hätte man allerdings das ganze Projekt auch mit einem realen Mikrocontroller aufbauen können.

Es gilt nun, einen digitalen Regler direkt in Hardware zu realisieren, ein nicht einfaches Unterfangen, denn leider kommt man nur näherungsweise an einen vorgegebenen Algorithmus heran (Der verwendete Algorithmus ist ein PI-Regler des Typ II, Nachzulesen im „Taschenbuch der Regelungstechnik" von Lutz/Wendt).

Da eine digitale Regelung sehr sensibel ist, bezüglich Abtastung und Weitergabe von gewandelten Analogsignalen, sind sehr schnelle A/D und D/A-Wandler von Nöten, vor allem wenn man das Gerät auch für Frequenzen größer 100 KHz realisieren

möchte. Man sollte minimal mit dem 8-fachen Wert abtasten, im Fall der Diplomarbeit mit 10 MegaSample.

Man sollte auch darauf achten, wenig IP-Cores zu benutzen. Zum einen sind sie nicht immer kostenlos und frei verwendbar und zum anderen muss man ressourcenfressende Funktionen mit in Kauf nehmen, die man eigentlich gar nicht benötigt. Deshalb ist es auch notwenig, bei begrenztem Raumangebot, alles selbst zu programmieren.

Ein weiterer Aspekt ist, komplementäre MOSFET's mit guten Leistungsdaten zu finden. An für sich sollte dies kein Problem darstellen, denn schließlich gibt es Audioverstärker mit MOSFET-Endstufen. Dennoch ist es wirklich schwer MOSFET's zu finden, die den Leistungsanforderungen dieser Diplomarbeit entsprechen. Denn im Gegensatz zu Audioanwendungen muss hier eine gewisse Belastung des MOSFET mit einer zum Beispiel konstanten Leistung länger ausgehalten werden.

Ein realer Hardwaretest konnte noch nicht durchgeführt werden, da die entsprechende Platine noch nicht gefertigt ist. Bisher wurden alle Teile simuliert mit Programmen wie ModelSIM und PSPICE und FPGA-relevante Blöcke auch synthetisiert und auf einem FPGA unter Echtzeitbedingungen getestet.

Dennoch ist es durchaus möglich, dass der Regelalgorithmus im FPGA nicht richtig stabil arbeitet. Was in der (begrenzten) Simulation funktioniert, heißt noch lange nicht, dass es auch in Realität funktionieren muss, zumindest nicht in der geforderten Geschwindigkeit.

4 Ausblick

Für die zukünftige Anwendung der elektronische ACDC-Last stehen die Wege noch offen. Denkbar wäre eine Realisierung des FPGA's in einem ASIC da dieser einen perfekten IP-Schutz bieten würde, andererseits aber auch einen universellen Einsatz der Steuerung erlaubt. Es wäre ein Einsatz des ASIC in Mini-Lasten denkbar oder auch in Hochleistungslasten, denn letztendlich ist das Leistungsspektrum der ACDC-Last von den angeschlossenen MOSFET's und den weiteren verwendeten Bauelementen (A/D-Wandler, D/A-Wandler, Operationsverstärker, usw) abhängig.

Denkbar wäre auch eine Kopplung der ACDC-Last mit einem PC über beispielsweise USB, um die in dem FPGA verarbeiteten Daten zu visualisieren oder um digitale Filter verschiedenster Art mit zu benutzen. Hierzu steht der Expansionsport und der freie Platz im FPGA zur Verfügung.

Systementwicklung einer RISC Implementierung der JAVA Virtual Machine

Attila Zabos, Eduard Steindl[2], Irineusz Janiszewski, Hermann Meuth , Bernhard Kreling[2], und Bernhard Hoppe

FB Elektrotechnik/Automatisierungstechnik, Forschungsgruppe µSYST,[2] FB Informatik
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

Einleitung

In diesem Artikel wird die Systementwicklung eines Mikroprozessor- Kerns (*attoJAVA- Core*) beschrieben, der die JAVA Virtual Machine (JVM) als elektronische Schaltung implementieren soll. Dieser Prozessor soll die Maschinenbefehle der JVM, sog. JAVA Bytecode, direkt ausführen, was den Ablauf von JAVA- Programmen beschleunigt und gleichzeitig den Leistungsbedarf und den Schaltungsaufwand soweit reduziert, dass JAVA als Programmiersprache für Anwendungen in Eingebetteten Systemen (Embedded Systems) interessant wird.

Im Gegensatz zu bereits verfügbaren oder veröffentlichten JAVA- Prozessoren, soll der attoJAVA-Kern nur die wichtigsten der insgesamt 226 Bytecode Instruktionen der JVM direkt ausführen (Reduced Instruction Set, RISC). Die verbleibenden Instruktionen werden im Rahmen eines vorgeschalteten Kompilationsschrittes auf den attoJAVA Befehlssatz abgebildet. Dazu wird neben der Hardware ein neuer Compiler benötigt, der das Bytecode- Programm vor der Laufzeit analysiert. Mit dieser Parallelentwicklung von Elektronik und Software (Hardware/Software Co- design) wird sich der Gatteraufwand für den attoJAVA Prozessor soweit reduzieren lassen, dass jede verfügbare Designlösung unterboten werden kann, ohne dass dies negative Auswirkungen auf die Leistungsdaten hätte oder die Nutzung in eingebetteten (Echtzeit-) Systemen einschränken würde.

Um den Rechnerkern später universell einsetzen zu können, wird der Prozessor unter Verwendung der Hardwarebeschreibungssprache VHDL entwickelt. Die Logiksynthese ermöglicht dann die Abbildung auf beliebige Zieltechnologien. Zu Evaluations- und Testzwecken wird die XILINX Virtex CPLD-Familie und der 0.35 µm CMOS CSD- Prozess von Austria Microsystems (AMS) verwendet. Layout-Studien und die Resultate des Synthesewerkzeugs *Synopsys* zeigen, dass der Prozessor in CSD-(CPLD-) Technologie mit maximal 50 MHz (10 MHz) getaktet werden kann. Die Chipfläche (CPLD-Größe) erreicht dabei nur moderate 10 mm² (300 k Gatter). Die elektrische Leistungsaufnahme der ASIC- Lösung bleibt unterhalb von 2mW pro MHz- Taktfrequenz.

1 Beschreibung der Designanforderungen

Obwohl Java mittlerweile einen Standard bei Internet- und Kommunikationsanwendungen darstellt, konnte sich diese Sprache bei eingebetteten Systemen für Steuer- und Regelanwendungen weniger durchsetzen. Bei eingebetteten Systemen sind nämlich typischerweise Rechnerleistung, Ein/Ausgabe-Möglichkeiten und Speicherressourcen beschränkt, um kosteneffiziente, platz- und stromsparende Systeme zu realisieren. Will man aber JAVA- Programme nach dem üblichen Schema ablaufen lassen, wird ein leistungsfähiger Standardmikroprozessor (z.B. der Pentium- Klasse) benötigt, der die JVM emuliert [1]. Um JAVA- Plattformen mit reduziertem Ressourcenbedarf zu realisieren, wurden deshalb in den letzten Jahren verstärkt Mikroprozessoren entwickelt [2-5], die JAVA- Bytecode direkt ausführen können. Diese Rechner sind nicht als elektronische Schaltungen erhältlich, sondern können als HDL- Modelle (sog. *Intellectual Property, IP*) teilweise kostenfrei oder gegen Lizenzgebühren bezogen und in komplexere Designs (*System on chip*) eingebunden werden. Der Hauptnachteil dieser IP-Komponenten ist der sehr große Gatterbedarf, der aus der Komplexität des JVM- Befehlssatzes resultiert. Der *picoJavaII*- Kern [3] benötigt beispielsweise 125.000 Gatter. Im Vergleich dazu liegt der Gatterbedarf eines Standard- Controllers (z. B. 8051) unterhalb von 10.000 Gattern!

Um den Anforderungen im kostensensiblen Embedded Bereich noch besser gerecht zu werden, wurde mit Entwicklung eines ebenfalls bytecode-orientierten Prozessors, des *attoJAVA*- Rechnerkers, begonnen. Das besondere ist die Kombination der Rechnerhardware mit einem Compiler, der vor der Laufzeit des Programms alle komplexen, nicht durch Hardware unterstützten Befehle auf den implementierten Befehlsatz abbildet (Bild 1). So lässt sich der gesamte Befehlsatz der JVM auf effiziente Weise

abdecken. Der attoJAVA- Prozessor wird technologieunabhängig auf der Basis der Hardwarebeschreibungssprache VHDL entwickelt. Damit ist das Design uneingeschränkt wiederverwertbar [6] und kann problemlos an interessierte Partner in Industrie und im Hochschulbereich transferiert werden.

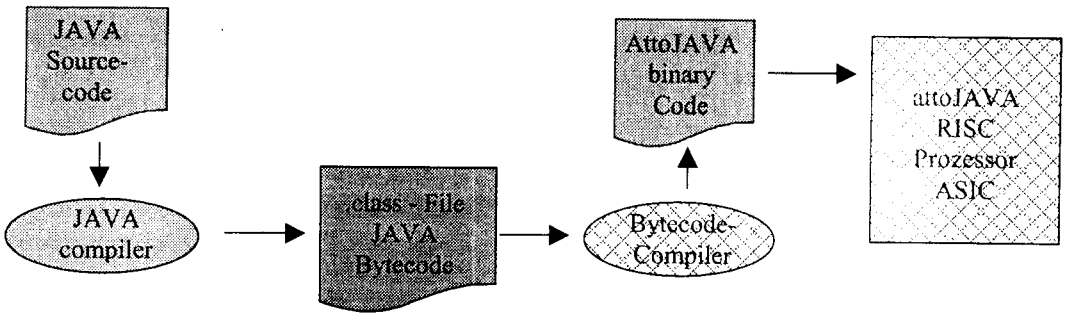


Bild 1: Ausführung einer JAVA Applikation. Die neu zu entwickelnden Komponenten sind schraffiert.

1.1 Aufbau, Befehlsatz und Implementierung

Der attoJAVA- Prozessor ist wie die JVM ein Stapelrechner (Stackprozessor). Um Adresskonflikte zu vermeiden, wurde die Harvard Architektur gewählt, die getrennte Adressbusse für Daten- und den Programmspeicher verwendet. Insgesamt unterstützt Prozessor nach derzeitigem Stand 30% der 226 Bytecodebefehlen. Darüber hinaus wurden auch sog. erweiterte Codes definiert, die den Zugriff auf Register oder Speicherbereiche sowie den Aufruf von Unterprogrammen ermöglichen. Mit Methoden des Hardware/Software Co- Designs [7] wird derzeit die optimale Untermenge von Bytecodeinstruktionen ermittelt, mit der alle JVM- Maschinenbefehle effizient dargestellt werden können. Der verkleinerte Befehlssatz senkt den Gatter- und Strombedarf des Prozessors ohne Nachteile bei den Leistungsdaten soweit, dass sich eingebettete Systeme kostengünstig darstellen lassen. In AMS CSD Technologie (0,35 µm CMOS) beträgt der Gatterbedarf nach ersten Syntheseläufen für eine 50 MHz- Version nur 15.000 Gatter, während für den picoJAVAII- Kern 125.000 Gatter benötigt werden. Die gegenseitige Beeinflussung von Taktfrequenz und Gatterbedarf zeigt Bild 2 auf der Basis von Synopsys- Synthese- Resultaten. Für Testzwecke und zur Evaluierung des Gesamtsystems aus Prozessor und Compiler haben wir auch untersucht, ob sich das Design auch für VirtexII- CPLDs der Firma XILINX aufbereiten lässt. Hier erscheint ein CPLD- Baustein mit 300.000 Gatteräquivalenten ausreichend zu sein.

1.2 Der attoJAVA- Bytecode Compiler

Die wichtigste Softwarekomponente der hier vorgestellten neuen JAVA Plattform wird der neue Compiler sein, der JAVA Bytecode in attoJAVA- Instruktionen übersetzt. Diese Übersetzung findet *vor* dem Ablauf des JAVA- Programms statt (*ahead of time, AOT*), während bei den JAVA- Systemen, die auf der Basis von Standardprozessoren arbeiten, der JAVA- Bytecode erst während der Programmausführung Stück für Stück in Maschinenbefehle des verwendeten Prozessors umgewandelt wird (Just in Time Compilation, JIT).

Der AOT- Ansatz hat keinerlei Nachteile für eingebettete Systemanwendungen, denn für den hier typischen Echtzeitbetrieb, müssen die Antwortzeiten des Systems ohnehin im Voraus bekannt sein, was JIT- Ansätze ausschließt. Den geplanten Ablauf der AOT- Kompilation des JAVA- Quellcodes zeigt Bild 3. Man erkennt, dass der JAVA- Compiler zunächst verschiedene sog. Class- Files mit JAVA-Bytecode erzeugen muss, die dann vom AOT- Compiler zusammen mit benötigten Methoden[1] aus dem Application Programming Interface (API) geladen werden. Um die Laufzeitstrukturen zu erzeugen, ersetzt der AOT- Compiler alle symbolischen Referenzen durch direkte Adressen. Die komplexeren Bytecode- Instruktionen werden in Unterprogramme aus attoJAVA- Befehlen umgewandelt. Am Ende

[1] „Methode" bezeichnet ein Unterprogramm in einer objektorientierten Programmiersprache.

schreibt der Compiler eine Binärdatei in den Programmspeicher, die dann vom attoJAVA- Prozessor ausgeführt werden kann.

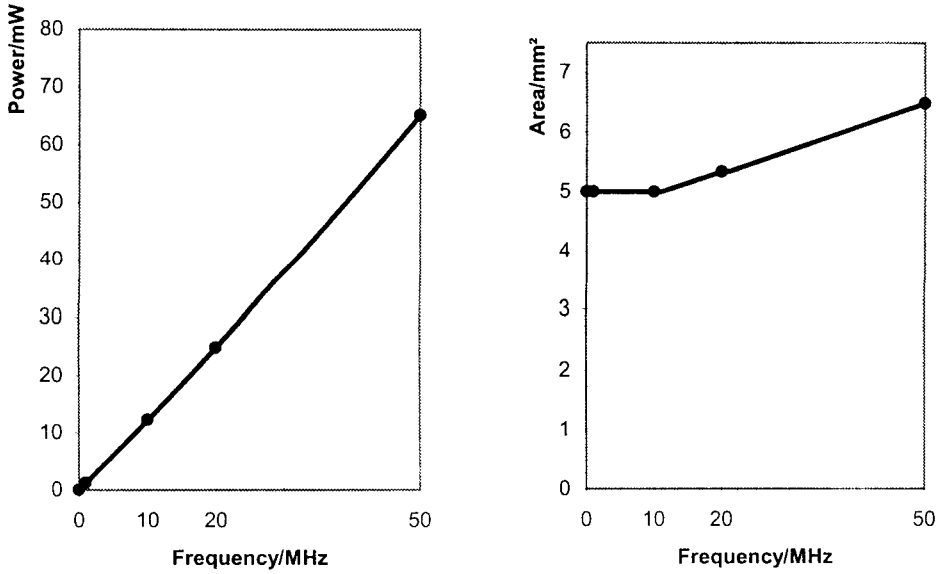


Bild 2: Verlustleistung (Power) und Chipfläche (Area) des attoJAVA in CSD- Technologie in Abhängigkeit von der Taktfrequenz (Frequency).

1.3 Prozessorarchitektur im Überblick

Um den gesamten Hardwareaufwand des Systems zu minimieren, wird eine möglichst einfache, aber flexible und leistungsfähige Rechnerarchitektur angestrebt. Die kritische Komponente für den erzielbaren Datendurchsatz des Rechners ist die arithmetisch logische Einheit (ALU), die in Übereinstimmung mit den Vorgaben für die JVM mit einem 32 Bit breiten Datenpfad implementiert werden soll. Da die JAVA- Bytecode Befehle unterschiedlich lang sein können, wird ein Befehlszugriff- Schieberegister (*Instruction Prefetch Queue, IPQ*) vorgesehen, damit die Instruktionen schneller geladen werden können. Eine Speicherschnittstellen- Einheit (*Memory Interface Unit, MIU*) steuert den Datentransfer zwischen internen (on-chip) und externen Speichern. Der interne Stapelspeicher soll, um möglichst schnelle Datenzugriffe zu ermöglichen, mit einem Schreib/Lesespeicher mit gleichzeitigem Schreib/Lesezugriff (*Dual- Port- RAM*) realisiert werden.

Taktfrequenz und Gatteraufwand des Prozessors werden von der Steuereinheit (*Control Unit*) maßgeblich beeinflusst. In anderen Entwürfen [3] werden hierfür geschachtelte Schaltungen mit 5 oder sogar noch mehr Pipelinestufen eingesetzt. Dadurch steigen zwar die Datendurchsatzraten, gleichzeitig kann es bei Pipelinestrukturen aber auch zu Zugriffskonflikten kommen, etwa wenn ein Befehl auf Daten zugreifen will, die von einem noch nicht komplett abgearbeiteten vorherigen Befehl generiert werden. Um diese Fälle abzufangen, sind logische Schaltungen vorzusehen, die wiederum zu Laufzeitverlängerungen und zusätzlichen Gatterbedarf führen, was die Vorteile einer Pipeline als Kontrolleinheit teilweise wieder aufwiegt. Aus diesen Gründen wird die Steuereinheit des attoJAVA- Kerns nur zweistufig ausgeführt. In der ersten Stufe wird der jeweilige Befehl geladen, in der zweiten Stufe werden die zugehörigen Steuersignale erzeugt.

Da die neue JAVA- Plattform sowohl aus Hardware- wie auch aus Software- Komponenten besteht und sich die Komplexität von Core und Compiler gegenseitig stark beeinflussen, wird die Steuereinheit als hierarchischer Zustandsautomat entwickelt. Im Gegensatz zu den üblichen Realisierungen über mikrokodierte Festwertspeicher (ROM), lassen sich Zustandsautomaten flexibel erweitern, wenn zusätzliche Befehle für eine effizientere Kompilation benötigt werden. Zustandsautomaten lassen sich grafisch über Zustandsdiagramme definieren und diese Diagramme können mit geeigneten CAE- Werkzeugen (z. B. HDL- Designer von Mentor Graphics) automatisch in VHDL- Code umgesetzt und in das HDL-Model des Prozessors eingebunden werden. Das Blockdiagram des Prozessors zeigt Bild 4.

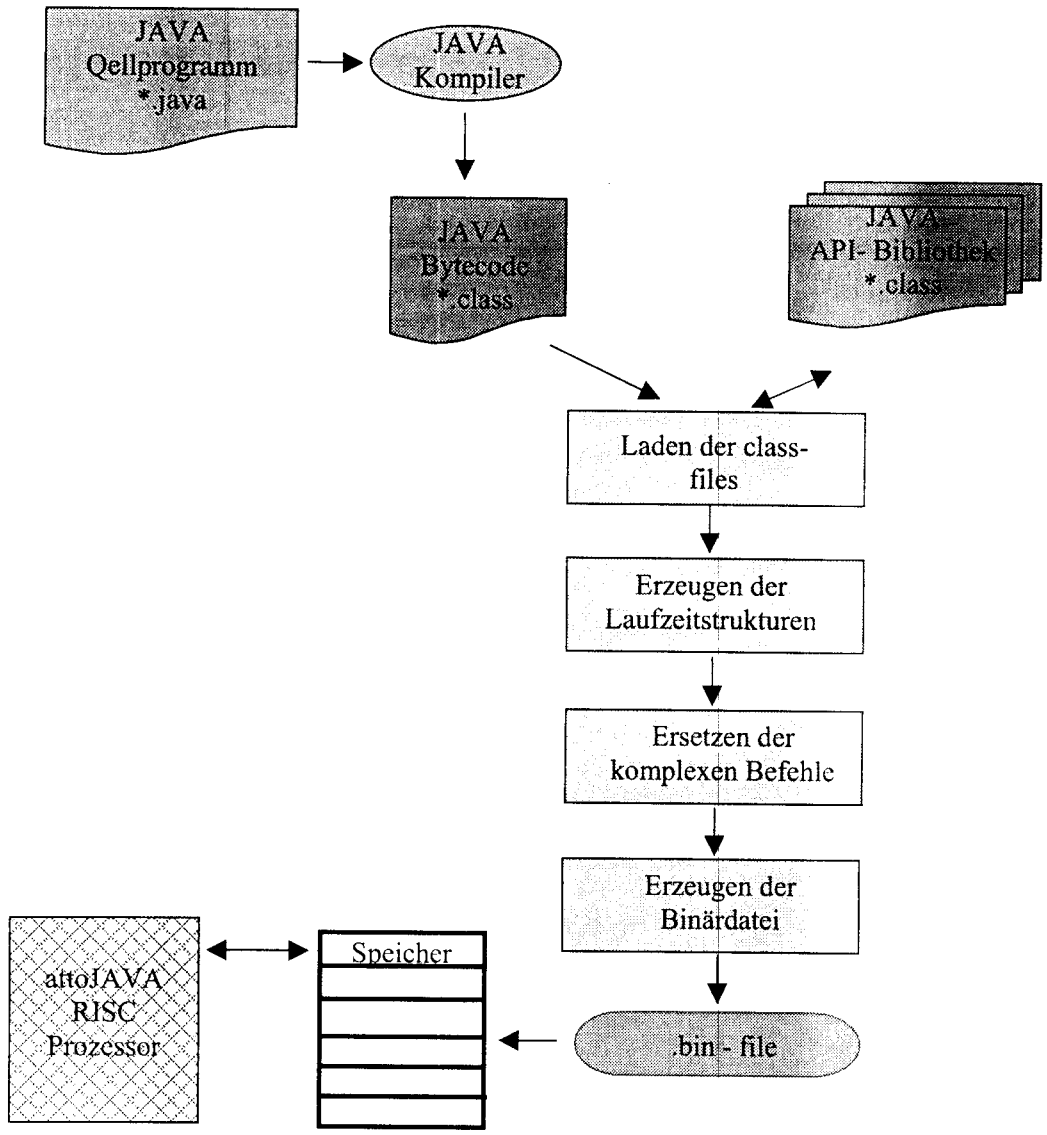


Bild 3: Ablauf der Kompilation von JAVA- Applikationen für den attoJAVA- Kern.

2 Prozessorentwurf

Bei der Prozessorentwicklung wird die Sprache VHDL bei der Designeingabe eingesetzt. Die erstellten VHDL- Modelle werden mit dem HDL- Simulator *Modelsim* getestet. Die Logiksynthese wird mit dem Werkzeug *FPGACompilerII* bzw. *Design Compiler* der Firma Synopsys durchgeführt. Bei der Synthese werden Xilinx V300EFG456 Virtex CPLD Bausteine und der AMS 0.35 µm CSD- Prozess als Zieltechnologien verwendet. Die Abbildung des Logikentwurfs auf die CPLD- Strukturen (Platzierung und Verdrahtung) und die anschließende Erzeugung der entsprechenden Simulationsmodelle mit allen parasitären Laufzeiten erfolgte mit den Werkzeugen des Xilinx *WebPACKs*.

2.1 Entwurfsvorgaben

Das primäre Ziel des attoJAVA- Projekts ist es, eine ressourcen- effiziente JAVA- Plattform zu entwickeln, bei der nur Teile des Befehlssatzes der JVM mittels eines Prozessorkerns in ASIC- oder CPLD- Technologie implementiert werden, während alle nicht vorhandenen Befehle über einen zusätzlichen Kompilationsschritt mittels komplexerer Befehlsfolgen der in Hardware verfügbaren Instruktionen dargestellt werden. Damit der Projektumfang handhabbar bleibt, unterstützt der Prozessor zunächst nur Festkomma- Arithmetik und eine einfädige Programmabarbeitung. Es ist geplant, Fließkomma-Operationen und ein Echtzeitbetriebssystem (RTOS) für Multitasking später einzubeziehen.

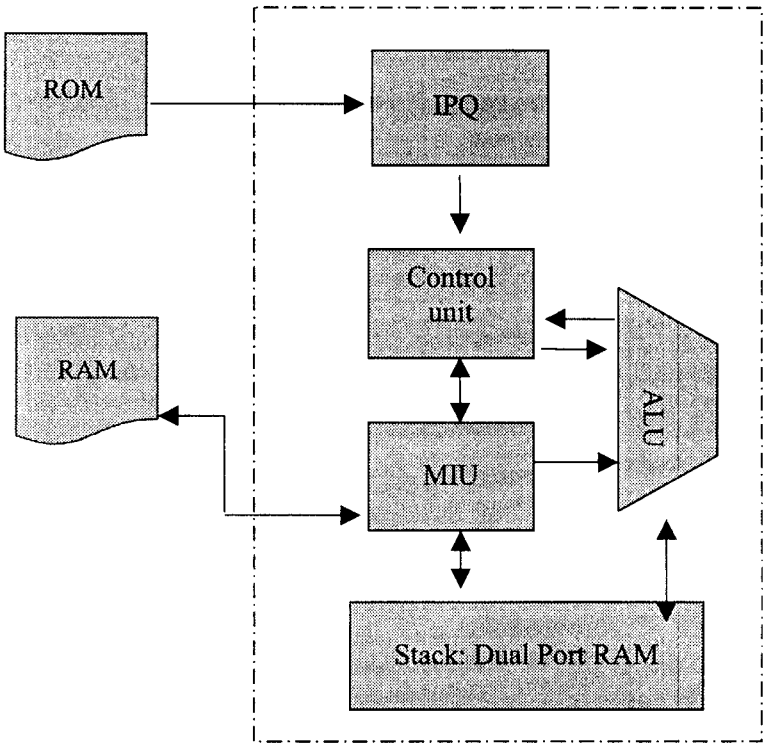


Bild 4: Blockdiagram des attoJAVA- Prozessorkerns.

Aus typischen Steuer- und Regelaufgaben wurden die folgenden Zielvorgaben für die attoJAVA-Entwicklung abgeleitet:

- Der Recherkern soll eine optimal gewählte Untermenge des JVM Befehlssatzes implementieren, die einen einfachen Compilerentwurf ermöglicht und ausreicht, um einfache Regelalgorithmen auch ohne Kompilation direkt abzuarbeiten;
- Der Prozessor soll mit 50 MHz Taktfrequenz im 0.35µm CMOS CSD- Prozess arbeiten, damit kurze Systemantwortzeiten für Echtzeitanwendungen erreicht werden;
- Der Rechner soll möglichst geringen Gatter- und Strombedarf aufweisen, damit der Prozessor in kostensensitiven und portablen Systemen eingesetzt werden kann.

Dementsprechend wird der attoJAVA- Befehlssatz so ausgewählt, dass alle JVM- Instruktionen mit moderatem Softwareaufwand kompiliert werden können. Insgesamt sind derzeit 66 grundlegende Bytecode- Instruktionen identifiziert worden, die in folgende Kategorien fallen:

- Spezielle, einfache Lade/Speicher- Befehle,
- alle arithmetisch/logischen Operationen der JVM,
- verschiedene Verzweigungsbefehle,
- Stack- Manipulationen.

Zusätzlich wurden 12 erweiterte Befehle definiert, die den Zugriff auf Hardwarekomponenten erlauben. Im einzelnen sind dies

- Register- Speicher Transferoperationen und
- Unterprogrammaufrufe und Rücksprünge ins Hauptprogramm.

2.1.1 Block- Spezifikationen

Wie schon in Absatz 2.3 ausgeführt, lässt sich der attoJAVA- Kern mit folgenden Funktionsblöcken aufbauen:

2.1.1.1 Instruction Prefetch Queue

Der Befehlssatz der JVM ist Byte-orientiert, allerdings variiert die Länge der Operanden. Deshalb wird der Zugriff auf Befehle mit einer Instruction Prefetch Queue (IPQ) beschleunigt. Der IPQ- Block lädt

die Daten aus dem Programmspeicher in ein internes Schieberegister, dessen letzte vier Einträge (4 Byte) fest mit dem 32Bit breiten internen Befehlsbus des Prozessors verbunden sind. Wegen der variablen Instruktionslänge muss nicht notwendigerweise jedes Byte auf dem Bus gültig sein. Deshalb werden die Busleitungen, die gültige Bits tragen, mit separaten Steuerleitungen an die Steuereinheit gemeldet.

2.1.1.2 Arithmetisch Logische Einheit

Obwohl in JAVA Datentypen verwendet werden können, die bis zu 64 Bit lang sind, ist beim attoJAVA eine 32 Bit ALU vorgesehen, denn die langen Datentypen kommen bei Embedded Anwendungen nur selten vor. Komplexere mathematische Operationen wie die Division oder die Multiplikation werden mit sequentiellen Algorithmen abgearbeitet. Die Multiplikation von zwei 32Bit-Eingabeworten dauert dementsprechend dann beispielsweise 32 Takte. Alle elementaren Operationen (Addition, Subtraktion, Schiebeoperationen, Inversion, etc.) werden hingegen innerhalb einer Taktperiode ausgeführt. Dieser Aufbau deckt alle typischen Anforderungen mit guten Leistungsdaten ab und stellt einen Kompromiss zwischen ALU- Komplexität und erzielbarem Datendurchsatz dar.

2.1.1.3 Memory Interface Unit

Diese Speicherschnittstelle (MIU) entkoppelt den externen und den internen Datenbus. Das Schnittstellenprotokoll spezifiziert auch die Datenübergabe an die Steuereinheit (CU). Wenn der Prozessor daher mit anderen externen Speichern kommunizieren soll, ist lediglich die MIU- Komponente anzupassen.

2.1.1.4 On Chip Stack

Da der attoJAVA- Prozessor ein Stapelrechner ist, sind häufige Datenaustausche zwischen dem Rechnerkern und dem Stapelspeicher und externen Speichermodulen zu erwarten. Damit hier keine größeren Verzögerungen auftreten, wurde ein Teil des Stacks mit internen on-chip Speicherbereichen realisiert. Das verwendete Dual- Port- RAM umfasst aufgrund der Chipflächenlimitierung 2.024 Bytes. Dies entspricht bei einer Stackbreite von 32 Bit 512 Stapeleinträge. Diese Speichertiefe sollte für typische Programme ausreichen [8].

2.1.1.5 Steuereinheit

Die Steuereinheit (CU) ist die komplexeste Struktur des Rechners. Dieser Block steuert die gesamte Datenverarbeitung über entsprechende Kontrollsignale für die ALU und die Speicherschnittstellen und regelt den Zugriff auf alle interne Busse des Rechners. Um die CU während des Entwurfsprozesses flexibel ändern zu können, wird die Steuereinheit mittels hierarchischer Zustandsgraphen beschrieben. Die CU erhält die abzuarbeitenden Befehle vom IPQ- Block, dekodiert diese und erzeugt die entsprechenden Steuersignale. Dabei wickelt die CU alle Datentransfers über das Memory Interface (MIU) ab. Der MIU- Block liefert z. B. die Operanden für die ALU und speichert die Ergebnisse wieder im internen oder externen Stapelspeicher ab. Der Operandenzugriff kann besonders schnell erfolgen, wenn mindestens ein Operand im internen Speicher gehalten wird, ansonsten sind zwei Taktzyklen zum sequentiellen Laden der Operanden nötig.

2.2 Konkurrierende Entwurfsziele

Da nicht die gesamte Funktionalität der JVM in Hardware implementiert werden soll, wird ein zusätzlicher Compiler benötigt, der ein für die JVM kompiliertes Class- File in eine Form bringt, die vom attoJAVA- Prozessor abgearbeitet werden kann. Mit Methoden des Hardware- Software- Codesigns soll die optimale Partitionierung in Hardware- Befehle und Softwareaufwand bestimmt werden. Hierbei sind folgende konkurrierende Entwurfsziele zu beachten:

Werden nur wenige Instruktionen von der Prozessorhardware unterstützt, erhält man einen kleinen und damit preisgünstigen Prozessorchip. Die Kompilierentwurf ist dann aber sehr kompliziert und weil die meisten JVM- Befehle durch lange Sequenzen einfacher Befehle dargestellt werden müssen, reduziert sich die Ausführungsgeschwindigkeit für JAVA- Programme. Werden hingegen möglichst viele JVM-Instruktionen in den attoJAVA- Befehlssatz übernommen, kann der Prozessor JAVA- Applikationen sehr schnell ausführen, der Compilerentwurf wird einfach, aber der Prozessorchip wird groß, erzeugt viel Verlustleistung und ist damit nicht mehr für den Embedded- Bereich geeignet. Weiterhin ist zu beachten, dass die vollständige Kompilierbarkeit von Class- Files einen gewissen Grundvorrat an Befehlen voraussetzt, die vom attoJAVA- Prozessor verstanden werden. Softwareanteile beinhalten im

übrigen alle JVM- Implementierungen [9], denn einige Bytecodeanweisungen benötigen in jedem Fall Unterstützung durch Softwarekomponenten, die in der Regel im Betriebssystem integriert sind [2-5].

2.3 Der laufzeitkritische Pfad

Da eine hohe Taktfrequenz angestrebt wird, ist genau zu analysieren, welcher Datenpfad den gesamten Datendurchsatz limitiert. Zunächst ist zu erwarten, dass die komplexen arithmetischen Operationen in der ALU zeitkritisch sind. Da der attoJAVA- Prozessor aber im wesentlichen Steuer- und Regelanwendungen mit geringer Rechenbelastung abarbeiten soll, wurden sequentielle Verfahren für die Multiplikation und die Division eingesetzt, die die mathematischen Operationen auf viele und damit kurze Taktperioden verteilen. Deshalb kann der kritische Pfad des attoJAVA- Prozessors im Prinzip überall im Design liegen, sei es in den Logikblöcken oder sei es aufgrund von Leitungsverzögerungen zwischen speichernden Elementen.

2.4 Entwurfsverifikation

Die hier beschriebene JAVA- Plattform besteht aus komplexen Hardware- und Softwarekomponenten, die interagieren. Deshalb ist es besonders wichtig, die korrekte Funktion des Gesamtentwurfs genau zu prüfen. Dies geschieht unter Verwendung der folgenden gestuften Verifikationsstrategie.

2.4.1 Testbenches für den Modultest

Die einzelnen Module werden zunächst mit eigenen VHDL- basierten Testumgebungen, sog. Testbenches, überprüft. Lediglich die Steuereinheit, lässt sich als komplizierte Zustandsmaschine einfacher im Rahmen der Gesamtverifikation des attoJAVA- Cores testen. Jede Operation, die die ALU ausführen kann, wird dabei mit zufällig ausgewählten Operanden getestet. Die Ergebnisse werden mit vorberechneten Resultaten verglichen. Bei Abweichungen erzeugt die Testbench ein Fehlersignal. Die Testumgebung des IPQ- Moduls generiert eine Folge von Programmzählerwerten. Für die betreffenden Adressen, die im internen und im externen Speicherbereich liegen können, liest die IPQ die Speicherinhalte aus. Die Testbench vergleicht auch hier wieder die ausgelesenen Daten mit Sollwerten.

Die MIU- Einheit kann Einzel- sowie Parallel- Zugriffe auf Speicher bearbeiten und verfügt dementsprechend über zwei parallele Schnittstellen (Ports). Gerade bei den Parallelzugriffen auf externe Speicheraddressen sind detaillierte Tests nötig, weil hier Zugriffskonflikte auftreten können. Wie im Falle des IPQ- Moduls erzeugt die Testumgebung eine Folge von Speicheranforderungen mit verschiedenen Adressen. Dabei werden auch Zugriffe über beide Ports durchgeführt, beispielsweise Lese Port A und gleichzeitig Port B, oder Lese Port A und Schreibe über Port B, etc. .

2.4.2 System Test

Nach dem Funktionstest der einzelnen Module erfolgt die Verifikation des Systems mittels spezieller JAVA- Programme, die gezielt das Zusammenspiel zwischen Core und Compiler abprüfen. Diese Programme wurden auf einer Standard- JAVA- Umgebung (Personalcomputer, PC) entwickelt und während die Programme auf dieser Plattform ablaufen, werden die Ergebnisse als Referenzdaten für den Test des attoJAVA/Compiler- Systems protokolliert. Die Struktur der Testumgebung für das attoJAVA- System ist in Bild 5 gezeigt. Damit die vom JAVA- Compiler des PCs zunächst für eine Ausführung auf der JVM erstellten binären Dateien auch auf dem attoJAVA- Rechner laufen, sind einige Aufbereitungsschritte nötig. Die Binärdateien enthalten außer den Bytecode Befehlen noch weitere Informationen und symbolische Referenzen. Der attoJAVA- Compiler löst die Referenzen auf und extrahiert die Bytecode- Instruktionen, die dann vor der Ausführung des Programms auf dem attoJAVA-Rechner in einer separaten Datei gespeichert werden. Diese Datei wird vom code_mem- Modul der abgebildeten Testumgebung gelesen und dem attoJAVA- Core zur Verfügung gestellt. Die weiteren Komponenten der Testbench, wie der Clock- und Reset- Generator (syst_clk_rst), der Datenspeicher (data_mem), der Zeitgeber (timer), der Adressdekoder (addr_dec) und die Busweiche (bus_switch) bilden eine reale Prozessorumgebung nach. Die beim Ablauf der JAVA- Applikationen erzeugten Daten werden in ein File geschrieben und mit den Referenzdaten verglichen.

Diese Testumgebung wird im Zuge des Designprozesses wiederholt verwendet: zuerst für den funktionalen Test des attoJAVA- HDL- Modells, dann zur Verifikation der synthetisierten Gatterversion des Cores und schließlich zum Test des Zeitverhaltens des rückgelesenen Designs (Potlayout- bzw. Post- Place&Route- Test).

Um die Qualität der JAVA- Testprogramme und der gesamten Testumgebung zu bewerten, wurde die Codeabdeckung, also der Prozentsatz der bei der Ausführung der Testprogramme angesprochenen

VHDL- Anweisungen, ermittelt. Bild 6 zeigt das Ergebnisfenster der Modelsim- Simulators nach der Simulation des jetzigen Standes des attoJAVA- Modells. Die Abdeckung liegt weit oberhalb von 90% und ist damit mehr als ausreichend.

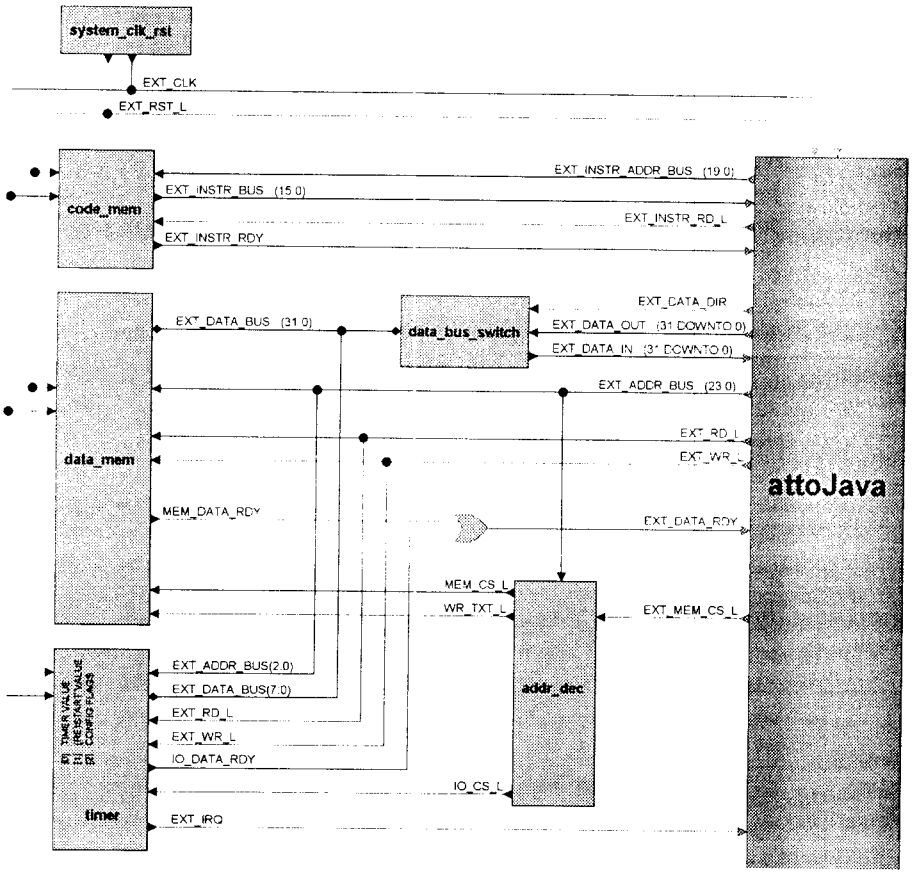


Bild 5: Test-bench des attoJAVA- Prozessors, Erklärungen im Text.

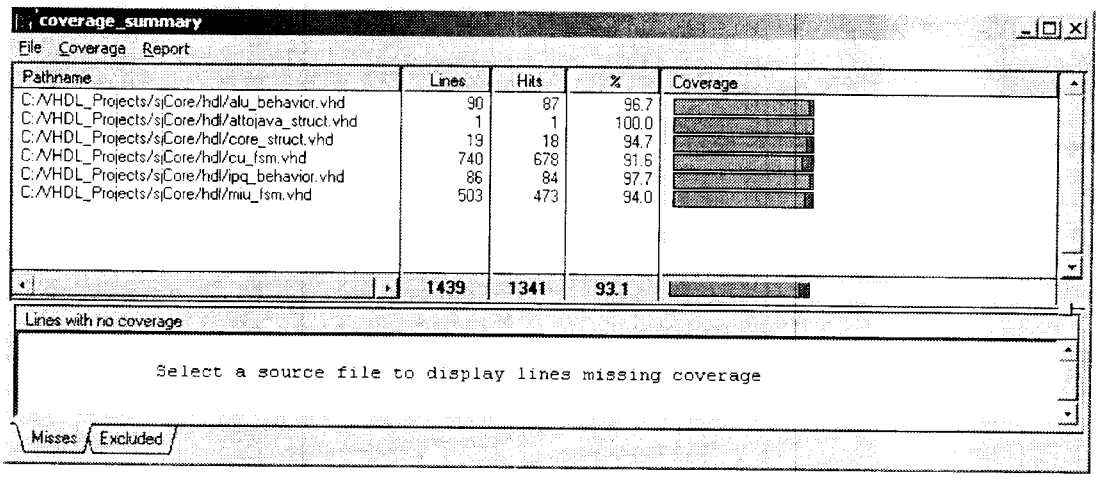


Bild 6: Die Test- Abdeckung des VHDL- Codes der System- Testbench liegt bei 93.1%.

3 Zusammenfassung

Der hier vorgestellte attoJAVA- Mikroprozessor soll die wichtigsten 66 Instruktionen aus dem Befehlssatz der JAVA Virtual Machine (JVM) direkt verarbeiten. Der vollständige Befehlssatz der JVM wird über einen neu entwickelten Compiler zugänglich, der alle nicht vom Prozessor per Hardware unterstützten Befehle in Folgen von attoJAVA- Instruktionen umwandelt und zwar vor der Laufzeit der jeweiligen Applikation. Dieser Hardware/Software- Codesign- Ansatz ermöglicht es, eine JAVA- Plattform bereitzustellen, die vom nötigen Hardwareaufwand jede existierende IP- Lösung unterbietet, ohne dass sich die Leistungsdaten verschlechtern oder dass die Echtzeitfähigkeit bei Embedded Anwendungen eingeschränkt wird. Aufgrund dieser Eigenschaften unterstützt der attoJAVA- Prozessor ganz wesentlich zukünftige Anwendungen der Sprache JAVA im kostensensitiven Marktsegment der eingebetteten Steuer- und Regelsysteme.

Das Prozessordesign wird auf der Basis von VHDL- Modellen mittels Logiksynthese durchgeführt und ist im Stadium der Systemanalyse und -entwicklung. Nach Abschluss aller Designarbeiten ist die Designwiederverwendung als Soft- IP- Komponente ohne Probleme möglich. Die Zieltechnologien, die für Test- und Evaluierungszwecke ausgewählt wurden, sind VirtexII CPLDs und der 0.35µm CSD CMOS Prozess von AMS. In der CSD- Technologie sind Taktfrequenzen von 50 MHz zu erwarten bei ca. 15.000 Gattern. Die Verlustleistung liegt nach ersten Simulationen in CSD- Technologie bei 2mW pro MHz Taktfrequenz.

In Zukunft sollen die hier beschriebenen Arbeiten fortgesetzt werden. So soll ein Testdesign des Prozessors auf einem komplexen CPLD emuliert und später als Chip gefertigt werden. Mittelfristig sind die folgenden Fragestellungen von besonderem Interesse:

- Der Gatterbedarf und Verdrahtungsaufwand der über Zustandsautomaten definierten Steuereinheit bestimmt im wesentlichen den Hardwareaufwand für diesen Prozessor. Deshalb sollen alternativ mikrokodierte Steuereinheiten entwickelt und mit der bisherigen Lösung verglichen werden;
- Um den aufwendigen Postproduktionstest für den komplexen Rechnerkern zu vereinfachen sollen Selbstteststrategien (Built in Self Test, BIST) für den attoJAVA- Core entwickelt werden [10];
- Um vollständige Kompatibilität zur JVM zu erreichen, sind die Fließkommaoperationen der JVM einzubeziehen und ein Echtzeitbetriebssystem wird benötigt, dass die Parallelverarbeitung von unterschiedlichen Aufgaben (Multitasking) ermöglicht.

Die Autoren bedanken sich beim Zentrum für Forschung und Entwicklung der FH- Darmstadt für die gewährte finanzielle Unterstützung.

4 Literaturverweise

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

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

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

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

[5] R&D Technologies Ltd: African Blue, A JAVA Byte Code compatible Processor, http://www.r-and-d.de

[6] M. Keating und Pierre Bricaud: „Design Reuse Methodology Manual for System on Chip Designs" Kluwer Academic Publishers, Dordrecht, 1998

[7] De Micheli, G. und Gupta, R.: „Hardware- software co- design", Proc. IEEE, Vol. 85 (3), pp. 349 – 365, 1997

[8] Philip Koopman: „Stack Computers the New Wave", La Honda: Mountain View Press, 1989

[9] Harlan McGhan und Mike O'Connor: „Pico JAVA: A Direct Execution Engine For JAVA Bytecode", IEEE Computer, S. 22- 30, 1998

[10] Li Chen und Sujit Dey: „Software Based Self- Testing Methodology for Processor Cores", IEEE Trans. on Computer Aided Design of Integrated Circuits and Systems, Vol. 20, No. 3, S. 369- 380, 2001

Data Management for Electronic System Design Environment

Dr. Uwe Schmid, Prof. Dr. Johann Siegl
Mentor Graphics Corp., Data Management Systems Business Unit, Nürnberg
www.mentor.com/dms

In order to shorten design cycles, collaborative and team design is needed in a distributed environment. Object-oriented data management systems are now available as servers to facilitate the exchange of information between team members within a design community, and to identify objects more easily (e.g. status and version information) and their relation to other objects. An integrated client/server database is used as both an information source and an exchange platform for released objects. This makes data management systems information hubs in today's design community.

Introduction

When preparing a design for a given application, designers use tool suites for design definition, verification and for preparing a layout for a target technology. Figure 1 shows well-known tool suites for electronic system design.

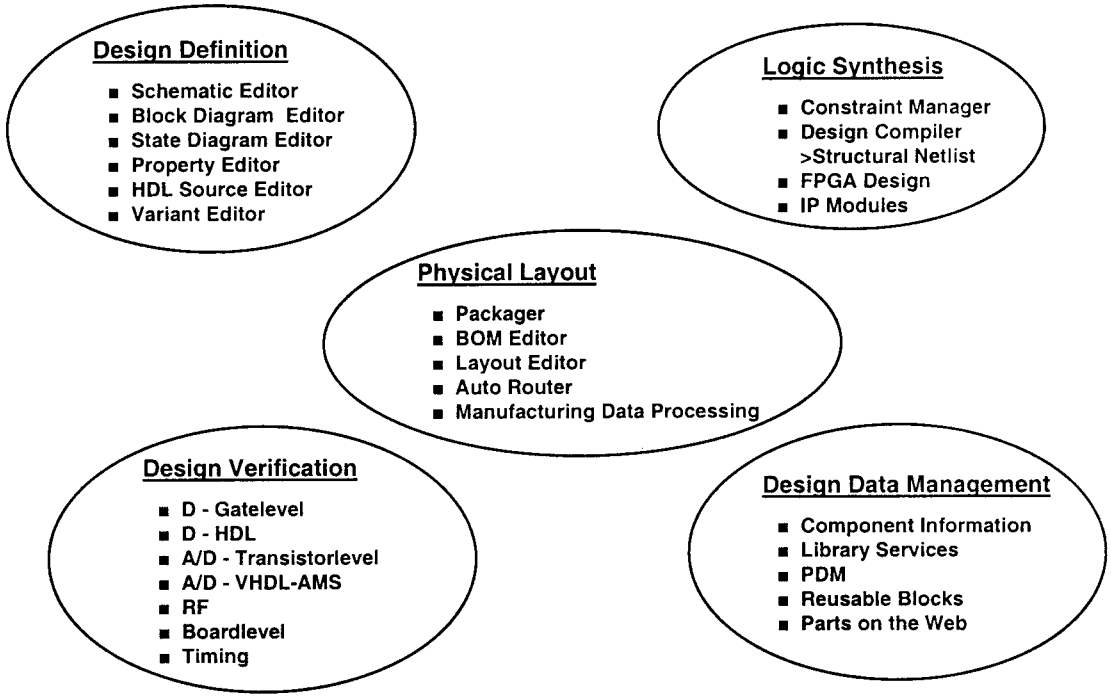


Figure 1: Typical Supporting Infrastructure for a Design Environment

Design methods require libraries, which are used to create design specific objects that are stored in a workspace (a directory or file-based repository). Setup data is needed to define initial design constraints and determine preliminary library references. Each design tool suite has its own database for library and setup information. The existence of these separate databases raises the following question: *How can the local environment be expanded to bring all necessary information*

together in a workgroup database, which can be accessed by all team members of a workgroup so that collaborative design is supported?

Designers working in a desktop environment work with specific design tools and local repositories in the user environment. Within the local repositories are working partitions for a specific design. Once a design module has reached a certain state of maturity, it is released to the workgroup environment, and the information is transferred to a master database on the workgroup server.

Figure 2 shows a design environment that uses a workgroup database for all team members who have controlled access for capturing and distributing exchangeable information. This establishes interaction, communication and coordination between the desktop and workgroup environments.

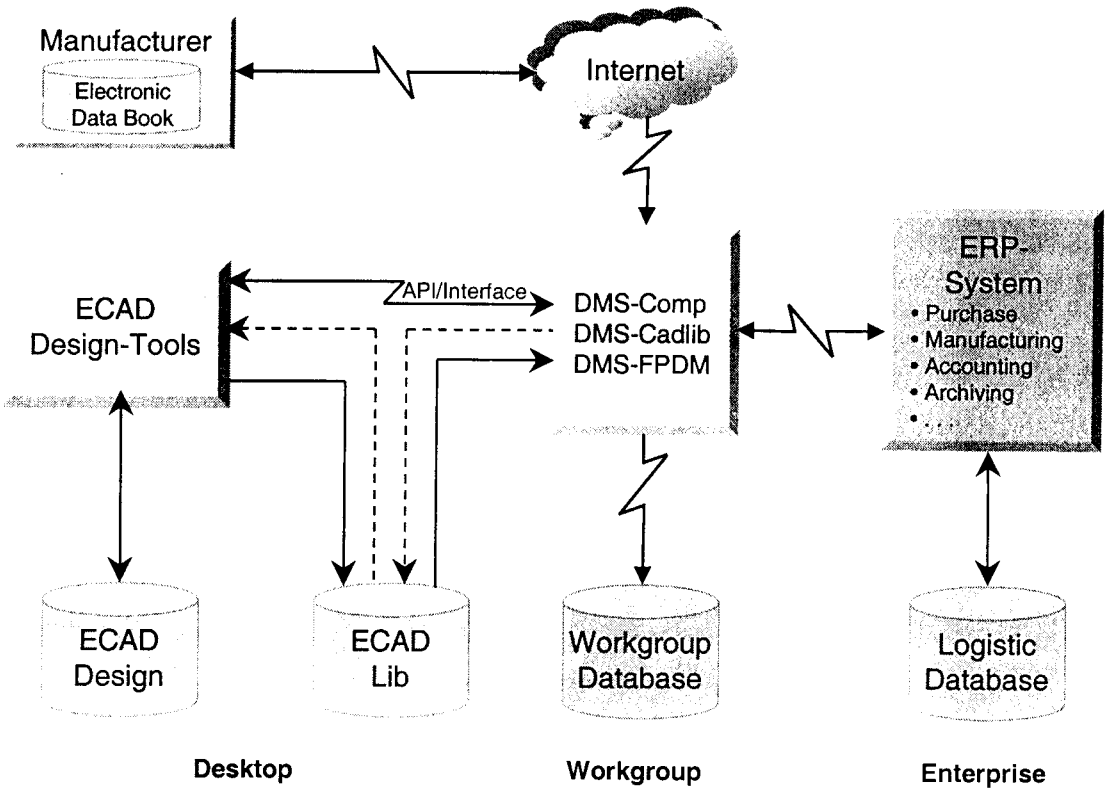


Figure 2: Team Design Using Distributed Databases Including Access to an Enterprise Server

Exchange Platform

Several design steps make up a design process, and designers need access to local repositories for each step. In order to bring exchangeable information into a workgroup database, a data management system must have a general database access layer and an interface to clients. It must also offer an object-oriented view and arrange all captured information in the workgroup database in object classes. This allows designers to have transparent access to the data management system throughout each design step.

The basic concept is to use a complete information exchange platform, which is shown in Figure 3. One portion of the figure shows the local repositories of the design methods at the desktop of a user. Different views of design primitives and reusable blocks are available in libraries. The same applies for setup data. The workspace is a local repository for non released design specific

objects. Libraries are downloaded from the master database of the workgroup server to the local repositories.

The data management services can be limited to desktop applications. These services have to be expanded to workgroup or even enterprise applications. With the Standard GUI it is possible to define a logical select for retrieving objects in distributed servers and get a hitlist back. As WebClient a defined logical select can be transferred to a known WebServer.

Distributed environments of design and manufacturing resources need client/server architecture, a transportation layer for object exchange, workflow control, powerful search&view functions, release management, configuration control, and version control. If an object is selected, the user will get information about possible views of the selected object (symbol view, model view, physical view, related documents) and all related information including references to other objects, status information, versioning etc.

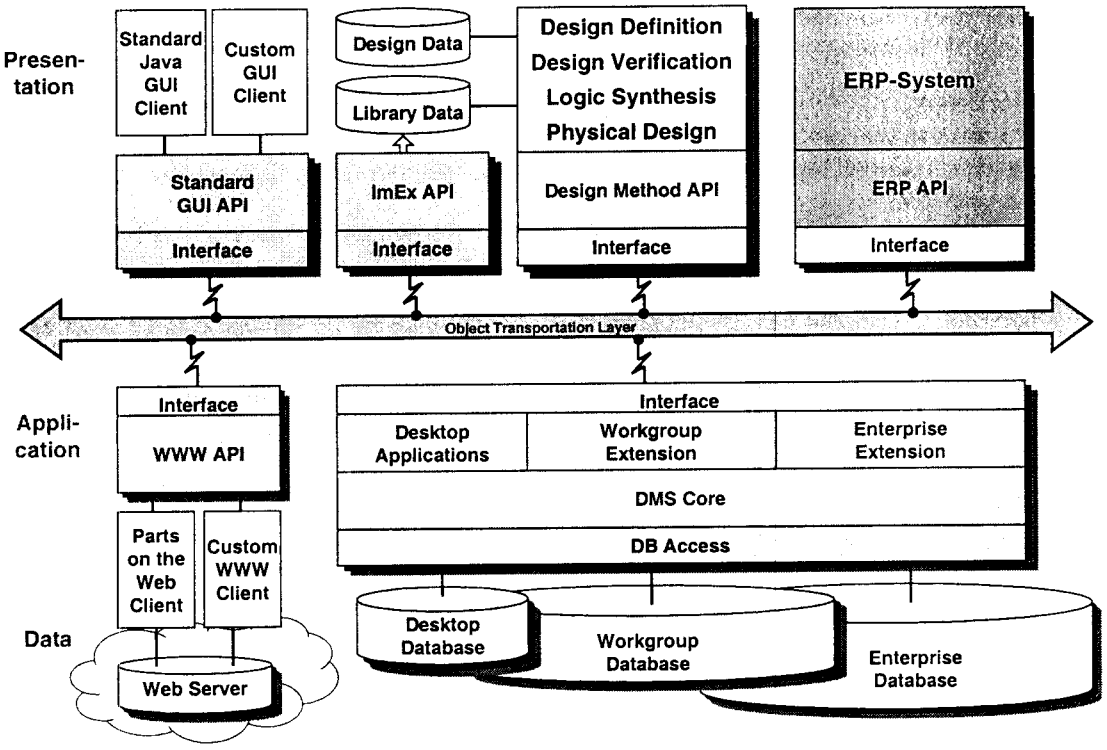


Figure 3: Basic Architecture of an Information Exchange Platform

Integration into Design Processes

There are different design processes introduced for electronic system design. As an example design definition by schematic entry and physical design by layout tools need part libraries, symbol libraries, model libraries, footprint libraries, padstack libraries, e.g. For this specific design process Data Management Services have to be integrated into design tools for e.g.

- "Instantiation Process" (Virtual and Physical Instantiation);
- "Assign_to Function" for assigning physical parts to virtual parts;
- Bill of Materials (BOM) Exchange;
- Design Container Exchange.

Designers can look for parts or existing design objects (e.g. reusable blocks) already released in the master database of a workgroup server by doing a logical search. By selecting an object, they

can view all relevant information about the object, including its relations to other objects. Within the design entry phase, virtual parts are represented by a symbol without an assigned part number (virtual instantiation). Physical parts with existing part numbers can be instantiated into a design (physical instantiation). Using the assign_to function, designers can look for available physical parts that match the attributes for instantiated parts having no part number (virtual instantiated parts).

Upon completing the design entry step, a packager creates a physical part list from the instantiation list. Parts lists are critical for the exchange of information and are used to create BOMs. BOMs are transferred to the exchange platform, where a BOM manager introduces and handles variants for the design.

After packaging and generating the local database for the physical design, the layout process can start. Design data is stored and captured in the local design workspace. After post processing, all relevant design data in the workspace is processed and packed into the design container. This packing process concentrates all necessary information needed for archiving the design, including the document structure.

When libraries are downloaded to local repositories in the workspace at the desktop of a user, they have to be under the strict control of the data management system. New library objects must be first introduced into the master database of a workgroup server, and then downloaded to the local repositories.

A major problem is synchronizing and updating local repositories from the master database of a workgroup server. To make this synchronizing and updating process easier a desktop data management system can be introduced into the design environment to (1) avoid inconsistencies between local libraries and libraries available at the exchange platform, and (2) establish full control of distributed libraries. Overall database control consistency is obtained by synchronizing the processes between the desktop database and workgroup database. Object transportation layer controls the transfer of ordinary objects with their attributes, graphical objects with their attributes and lists of objects.

Having access to distributed information at desktop, workgroup or enterprise level, it is possible to get correct information at the right time for a specific task in complex business processes. There is full control throughout the lifecycle for each recorded object possible. If an object is transferred from one task to another task, check functions can be performed to make sure that the stage of maturity or the version is correct. Figure 4 shows the bridge between supply chain control and data management in electronic system design environment.

IMS CHIPS Siliziumtechnologie

H. Richter

Institut für Mikroelektronik Stuttgart
Allmandring 30a, 70569 Stuttgart
http://www.ims-chips.de

Übersicht

- 1. Einleitung: IMS CHIPS Aufgaben und Historie
- 2. Silizium Technologie Überblick
- 3. qualifizierter 0.8 µm CMOS Prozess
- 4. Sonderprozesse auf MOS Basis:
  - ↳ Beispiel 300V (D)MOS Prozess
- 5. Membran- und Stencil Masken
- 6. Zusammenfassung

IMS CHIPS(I)

Aufgaben

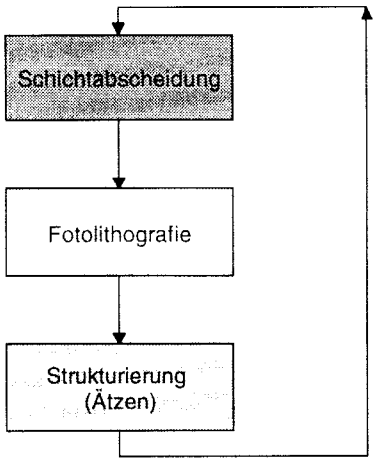
- Entwurf mikroelektronischer Schaltungen und Systeme
- Herstellung mikroelektronischer Versuchs-, Prototyp- und Serienschaltungen
- Forschungs- und Entwicklungsprojekte
- Aus- und Weiterbildung

IMS CHIPS(II)

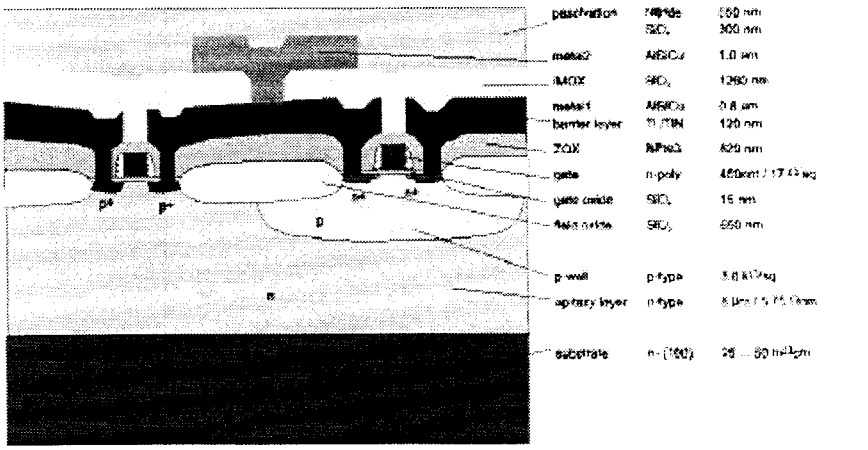
Geschichte

- gegründet 1983 durch Land BW
- Reinraumbetrieb seit 1987
- GATE FOREST 1989 eingeführt
- 0.8 µm Technologie seit 1995 in Produktion
- z.Z. 75 Mitarbeiter + 20+ Studenten
- Jahresetat: 9 MEURO

2 - Silizium Technologie

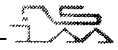


2 - 0.8µm CMOS Prozess



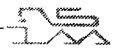
2 - IC Trends

	1995	1997	1999	2001	2006
Technologie	0.35µm	0.25µm	0.18µm	130nm	70m
DRAM	64M	256M	1G	4G	64G
Gatter / mm^2	12k	20k	35k	70k	160k
Wafer (mm)	200	200	300	300	450
Metallebenen	5	6	6-7	7-8	8-9
Masken	18	20	22	23	25

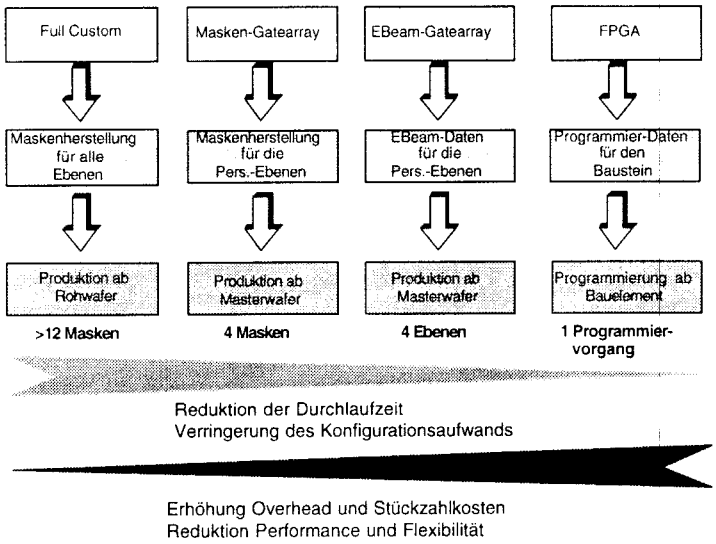


3 - qualifizierter 0.6 µm CMOS Prozess

- Umfeld und Randbedingungen
- GATE ARRAY Architektur
- Prozesscharakteristika
- Qualifizierungen
- CMOS Imager



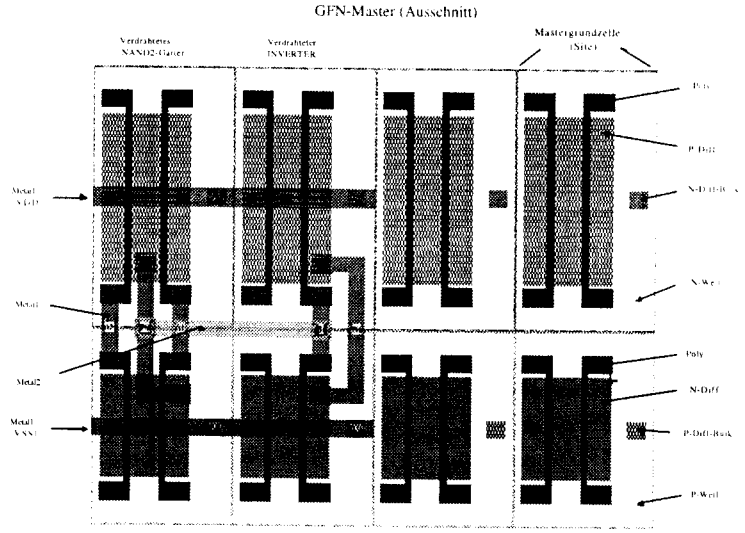
3 - IC Herstellungsmethoden



3 - GATE FOREST Charakteristika

- kanallose Sea-of-Transistor Architektur
- maskenlose Prototypen und Kleinserienfertigung
- Komplexität bis 80.000 Gatter
- Stückzahlen 100 bis 100.000 Stück
- Digital mit A/D Schnittstellen, 5V / 3.3V

3 - GATE FOREST Layout



HR/MPC02/11

3 - GATE FOREST Master Familie

GATE FOREST Master Familie

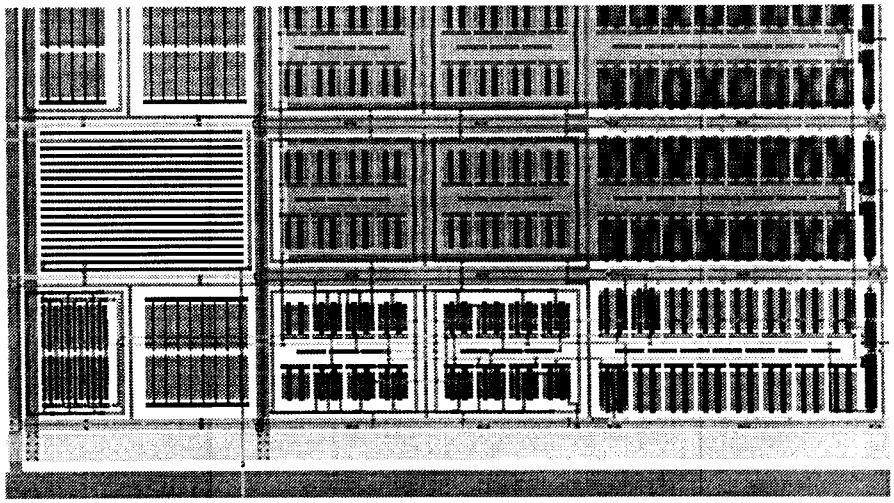
Master type	Maximum usable functions digital gates	analog tiles	Maximum I/O Pins	Standard package
GFN 001	300	-	34	C / PLCC28
GFN 002	1600	-	56	C / PLCC28
GFN 004	3500	8	82	C / PLCC44
GFN 012	9500	14	136	C / PLCC68
GFN 024	18000	20	184	C / PLCC84
GFN 060	45000	31	284	C / PQFP100
GFN 090	63000	38	320	C / PQFP160
GFN 120	80000	46	408	C / PQFP208

HR/9943/12

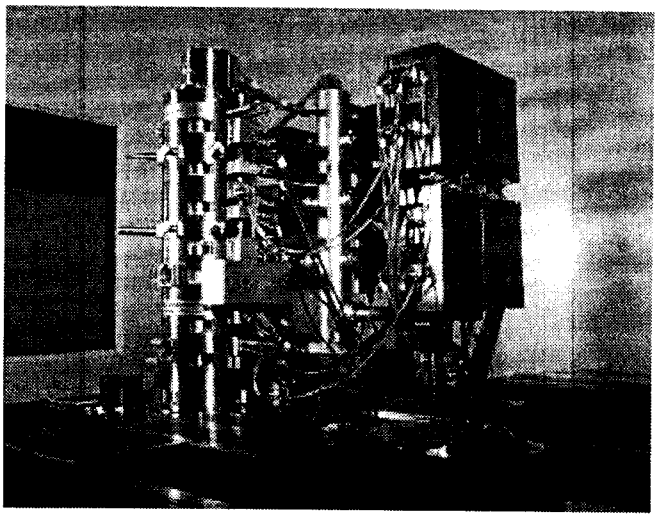
3 - Analoge Komponenten

- Komparatoren
  - getaktet, 50 MHz, Offset < 10mV
  - kontinuierlich, 2MHz, Offset < 6mV
- Operationsverstärker
  - standard 80 dB, 10 MHz, Offset < 10mV
  - Ausgangstreiber 60 dB, 30 MHz, Offset < 20mV
- Schalter, Multiplexer
- Widerstandsnetzwerke
- Band gap Referenz
- Power-on-reset

3 - Analog-Elemente

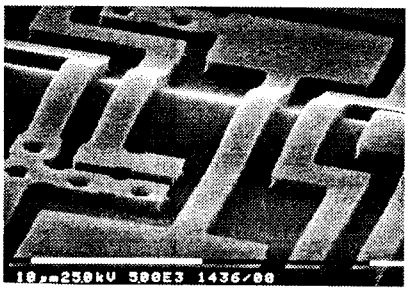


3 - Elektronenstrahlschreiber

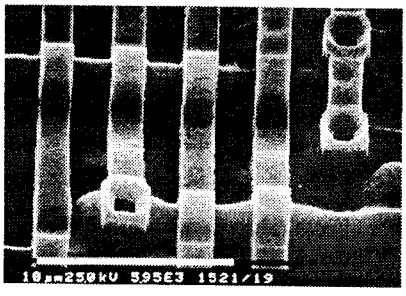


3 - GATE FOREST Personalisierung

Contact / Metal1



Via / Metal2



3 - Qualifizierung

- ISO 9001 und DIN EN 100 114/VI zertifiziert
  - standardisierter, verifizierter Design Ablauf
  - kontrollierte Produktion basierend auf qualifizierten Einzelprozessen mit umfangreichen Prozesskontrollen
  - 100% Wafer und Bauelemente Test
  - qualifizierte Zulieferer
  - produktbegleitenden Qualitäts- und Zuverlässigkeitsuntersuchungen

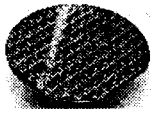
3 - Produktionsnetzwerk

Qualifiziertes GATE FOREST Produktionsnetzwerk

Infineon

ISO 9001

bis 1999



MICRONAS

ISO 9001, QS9000

ab 1999

ams

ISO 9001, DIN EN 100 114/VI
Entwurf
Waferherstellung + Test
Keramikmontage
BE Test

ASAT

ISO9001, SAC1.0, QS9000

Frankreich (bis 3/97)

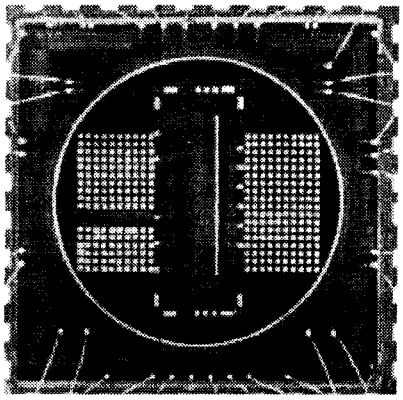
Wales (4/97 bis 6/99)

Hong Kong (ab 6/99)

microtec

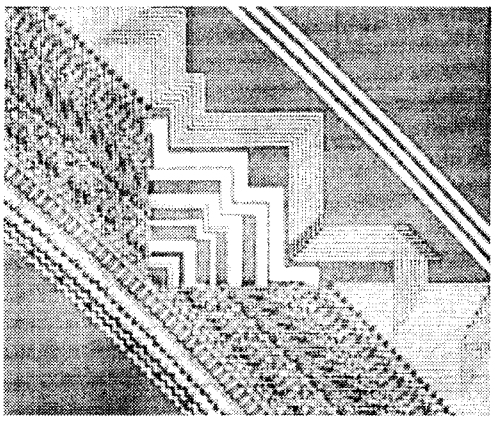
ISO 9001, DIN EN 45001

3 - CMOS Imager: Ringsensor (i)



- 2048 Pixel (9.4µm x 18.8µm)
- Integrierend oder log. HDRC Modus
- Ausgangspegel: 0,5V (log.) oder 2V (lin.)
- > 20 Mpixel / sec

3 - CMOS Imager: Ringsensor (ii)



Besonderheiten:

Strukturen mit beliebigen Winkeln im Fotodioden-bereich

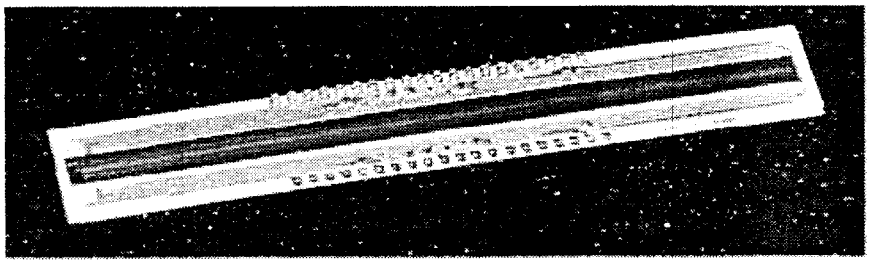
Gridorientiertes Design der Analog- und Digitalelektronik

Plazierung und Verdrahtung mit C++ Programm

Anwendung:

Oberflächeninspektion vom Stäben, Kolben, Glasrohren etc.

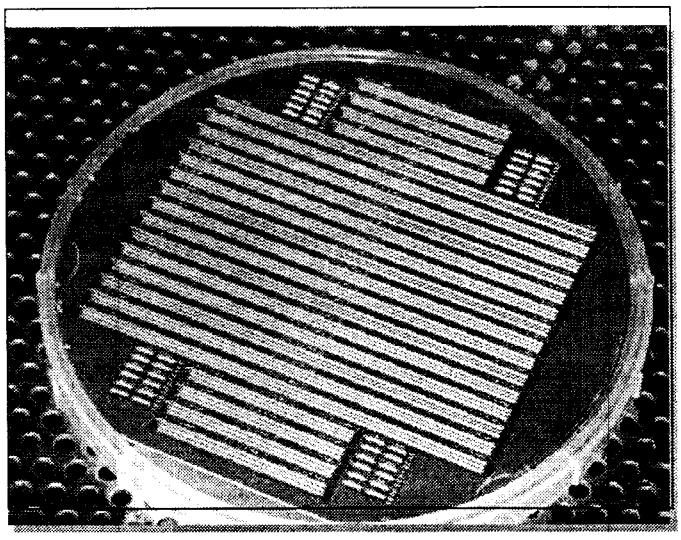
3 - CMOS Imager: Lange Zeile (i)



Scannerzeile für satellitengestützte Erdbeobachtung

- 10240 Pixel, 10.8µm Pixelpitch
- Integrierender Sensor, programmierbare Integrationszeit
- Hohe Datenrate (80MS/s)
- Correlated Double Sampling CDS
- Ansteuerlogik auf dem Chip

3 - CMOS Imager: Lange Zeile (ii)



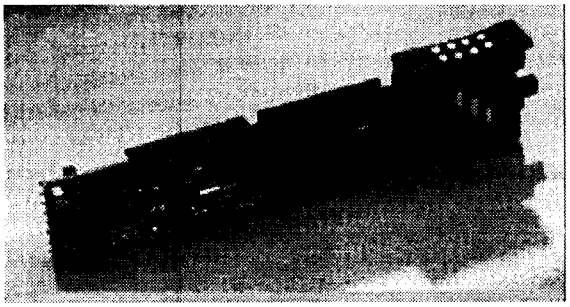
3 - Technologie Roadmap

Generation	1.2 µm digital	0.8 µm digital	0.8 µm mixed signal	0.5 µm digital	0.5 µm mixed signal
Entwicklung Produktion	1991 - 1992 1992 - 1995	1994 - 1995 1995 -	1995 - 1997 1997 -	2001 - 2002 2002 -	2001 - 2003 2004 -
Komplexität	1k bis 20k Gatter	2k bis 80k Gatter		20k bis 500k Gatter	
Gatterdichte	700 / mm^2	1200 / mm^2		3000 / mm^2	
Analog-Kennzahlen			8 Bit / 1MHz		10 Bit / 3MHz
Verdrahtung	6 µm	4 µm		2,4 µm	
Stückzahlen	bis 20.000	bis 100.000	bis 10.000	bis 500.000	bis 50.000
Technologie Partner	Intermetall Siemens	Siemens Micronas	Siemens	Micronas	Micronas

4 - HV MOS Prozess

- Anwendung
- Spezifikation
- DMOS Transistor
- Prozessdaten
- Schaltung und Layout
- Chip
- Ergebnisse

4 - Anwendung HV ASIC



Firma Metec fertigt Braille-Module, die Blinden die Arbeit am PC ermöglicht.

Zwei ASICs werden im Modul benötigt:

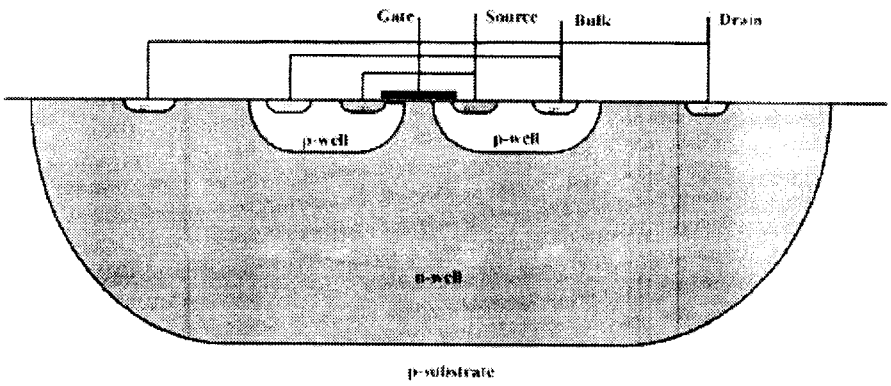
-NV-ASIC: setzt Mitteilungen des PCs in Ansteuersignale für den HVASIC um
-HV-ASIC: steuert Piezo-Aktuatoren, die Braille-Stifte heben und senken

4 - HV ASIC Spezifikation

Spezifikationen des HV - ASICs

- $V_{sperr} \geq 220V$ ($V_{Abbruchr} \geq 250V$)
- Risetime, Falltime=50msec...150msec (HV=200V, C_L=12nF)
- Leckstrom≤20µA (alle Eingänge „high" oder alle Eingänge „low")
- Kurzschlussstrom ≤200µA

4 - HV DMOS Transistor

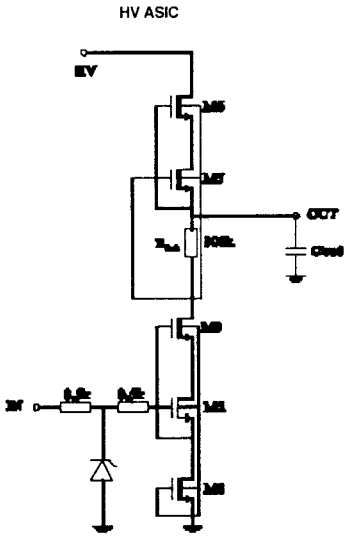


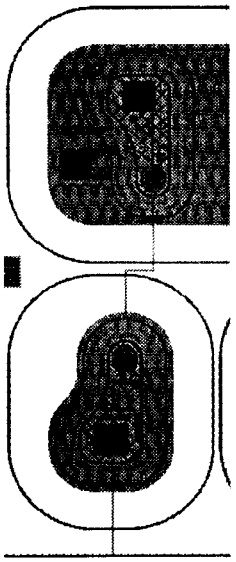
4 - HV Prozess

Eckdaten des Prozesses

- Ausgangsmaterial p-Si 600Ωcm ± 200 Ωcm
- Drive-In n-Wanne 500h, 1200 °C
- Drive-In p-Wanne 11h, 1200 °C
- 100nm Gateoxid
- 600nm Polygate n-dotiert, Lpoly= 5 µm
- single metal
- 14 Maskenebenen
- 9 Implantationen

4 - Schaltung und Layout



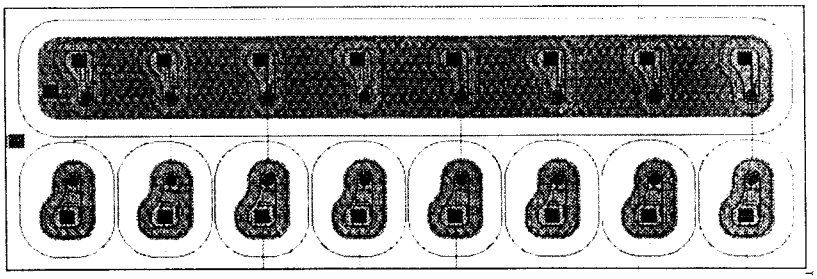


HR/MPC02/29

4 - HV - Chip

Layout HV-ASIC

Chip-Größe: 2.3x6.2 mm^2=14.3 mm^2

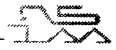


HR/MPC02/30

4 - HV Ergebnisse

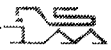
Experimentelle Ergebnisse

• Abbruchspannung Chip:	260V...290V
• Risetime, falltime:	40msec ...150msec
• Leckstrom:	< 20µA
• Kurzschlussstrom:	< 200µA
• Funktionaltest	Ausbeute 87% Nullserie
• Test von Modulen bei Metec:	Dauertest, stat. Test erfolgr.
• Burnin 168h (200V, 50Hz,125 C)	0/30
• Lebensdauertest 1000h	1/30

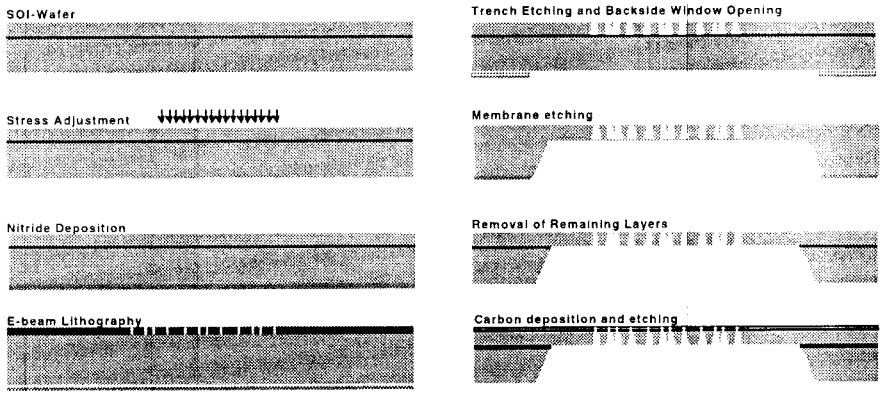


5 - Loch- und Membran Masken

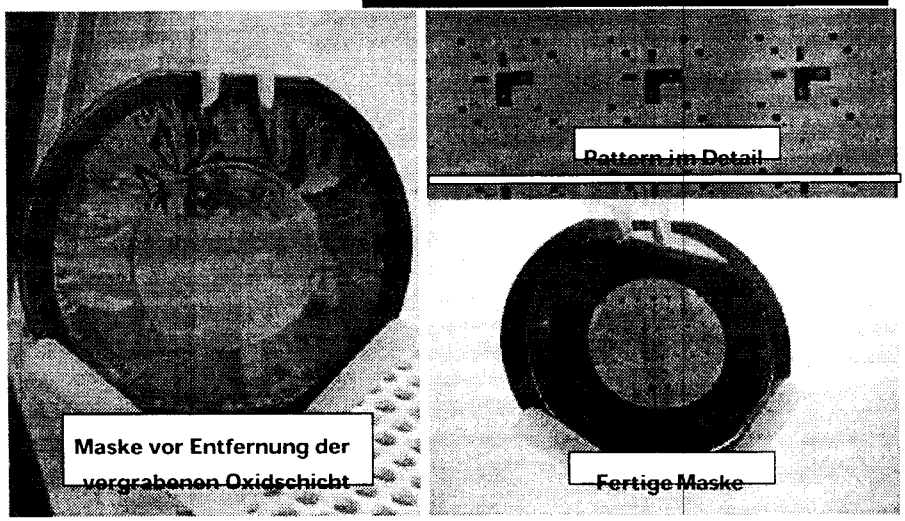
- Technologische Grundlagen:
  - ↳ (SOI) Wafer Flow Prozess
- Lochmasken
- Maskenstrukturen
- Membranmasken



5 - SOI-Wafer Flow Prozess (I)

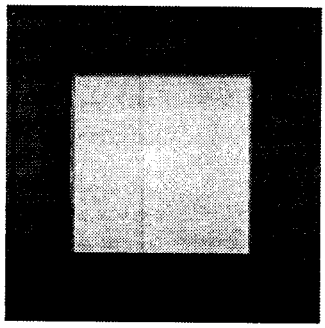


5 - SOI Wafer Flow Prozess (II)



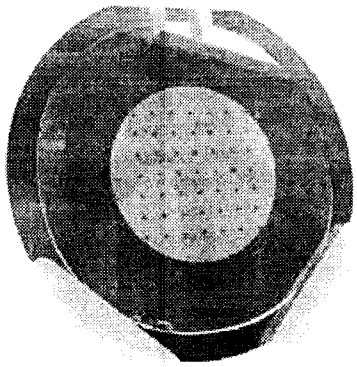
5 - Lochmasken (i)

- Optische Gitter



- Herstellung:
  - SOI Wafer Flow Prozess
- Membran:
  - 1.8µm Silizium
  - 25mmx25mm
- Minimale Strukturgröße:
  - 0.9µm
- Öffnungsgrad:
  - ~50%
- Einsatzgebiet:
  - IR Optik

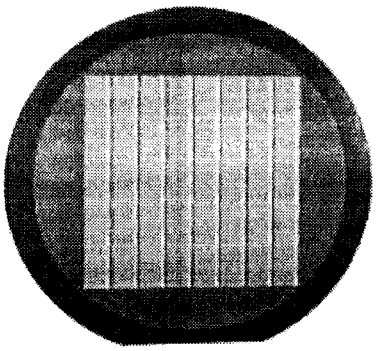
5 - Lochmasken (ii)



- Herstellung:
  - SOI Wafer Flow Prozess
- Membran:
  - 3µm Silizium
  - 126mm ∅
- Minimale Strukturgröße:
  - 0.2µm
- Öffnungsgrad:
  - 37%
- Einsatzgebiet:
  - Verifikation der optischen Eigenschaften des Ionenprojektors

5 - Lochmasken (iii)

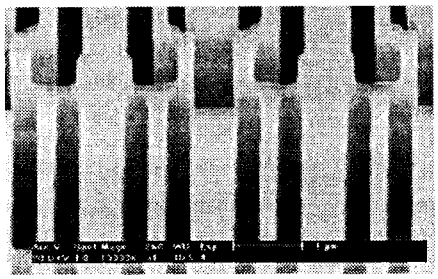
- IPL Testmasken



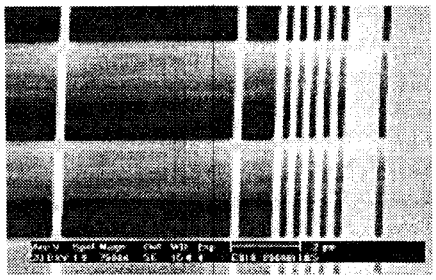
- Herstellung:
  - SOI Wafer Flow Prozess
- Membran:
  - 1.8µm Silizium
  - 126mm ∅
- Minimale Strukturgröße:
  - 0.35µm
- Öffnungsgrad:
  - ~20 %
- Einsatzgebiet:
  - Komplementärdesigns zur Abbildung von in sich geschlossenen Strukturen

5 - Maskenstrukturen (i)

- Realstruktur
  - Minimale Strukturgröße 0.3µm
  - Strukturtiefe 3µm
  - DRAM-Pattern

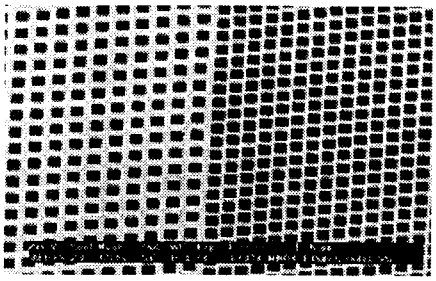


- Teststruktur
  - Minimale Strukturgröße 0.2µm
  - Strukturtiefe 3µm
  - 200nm Linien und Gräben

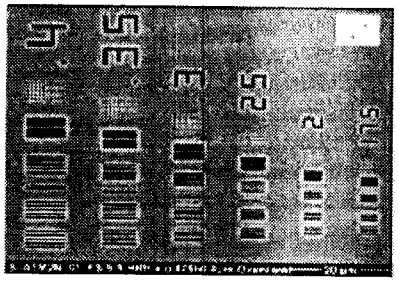


5 - Maskenstrukturen (ii)

- Stabilitätstest
  - Minimale Strukturgröße 0.3µm
  - Strukturtiefe 3µm
  - Öffnungsgrad >60%

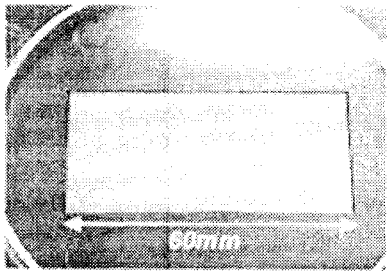


- Auflösungstest
  - Minimale Strukturgröße 175nm
  - Strukturtiefe 3µm
  - 175nm Linien und Löcher



5 - Membranen (i)

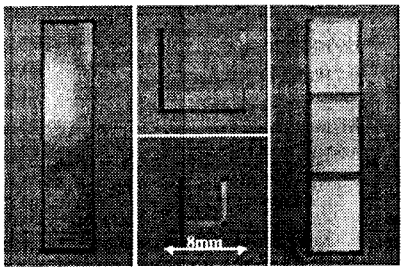
- Strahlaustrittsfenster



- Herstellung:
  - SOI Wafer Flow Prozess
- Membran:
  - 0.5µm Silizium
  - 60mmx30mm
- Minimale Strukturgröße:
  - unstrukturiert
- Einsatzgebiet:
  - Austrittsfenster für weiche Röntgenstrahlung BESSY

5 - Membranen (ii)

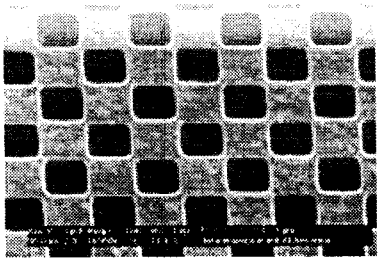
- Polykristalline Membranen



- Herstellung:
  - Modifizierter SOI Wafer Flow Prozess
- Membran:
  - 0.2µm Poly SiGe (50% Ge)
  - 25mmx5mm / 8mmx8mm
- Minimale Strukturgröße:
  - unstrukturiert
- Einsatzgebiet:
  - Membranmasken

5 - Absorbermasken (i)

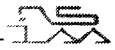
- Absorbermaske mit durchgängiger Membran



- Herstellung:
  - Wafer Flow Prozess
- Membran:
  - 0.18µm Siliziumnitrid
  - 5mm x 5mm
- Absorber:
  - 0.15µm Aluminium
- Minimale Strukturgröße:
  - 0.25µm
- Öffnungsgrad:
  - 1-50%
- Einsatzgebiet:
  - Messtechnik

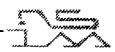
6 - Zusammenfassung

- Basis für IMS CHIPIS Si Technologien:
  - qualifizierte, industriekompatible CMOS Prozesse
- breites Produktspektrum auf dieser Basis:
  - Digital / Mixed-Signal / Sensor ASICs
- abgeleitete Sonderprozesse :
  - 300V (D)MOS Prozess
- Si Technologien für nicht-elektrische nm Strukturen
  - Loch- und Membran- Masken



6 - Ansprechpartner

• CMOS Technologie	Dr. W. Appel	appel@ims-chips.de
• CMOS Imager	H.G. Graf	graf@ims-chips.de
• ASIC Entwurf	Dr. H. Richter	richter@ims-chips.de
• HV MOS	Dr. R. Grube	grube@ims-chips.de
• Masken	Dr. R. Springer	springer@ims-chips.de



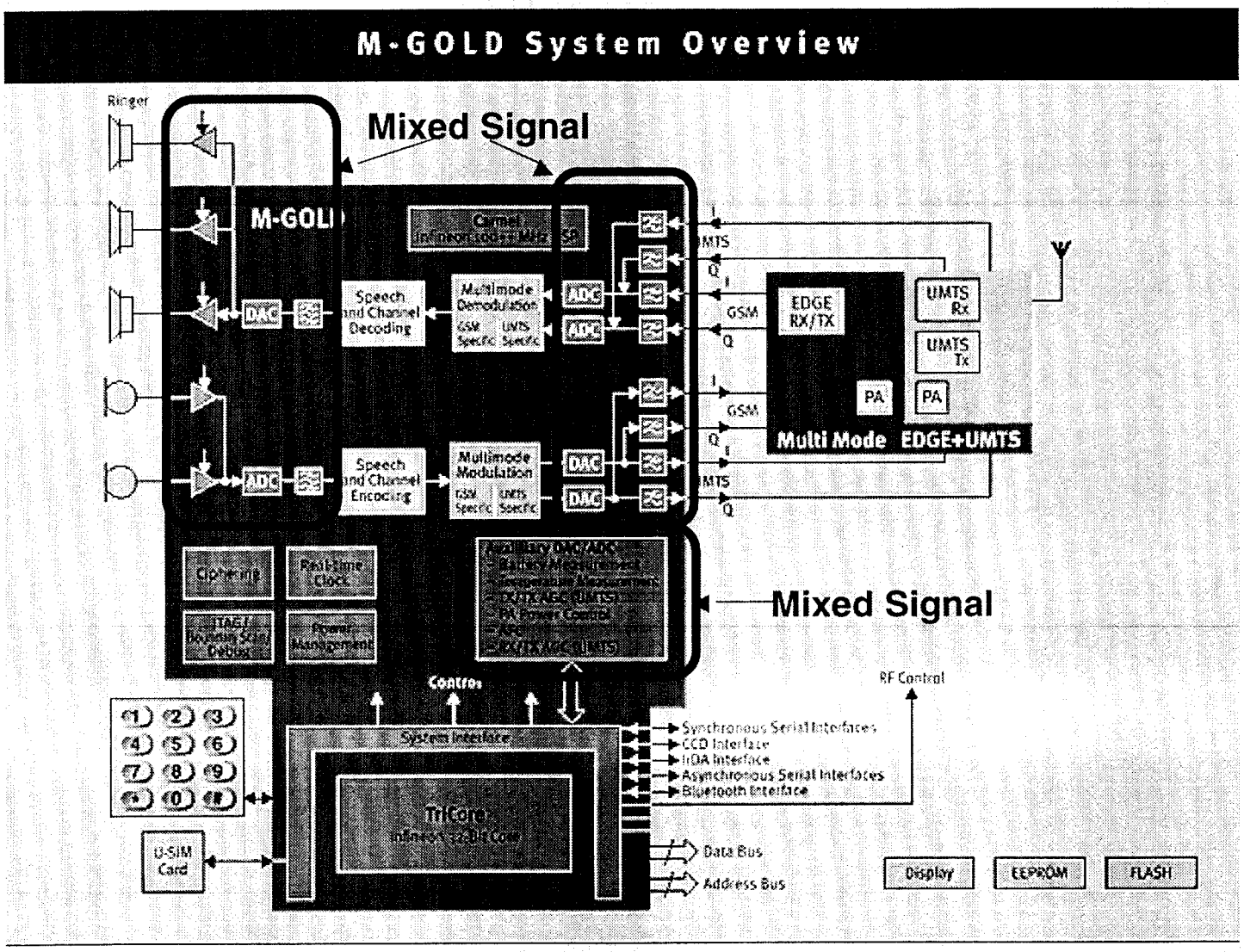
Mixed-Signal Design Methodik für die SoC-Integration in Very Deep Sub-Micron Technologien

Dr. Josef Eckmüller
Infineon Technologies
MPC Workshop
25.01.02, Göppingen

Arbeitsgebiet

- Entwicklung von SoC's für die Telekommunikation
  - drahtgebundene Kommunikation
    - z.B. ISDN, Breitband Kommunikation (ADSL, VDSL), High Speed Kommunikation IC's (fiber optic)
  - drahtlose Kommunikation
    - z.B. Cellular (GSM, GPRS, UMTS), Bluetooth, Wireless LAN
- Entwicklung der Mixed Signal Module findet in unserem Bereich statt
  - CMOS Technologie (derzeit 130nm)
  - ADC's, DAC's, Verstärker, Filter, PLL's
  - Schwerpunkt: Mixed Signal Design Methodik

Beispiel: UMTS Basisband Chip



Motivation

- Konsequenzen des technologischen Fortschritt in der Halbleiterindustrie
  - für die Mixed-Signal Entwicklung und
  - die Folgen für die Mixed-Signal Design Methodik
  - Lösungsansätze aus heutiger Sicht
- Industrielle Anwendung neuer Design-Methodiken für die Mixed Signal Entwicklung

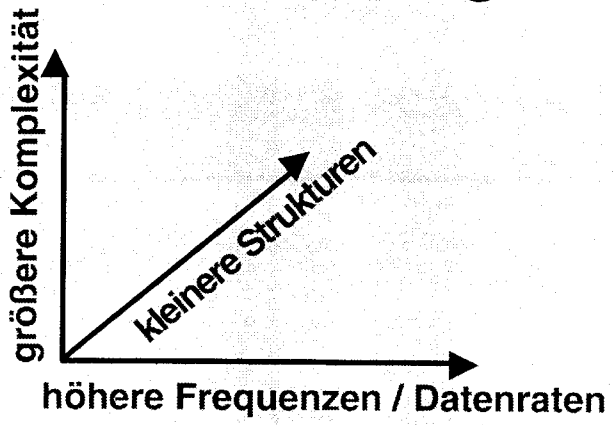
SIA Roadmap

Jahr	*2000*	*2002*	*2004*	*2006*	*2008*	*2010*	
min. Struktur	180	130	100	70	50	35	nm
DRAM	1G		8G		64G		
Fmax für mP	1250	2100	3500	6000	10000	13500	MHz
# IO	2304	3042	3840	4424	4416		
Pvmax	90	130	160	170	174	183	W
Amax	450	567	622	713	817	937	mm^2
VDD	1.8/1.5	1.5/1.2	1.2/0.9	0.9/0.6	0.6/0.5	0.6/0.3	max

Stand 1999

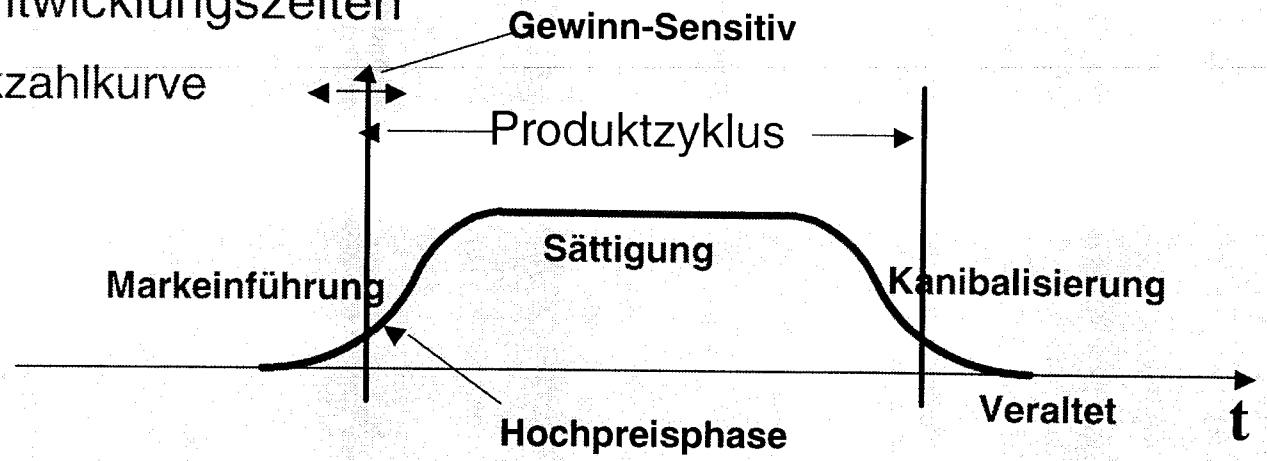
Anforderungen an MS Design Methodik

Technische Herausforderungen



Wirtschaftliche Herausforderungen

- kurze Entwicklungszeiten
  - Stückzahlkurve

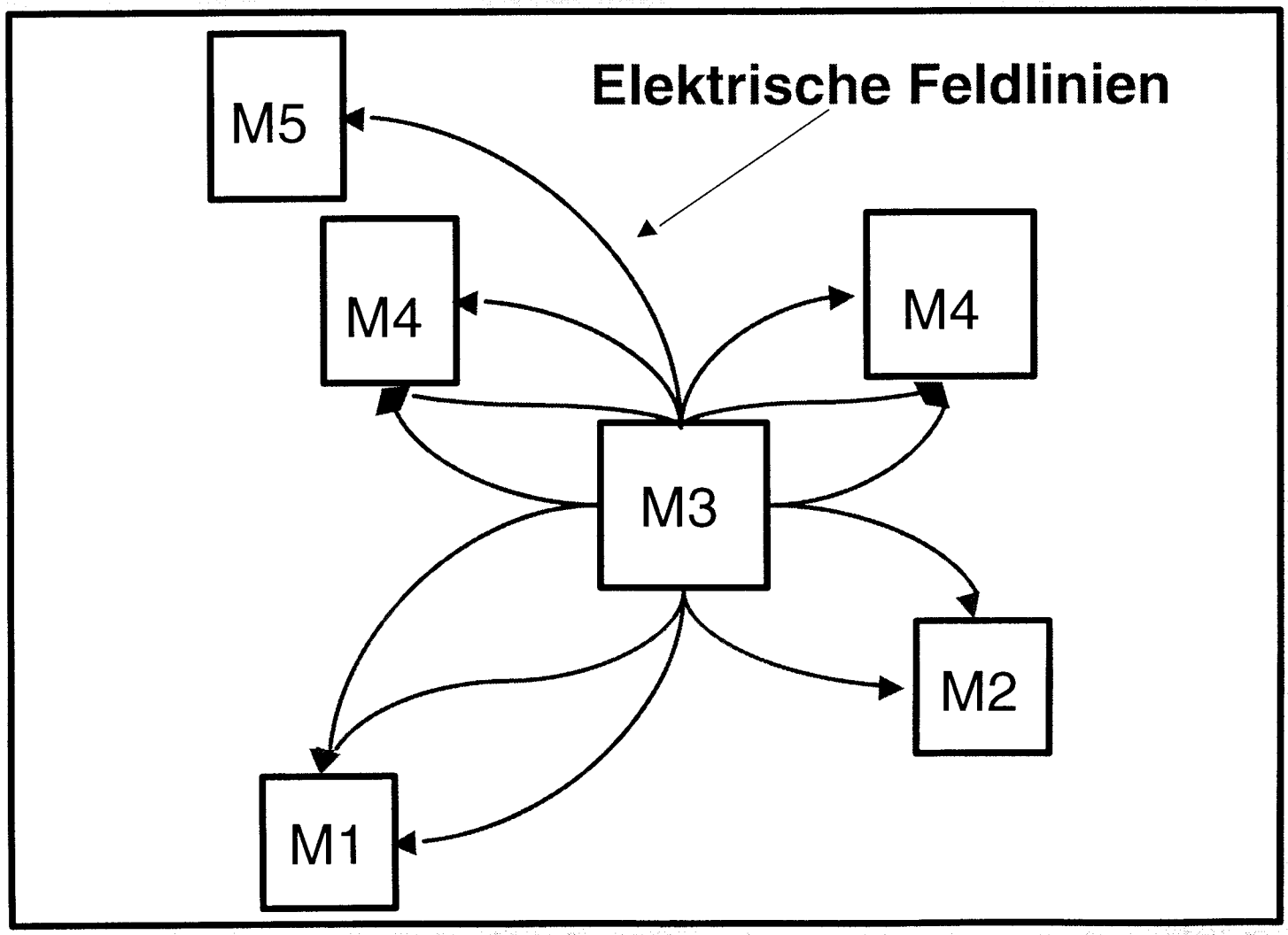


- Kostenkon' olle

Dr. Josef Eckmüller
Infineon Technologies
MPC Workshop
25.01.02, Göppingen

Kleinere Strukturen - Extraktion der Parasiten

- schematischer Querschnitt von Leitungen in einer modernen Technologie mit 5 Metallebenen

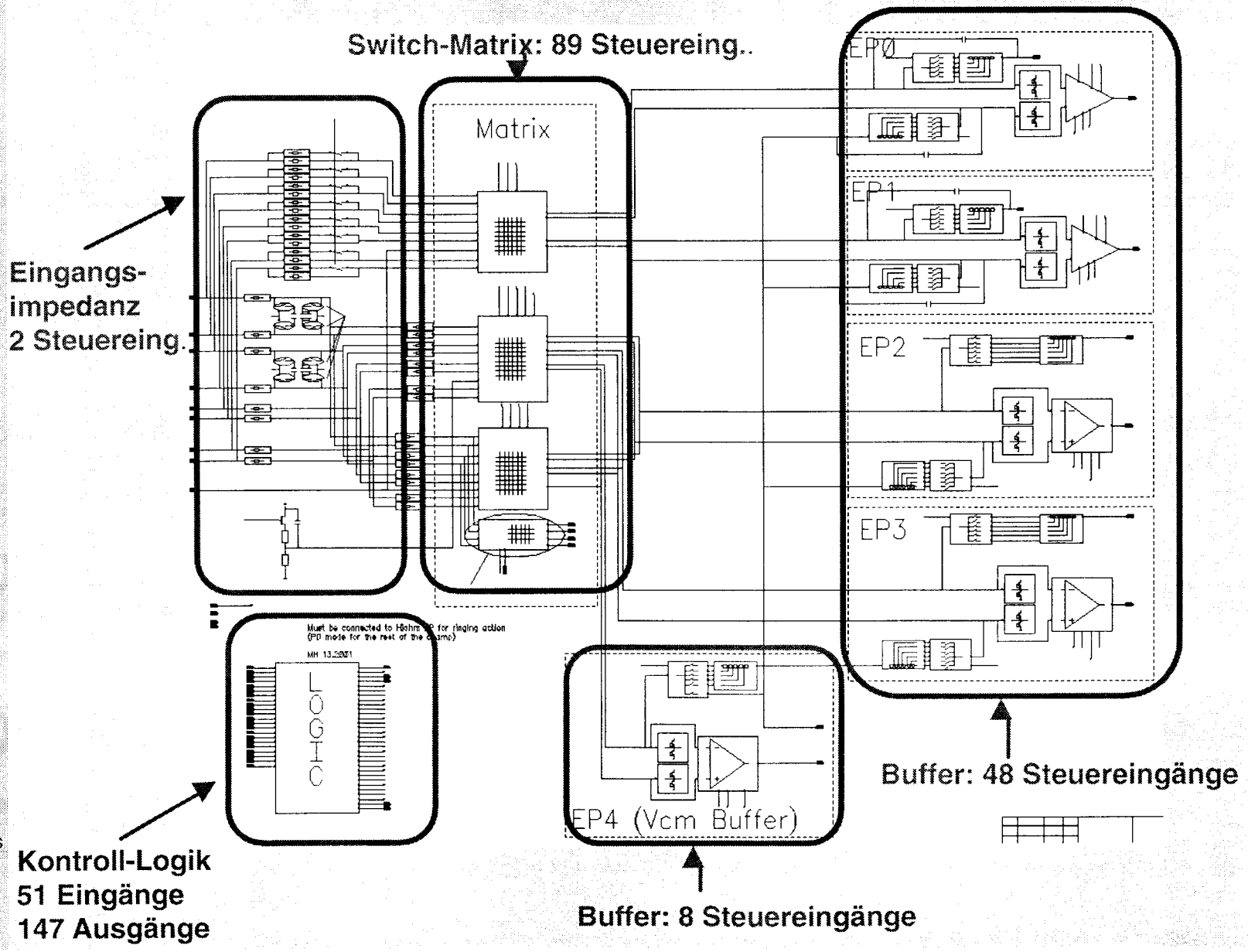


Kleinere Strukturen - Extraktion der Parasiten

- insgesamt nimmt die Bedeutung der parasitären Effekte zu
  - kleinere Parasiten aber höhere Frequenzen
- vereinfachte geometrische Berechnung der parasitären Kapazitäten (Plattenkapazität) nicht mehr möglich
  - Leiterbahnbreite/Leiterbahnhöhe<1.0
  - bisher verwendete Layout-Extraktoren wie z.B. DIVA sind zu ungenau
  - Berücksichtigung der Streufelder für Berechnung der Parasiten (3D Feldberechnung)
- Rechenaufwand heute dafür noch so hoch, dass dies für größere Analogschaltungen nicht praktikabel

  ⇒3D Segmentierung des Layouts und parallele Berechnung der Einzelsegmente

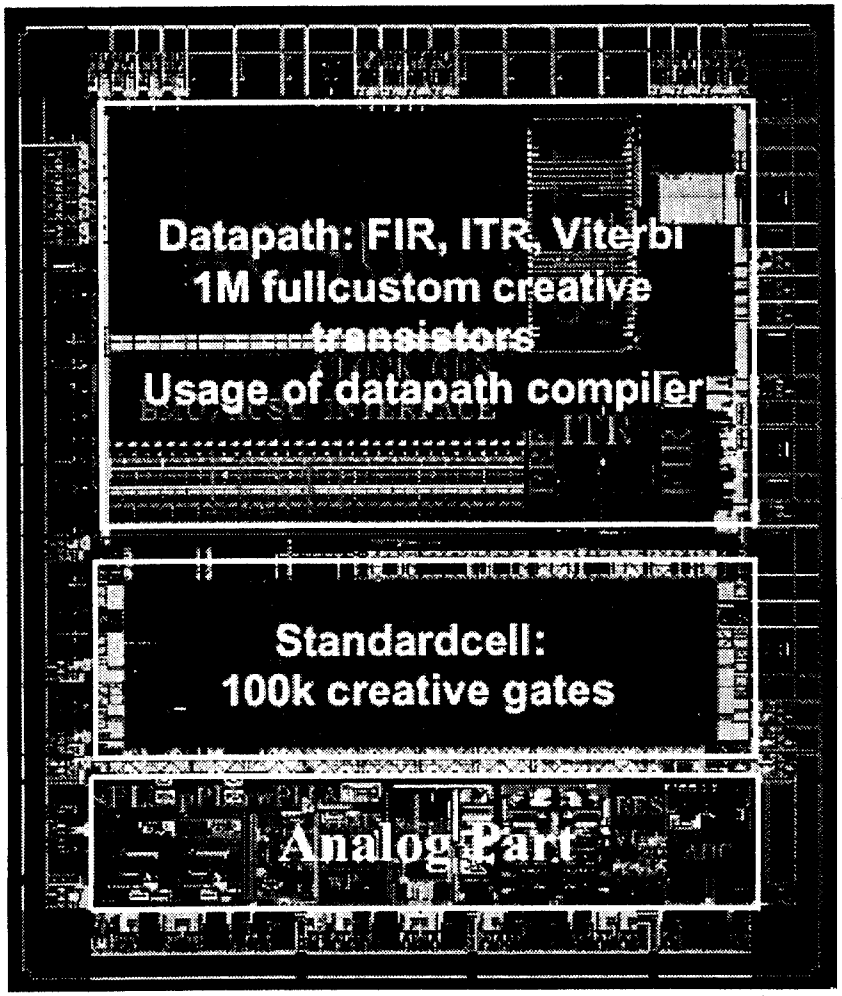
Komplexität - Mixed Signal Verifikation



Komplexität - Mixed Signal Verifikation

- Komplexität muss durch Abstraktion begegnet werden
  - analoge Komponenten durch Modelle (VHDL, VHDL-AMS) nachbilden, die Funktion wiedergeben
  - Schaltungsperformance muss nicht genau nachgebildet werden
- 3 Verifikationsaufgaben
  - Modelle gegenüber zugehöriger Schaltung und Spezifikation verifizieren
  - Zusammenspiel zwischen Digital- und Analogteil des MS Makros verifizieren (bigA-smallD-System)
  - Interface zwischen MS Makro und Digitalteil des SoC (bigD-smallA-System) verifizieren
- bisher existiert für diese Verifikationsaufgaben keine methodische Vorgehensweise wie z.B. die formale Verifikation digitaler Systeme

Höhere Frequenzen - Substrate Noise und Crosstalk



- Layout eines 750Mbit/s Read Write Channel HDD
- analoge (AGC, ADC, CTF, PLL, DAC) und digitale Komponenten zusammen integriert
- Hochintegration in CMOS: Digitalteil, Analogteil, HF-Teil in CMOS mittlerweile integrierbar
- bisherige Vorgehensweise: HF in bipolar, digital und analog zusammen aber bei niedrigen Frequenzen

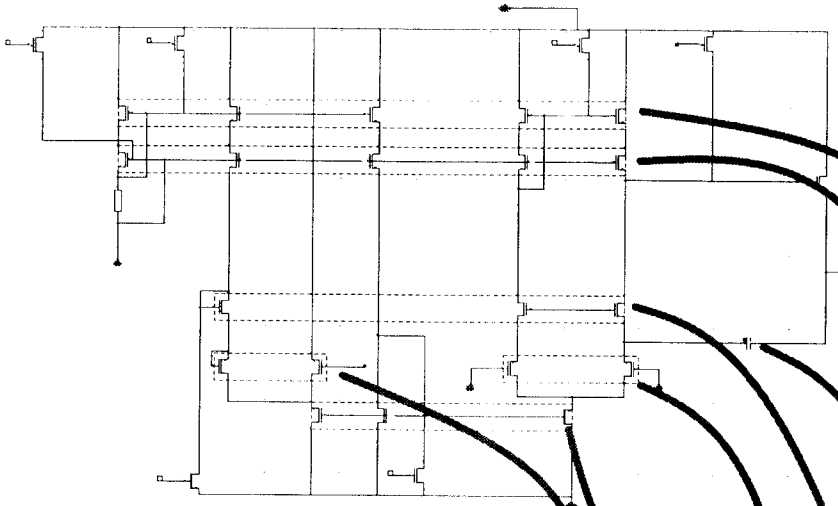
Höhere Frequenzen - Substrate Noise und Crosstalk

- bisher wenig Erfahrung wie stark sich gemeinsam integrierte Digital- und Analogteile bei hohen Betriebsfrequenzen beeinflussen
- aktiver Digitalteil erzeugt durch Stromspitzen Störspannungen im Substrat, die Performance des Analogteils beeinflussen
- Übersprechen zwischen hochfrequent getakteten Leitungen und Analogteil
- Methodische Probleme
  - Analyse des Substrat Noise
    - Modellierung des Digitalteils als Störquelle
    - Modellierung und Parameterextraktion der Substrat-Übertragungsstrecke
  - Optimierung des Floorplannings
  - erste Tools verfügbar z.B. SubstrateStorm von Simplex

kurze Entwicklungszyklen - Automatisierung

- Analoglayout wird meist manuell erstellt
  - langwierig
  - Einhaltung der Design-Regeln nicht garantiert
    - Iterationen zwischen partieller Layout-Erstellung und Verifikation
  - jedes Schematic-Device korrespondiert mit einem Layout-Element
    - Platzierung und Verdrahtung aufwendig

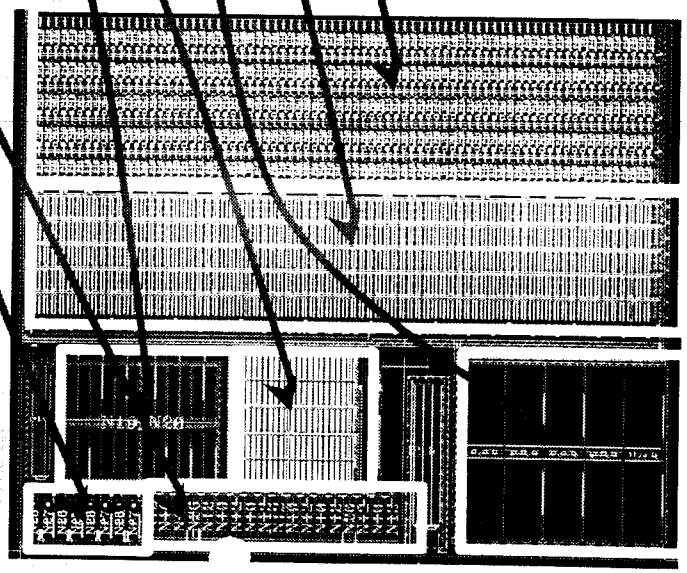
kurze Entwicklungszyklen - Automatisierung



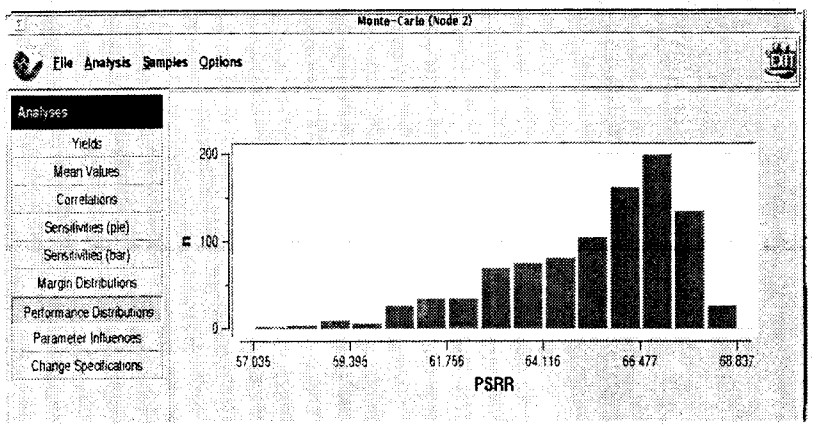
- Entwicklung von Modulgeneratoren
  - Parametrisierbar
  - Zusammenfassung von Schematic-Devices zu einem Layout-Element
  - implizite Berücksichtigung von Matching

- Vorteile
  - Wiederverwendbar
  - wenige Layout-Elemente
    - einfache Platzierung und Verdrahtung
    - gute Basis für automatisches Platzieren und Verdrahten
  - correctness by construction

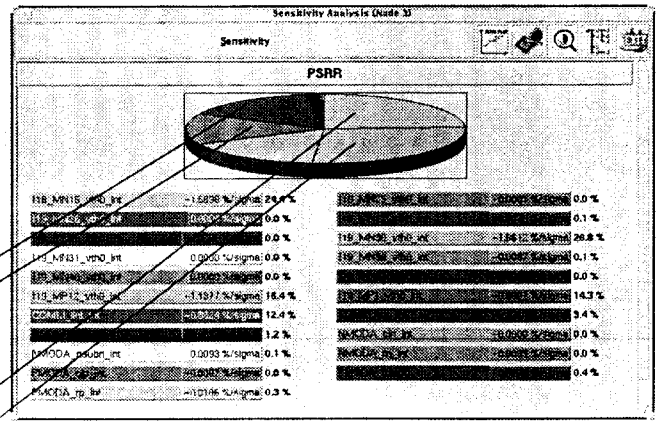
$\Rightarrow$ **deutliche Effizienzsteigerung**



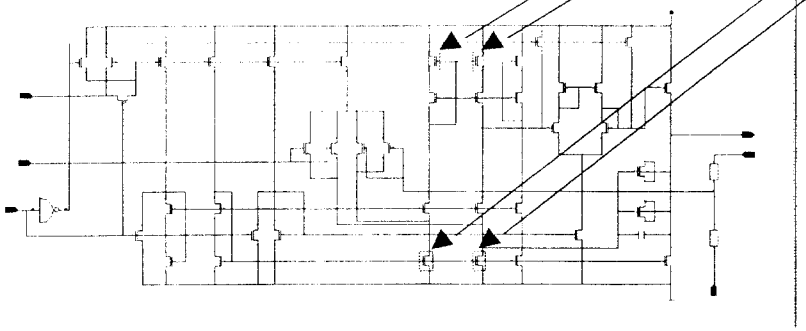
Kostenkontrolle - Ausbeute Analyse und Optimierung



- **Statistische Simulation**
  - z.B. PSRR eines OpAmp
  - Berücksichtigung globaler und lokaler Schwankungen



- **Empfindlichkeitsanalyse**
  - PSRR gegenüber globaler und lokaler (Mismatch) Prozessvariation
  - Identifizierung Mismatch-Sensitiver Elemente



Dr. Josef Eckmüller
Infineon Technologies
MPC Workshop
25.01.02, Göppingen
Seite 15

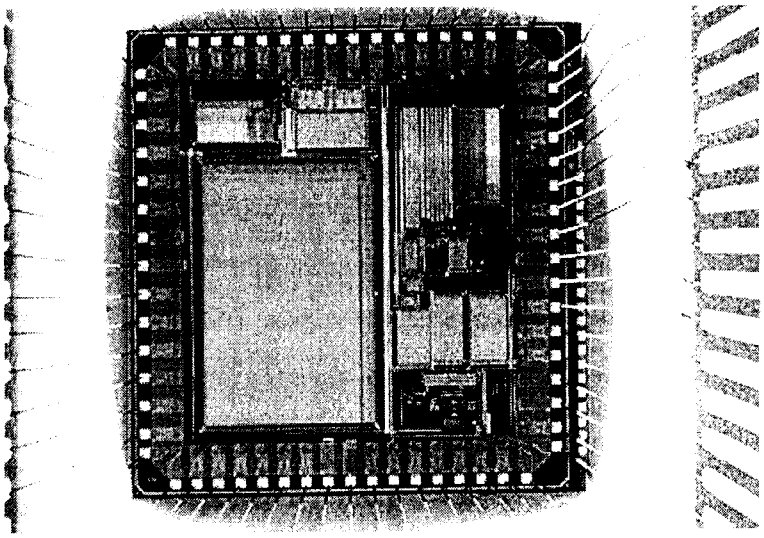
Zusammenfassung

- technologischer Fortschritt in der Halbleiterindustrie ermöglicht SoC-Entwürfe mit
  - höherer Komplexität
  - kleinerer Strukturgröße
  - höheren Datenraten
- neben technischen Herausforderungen existiert Forderung nach Steigerung der Effizienz der MS Entwicklung

⇒ neue methodische Herausforderungen für die Mixed-Signal Entwicklung

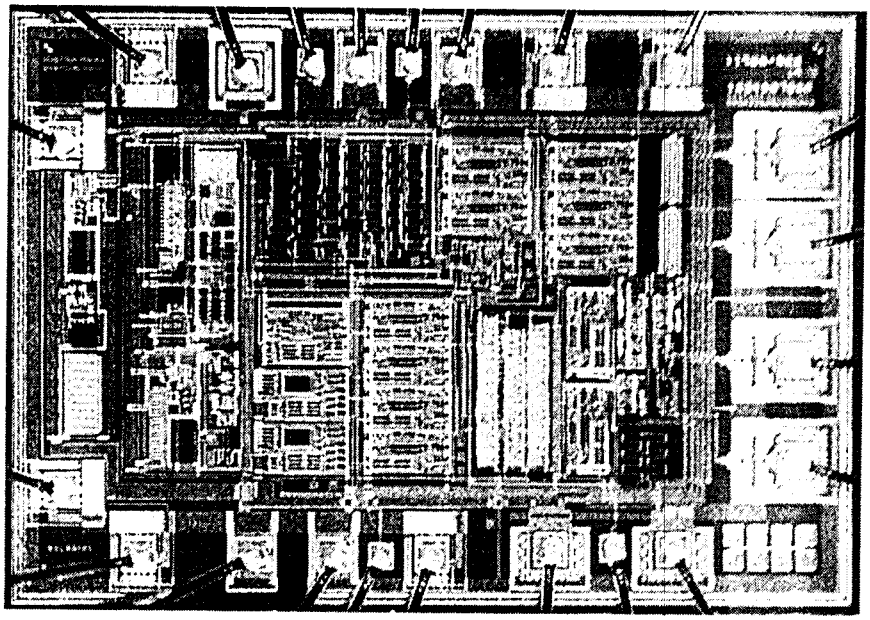
- praktische Beispiele wurden angeführt, um einige Problem zu verdeutlichen

Thermologger_V4b



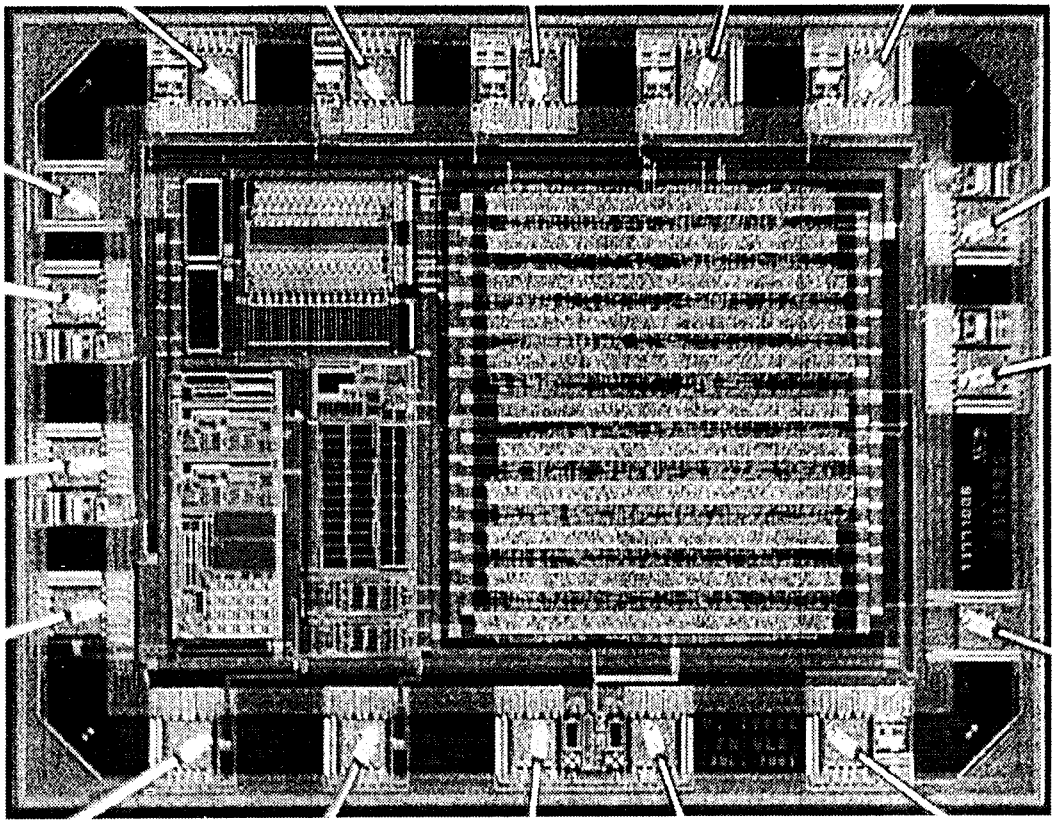
Entwurf:	Fachhochschule Offenburg Bearbeiter: Markus Fischer Carsten Störk Jürgen Hauser Betreuer: Prof. Dr. - Ing. Dirk Jansen
Layouterstellung:	Fachhochschule Offenburg (Mixed-Signal-Entwurf)
Technologie:	Alcatel Mietec 0.5µm CMOS-AD
Chipfertigung:	Europractice, Run 552
Herstelldatum:	Juli 2001
Kostenträger:	MPC-Mittel FH-Verbund Baden_Württemberg
Chipdaten:	Chipgröße: 4,5 x 5,3 mm² Chipgehäuse: JLCC 68 Komplexität: ca 40000 Transistoren
Funktion:	Der Chip enthält einen Temperatursensor, den FHOP-Prozessorkern, ein 8kB RAM, BIOS-ROM, PLL sowie Ein- und Ausgabeeinheiten. Der Chip soll die Temperatur aufzeichnen und in das RAM ablegen. Für die Kommunikation stehen 2 Parallel-Ports, ein Seriell-Port sowie die PSK-Schnittstelle zur Verfügung. Der PLL erzeugt den Takt für die PSk-Schnittstelle. Die Steuerung der Gesammtfunktion erfolgt mit dem FHOP. Das Programm hierfür ist im ROM abgelegt. Es kann aber auch noch zusätzlicher Code in das RAM abgelegt werden, sodass der Chip ein komplettes Controllersystem darstellt.
Testergebnisse:	Der Chip funktionierte nur teilweise. Nachgewiesen sind folgende Funktionen: Prozessorkern, Schnittstellen, PLL, analoge Schnittstellenkomponenten, Resetfunktion. Der Chip antwortet nach einem Reset mit seiner Kennung im definierten 125kHz Format. Wegen vermutlicher Fehler im Bios konnte jedoch keine Kommunikation hergestellt werden. Damit sind viele Funktionalitäten nich überprüfbar. Weitere Tests werden in Zukunft den Fehler näher einzugenzen erlauben

Laderegler für Solarsysteme SOLAR1 R1



Entwurf:	Fachhochschule Ulm Bearbeiter: Thomas Brenner Betreuer: Prof. Dipl.-Phys. Gerhard Forster		
Layouterstellung:	Fachhochschule Ulm (Mixed Signal-Entwurf) Analog-Teil: Standardzellen + Full Custom Design Digital-Teil: Standardzellen		
Technologie:	CXZ 0,8 µm CMOS A/D High Voltage, Fa. AMS		
Chipfertigung:	Fa. AMS, Österreich, über Europractice		
Herstelldatum:	III. Quartal 2001		
Kostenträger:	MPC-Gruppe Baden-Württemberg		
Chipdaten:	Chipfläche:	2,5 x 1,7 mm^2	
	Gehäuse:	LCC 44	
	Funktionsblöcke:	Analogteil:	Spannungsregler, 11 Komparatoren, 3 OPV, PWM
		Digitalteil:	ca. 150 Gatter, 4 Treiber
Funktion:	Die Energieausbeute in einem Solarsystem, bestehend aus Solarpanel, Blei-Akku und Verbraucher, wird optimiert und der Akku (12 V oder 24 V Nennspannung) in einem hinsichtlich der Lebensdauer günstigen Betriebszustand gehalten. Es handelt sich um ein Redesign bei gleichzeitiger Funktionserweiterung eines bestehenden ICs (siehe MPC-Workshop Juli 2001).		

Modem für den Schmalband-Amateurfunk



Entwurf:	Fachhochschule Ulm Bearbeiter: Volker Typke Betreuer: Prof. Arnold Führer
Layouterstellung:	Fachhochschule Ulm Mixed-Signal-Technologie
Technologie:	0,8 µm CMOS CYE-Prozeß
Chipfertigung:	Fa. Austria Mikro Systeme (AMS), über Europractice
Herstelldatum:	4. Quartal 2001
Kostenträger:	MPC-Gruppe Baden-Württemberg
Chipdaten:	Chipfläche: L 2310 µm B 1760 µm = 4,06 mm^2
Funktion:	ASIC zur drahtlosen digitalen Datenübertragung in der Betriebsart Packet Radio eines Amateurfunkgeräts. Der Baustein arbeitet nach dem Minimum Shift Keying Verfahren und kann direkt (es sind nur wenige diskrete Bauelemente erforderlich), an den Mikrofoneingang bzw. den Lautsprecherausgang der Amateurfunkstation angeschlossen werden. Sie finden eine genauere Beschreibung auf Seite 7.