MPC

MULTI PROJEKT CHIP GRUPPE
BADEN-WÜRTTEMBERG

Herausgeber: Hochschule Ulm **Ausgabe:** 40 **ISSN** 1862-7102 **Workshop:** Konstanz Juli 2008

03 Systemdesign mit SystemC
F. Kesel, HS Pforzheim

13 Ein Signalverarbeitungssystem zur Verbesserung der Kommunikation in Fahrzeugen
J. Pittermann, S. Stenzel, J. Freudenberger, HS Konstanz

19 Designstudie für ein Laser-Radar-IC
G. Vallant, G. Forster, HS Ulm

29 Von funktionalen Programmen zur Hardwarebeschreibung digitaler Filter
T. Häberlein, M. Brettschneider, HS Albstadt-Sigmaringen

39 Development of an Electro Physiological Front End with SPI-Bus Connection
D. Bau, HS Offenburg

47 Organic/Paper Substrate RFID
M. Bartel, HS Aalen

57 Sicherer Entwurf von Hochfrequenzschaltungen durch Verwendung eines echten statistischen Designkits
H.-J. Wassener, R&D Atmel RFA

71 VLSI-basierte Hardwarearchitekturen für Hodgkin-Huxley-Neuronenmodelle
M. Beuler, W. Bonath, FH Gießen-Friedberg

Cooperating Organisation
Solid-State Circuit Society Chapter
IEEE German Section

Tagungsband zum Workshop der Multiprojekt-Chip-Gruppe Baden-Württemberg, Konstanz, 4. Juli 2008

Inhaltsverzeichnis

Systemdesign mit SystemC 3
F. Kesel, HS Pforzheim

Ein Signalverarbeitungssystem zur Verbesserung der Kommunikation in Fahrzeugen 13
J. Pittermann, S. Stenzel, J. Freudenberger, HS Konstanz

Designstudie für ein Laser-Radar-IC 19
G. Vallant, G. Forster, HS Ulm

Von funktionalen Programmen zur Hardwarebeschreibung digitaler Filter 29
T. Häberlein, M. Brettschneider, HS Albstadt-Sigmaringen

Development of an Electro Physiological Front End with SPI-Bus Connection 39
D. Bau, HS Offenburg

Organic/Paper Substrate RFID 47
M. Bartel, HS Aalen

Sicherer Entwurf von Hochfrequenzschaltungen durch Verwendung eines echten statistischen Designkits 57
H.-J. Wassener, R&D Atmel RFA

VLSI-basierte Hardwarearchitekturen für Hodgkin-Huxley-Neuronenmodelle 71
M. Beuler, W. Bonath, FH Gießen-Friedberg

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

Systemdesign mit SystemC

Frank Kesel

Hochschule Pforzheim

frank.kesel@hs-pforzheim.de

Der Entwurf eines „System-on-Chip“ ist heute geprägt durch komplexe Hardware- und Softwareentwicklungen und die Systemintegration. Die Modellierung und der Entwurf der Hardware auf RT-Ebene weist heute kein ausreichendes Abstraktionsniveau für die Systementwicklung mehr auf. Heute werden Systeme auf höheren Abstraktionsniveaus, wie beispielsweise der Transaktionsebene, entwickelt, um z.B. Entwurfsalternativen zu überprüfen oder Hardware und Software gleichzeitig entwickeln zu können.

SystemC als Entwurfssprache auf Systemebene wird hierzu in zunehmend stärkerem Maße benutzt. Der Beitrag stellt die wesentlichen Konzepte von SystemC dar und zeigt die Unterschiede in den Modellierungskonzepten zwischen der RT-Ebene und der Transaktionsebene anhand von SystemC. Ferner wird diskutiert, wie sich die Leistungsfähigkeit der Simulation des Systems durch das höhere Abstraktionsniveau verbessern kann.

1. Modellierung von Systemen

1.1. Problematik des SOC-Entwurfs

Integrierte elektronische Systeme, im Englischen auch als „System-on-Chip“ (SOC) bezeichnet, bestehen aus digitaler und analoger elektronischer Hardware und auf der Hardware ablaufende Software. Bei der Hardware handelt es sich sehr häufig um komplexe Rechnersysteme, bestehend aus einem oder mehreren Mikroprozessoren, Spezialprozessoren, integrierten Speichersystemen, Bussystemen und daran angeschlossenen Peripherieschaltungen. Die Implementierung findet entweder durch ASICs oder immer häufiger auch durch programmierbare FPGAs oder Kombinationen aus beiden statt. Während alleine der Entwurf der Hardware heute schon durch enorme Komplexitätsprobleme geprägt ist, so kommt noch die Entwicklung von nicht minder komplexer Software hinzu sowie die Integration von Software und Hardware zu einem funktionsfähigen System.

Der korrekte Entwurf solcher Systeme innerhalb kurzer Zeit stellt daher eine gewaltige Herausforderung dar. In den vergangenen vierzig Jahren ist man im Hardwareentwurf der ständig steigenden Komplexität dadurch begegnet, dass man den Entwurf der Schaltungen auf immer abstrakteren Modellierungsebenen durchgeführt hat und diesen seit etwa fünfzehn Jahren mit Hilfe von Hardwarebeschreibungssprachen, wie VHDL, auf Register-Transfer-Ebene (RTL: Register-Transfer-Level) durchführt [1]. Die Erzeugung eines ASIC-Layouts oder der Programmierinformationen für ein FPGA überlässt man dann weitgehend entsprechenden EDA-Werkzeugen (EDA: Electronic Design Automation). Vergleichbare Entwicklungen gab es auch im SW-Entwurf, z.B. durch entsprechende Programmiersprachen und den Einsatz von Compilern.

Die Hardware wird also auf einer bestimmten Abstraktionsebene modelliert und durch manuelle oder automatische Verfeinerungen, beispielsweise durch Logiksynthese, in „tiefere“ Abstraktionsebenen überführt, bis hin zu einer physikalischen Realisierung. Je abstrakter das Modell ist, desto ungenauer ist die Modellierung, beispielsweise hinsichtlich der Mikroarchitektur oder des zeitlichen Verhaltens. So werden z.B. die Gatterlaufzeiten in einem RTL-Modell nicht berücksichtigt. Andererseits gewinnt man durch steigende Abstraktion an Übersicht und kann die Funktionalität schneller beschreiben. Ebenfalls wichtig ist die Tatsache, dass ein Modell auf dem Entwicklungsrechner mit steigendem Abstraktionsgrad in der Regel auch schneller ausgeführt werden kann.

Die Komplexität heutiger HW-SW-Systeme führt dazu, dass der HW-Entwurf auf RT-Ebene keinen ausreichenden Abstraktionsgrad mehr aufweist. Erforderlich ist beispielsweise die Untersuchung von Entwurfsalternativen im Hinblick auf die Aufteilung der Funktionalität auf Hardware und Software („HW/SW-Partitioning“, [2]). Ferner möchte man möglichst frühzeitig ein Modell der Hardware haben, welches eine gleichzeitige Entwicklung der Software ermöglicht („HW/SW-Co-Design“). Hierfür wird in der Regel die von RTL-Modellen gebotene Genauigkeit nicht benötigt - sie behindert in diesen Aufgaben eher durch langsame Simulationszeiten und aufwändige Erstellung der RTL-Modelle. Der anstehende und von einigen Firmen schon vollzogene Übergang zu einem neuen Modellierungsparadigma wird durch den Begriff „Electronic System Level“ (ESL, [2]) geprägt, damit ist die Modellierung des Systems bestehend aus Hardware und Software gemeint.

1.2. Entwurf auf Systemebene

Der Entwurf eines Systems beginnt üblicherweise mit einer Spezifikation. Sehr häufig versucht man für das System oder Teile davon ausführbare Modelle zu erstellen, die nur die reine Funktionalität darstellen und noch keine Entwurfsentscheidungen wie z.B. HW/SW-Partitionierung oder HW-Architektur beinhalten, dies wird häufig als „ausführbare Spezifikation" bezeichnet. Hierfür werden Werkzeuge wie beispielsweise Matlab oder Programmiersprachen wie C++ verwendet. Wie schon erwähnt ist ein anschließender direkter Übergang zu einem RTL-Modell der Hardware nicht sinnvoll, da man beispielsweise hinsichtlich der Architektur der Hardware oder der Partitionierung noch Alternativen untersuchen möchte oder schon mit der Entwicklung der Software beginnen möchte.

Die Lücke zwischen der ausführbaren Spezifikation und der RT-Ebene kann durch Modelle auf der so genannten Transaktionsebene (Transaction Level Modeling, TLM, [2], [6]) geschlossen werden. Der Begriff der Transaktions-„ebene" ist dabei missverständlich, da es sich im Sinne der Modellierungsebenen und der damit verbundenen Abstraktionsgrade um mehrere Ebenen handelt. TLM definiert eher eine Modellierungstechnik, die am besten durch den ursprünglichen, englischen Begriff des „transaction based", also der Modellierung der Transaktionen eines Systems, beschreiben lässt.

1.3. TL-Modellierung

Die wesentliche Idee von TLM, siehe z.B. [3], besteht darin, die Funktionalität des Systems aufzuteilen in Rechner-Komponenten („computation components") und in Kommunikationseinrichtungen („communication"), wie z.B. Bussysteme, welche die Komponenten verbinden. Der Fokus von TLM liegt dabei in der Modellierung der Kommunikationseinrichtungen auf einer höheren Abstraktionsebene, da die Kommunikation in modernen Systemen einen Großteil der Funktionalität bestimmt. Man abstrahiert beispielsweise Bussysteme dadurch, dass keine einzelnen Bussignale mehr beschrieben werden und auch das zeitliche Verhalten nicht zyklengenau dem konkreten Busprotokoll entsprechen muss.

Die Kommunikation wird durch so genannte „Kanäle" („channels", [3]) modelliert. „Transaktionen" werden dann durch den Aufruf von „Interface"-Funktionen der Kanäle implementiert. Damit können im TL-Modell unnötige Details der Kommunikationseinrichtungen, wie z.B. Adress-, Daten- und Steuerbussignale, zunächst vernachlässigt und erst zu einem späteren Zeitpunkt durch ein RTL-Modell genauer modelliert werden. Dies erhöht insbesondere die Simulationsgeschwindigkeit gegenüber einem detailgenaueren RTL-Modell, so dass beispielsweise Entwurfsalternativen schnell untersucht werden können oder mit der Entwicklung der Software begonnen werden kann.

Wie schon erwähnt, handelt es sich bei TLM nicht um eine einzige Ebene, wie z.B. die Register-Transfer-Ebene. Ferner ist TLM derzeit auch noch nicht so exakt definiert wie die RT-Ebene. Wenn man daran gehen möchte, die Aufteilung des Systems in SW und HW mit Hilfe von TL-Modellen zu modellieren, müssen, neben den Programm- und Datenspeichern, in der Regel die vom Programmierer erreichbaren Register des Rechnersystems modelliert werden, so dass mit der Entwicklung der Software begonnen werden kann. Ein solches Modell der Hardware wird als „Programmer's View" (PV) bezeichnet. Für eine Prozessor-„Komponente", die im System integriert ist, könnte man dies z.B. durch einen Instruktionssatzsimulator (ISS) implementieren.

Eine Unterscheidung der TLM-Abstraktionsebenen kann bezüglich der zeitlichen Modellierungsgenauigkeit vorgenommen werden [3]. Hier unterscheidet man zwischen Modellen, die die Zeit gar nicht modellieren („untimed", PV), Modellen, die mit geschätzten Ausführungszeiten versehen sind („approximate timed", PV+timing), und Modellen, die taktzyklengenau sind, dies entspricht dann der zeitlichen Genauigkeit von RTL-Modellen.

Einige Konzepte von TLM lassen sich am besten mit objektorientierter Programmierung (OOP) realisieren, so zum Beispiel die „Interfaces" oder Schnittstellen, die eine direkte Entsprechung in OOP-Sprachen, wie z.B. C++, haben und mit Hardwarebeschreibungssprachen (HDL: Hardware Description Language) wie VHDL oder Verilog nicht realisierbar sind. Andererseits fehlt in C++ beispielsweise die für TL-Modelle notwendige Möglichkeit, Nebenläufigkeiten zu modellieren, wie es wiederum mit HDLs möglich ist. Viele wesentliche Ideen der TLM wurden z.B. in „SpecC" durch Gajski realisiert [7]. Eine andere Entwicklung im Hinblick auf TLM-Sprachen stellt „SystemC" dar [4], [5], [8]. Während SystemC zunächst hauptsächlich für die RTL-Modellierung eingesetzt werden konnte, intergrierte man dann in der weiteren Entwicklung von SystemC einige Konzepte von TLM-Sprachen wie SpecC, so dass es heute möglich ist, mit SystemC mehrere Abstraktionsebenen von der Spezifikation bis zur RT-Ebene zu modellieren. SystemC entwickelt sich daher zur Standardsprache für das ESL-Design [2].

Wir wollen in den folgenden Abschnitten daher zunächst SystemC anhand der Modellierung auf RT-Ebene vorstellen und dann zeigen, welche Konzepte von SystemC benutzt werden, um Systeme auf Transaktionsebenen zu modellieren. Im Rahmen des vorliegenden Beitrags können wir natürlich nicht alle Details von SystemC darstellen. Wir beschränken uns daher auf die Erläuterung einiger wesentlicher Kon-

zepte und verweisen für eine ausführliche Darstellung von SystemC z.B. auf [4], [5] und [8].

2. Einführung in SystemC

2.1. Historische Entwicklung

SystemC entstand durch Beiträge aus dem universitären Umfeld, von EDA-Firmen und Unternehmen der Halbleiterindustrie. Einige dieser Organisationen schlossen sich 1999 zur OSCI (Open SystemC Initiative) zusammen, einer unabhängigen, nichtkommerziellen Vereinigung [8]. Die erste im Jahr 2000 veröffentlichte „Release 1.0“ implementierte wesentliche Konzepte von HDLs auf RT-Ebene, so dass SystemC zunächst nur als weitere HDL-RTL-Sprache empfunden wurde.

SystemC ist keine eigenständige Programmiersprache, sondern eine Erweiterung von C++. Die Erweiterung wird durch Klassenbibliotheken implementiert, die von der OSCI bezogen werden können. In diesen Klassenbibliotheken ist auch ein ereignisgesteuerter Simulator implementiert. Die Entwicklung eines SystemC-Modells besteht darin, den C++-Quellcode unter Verwendung der SystemC-Klassen zu schreiben und dann mit einem C++-Compiler, wie Visual C++ oder gcc, zu übersetzen. Der durch den Compiler erzeugte und auf dem PC oder einer Workstation ausführbare Code beinhaltet dann den Simulator. Ebenso ist es möglich, den SystemC/C++-Code z.B. für Simulationsprogramme wie Modelsim zu kompilieren und zu simulieren.

Die schon erwähnte Realisierung von TLM-Konzepten, wie Kanälen und Interfaces, führte im Jahr 2002 zur „Release 2.0“ und damit zu einer echten Sprache für die Beschreibung von Systemen. Mit einigen weiteren Verbesserungen in der „Release 2.1“ aus dem Jahre 2005 konnte SystemC auch beim IEEE standardisiert werden (IEEE-Standard 1666). SystemC hat sich mittlerweile in vielen Firmen, die sich mit dem Entwurf von elektronischen Systemen beschäftigen, etabliert und wird auch in zunehmend stärkerem Maß von EDA-Werkzeugen unterstützt.

Die Arbeit der OSCI ist in verschiedene „Working Groups“ aufgeteilt, die sich derzeit mit folgenden Weiterentwicklungen beschäftigen:

- Analog/Mixed Signal (AMS) Modellierung
- Definition einer synthesefähigen Untermenge von SystemC
- Transaction-Level Modeling (TLM)
- SystemC Verification Library (SCV)

2.2. Wesentliche Konzepte

Wie schon erwähnt, entstand SystemC als C++-„Nachbau“ von HDLs, wie z.B. VHDL, und weist daher einige Gemeinsamkeiten mit HDLs auf:

- Modellierung von Nebenläufigkeiten durch Prozesse
- Hierarchische Strukturierung durch Module (vgl. VHDL: Entity und Architecture)
- Verbindung der Prozesse durch Signale
- Modellierung der Zeit
- Ereignisgesteuerte Simulation
- Hardwarespezifische Datentypen

```
#include <systemc.h>

class Multiplier : public sc_module
{
public:
   //PORTS
   sc_in<bool> clock,  reset;
   sc_in<int>  a, b;
   sc_out<int> y;

   //PROCESSES
   void mult_proc(void);
   void reg_proc(void);
   //CONSTRUCTOR
   Multiplier(sc_module_name name);
   SC_HAS_PROCESS(Multiplier);

private:
   //Internal Signals
   sc_signal<int> y_sig;
};
```

Listing 1: Klassendeklaration „Multiplier“

Die Unterschiede zwischen SystemC und traditionellen HDLs sind darin zu sehen, dass SystemC objektorientiert ist und durch das Konzept der Kanäle und Interfaces für die Systemmodellierung besser geeignet ist.

Listing 1 zeigt ein erstes Beispiel eines SystemC-Modells auf RT-Ebene. Die hierarchische Strukturierung eines SystemC-Designs erfolgt über „Module“, wobei es sich dabei um C++-Klassen handelt, die von „sc_module“ abgeleitet werden. Ein Modul wird dann auf der nächsten hierarchischen Ebene als Objekt instanziert. In Listing 1 ist die Definition der Klasse gezeigt (Header-Datei), die Implementierung der Funktionen ist in Listing 2 zu sehen (cpp-Datei).

Ein Modul kann folgendes beinhalten:

- Ports, um das Modul auf der nächsten Ebene mit anderen Modulen zu verbinden („Schnittstelle“).

- Instanzierte Module der nächst tieferen Ebene und damit die Beschreibung der Struktur oder des hierarchischen Aufbaus.
- Signale, die Prozesse oder Module verbinden.
- Prozesse, die das (nebenläufige) Verhalten eines Moduls beschreiben.
- Member-Variablen und Methoden, die typischerweise als Hilfsfunktionen im Modul verwendet werden.
- Konstruktor des Moduls.

```
#include "Multiplier.h"
// constructor
Multiplier::Multiplier
(sc_module_name name) : sc_module(name)
{
   SC_METHOD(mult_proc);
   sensitive << a << b;
   SC_METHOD(reg_proc);
   sensitive_pos << clock;
   sensitive << reset;
}

void Multiplier::reg_proc(void)
{
   if ( reset.read() == true) {
      y.write( 0 );
   } else {
      y.write( y_sig.read() );
   }
}

void Multiplier::mult_proc(void)
{
   y_sig.write( a.read() * b.read());
}
```

Listing 2: Implementierung der Funktionen der Klasse „Multiplier“

Ports sind C++-Objekte (z.B. Klassen „sc_in“ und „sc_out“ in Listing 1) und sind Instanzen von Template-Klassen, siehe z.B. [9], so dass der Datentyp als Template-Parameter übergeben werden kann. Auch Signale sind C++-Objekte (z.B. „y_sig“ in Listing 1), deren Datentyp ebenfalls über einen Template-Parameter angegeben werden kann. Signale kennen ferner Methoden, die auf sie angewendet werden können, z.B. die Methoden „read“ und „write“ in Listing 2, um ein Signal lesen oder schreiben zu können.

Die Verbindung zwischen einem Signal und einem Port eines Moduls erfolgt durch den Bindungsoperator „()“. In Listing 3 ist die Main-Routine des Simulationsmodells gezeigt, in welcher, neben einem Taktobjekt, der Multiplizierer sowie eine Testbench instanziert wird. Zur Verbindung der drei Module sind entsprechende Signale als Objekte instanziert. Die Verbindung eines Signals mit den Ports der Modul-Objekte erfolgt über den Bindungsoperator in folgender Weise: `Modul.Port(Signal);`

Nach der Bindung ist der Port ein „Zeiger“ auf das Signal, so dass der Aufruf einer Methode des Ports, wie in Listing 2 z.B. für den Port „y“ gezeigt, tatsächlich die Methode des angeschlossenen Signals aufruft. Damit erreicht man nun den gleichen Vorteil wie in VHDL: Ein Modul kann unabhängig von der Umgebung geschrieben werden, in der es später instanziert wird. Der Datenaustausch erfolgt ausschließlich über die Ports bzw. über die Methoden des Ports und damit die Methoden des später anzuschließenden Kanals. Es muss nur sichergestellt werden, dass Port und Signal vom gleichen Typ sind.

```
#include <systemc.h>
#include "Testbench.h"
#include "Multiplier.h"

int sc_main(int argc, char * argv[])
{
   //Object instantiation
   sc_clock    clock("clock", 10, SC_NS);
   Testbench   tb("tb");
   Multiplier  mult("mult");

   //Signal instantiation
   sc_signal<bool> reset;
   sc_signal<int>  a, b, y;
   //Connect signals
   tb.clock(clock);
   tb.reset(reset);
   tb.a(a);
   tb.b(b);
   tb.y(y);

   mult.clock(clock);
   mult.reset(reset);
   mult.a(a);
   mult.b(b);
   mult.y(y);

   //Start simulation
   sc_start(1, SC_US);

   return 0;
}
```

Listing 3: Main-Routine des Beispiels

Prozesse werden als C-Funktionen beschrieben und durch ein Makro im Konstruktor beim Simulator „registriert“. In Listing 2 sind zwei Prozesse (reg_proc und mult_proc) durch das Makro „SC_METHOD“ als Prozesse beim Simulator registriert. Das besondere an den Prozess-Funktionen ist, dass keine Argumente übergeben werden und auch kein Wert zurückgeliefert wird, da nur der Simulator die Prozesse ausführt. Der Benutzer steuert die Ausführung der Prozesse im Simulator durch die von VHDL bekannten „Sensitivitätslisten“, die in SystemC direkt nach der Registrierung eines Prozesses durch die „sensitive“-Anweisungen angegeben werden. Sobald sich also ein Signal oder ein Port, der ja nur ein „Zeiger“ auf ein angeschlossenes Signal ist, ändert, wird der Prozess vom Simulator ausgeführt.

SystemC kennt zwei unterschiedliche Arten von Prozessen, neben den schon angesprochenen „Methods“ ist der „Thread“ die zweite Prozessvariante. Er wird mit dem Makro „SC_THREAD“ als solcher beim Simulator „registriert“.

„Methods“ haben folgende Eigenschaften:

- Werden komplett ausgeführt und geben danach die Kontrolle an den Simulator zurück.
- Da kein Kontextwechsel notwendig ist, ist die Behandlung durch den Simulator weniger aufwändig.
- „Methods“ können nicht durch den Aufruf von „wait()“-Funktionen suspendiert werden.
- Lokale Variablen sind nicht statisch.
- Das Hauptanwendungsgebiet sind RTL-Modelle in SystemC.

„Threads“ haben im Vergleich dazu folgende Eigenschaften:

- Sie werden nur einmal vom Simulator ausgeführt und zu Beginn der Simulation gestartet.
- Durch „wait()“-Funktionen können „Threads“ suspendiert werden. Dies erfordert einen Kontextwechsel im Simulator und ist daher aufwändiger als die Behandlung von „Methods“.
- Da „Threads“ nur einmal ausgeführt werden, wird typischerweise eine Endlosschleife programmiert, die dann durch „wait()“-Funktionen immer wieder suspendiert wird.
- Lokale Variablen sind statisch.
- Das Hauptanwendungsgebiet sind Testbenches und TL-Modelle in SystemC.

Listing 4 zeigt ein Beispiel für einen „Thread“-Prozess, die Definition der Modul-Klasse in der Header-Datei wurde aus Platzgründen weggelassen. Der Prozess ist sensitiv auf die fallende Flanke des Taktes „clock“. Dies bedeutet, dass der Prozess durch eine „wait()“-Funktion so lange suspendiert wird, bis die nächste fallende Flanke am Takt anliegt und dann die Prozessausführung vom Simulator fortgeführt wird. Nach 20 Schleifendurchläufen ist die Ausführung am Ende des Prozesses angekommen. Nun wird der Prozess vom Simulator für immer, d.h. also für die weitere Ausführung der Simulation, suspendiert.

```
#include "Testbench.h"
// constructor
Testbench::Testbench(sc_module_name name) :
sc_module(name)
{
   //Register processes with simulator
   SC_THREAD ( stimulus_proc );
   sensitive_neg << clock;
}

void Testbench::stimulus_proc (void)
{
   int ai, bi;

   reset.write(false);
   a.write(0);
   b.write(0);
   wait();
   for ( int i = 0; i < 20; i++ ) {
      ai = i;
      bi = i+1;
      a.write(ai);
      b.write(bi);
      wait();
      cout <<"a= "<<ai<<", bi= "<<bi<<",
        y= "<<y.read()<<endl;
   }
}
```

Listing 4: Implementierung der Testbench

Das bisher vorgestellte Beispiel des Multiplizierers mit seiner Testbench ist ein typisches Beispiel, wie man mit SystemC ein RTL-Modell schreiben würde. Aus Listing 2 kann man die RTL-Struktur entnehmen: Der „reg_proc“-Prozess beschreibt ein Register, welches mit der steigenden Flanke des Taktes Daten übernimmt und der „mult_proc“-Prozess beschreibt eine kombinatorische Transferfunktion. Nun stellt sich natürlich die Frage, welchen Mehrwert SystemC gegenüber HDLs wie Verilog und VHDL bietet? Dieser Frage soll im nächsten Abschnitt nachgegangen werden.

3. TLM mit SystemC

Wir wollen im Folgenden die Idee der TL-Modellierung anhand des Beispiels eines einfachen Systembusses diskutieren. Während ein RTL-Modell eines Bussystems die detaillierte Verbindungsstruktur des Busses, also die Signale, sowie sein zyklengenaues Verhalten modellieren muss, wird bei TLM beides abstrahiert. Wir gehen im Folgenden hauptsächlich auf die Abstraktion bezüglich der Signale ein.

Das Bussystem wird ersetzt durch einen so genannten Kanal, welcher zunächst wie ein Modul als C++-Klasse beschrieben wird und daher auch als Objekt instanziert wird. Die Kommunikation mit anderen Modulen findet nun allerdings nicht mehr über Signale statt, sondern über den Aufruf von Methoden des Kanals durch ein Modul, an welchen der Kanal angeschlossen oder gebunden ist.

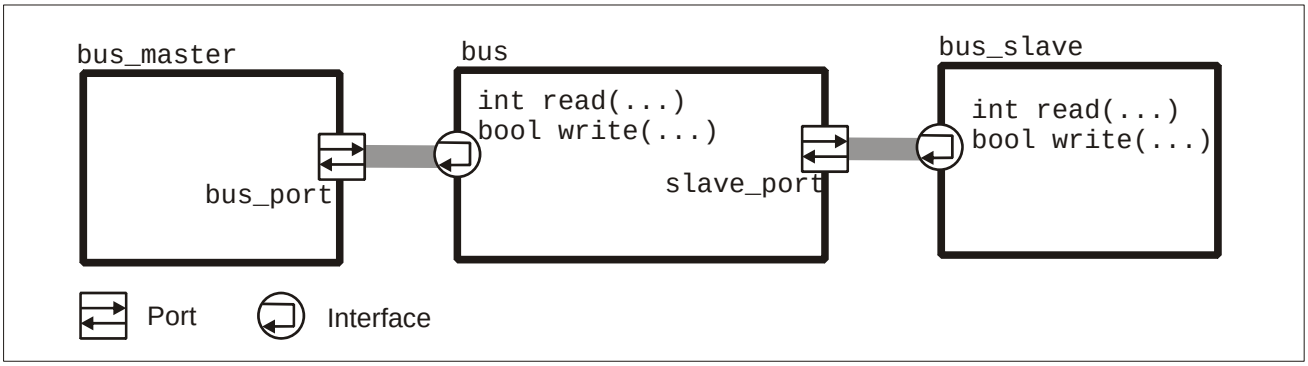


Abbildung 1: Busmodell

Zu diesem Zweck werden im Modul wieder Ports verwendet. Allerdings werden die Ports nun nicht an Signale der nächsten Ebene gebunden, sondern der Kanal wird direkt an den Port gebunden.

Bei den Ports handelt es sich um Objekte der Template-Klasse „sc_port“. Der Template-Parameter ist hier das so genannte „Interface“, also die Schnittstelle, welches eine abstrakte C++-Klasse ist [9]. Das „Interface“ definiert die Methoden, die über den Port dann aufrufbar sind, siehe Listing 5. Da der Kanal später an den Port gebunden ist, werden wiederum effektiv die Methoden des Kanals aufgerufen. Der Kanal wird daher von der Interface-Klasse abgeleitet und muss die Methoden des Interfaces implementieren.

SystemC kennt zwei Arten von Kanälen:

- Primitive Kanäle: Hierzu gehören zum Beispiel die Signale oder die FIFOs. Primitive Kanäle sind vordefiniert und werden vom Simulator durch den „Evaluate-Update-Mechanismus“ behandelt [4]. Dies ist der gleiche Mechanismus wie z.B. in VHDL, der dafür sorgt, dass ein Signal den Wert nicht sofort zugewiesen bekommt, sondern erst nach Abarbeitung des Prozesses. Damit wird sichergestellt, dass sich die Abarbeitungsreihenfolge der Prozesse im Simulator während eines Deltazyklus nicht auf das Ergebnis der Simulation auswirkt [1], [4].
- Hierarchische Kanäle: Diese sind nicht in SystemC eingebaut, wie die primitiven Kanäle, sondern werden selbst geschrieben. Sie stellen gewissermaßen die Verallgemeinerung des Signals dar. Es handelt sich dabei um Module, die von Interface-Klassen abgeleitet werden und daher die Interface-Methoden implementieren. Diese Kanäle können vom Simulator nicht durch den „Evaluate-Update-Mechanismus“ behandelt werden, so dass der Benutzer selbst für die Korrektheit der Simulation sorgen muss. Hierarchische Kanäle sind *das* wesentliche Modellierungselement für TL-Modelle.

Abbildung 1 zeigt ein einfaches Busmodell, mit denen wir die wesentlichen Modellierungstechniken erläutern wollen. Das Interface des Busmodells kann Listing 5 entnommen werden. Im Interface sind die beiden Methoden „read“ und „write“ definiert. Da es sich um eine abstrakte Klasse handelt, sind nur die Funktionsköpfe definiert, die Implementierung der Methoden erfolgt ausschließlich in den von dieser Klasse abgeleiteten Objekten [9].

```
class simple_bus_if: public virtual
sc_interface
{
public:
  // BUS/Slave interface
  virtual int read(unsigned int address) = 0;
  virtual bool write(int data, unsigned int
   address) = 0;

};
```

Listing 5: Interface des Busmodells

Listing 6 zeigt die wesentlichen Details der Header-Datei für den Bus-Master. Für die Definition der Klasse wird hier das Makro „SC_MODULE“, verwendet, welche die Definition eines Moduls aus Listing 1 zusammenfasst. An den Port „bus_port“ soll, wie in Abbildung 1 gezeigt, später ein Kanal mit dem Interface „simple_bus_if“ aus Listing 5 gebunden werden. Daher wird der Template-Parameter des Ports auf dieses Interface gesetzt.

```
#include <systemc.h>
#include "simple_bus_if.h"

SC_MODULE(simple_bus_master)
{
public:
  // ports
  sc_in_clk clock;
  sc_port<simple_bus_if> bus_port;

  …

  //helper functions
  void read_bus(int address, int data);
  void write_bus(int address, int data);
};
```

Listing 6: Ausschnitt aus der Definition des Bus-Master-Moduls

Der Bus-Master besteht im Wesentlichen aus einem Thread-Prozess „main_action“ und zwei Hilfsfunktionen „read_bus“ und „write_bus“, die in Listing 7 und Listing 8 gezeigt sind.

```
#include "simple_bus_master.h"

// constructor
sim-
ple_bus_master::simple_bus_master(sc_module_n
ame name)
    : sc_module(name)
{
   // process declaration
   SC_THREAD(main_action);
   sensitive_pos << clock;
}

void simple_bus_master::main_action()
{
   while(1){
      wait();
      write_bus(100, 20);
      …
      wait();
      read_bus(101, 30);
      …
   }
```

Listing 7: Konstruktor und Prozess des Bus-Masters (Ausschnitt)

Der Bus-Master-Prozess wird als Endlosschleife betrieben und beschreibt die Buszugriffe des Masters durch die Hilfsfunktionen. In den Hilfsfunktionen in Listing 8 findet nun der Zugriff auf den Bus-Kanal über den Port „bus_port“ statt. So wird beispielsweise mit „`bus_port->read(address);`“ die Methode „read“ des an den Port gebundenen Kanals aufgerufen, in diesem Fall also des Bus-Kanals aus Abbildung 1.

```
void simple_bus_master::read_bus(int address,
int data)
{
   int idata;
   idata = bus_port->read(address);
   if(data == idata){
      cout << "At Time " <<
       sc_simulation_time();
      cout << "ns: Master Access OK. Read
       Address: " << address;
      cout << " Data expected: " << data << "
       Data received: " << idata << endl;
   }
   …
}

void simple_bus_master::write_bus(int ad-
dress, int data)
{
   bus_port->write(data, address);
   …

}
```

Listing 8: Hilfsfunktionen des Bus-Masters (Ausschnitt)

Die Definition des Bus-Kanals kann Listing 9 entnommen werden. Diese Klasse wird nun sowohl von „sc_module“ als auch vom Interface „simple_bus_if“ abgeleitet. Damit müssen die Interface-Funktionen „read“ und „write“ durch den Kanal implementiert werden, dies ist in Listing 10 beschrieben. Es handelt sich um die Funktionen die durch die Bindung des Bus-Kanals an den Port des Masters tatsächlich beim Aufruf in Listing 8 ausgeführt werden.

```
#include <systemc.h>
#include "simple_bus_if.h"

class simple_bus: public simple_bus_if, pub-
lic sc_module
{
public:
   // ports
   sc_in_clk clock;
   sc_port<simple_bus_if> slave_port;
   …
   //Implementation of BUS interface
   int read(unsigned int address);
   bool write(int data, unsigned int ad-
dress);
   …

};
```

Listing 9: Ausschnitt aus der Definition des Bus-Moduls

Zu bemerken ist, dass der Bus-Kanal selbst auch einen Port besitzt und daran, wie in Abbildung 1 gezeigt, das Slave-Modul gebunden wird.

```
#include "simple_bus.h"
…
int simple_bus::read(unsigned int address)
{
   …
   return (slave_port->read(address));
}

bool simple_bus::write(int data, unsigned int
address)
{
   slave_port->write(data, address);
   …
   return (1);
}
```

Listing 10: Ausschnitt aus der Implementierung der Interface-Funktionen des Busses

Diese Bindung wird nun im Bus-Kanal verwendet, um in den „read“- und „write“-Funktionen jeweils wieder entsprechende Interface-Funktionen des Slaves über den Port „slave_port“ des Bus-Kanals aufzurufen. Auch der Slave wird vom Interface „simple_bus_if“ abgeleitet und implementiert die Interface-Funktionen, die nun allerdings unterschiedlich zu den Implementierungen des Busses sein können. Zusammenfassend führt also der Aufruf der „read“- und „write“-Funktionen im Master zu einem geschachtelten Funktionsaufruf bis in den Slave hinein, der dann entsprechende Daten liefert oder Daten an der angegebenen Adresse abspeichert. Auf die Darstellung des Quell-

codes des Slaves verzichten wir an dieser Stelle aus Platzgründen.

Am Beispiel des Bus-Kanals wird klar, dass ein hierarchischer Kanal auch Modul sein kann, da er selber wieder Ports besitzt. Wir können also folgendes feststellen:

- Ein SystemC-Objekt ist ein Modul, wenn es Ports besitzt, an welche entsprechende Kanäle gebunden werden können.
- Ein SystemC-Objekt ist ein Kanal, wenn es Interface-Methoden implementiert und damit an den Port eines Moduls gebunden werden kann.
- Ein SystemC-Objekt kann sowohl als Kanal als auch als Modul fungieren.

```
#include "top.h"

//constructor
top::top(sc_module_name name)
    : sc_module(name), C1("C1", 10, SC_NS)
{
    // create instances
   master = new simple_bus_master("master");
   slave1 = new simple_bus_slave("slave1");
   bus = new simple_bus("bus", false);

   // connect instances
   …
   master->bus_port(*bus);
   bus->slave_port(*slave1);
}
```

Listing 11: Implementierung des Konstruktors des Top-Levels

In Listing 11 ist noch gezeigt, wie Master, Bus und Slave in einem Modul (hier: „Top-Level“ des Designs) der nächsten hierarchischen Ebene instanziert und gebunden werden. Die Objekte werden in diesem Beispiel zunächst dynamisch über Zeiger erzeugt oder instanziert. Daher erfolgt die anschließende Bindung über die Zeiger-Notation: An den Port des Masters wird das Bus-Objekt gebunden und an den Port des Busses wird das Slave-Objekt gebunden.

Abgesehen von der Zeiger-Notation handelt es sich hier also um den gleichen Mechanismus, wie wir dies schon bei den RTL-Modellen und der Bindung von Ports und Signalen im vorangegangenen Abschnitt gesehen haben. Der Unterschied liegt nur darin, dass es sich im Fall von RTL-Modellen um primitive Kanäle (= Signale) handelt und Fall von TL-Modellen um hierarchische Kanäle. Im Übrigen werden auch Signale ähnlich wie hierarchische Kanäle modelliert: Bei den in Listing 2 verwendeten „read“- und „write“-Methoden handelt es sich um Interface-Funktionen des Signals.

Der höhere Abstraktionsgrad im Vergleich von TL-Modellen und RTL-Modellen zeigt sich darin, dass sämtliche Bussignale, also Steuer-, Adress- und Datenbus, und die eigentliche Buslogik durch den Bus-Kanal ersetzt werden. Damit können also im Vergleich zu den RTL-Modellen eine Vielzahl von sonst notwendigen Signalen, Modulen und Prozessen durch einen (hierarchischen) Kanal ersetzt werden und dies kann die Ausführung der Simulation erheblich beschleunigen.

Ebenso wird auch das zeitliche Verhalten des Busses sehr stark abstrahiert. Die Buszugriffe werden im Bus-Master über einen „Takt“ gesteuert, um in der Simulation eine zeitliche Abfolge der Buszugriffe zu sehen. Es handelt sich dabei aber nicht um den Bustakt eines konkreten Busprotokolls und man könnte auch ein SystemC-Modell schreiben, welches den zeitlichen Ablauf überhaupt nicht modelliert.

Wie stark die größere strukturelle und zeitliche Abstraktion die Simulations-Performance erhöht, verglichen mit RTL-Modellen, soll im nächsten Abschnitt diskutiert werden.

4. Simulations-Performance

Die „Performance“ (dt.: Leistungsfähigkeit) einer HDL-Hardware-Simulation kann z.B. dadurch bestimmt werden, dass wir eine bestimmte Dauer der Modellzeit simulieren und die dafür benötigte Rechnerzeit messen. Wenn wir allerdings ein synchrones, getaktetes RTL-Modell simulieren ist die benötigte Rechnerzeit abhängig davon, mit welcher Taktfrequenz das Modell getaktet wird. Da es sich um eine diskrete ereignisgesteuerte Simulation handelt, werden nur Zeitpunkte simuliert, in denen sich der Zustand des Modells ändert, bei einem RTL-Modell sind dies hauptsächlich die Taktzeitpunkte.

Häufig wird daher nicht die Modellzeit benutzt, sondern die Anzahl der simulierten Taktzyklen, um einen von der Taktfrequenz unabhängigen Vergleich zu ermöglichen. Die Anzahl der simulierten Taktzyklen lässt sich aus Modellzeit und Taktfrequenz bestimmen. Wir können daher die Performance definieren als:

Performance = Anzahl Taktzyklen / Rechnerzeit

Die Einheit der Performance ist daher „Zyklen/sec“ oder „cycles/s“ und wird auch als „Simulationsfrequenz“ bezeichnet. Wenn wir z.B. ein RTL-Modell mit einer Taktfrequenz von 50 MHz betreiben und eine Modellzeit von 10 ms simulieren, dann sind das 500.000 Taktzyklen. Wenn wir für die Simulation eine Rechnerzeit von 30 sec benötigen, dann ergibt sich eine Performance von 16,7 Kcycles/s.

Für den Vergleich der Performance zwischen RTL-Modellen und TL-Modellen existieren derzeit nur wenige Vergleichsdaten von realen Modellen. Ghenassia formuliert in [6] eine Zielvorstellung für die Leistungs-

fähigkeit von TL-Modellen: Im Vergleich zu einem synthesefähigen RTL-Modell sollte ein zyklengenaues TL-Modell um einen Faktor 100 schneller sein und ein TL-Modell mit geschätztem Timing um den Faktor 1000.

In [10] sind die Ergebnisse für einen SDRAM-Controller der Fa. Prosilog veröffentlicht. Das synthesefähige RTL-Modell des Controllers erreicht eine Performance von 2,9 Kcycles/s und das SystemC-TL-Modell des Controllers, welches annähernd taktzyklengenau ist, eine Performance von 44,5 Kcycles/s. Das TL-Modell ist somit um den Faktor 15 schneller als das RTL-Modell.

Der Autor des vorliegenden Beitrags konnte im Rahmen eines Industrieprojektes ebenfalls einen Vergleich zwischen dem RTL-Modell eines komplexen Timer-Moduls und einem vom Autor entwickelten SystemC-TL-Modell des Timers, welches auch annähernd taktzyklengenau ist, durchführen. Das synthesefähige RTL-Modell kam auf eine Performance von 3,7 Kcycles/s und das TL-Modell auf eine Performance von 59,3 Kcycles/s und damit war das TL-Modell um einen Faktor 16 schneller als das RTL-Modell.

Die Zielvorstellungen von Ghenassia dürfen also vor dem Hintergrund von Messungen an realen Modellen kritisch bewertet werden. Nach Erfahrungen des Autors hängt die Leistungsfähigkeit der TL-Modelle stark davon ab, wie zeitgenau die Modelle sind. Ist also beispielsweise das Zeitverhalten ein wesentlicher Bestandteil der Funktionalität einer Komponente, wie bei einem Timer oder einem SDRAM-Controller, so wird man das TL-Modell annähernd taktzyklengenau auslegen müssen und damit rechnen müssen, dass solche Module nur etwa einen Faktor 10-15 schneller sein werden im Vergleich zu synthesefähigen RTL-Modellen.

Das im vorangegangenen Abschnitt diskutierte Beispiel des (sehr einfachen) Bussystems, was man als TL-Modell mit geschätztem Timing bezeichnen könnte, kann z.B. auf eine Performance von über 1 Mcycles/s kommen, allerdings nur wenn man die Textausgaben auskommentiert. Mit Textausgabe ergibt sich dann nur eine Performance von 88 Kcycles/s. Dieses Beispiel, für das leider kein RTL-Vergleich vorliegt, zeigt, dass neben der zeitlichen Genauigkeit des Modells auch weitere Faktoren die Leistungsfähigkeit der Simulation beeinflussen können.

In einem Modell eines Gesamtsystems werden Komponenten – wie die oben angesprochenen Peripherieeinheiten – enthalten sein, deren Zeitverhalten in einem TL-Modell relativ genau modelliert werden muss und andere Komponenten, wie das Bussystem oder die Prozessoren, deren Zeitverhalten zunächst nicht sehr genau modelliert werden muss, so dass man für diese System-Komponenten eher mit höheren Performance-Verbesserungen im Vergleich zu ihren RTL-Modellen rechnen kann. Nach Ansicht des Autors dürfte daher die an den beiden Beispielen ermittelte Performance-Verbesserung um den Faktor 15 eher die untere Grenze für das Modell eines Gesamtsystems sein. Ob ein Faktor 1000 für die Performance-Verbesserung realistisch ist, muss an konkreten Systembeispielen ermittelt werden.

5. Zusammenfassung

SystemC ist eine Erweiterung von C++ zur Modellierung von Hardware. Wesentliche Konzepte von HDLs, wie Nebenläufigkeit, Reaktivität oder hierarchische Modularisierung, sind in SystemC implementiert. Neben der RT-Ebene können mit SystemC auch höhere Abstraktionsebenen, wie die TL-Ebenen oder die Spezifikationsebene modelliert werden. SystemC bietet hierzu Mechanismen wie die Objektorientierung sowie Interfaces und Kanäle an, die in bislang benutzten HDLs wie VHDL oder Verilog nicht vorhanden sind. Ferner ist SystemC eine Sprache mit der sich sowohl Hardware als auch Software modellieren lässt und damit gut geeignet für die Systemmodellierung.

Weitere Vorteile von SystemC sind die Standardisierung durch den IEEE und die Unterstützung durch die Industrie und akademische Institutionen. Darüber hinaus kann ein SystemC-Modell mit jedem C++-Compiler übersetzt werden und auch HDL-Simulatoren, wie z.B. Modelsim von MentorGraphics, bieten die Möglichkeit SystemC-Modelle zu kompilieren und zu simulieren.

Zu den Nachteilen von SystemC wäre zu bemerken, dass wir uns derzeit in einer relativ frühen Phase der Entwicklung befinden, andere HDLs sind hier natürlich deutlich „erwachsener“. Speziell beim Thema TLM ist vieles noch nicht endgültig definiert und bis zur Einführung eines endgültigen TLM-Standards durch die OSCI-Arbeitsgruppe wird noch einige Zeit vergehen. In diesem Zusammenhang muss auch das Thema Synthese erwähnt werden, speziell die Synthese von TL-Modellen. Es gibt für die Synthese von SystemC-Modellen schon kommerzielle Werkzeuge, z.B. von Forte [11], aber der Markt ist noch nicht sehr groß. Einen weiteren Impuls dürfte auch hier die Einführung eines Synthese-Standards für SystemC durch die OSCI-Arbeitsgruppe ergeben.

Wenn es möglich wird, mit guten Ergebnissen aus SystemC-TL-Modellen entsprechende RTL-Modelle zu synthetisieren, dürfte dies der Verbreitung von SystemC einen erheblichen Schub geben. Derzeit besteht das Problem, dass man das RTL-Modell durch manuelle Verfeinerung des TL-Modells schreiben muss, so dass in einigen Firmen sich die Entwicklung eines TL-Modells in SystemC noch nicht aufdrängt. Firmen die allerdings die Entwicklung sehr großer

Systeme mit größeren Softwareanteilen betreiben sind heute schon gezwungen, schnelle SystemC-Modelle der Hardware zu entwickeln, um eine Co-Entwicklung der Software auf einem schnellen Modell der Hardware betreiben zu können. In diesem Sinne dürfte es auch für die Anbieter von IP-Blöcken (IP: Intellectual Property) sinnvoll sein, den Kunden zusätzlich zur Netzliste oder dem synthesefähigen Code ein schnelles Simulationsmodell des IP-Blocks in SystemC anzubieten.

6. Literatur

[1] Frank Kesel, Ruben Bartholomä: “Entwurf von digitalen Schaltungen und Systemen mit HDLs und FPGAs”, Oldenbourg, 2006

[2] B. Bailey, G. Martin, A. Piziali: “ESL Design and Verification”, Morgan Kaufman, 2007

[3] L. Cai, D. .Gajski: “Transaction Level Modeling: An Overview”, Int’l Conference on Hardware/Software Codesign and System Synthesis, 2003

[4] Thorsten Grötker, Stan Liao, Grant Martin und Stuart Swan: „System Design with SystemC”, Kluwer Academic Publishers, 2002

[5] David C. Black, Jack Donovan: “SystemC: From the Ground Up”, Kluwer Academic Publishers, 2004

[6] Frank Ghenassia: “Transaction-Level Modeling with SystemC. TLM Concepts and Applications for Embedded Systems”, Springer, 2006

[7] D. Gajski, J. Zhu, R. Dömer, A. Gerstlauer, S. Zhao: „SpecC: Specification Language Methodology“, Kluwer Academic Publishers, 2000

[8] www.systemc.org

[9] H. Schildt: „Teach Yourself C++“, McGraw-Hill, 1998

[10] Chip Design Magazine, Feb./Mar. 2006, www.chipdesignmag.com

[11] www.forteds.com

Ein Signalverarbeitungssystem zur Verbesserung der Kommunikation in Fahrzeugen

Johannes Pittermann, Sebastian Stenzel, Jürgen Freudenberger

HTWG Konstanz, Brauneggerstraße 55, 78462 Konstanz

juergen.freudenberger@htwg-konstanz.de

Die Kommunikation zwischen der vorderen und hinteren Sitzreihe im Auto gestaltet sich aufgrund von Fahrgeräuschen und der Akustik der Fahrgastzelle (Sitzposition und Blickrichtung der Sprecher) als relativ schwierig. Beispielsweise neigen Fahrer häufig dazu, sich umzudrehen, um von den hinteren Passagieren besser verstanden zu werden, wodurch sie allerdings unbewusst die eigene Sicherheit und die der Mitfahrenden gefährden.

In diesem Artikel beschreiben wir Möglichkeiten zur Verbesserung der Sprachverständlichkeit, wobei wir den Fall betrachten, in dem Fahrer oder Beifahrer zu den Mitfahrern auf der Rückbank sprechen. Wir stellen ein Innenraumkommunikationssystem vor, welches das Sprachsignal von vorne nach hinten, zusätzlich zur natürlichen Ausbreitung, aufbereitet und verstärkt - unter besonderer Berücksichtigung der Innenraumakustik. Insbesondere beschreiben wir unsere Ansätze zur Rausch- und Rückkopplungsunterdrückung mittels spektraler Subtraktion, Frequenzverschiebung und Feedbackunterdrückung, welche wir in unserem System verwenden.

1. Einführung

Fahrzeuge der Ober- und Luxusklasse verfügen zwar zumeist über eine geräuschgedämmte Fahrgastzelle, dennoch gestaltet sich eine Unterhaltung zwischen Fahrzeuginsassen der vorderen und hinteren Sitzreihe als eher schwierig. Im Gegensatz zur "normalen" Kommunikation (mit Blickkontakt) gibt es im Auto Fahrgeräusche, außerdem sind die Positionen und Blickrichtungen der Gesprächspartner durch die Anordnung der Sitze festgelegt, wodurch die Kommunikation zusätzlich erschwert wird. Nicht zuletzt dadurch, dass Fahrer häufig versucht sind, sich zu ihren Gesprächspartnern umzudrehen, empfinden es die Fahrgäste als unangenehm, längere Gespräche zu führen. Aus Gründen des Komforts und der Sicherheit ist daher ein System wünschenswert, welches die natürlichsprachliche Kommunikation unterstützt bzw. verbessert.

In diesem Artikel stellen wir ein Innenraumkommunikationssystem vor, welches die Kommunikation in Fahrzeugen verbessert. Dieses System funktioniert prinzipiell wie eine Sprechanlage zwischen den einzelnen Sitzen. Darüber hinaus kann es auch als Front-End für weitere Anwendungen wie bspw. Freisprechtelefonie, Sprachbedienung von Geräten, Sprechfunk oder Dialogsysteme dienen. Ähnliche Ansätze wurden u.a. in [1], [7], [5] und [8] beschrieben.

Klassischerweise verbindet man mit Kommunikationssystemen eine bidirektionale Kommunikation, d.h. man erwartet, dass ein solches System sowohl die Kommunikation von vorne nach hinten wie auch von hinten nach vorne verbessert. Die praktische Erfahrung wie auch Messungen haben jedoch ergeben, dass besonders die Kommunikation von vorne nach hinten eine höhere Aufmerksamkeit in Bezug auf Signalverbesserungen verlangt [3]. Zum einen ist die Abstrahlcharakteristik des Mundes im Kopf nach vorne gerichtet, insbesondere für höhere Frequenzen, zum anderen verfügen bereits Mittelklassefahrzeuge mit Freisprecheinrichtung über alle notwendigen Mikrofone und Lautsprecher, um die Kommunikation von vorne nach hinten zu verbessern. Darüber hinaus wurde bereits in Messungen nachgewiesen, dass bei derzeitigen Mikrofonpositionen (keine Nahbesprechungsmikrofone) durch einfache Verstärkung ohne weitere Signalverarbeitung eine Verbesserung der Signalqualität um 10 dB bei der Kommunikation von vorne nach hinten möglich ist, wohingegen die erwartete Verbesserung von hinten nach vorne eher niedriger liegt. Für die Kommunikation zwischen Fahrer und Beifahrer ist keine Verbesserung zu erwarten [5].

Für Innenraumkommunikationssysteme gelten strenge Einschränkungen in Bezug auf maximale Verstärkung und Signalverzögerung vom Sprecher zum Zuhörer, da ansonsten ungewollte Irritationen entstehen können. Bei einer hohen Verstärkung und langer Verzögerung beispielsweise nimmt der Sprecher sein eigenes Echo wahr. Darüber hinaus bewirken kurze Verzögerungszeiten und hohe Verstärkung beim Zuhörer den Eindruck, der Sprecher säße in Richtung des Lautsprechers (meist hinter oder neben dem Fahrgast). Eine derartige Fehllokalisierung wird eher als unangenehm empfunden und sollte daher ver-

mieden werden. Normalerweise sollte die Gesamtverzögerung (inklusive A/D- und D/A-Wandlung) 10ms nicht übersteigen [8]. Aufgrund dieser Einschränkung arbeiten die Signalverarbeitungsalgorithmen typischerweise im Zeitbereich, um blockweise Verarbeitung zu vermeiden. Wir stellen daher einen Ansatz zur spektralen Subtraktion vor, bei dem das Filter im Frequenzbereich berechnet wird, die eigentliche Filterung aber im Zeitbereich stattfindet. Dieses Filter ist nicht nur darauf ausgelegt, Hintergrundgeräusche zu unterdrücken, sondern auch darauf, Pfeifgeräusche durch akustische Rückkopplung zu verringern.

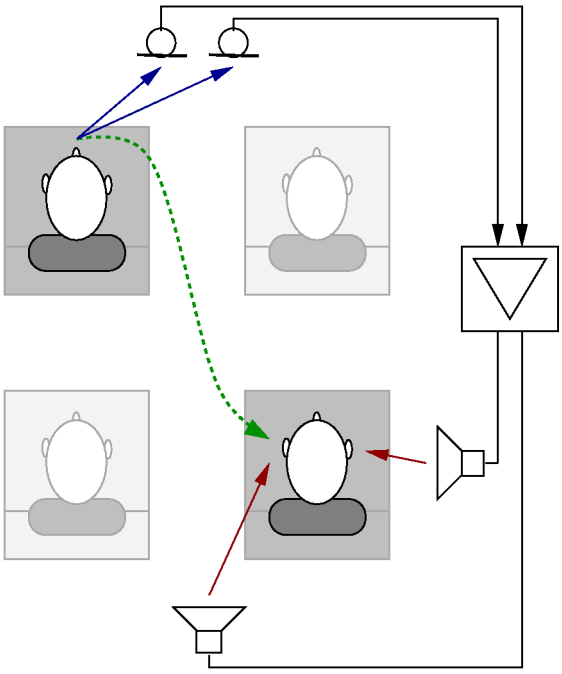


Abbildung 1: Sitzpositionen im Auto und die akustische Ausbreitung der Sprachsignale vom Fahrer zum Fahrgast hinten rechts.

In diesem Beitrag stellen wir folglich ein Halbduplex-Kommunikationssystem vor, welches Sprache von der Vordersitzen zu den Rücksitzen zusätzlich zum akustischen Pfad verstärkt. Die entsprechende geometrische Anordnung der Insassen, Mikrofone und Lautsprecher ist in Abbildung 1 dargestellt. Hierbei ist zu beachten, dass unser vorgestelltes System speziell darauf ausgelegt ist, die durch Freisprecheinrichtung und Audiosysteme bereits vorhandenen Mikrofone und Lautsprecher zu verwenden – im Gegensatz zu den Systemen in [5] und [8] stehen unserem System also keine optimal angepassten und positionierten Mikrofone und Lautsprecher zur Verfügung. Nichtsdestotrotz erreichen wir mit dem System eine sowohl hörbare wie auch messbare Verbesserung der Kommunikation zwischen Vorder- und Rücksitzen.

Um die Komplexität in der Simulation zu verringern, betrachten wir in unseren Experimenten und Simulationen ausschließlich die Kommunikation vom Fahrer zum Mitfahrer hinten rechts. Der stets vorhandene direkte akustische Pfad ist durch eine gestrichelte Linie angedeutet, weitere akustische Ausbreitungspfade sind als durchgezogene Linien dargestellt. Die Sprachsignale werden von zwei im Rückspiegel angebrachten Mikrofonen erfasst und das aufbereitete Signal wird über zwei Lautsprecher (einer in der Tür hinten rechts, einer in der Hutablage) ausgegeben. Obwohl dies nicht explizit in der Zeichnung dargestellt ist, betrachten wir in unseren Experimenten stets beide Pfade zum rechten und linken Ohr des Mitfahrers. Im verbleibenden Teil des Artikels beschreiben wir zunächst den Aufbau unseres vorgestellten Kommunikationssystems bevor wir auf unsere experimentellen Ergebnisse eingehen.

2. Systemaufbau

Der schematische Aufbau unseres vorgestellten Systems ist in Abbildung 2 dargestellt. Die Sprachsignale werden von beiden Mikrofonen aufgezeichnet und zunächst mit Hilfe eines Delay-and-sum-Beamformers aufbereitet. Für Frequenzen oberhalb 1kHz erreicht dieser Beamformer eine Signalverstärkung von 2-3dB, wohingegen die Verstärkung für niedrigere Frequenzen eher marginal ausfällt. Durch die Verwendung eines fixen (und nicht adaptiven) Beamformers kann die Komplexität der Feedbackunterdrückung gering gehalten werden, da hier lediglich ein Feedbackcanceler benötigt wird – bei einem adaptiven Beamformer würde jeweils ein Feedbackcanceler pro Mikrofonkanal benötigt.

Bedingt durch die akustischen Pfade von den Lautsprechern hinten zu den Mikrofonen vorne bildet das Innenraumkommunikationssystem einen geschlossenen Kreislauf, in welchem bei zu hoher Verstärkung Instabilitäten auftreten können. Daher besteht die Aufgabe des Feedbackcancelers darin, die akustische Rückkopplung zu schätzen und diese vom Ausgangssignal des Beamformers abzuziehen. Aufgrund der starken Korrelation zwischen Eingangssprachsignal und Lautsprechersignal, ähnlich wie auch bei Hörgeräten [4], gestaltet sich die Feedbackunterdrückung jedoch als extrem schwierig. Daher werden in der Regel weitere Ansätze benötigt, aufkommende Rückkopplungsgeräusche zu unterdrücken. Systeme wie die in [5] und [8] setzen adaptive FIR-Filter ein, um periodische Signalanteile zu prädizieren und zu unterdrücken. Für unser System verwenden wir die spektrale Subtraktion, um sowohl den Hintergrundgeräuschpegel zu verringern, wie auch Rückkopplungsfrequenzen zu dämpfen.

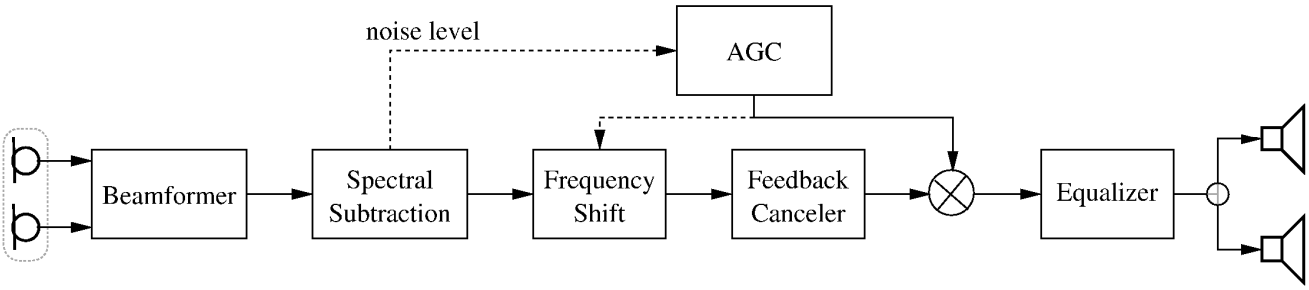


Abbildung 2: Schematischer Aufbau des Innenraumkommunikationssystems

Konventionelle Geräuschreduktion, wie z.B. spektrale Subtraktion, wird üblicherweise im Frequenzbereich durchgeführt. Das Ziel des zugrundeliegenden Algorithmus ist zunächst die Schätzung der Geräuschanteile im Kurzzeitspektrum des Signals und die daraus abgeleitete Berechnung eines Filters, mit welchem diese Geräuschanteile im Signal sinnvoll gedämpft werden. Bei einer Systemabtastrate von 16kHz arbeitet die spektrale Subtraktion typischerweise mit überlappenden Signalblöcken der Länge 256 oder 512 Samples, von welchen dann jeweils das Kurzzeitspektrum mittels FFT berechnet wird. Die damit verbundene Verzögerung wäre jedoch bereits 16ms bzw. 32ms, und damit für unser System nicht akzeptabel.

Um den Geräuschanteil im Signal möglichst präzise zu schätzen wird eine robuste Methode zu Sprachpausendetektion (VAD) benötigt, da Geräusche nur in Sprachpausen erfassbar sind. Bei einfachen rein leistungsbasierten VAD-Algorithmen werden häufig schnelle Wechsel zwischen Sprachaktivität und Pause festgestellt, obwohl diese so im Signal nicht vorhanden sind, da die Signalenergie zu diesem Zeitpunkt gerade leicht um den Schwellwert oszilliert. Daher betrachten wir zu jedem diskreten Zeitpunkt k zunächst ein modifiziertes Leistungsdichtespektrum $\widetilde{\Phi}_{xx}(f,k)$, welches rekursiv von zuvor berechneten Leistungsdichtespektren abhängt und sich wie folgt errechnet:

$$\widetilde{\Phi}_{xx}(f,k) = (1-\delta)\cdot\widetilde{\Phi}_{xx}(f,k-1) + \delta\cdot\Phi_{xx}(f,k)$$

mit Glättungskonstante $\delta \in [0.1,0.9]$ (typischerweise ca. 0.3) und dem aktuellen Leistungsdichtespektrum $\Phi_{xx}(f,k)$. In $\widetilde{\Phi}_{xx}(f,k)$ betrachten wir nun die Leistungen in drei Frequenzbändern (unter 750Hz, 750Hz bis 1875Hz und über 1875Hz).

Mit dem aktuellen Leistungsdichtespektrum $\Phi_{xx}(f)$ und dem in den vorigen Sprachpausen geschätzten Rauschleistungsdichtespektrum $\Phi_{nn}(f)$ lassen sich nun die Filter gemäß

$$\hat{H}^{ss}(f) = \sqrt{1-\frac{\Phi_{nn}(f)}{\Phi_{xx}(f)}}$$

berechnen und auf das Signal im Frequenzbereich anwenden bevor eine inverse FFT durchgeführt wird.

Da die dadurch entstehende Verzögerung für unsere Anwendung nicht tolerierbar ist, wenden wir die inverse FFT bereits auf die Übertragungsfunktion des Filters an und erhalten unser Ausgangssignal durch die Faltung von Eingangssignal mit den Filterkoeffizienten im Zeitbereich, wobei die Filterkoeffizienten alle 2ms (entsprechend einer Blocklänge von 64 Samples und einer Überlappung von 32 Samples) aktualisiert werden. Um sogenannte "musical tones" zu vermeiden, werden die Filterkoeffizienten im Zeitbereich rekursiv geglättet, d.h. mit dem $i$ten Filterkoeffizienten $\hat{h}_i^{ss}(k) = F^{-1}\{\hat{H}^{ss}(f)\}$ zum Zeitpunkt k erhalten wir

$$h_i^{ss}(k) = (1-\gamma)\cdot\hat{h}_i^{ss}(k-1) + \gamma\cdot\hat{h}_i^{ss}(k)$$

mit Glättungskonstante $\gamma \in [0.1,0.9]$.

Einerseits führt die häufige Aktualisierung des Rauschunterdrückungsfilters zu einer Erhöhung der Rechenkomplexität des Systems. Andererseits zeigen jedoch Simulationen, dass bereits eine kurze FFT-Länge von 64 oder 128 ausreicht, um den benötigten SNR-Gewinn von 3-5dB zu erhalten, wodurch die Gesamtkomplexität mit der konventioneller Geräuschreduktionsalgorithmen vergleichbar ist.

Um Rückkopplungssignale zu vermeiden, werden häufig auch Nichtlinearitäten in derartige Systeme eingebaut. Für unser System verwenden wir eine Frequenzverschiebung, deren Prinzip der in [6] veröffentlichten Methode ähnelt: das Einseitenbandsignal wird nicht um einen fixen Frequenzversatz verschoben, sondern um einen 5x pro Sekunde sinusförmig zwischen 0Hz und 10Hz wechselnden Versatz

("frequency warbling"). Die Frequenzverschiebung selbst wird nur bei einem hohen Geräuschpegel eingesetzt, da in diesem Fall das Signal stärker verstärkt werden muss, was wiederum die Rückkopplungsgefahr erhöht. Bei niedrigem Geräuschpegel, wo selbst die geringe Verzerrung durch die Frequenzverschiebung als störend empfunden werden kann, bleibt das Signal unverändert.

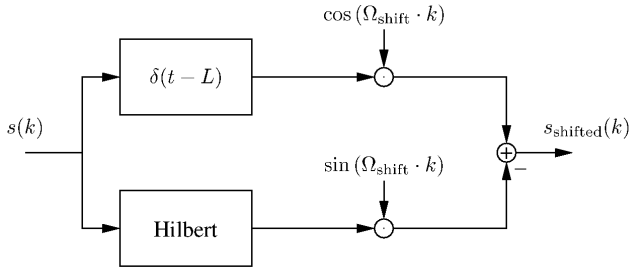


Abbildung 3: Blockdiagramm der Frequenzverschiebung

Bedingt durch die zugrunde liegende Faltung im Frequenzbereich, benötigt die Frequenzverschiebung ein Einseitenbandsignal, welches mit einer komplexen Exponentialfunktion multipliziert, sodass sich das Ausgangssignal wie folgt errechnet:

$$s_{\text{shifted}}(k) = \Re\{s_{\text{SSB}}(k) \cdot e^{\Omega_{\text{shift}} \cdot k}\}$$

mit dem Einseitenbandsignal $s_{\text{SSB}}(k)$ und dem Faktor $\Omega_{\text{shift}} = 2\pi j \cdot f_{\text{shift}}(k) / f_s$ mit variablem Frequenzversatz $f_{\text{shift}}(k)$ und Abtastfrequenz f_s. In unserer Implementierung verwenden wir den in Abbildung 3 gezeigten Ansatz mit einem Hilbertfilter. Das ideale Hilbertfilter ist ein akausales Filter mit unendlich langer Impulsantwort, welches sich somit zunächst nicht realisieren lässt. Daher verwenden wir ein approximiertes Hilbertfilter der Länge $2L+1$ Samples im unteren Zweig, und, um die Synchronität zu bewahren eine Verzögerung um L Samples im oberen Zweig. Die verzögerten bzw. gefilterten Signale werden mit einem Kosinus bzw. Sinus mit dem entsprechenden Frequenzversatz multipliziert und aufaddiert. Für Filter und Verzögerung wählen wir $L = 16$, was einen passablen Mittelweg zwischen Systemverzögerung (lediglich 1ms) und Qualität der Filterübertragungsfunktion darstellt. Das Diagramm in Abbildung 4 verdeutlicht, dass die Übertragungsfunktion nicht für alle Frequenzen gleich ist, was im Zusammenspiel eine leichte Verzerrung des Ausgangssignals mit sich bringt.

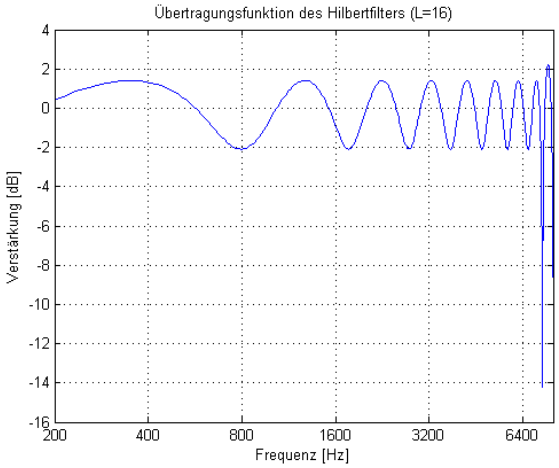


Abbildung 4: Übertragungsfunktion des Hilbertfilters

Zusätzlich zur Frequenzverschiebung verwenden wir einen Feebackcanceler, welcher mit Hilfe von linearer Prädiktion harmonische Signalanteile (insbesondere Rückkopplungssignale) detektieren und unterdrücken kann. Das Eingangssignal wird um 3ms verzögert und ein FIR-Filter der Länge $L = 30$ wird mittels LMS-Algorithmus derart adaptiert, dass die Differenz zwischen Originalsignal und verzögertem und gefiltertem Eingangssignal minimiert wird:

$$\hat{h}_i^{\text{fb}}(k) = \hat{h}_i^{\text{fb}}(k-1) + \left(\frac{\alpha}{\sum_j \|d(k-j)\|^2} \cdot s(k) \cdot d(k-i) \right)$$

mit dem $i$ten Filterkoeffizienten $\hat{h}_i^{\text{fb}}(k)$ des FIR-Filters zum Zeitpunkt k $(0 \le i \le L-1)$, Adaptionsrate α sowie Originalsignal $s(k)$ und verzögertem Signal $d(k)$.

Die Gesamtverstärkung des Innenraumkommunikationssystems muss sich andauernd der Fahrsituation anpassen. Anhand des von der spektralen Subtraktion geschätzten Geräuschpegels, passt die Automatische Verstärkungsregelung (AGC) die Verstärkung dergestalt an, dass das Signal nur dann verstärkt wird, sofern der Geräuschpegel darauf hindeutet, dass Unterstützung durch das System erforderlich ist. Unsere AGC basiert auf einer Tabelle mit drei Geräuschpegelregionen mit den zugehörigen Verstärkungen (-4dB, -2dB und 0dB) relativ zur maximalen Gesamtverstärkung.

Nach der AGC, wird das Signal entzerrt, unter Berücksichtigung der akustischen Verhältnisse der

jeweiligen Fahrgastzelle. Der Entzerrer ist als IIR-Filter realisiert, dessen Filterkoeffizienten anhand der relevanten akustischen Ausbreitungspfade zwischen Lautsprechern und Mikrofonen bestimmt werden.

In der Simulation fügen wir darüber hinaus noch künstlich eine Verzögerung um 2ms ein, um die Laufzeiten in den A/D- und D/A-Wandlern zu modellieren.

3. Ergebnisse

In unseren Experimenten verwenden wir die Impulsantworten der Fahrgastzelle einer Mercedes-Benz S-Klasse sowie Hintergrundgeräusche im gleichen Fahrzeug bei einer Geschwindigkeit von 100km/h. Aus Gründen der besseren Vergleichbarkeit der Ergebnisse skalieren wir Sprach- und Störsignale derart, dass wir ein zuvor festgelegtes Signal-Rauschleistungsverhältnis (SNR) am linken Mikrofon des Rückspiegels erzielen. Die Spektrogramme des Signals am linken Ohr des Fahrgasts hinten rechts, mit und ohne Unterstützung unseres Innenraumkommunikationssystems, bei einem SNR von 10dB, sind in Abbildung 5 dargestellt.

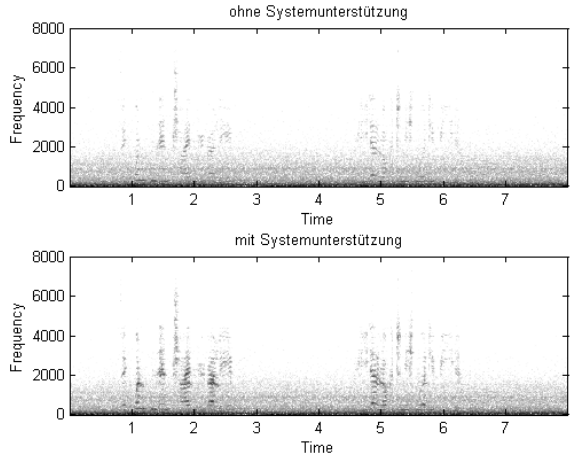


Abbildung 5: Spektrogramme des Sprachsignals am linken Ohr des Fahrgasts

Ein Vergleich beider Spektrogramme ergibt zum einen, dass der allgemeine Geräuschpegel gleich ist, was darauf zurückzuführen ist, dass in beiden Fällen das gleiche Hintergrundgeräusch anliegt. Andererseits ist eine Verstärkung des Sprachsignals durch das Innenraumkommunikationssystem deutlich erkennbar.

Zusätzlich zum subjektiven Höreindruck (Anhören und Vergleich der Ausgangssignale) und zum optischen Vergleich der Spektrogramme verwenden wir auch objektive Messgrößen für die Evaluierung unseres Systems. Zunächst betrachten wir die Signalverzögerungen, welche wir mit Hilfe von Composite-Sprachsignalen (Sprachsignale mit sekündlich wiederholten Rauschbursts der Länge 200ms) bei hohem SNR (im Stand, Motor ausgeschaltet) messen. Berechnungen der Kreuzkorrelationen bei ausgeschaltetem Innenraumkommunikationssystem ergeben eine Signallaufzeit von 61 Samples (3.8ms) auf dem direkten akustischen Pfad. Schalten wir nun unser System ein und ziehen die sich damit ergebende Kreuzkorrelation und die Kreuzkorrelation ohne System von einander ab, erhalten wir eine Systemlaufzeit von 110 Samples (7.1ms). Durch die nur sehr kleine Verzögerung erreicht die erste Wellenfront den hinteren rechten Fahrgast von vorne (und nicht vom Lautsprecher) ohne wahrnehmbares Echo und bewirkt somit dem Haas-Effekt [2] entsprechend den korrekten Eindruck, die Sprache käme auch tatsächlich vom Fahrer.

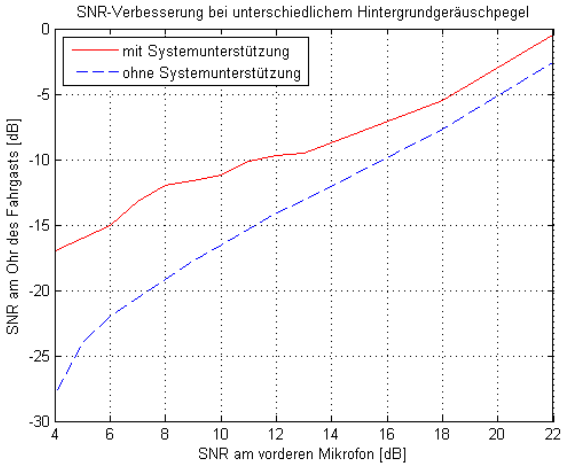


Abbildung 6: SNR (in dB) am vorderen Mikrofon und beim Fahrgast

Mit den so ermittelten Verzögerungen können wir nun die Signal-Rauschleistungsverhältnisse für unterschiedliche Hintergrundgeräuschpegel messen. Wie im Diagramm in Abbildung 6 dargestellt, messen wir zunächst das Referenz-SNR am linken Mikrofon im Rückspiegel. Dann bestimmen wir das SNR am linken Ohr des Fahrgasts hinten rechts - mit Unterstützung durch unser Innenraumkommunikationssystem und ohne. Vergleicht man nun die Kurven für beide Fälle, ist klar ersichtlich, dass unser System eine Verbesserung von mindestens 3dB (bei höherem Hintergrundgeräuschpegel sogar darüber) bewirkt.

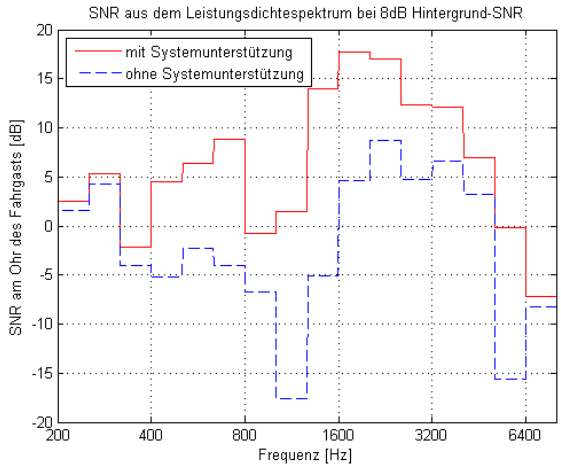


Abbildung 7: SNR (in dB) in relevanten Drittelok-tavbändern

Die Unterschiede im SNR zwischen den Dritteloktav-bändern sind in Abbildung 7 aufgeschlüsselt. Vergleicht man die gestrichelten Linien (ohne Systemunterstützung) und die durchgezogenen Linien (mit Systemunterstützung), fällt auf, dass die SNR-Verbesserung durch unser Innenraumkommunikationssystem vorrangig in sprachrelevanten Bändern zwischen 300Hz und 6400Hz geschieht, wodurch auch der subjektive Eindruck bestätigt wird, dass die Signalqualität tatsächlich verbessert wird.

4. Zusammenfassung

In dieser Veröffentlichung haben wir ein Halbduplex-Innenraumkommunikationssystem vorgestellt, mit welchem die akustische Ausbreitung von Sprachsignalen von der vorderen Sitzreihe zur hinteren Sitzreihe in einem Fahrzeug verbessert werden kann. Das System weist eine geringe Rechenkomplexität auf und wurde bereits auf einem digitalen Signalprozessor vom Typ *TMS320C6713* der Firma Texas Instruments implementiert.

Unsere Versuchsreihen mit realen Daten zeigen eine hörbare Verbesserung des Sprachsignals, welche nicht nur in Spektrogrammen visualisiert werden kann (siehe Abbildung 5) sondern auch durch Messungen des Signal-Rauschleistungsverhältnisses (SNR) bestätigt wird (siehe Abbildung 6 und Abbildung 7). Für die Halbduplex-Kommunikation zwischen Fahrgästen der vorderen Sitzreihe und Fahrgästen der hinteren Sitzreihe erzielt unser System eine SNR-Verbesserung zwischen 3dB und 7dB bei typischen Hintergrundgeräuschpegeln in der Fahrgastzelle eines Autos.

5. Quellenangaben

[1] B. M. Finn. “Integrated vehicle voice enhancement system and hands-free cellular telephone system”. European Patent EP 0 932 142 A2, Jul. 1999.

[2] H. Haas. “The influence of a single echo on the audibility of speech”. J. Audio Eng. Soc., 20:145–159, March 1972.

[3] E. Hänsler, G. Schmidt, editors. “Topics in Acoustic Echo and Noise Control: Selected Methods for the Cancellation of Acoustical Echoes, the Reduction of Background Noise, and Speech Processing”. Springer, 2006.

[4] J. M. Kates. “Signal Processing for hearing aids”. Kluwer Academic Publishers, 1998. Chapter 6 in M. Kahrs, K. Brandenburg: Applications of Digital Signal Processing to Audio and Acoustics.

[5] K. Linhard, J. Freudenberger. “Passenger in-car communication enhancement”. In Proc. EUSIPCO, Vienna, pages 21–24, 2004.

[6] G. Nishinimoya. “Improvement of acoustic feedback stability of public address system by warbling”. In Proceedings of the Sixth International Congress of Acoustics, pages 93–96, 1968.

[7] K. Schaaf, J. Schultz, and K. Tontch. “Digital voice enhancement for improved in-car communication”. In Proc. 3rd IFAC Workshop Advances in Automotive Control, Karlsruhe, Germany, March 2001.

[8] G. Schmidt and T. Haulick. “Signal processing for in-car communication systems”. Signal Processing, 86(6):1307–1326, 2006.

Designstudie für ein Laser-Radar-IC

Georg Vallant, Gerhard Forster
Institut für Kommunikationstechnik
Hochschule Ulm, Prittwitzstraße 10, 89075 Ulm
vallant@mail.hs-ulm.de forster@hs-ulm.de

Im Rahmen eines Entwicklungsvorhabens wird die vollständige Integration eines mehrkanaligen Laser-Radar-Systems für bildgebende Verfahren (LIDAR) angestrebt. Das System soll später in der Umfeldsensorik und dort im Besonderen in der dreidimensionalen Oberflächenvermessung in Echtzeit Anwendung finden. Es zeichnet sich durch eine Auflösung von 15 cm auf einer Entfernung von 1 km aus. Dabei kann eine Messung alle 10 µs erfolgen. Der Realisierung gingen mehrere Entwicklungsarbeiten voraus. Dieser Beitrag präsentiert die Grundlagen, den Systementwurf, den Schaltungsentwurf inklusive Simulation und das Layout eines 4-kanaligen Testchips in einer 0,35 µm-CMOS-Technologie (V_{DD} = 3.3 V). Ein Empfangskanal besteht aus einer Empfangsdiode, einem Transimpedanzverstärker mit einer Grenzfrequenz von 380 MHz, einem High-Speed-Komparator mit einer Verzögerungszeit von 1,4 ns und der Auswerteelektronik. Die Stromaufnahme eines Kanals beträgt 2,1 mA. Das System arbeitet mit einem internen Takt von 640 MHz, der mittels Frequenzsynthese aus einem externen Takt gewonnen wird. Anhand der messtechnischen Charakterisierung dieses Testchips soll die monolithische Realisierbarkeit von mindestens 64 Kanälen nachgewiesen werden.

1. Einleitung

Bisherige bildgebende LIDAR-Verfahren arbeiten größtenteils einkanalig [1,2]. Unser Entwicklungsziel war es, mehrere Empfangskanäle in einer sogenannten Detektorzeile, dicht nebeneinander angeordnet, monolithisch zu integrieren. Ein mehrkanaliges System verbessert die Möglichkeiten, großflächig Objekte abzuscannen. Das Verfahren kann dann in der erweiterten Abstandsdetektion für Flugobjekte (Assistenzhilfe für Hubschrauber bei schlechten Sichtverhältnissen) und der dreidimensionalen Oberflächenvermessung in Echtzeit Einsatz finden. Der wesentliche Vorteil gegenüber dem klassischen RADAR ist der Lichtausbreitungscharakter, der eine Detektion nichtmetallischer Objekte ermöglicht und aufgrund der geringeren Wellenlänge das Potential zu höherer Auflösung bietet.

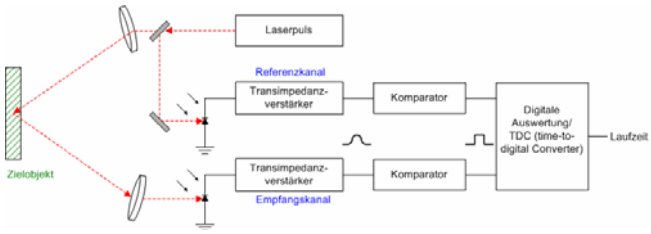


Bild 1: Entfernungsmessung mittels LIDAR

2. Systementwurf

Das Konzept sieht die Existenz eines einmal on-chip vorhandenen Referenzkanals vor. Daneben existieren *n* Empfangskanäle, von denen jeder einen Pixel repräsentiert. Das Prinzip der Entfernungsmessung ist in Bild 1 dargestellt: Ein über optische Elemente abgelenkter Laserpuls wird auf den Referenzkanal geführt. Die Empfangsdiode wandelt die einfallenden Photonen in 1 bis 2 ns breite Strompulse. Verwendet man eine Avalanche-Diode mit einem Quantenwirkungsgrad von 30, hat man im minimalen Fall 500 nA starke Pulse zu erwarten. Die Anstiegs- bzw. Abfallzeit beträgt ca. 100 ps. Die Strompulse werden nun von der ersten Stufe des Kanals, dem Transimpedanzverstärker, in einen Spannungspuls gewandelt. Der sich anschließende Komparator verfügt über eine feste Schaltschwelle und erzeugt einen Digitalpuls, welcher in der digitalen Auswerteelektronik einen Zähler startet. Währenddessen wurde der Laserpuls auf das Zielobjekt geführt und trifft nach einer gewissen Zeit, reflektiert und gedämpft, auf den Empfangskanal. Dessen Verarbeitungskette ist identisch mit der des Referenzkanals. Der erzeugte Digitalpuls hält den Zähler an und beendet die Distanzmessung. Der re-

sultierende Zählerstand N stellt das Ergebnis dar. Die Auflösung wird durch die Taktfrequenz des internen Zählers bestimmt. Die Entfernung berechnet sich nach der bekannten Formel aus der Lichtgeschwindigkeit:

$$d = \frac{t_{time\_of\_flight} \cdot c}{2} \qquad \text{wobei} \qquad t_{time\_of\_flight} = N \cdot T_{takt}$$

Dabei darf die Periodendauer des Takts nicht beliebig klein gewählt werden, weil einerseits der Zähler nur eine endliche Teilerfrequenz erlaubt, andererseits aber auch die zulässige Stromaufnahme begrenzt ist. An dieser Stelle gilt es, einen Kompromiss zu finden.

Das reflektierte Licht erfährt eine vom Reflexionsfaktor und der Entfernung des Zielobjekts abhängige Dämpfung. Dies führt zu einer extremen Dynamik (Faktor ~ 10^8), die ein elektronisches System nicht vollständig abdecken kann. Man beschränkt sich auf eine kleinere Dynamik, vom schwächsten Puls (500 nA), welcher mit der unteren Rauschgrenze zusammenfällt, bis zum stärksten Puls (100 µA), der die Empfangsstufe in die Begrenzung steuert. Die gesamte Dynamik ist nur mit optischen Mitteln erreichbar, indem eine Vorauswahl zwischen Nah- und Fernbereich des Lasers mit entsprechender Leistungsregulierung getroffen wird. In diesem Zusammenhang manifestiert sich das Problem des sog. Walk-Errors: Er beschreibt den Effekt, dass die begrenzte Bandbreite des Eingangsverstärkers und die nichtlineare Natur des Komparators zu einer Abhängigkeit der Verzögerungszeit von der Eingangsamplitude des Signals führen. Starke Signale führen zu einer geringen Verzögerungszeit, schwache Signale hingegen zu einer großen Verzögerungszeit. Dieser Laufzeitfehler verfälscht das Messergebnis im Fernbereich, verschlechtert also die Auflösung. Eine typische Walk-Error-Kurve ist in Bild 2 dargestellt.

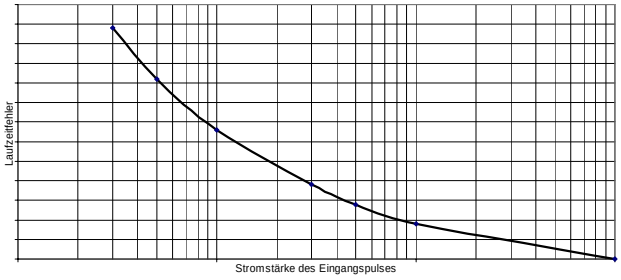


Bild 2: Abhängigkeit der Gesamt-Verzögerungszeit durch Transimpedanzverstärker und Komparator von der Eingangsamplitude

Eine effiziente und stromsparende Methode zur Korrektur dieses Fehlers besteht darin, zusätzlich eine Amplitudeninformation des empfangenen Pulses zu gewinnen und das Ergebnis im Nachhinein zu korrigieren. Dies lässt sich mit einem zusätzlichen in den Kanal eingebrachten Komparator mit einer zweiten, erhöhten Schaltschwelle realisieren.

Die an den Eingangsverstärker gestellten Anforderungen sind in Tabelle 1 zusammengefasst. Der Verstärker soll als Transimpedanzverstärker wirksam sein, mit niederohmigem Ein- und Ausgang: Ein Eingangsstrom von 1 µA soll in eine Spannung von 100 mV umgesetzt werden. Sein Eigenrauschen sollte minimal gehalten werden, da es sonst die Wahl der Komparatorschaltschwelle und somit die Dynamik des Systems nach unten hin begrenzt. Weiterhin ist die Verlustleistung kritisch. Die Stromaufnahme eines Eingangsverstärkers sollte aufgrund der angestrebten Integration von mehreren Kanälen 2 mA nicht übersteigen.

Tab. 1: Angestrebte Daten des Eingangsverstärkers

Transimpedanz	$R_f = 100\ \text{k}\Omega$
3dB-Grenzfrequenz	$f_{3dB} > 300\ \text{MHz}$
Eff. Rauschsspannung	$u_{r,eff} < 12\ \text{mV}$
Stromaufnahme	$I_{dd} < 2\ \text{mA}$

Die Komparatorlaufzeiten sind zunächst unerheblich, solange sie im Referenzkanal und im Empfangskanal gleich sind. Fehler entstehen wegen des Walk-Errors im Empfangskanal. Da der Walk-Error mit der Absolutlaufzeit des Komparators abnimmt, wird eine möglichst geringe Verzögerungszeit angestrebt. Geht man von einer Auflösung von 15 cm aus, so wird ein Walk-Error kleiner 1 ns benötigt. Dies sollte bei einer Verzögerungszeit von 2 ns möglich sein. Weiterhin sollte die Offsetspannung des Komparators kleiner als das Ausgangsrauschen des Eingangsverstärkers sein. Hierzu ist es nötig, trotz der hohen Anforderung an die Geschwindigkeit die Kanallängen der Transistoren nicht auf das Technologieminimum zu setzen. Tabelle 2 fasst die Zielspezifikationen für den Komparator zusammen.

Tab. 2: Angestrebte Daten des Komparators

Verzögerungszeit	$t_{pdh} = t_{pdl} < 2\ \text{ns}$
Walk-Error	$t_{walk} < 1\ \text{ns}$
Offsetspannung	$U_{offset} < 5\ \text{mV}$
Stromaufnahme	$I_{dd} < 0{,}3\ \text{mA}$

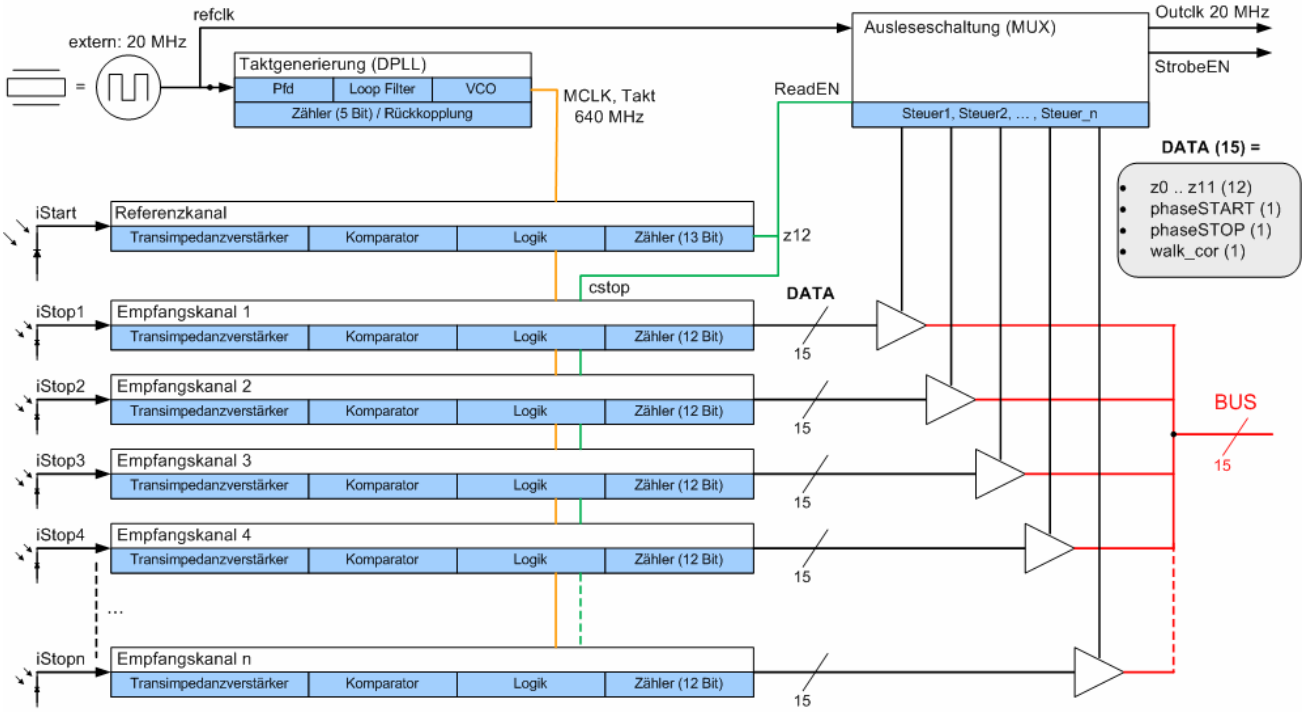


Bild 3: Das Gesamtsystem samt Einzelkomponenten

Bild 3 zeigt den Referenzkanal, die *n* Empfangskanäle sowie die Taktgenerierung. Letztere ist als DPLL implementiert und erzeugt nach dem Prinzip der Frequenzsynthese aus einem 20 MHz-Takt einen hochfrequenten 640 MHz-Takt, der zur Messung eingesetzt und an alle Kanäle geführt wird. Weiterhin ist eine Ausleseschaltung dargestellt. Quantisiert man die Strecke von 1 km in Teilabschnitte von 30 cm, muss bereits ein 12-Bit Zähler mit 4096 Zuständen verwendet werden. Da der Zähler erst nach Abschluss des Zählvorgangs ausgelesen werden muss, eignet sich ein asynchroner Zähleraufbau (Ripple-Counter), der ein wesentlich einfacheres Layout und geringere Verlustleistung als ein Synchronzähler ermöglicht.

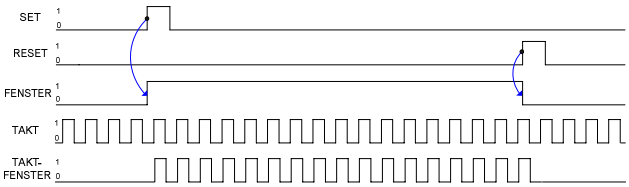


Bild 4: Zähleransteuerung mit Start- und Stopp-Puls

Eine Auswertelogik erzeugt entsprechend Bild 4 aus dem Startpuls (SET), der vom Referenzkanal auf alle *n* Empfangskanäle verteilt wird, und dem reflektierten Stopp-Puls (RESET) ein FENSTER, das mit dem freilaufenden Takt von 640 MHz zu einem TAKT-FENSTER verknüpft wird. Dieses TAKTFENSTER wiederum steuert mit jeder steigenden Flanke den Asynchron-Zähler an. Da jeder Kanal über ein eigenes 12-Bit-Wort (z0 bis z11) am Zählerausgang verfügt, müssen die Ergebnisse auf einem Datenbus nach außen geführt werden. Insgesamt besteht der Datenbus aus 15 Leitungen: phaseSTART und phaseSTOP werden zur Phasendetektion eingesetzt und erhöhen die Auflösung von 30 cm auf 15 cm; die Funktion beider Signale wird in Kapitel 3.4 erläutert. Das Signal walk_cor stellt das Ergebnis der in Kapitel 3.3 beschriebenen Schwellenkontrolle dar. Die Steuerung des Busses erfolgt mit Tristate-Buffern und einer Ausleseschaltung, die nicht-überlappende Phasentakte generiert. Die n Messergebnisse werden zeitlich versetzt nach dem Multiplex-Prinzip nach außen geführt. Zum Auslesen stehen der anschließenden Signalverarbeitung ein Auslesetakt Outclk und ein Enable-Signal StrobeEn zur Verfügung. Die Ausleseschaltung erzeugt nicht-überlappende Phasentakte Steuer_1 bis Steuer_n, die die Tristate-Buffer hochohmig oder durchlässig schalten, so dass immer nur ein Kanal auf den Datenbus gelangt, die anderen aber inaktiv bleiben. Dieses Prinzip, das sich mit einem wandernden Zeiger darstellten lässt, der zirkulierend immer wieder von vorne beginnt, soll in Bild 5 verdeutlicht werden.

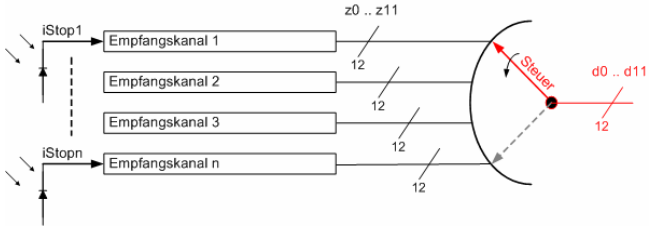


Bild 5: Prinzip der Ausleseschaltung

Der zeitliche Ablauf eines Messvorgangs ist in Bild 6 dargestellt. Um den Beginn einer Messung einzuleiten, wird ein globales high-aktives Enable-Signal systemON verwendet. Liegt systemON auf HIGH, ist das System bereit, mit dem Startpuls die Messung zu beginnen. Während dieser Phase (Measure/Count) detektieren die einzelnen Empfangskanäle die Rücksignale und halten ihre Zähler dementsprechend an. Nach einer festen Zeit von ca. 6,4 µs gibt der Referenzkanal das Logiksignal strobeEN aus, welches im HIGH-Zustand den oben beschriebenen Auslesevorgang ermöglicht. Hierzu verfügt der Referenzkanal über einen 13-Bit Zähler, dessen höchstwertigster Ausgang z12 verwendet wird. Eine weitere Aufgabe des Steuersignals z12 ist es, bei den Kanälen, welche aufgrund zu starker Dämpfung kein Rücksignal empfangen haben, die Fensterung automatisch zu beenden. Dem Auslesevorgang (Read) schließt sich mit dem Sprung von systemOn auf LOW eine Ruhephase (idle) an, in der das System ruht, bis systemON erneut eine Messung beginnt. Die zeitliche Wiederholrate der Messungen legt den duty cycle fest.

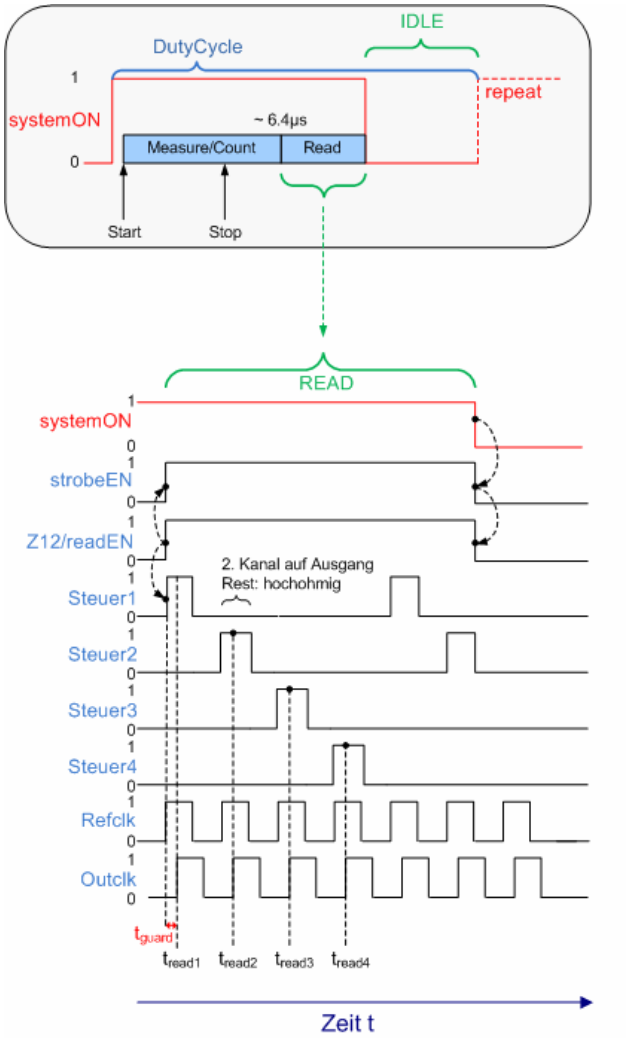


Bild 6: Ablauf des Mess- und Auslesevorgangs

3. Schaltungsentwurf

3.1. Transimpedanzverstärker

Um das Rauschen gering zu halten, wurde die single-ended-Variante nach Bild 7 einem differenziellen Aufbau vorgezogen [3]. Die Verstärkerstruktur basiert auf einer einfachen Inverterschaltung. Die hohe Anforderung an die Bandbreite bedingt die Verringerung der Lastimpedanz, hier durch den zusätzlichen Einsatz eines MOSFETs in Diodenschaltung. Er sorgt außerdem für eine Unabhängigkeit der Verstärkung von den Ausgangswiderständen der Transistoren M_1 bis M_3.

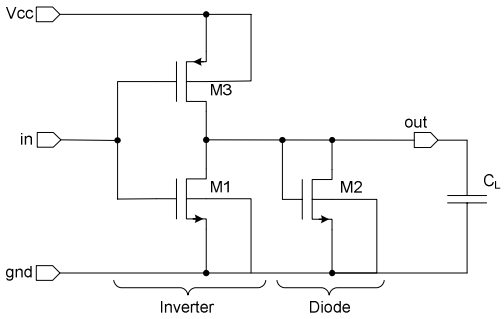


Bild 7: Inverterschaltung Amp mit Diodenlast

Die Spannungsverstärkung A_o einer solchen Stufe berechnet sich zu

$$A_o = -(g_{m1} + g_{m3}) \cdot (r_{o1} \parallel r_{o3} \parallel r_{o2} \parallel \frac{1}{g_{m2}})$$

$$A_o = -\frac{g_{m1} + g_{m3}}{g_{DS1} + g_{DS3} + g_{DS2} + g_{m2}}$$

Die Ausgangsleitwerte sind gegenüber den Steilheiten vernachlässigbar, was zu folgender Formel führt:

$$A_o \approx -\frac{g_{m1} + g_{m3}}{g_{m2}}$$

Leider ist damit noch nicht die Spannungsverstärkung erzielbar, die benötigt wird, um die erforderliche Transimpedanzverstärkung $R_f = 100\ \text{k}\Omega$ nur durch einen Widerstand festzulegen. Deshalb wird die Spannungsverstärkung durch Kaskadierung dreier Einzelstufen erhöht (Bild 8). Die Verstärkung einer Einzelstufe beträgt 15 dB, die Verstärkung nach Kaskadierung folglich 45 dB. Durch die zunehmende Phasendrehung wird es nötig, den Verstärker zu stabilisieren, da keine positive Phasenreserve bei der Transitfrequenz vorliegt. Aus diesem Grund wurde am Eingang eine Kompensationskapazität C_k vorgesehen, mit welcher die Schleifentransitfrequenz auf eine gewünschte Phasenreserve zwischen 45° und 60° verschoben

wird. Die Kapazität C_k sollte zwischen 300 fF und 1 pF betragen. Durch den pn-Übergang verfügt die Photodiode über eine Eigenkapazität, die im Idealfall bereits zur Stabilisierung des Verstärkers ausreicht. Bild 8 stellt den kaskadierten, stabilisierten und rückgekoppelten Transimpedanzverstärker dar.

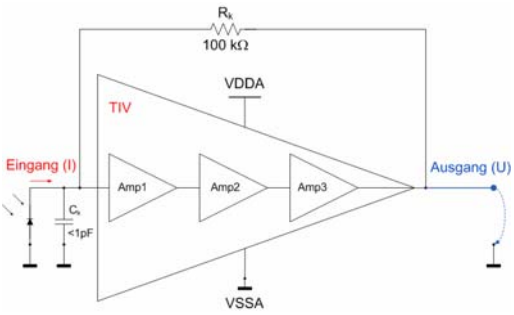


Bild 8: Kaskadierter und rückgekoppelter Verstärker

Ohne Rückkopplungswiderstand erhält man die Spannungsverstärkung

$$\underline{A} = \underline{A}_1 \cdot \underline{A}_2 \cdot \underline{A}_3$$

Mit Rückkopplungwiderstand R_k ergibt sich der Rückkopplungsfaktor $k_r = 1/R_k$ und damit die Transimpedanzverstärkung

$$\underline{A}_{ti} = \frac{\underline{A}}{1 + k_r \cdot \underline{A}}$$

$$\underline{A}_{ti} \approx \frac{1}{k_r} = R_k \text{ für } |\underline{A}| >> 1$$

Der Frequenzgang der Transimpedanz ist einmal linear und einmal logarithmisch in Bild 9 dargestellt. Die Grenzfrequenz f_{3dB} des Verstärkers beläuft sich auf 386 MHz.

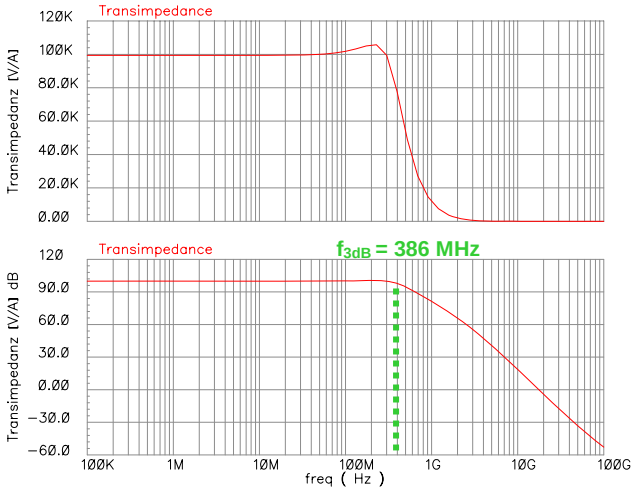


Bild 9: Transimpedanz des Verstärkers, oben im linearen Maßstab, unten im logarithmischen Maßstab

Die Kanallängen der Transistoren M_1 bis M_3 wurden aufgrund der hohen Geschwindigkeitsanforderung auf das Technologieminimum von 0,35 µm gebracht. Variationsmöglichkeiten bieten sich damit nur für die Wahl der Kanalweiten. Eine Möglichkeit, die Phasendrehung innerhalb der Regelschleife klein zu halten besteht darin, die Kanalweiten der Vorgängerstufe um einen Faktor a größer zu machen als die der Folgestufe. Ist F_0 die Fläche der dem Gesamtverstärker nachgeschalteten Stufe, so berechnet sich die Verstärker-Gesamtfläche wie folgt:

$$F_{ges} = a^3 F_0 + a^2 F_0 + a F_0$$

$$\frac{F_{ges}}{F_0} = (a^3 + a^2 + a)$$

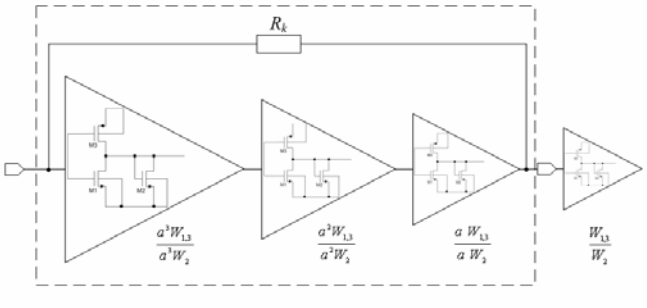


Bild 10: Größenstaffelung der Einzelverstärker

Tabelle 3 fasst die Simulationsergebnisse von 5 Varianten mit den Weiten $a = 1 \ldots 2$ zusammen. Mit zunehmenden Faktor a verringern sich die erforderlichen Kapazitätswerte C_k und die Laufzeit $t_{10/90}$. Ebenfalls verbessert sich das Rauschverhalten deutlich; dies wird jedoch mit höherer Stromaufnahme I_{dd} erkauft. Da die Stromaufnahme von besonderer Bedeutung ist, wurde letztlich die Designvariante 1 gewählt.

Tab. 3: Designvorschläge

Design #	1	2	3	4	5
a	1	1,3	1,6	1,8	2
I_{dd} [mA]	1,76	3,11	4,95	6,53	8,42
C_k [pF]	1,4	1,05	0,8	0,7	0,55
$t_{10/90}$ [ns]	1,17	0,95	0,78	0,75	0,70
$e_{n,i}$ [pA/$\sqrt{\text{Hz}}$]	13,54	7,09	4,25	3,34	2,52
$u_{r,eff}$ [mV]	12,29	8,85	6,86	5,73	4,87

3.2. Komparator

Für Hochgeschwindigkeitsanwendungen empfiehlt sich der in Bild 11 dargestellte dreistufige Aufbau [5]. Der verwendete Komparator arbeitet nicht-getaktet, also als sensing amplifier. Somit muss kein hochfrequenter Takt in den Analogteil geführt werden. Der Komparator arbeitet zudem mit einer festen Schaltschwelle, die ca. 50 mV über dem Arbeitspunkt des Transimpedanzverstärkers (ca. 1,25 V) gelegt wird. Die Einstellung dieser Schwelle erfolgt über einen von außen zugeführten Konstantstrom.

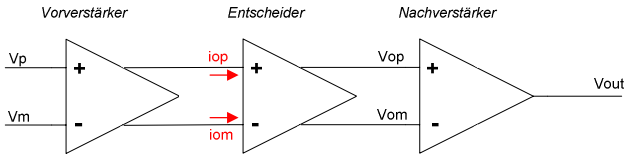


Bild 11: Dreistufiger Aufbau des Komparators

Der dreistufige Aufbau wird auch in Bild 12 deutlich, welches die dimensionierte Schaltung zeigt. Der Vorverstärker besteht aus einem Differenzverstärker, der jedoch nicht wie gewohnt mit aktiver Last arbeitet, sondern mit zwei Stromspiegeln ein Gegentakt-Stromsignal ausgibt (Transkonduktanzverstärker). Ein wesentlicher Vorteil dieser Anordnung ist der sehr geringe Miller-Effekt, da von einer Spannungsdifferenz am Eingang (V_p und V_m im Vorverstärker) zu einer Stromdifferenz (i_{op} und i_{om} im Entscheider) übergegangen wird und dies auch bei hohen Frequenzen zu einer sehr geringen Spannungsverstärkung des Vorverstärkers führt.

$$A_o = -g_{m1} \cdot r_{o,31} = -\frac{g_{m1}}{g_{m31}} = \frac{\sqrt{2 \cdot k_n' \cdot (\frac{W_1}{L_1}) \cdot \frac{I_{ss}}{2}}}{\sqrt{2 \cdot k_p' \cdot (\frac{W_{31}}{L_{31}}) \cdot \frac{I_{ss}}{2}}}$$

$$|A_o| = \sqrt{\frac{k_n' \cdot W_1}{k_p' \cdot W_{31}}} \cong \sqrt{3 \cdot \frac{W_1}{W_{31}}} \quad \text{für gleiche } L$$

Der Entscheider wandelt, unter Anwendung einer Mitkopplung über die Transistoren MN_6 und MN_7, die Stromdifferenz wieder in eine Spannungsdifferenz. Zudem entkoppelt der Entscheider den Vorverstärker vom Nachverstärker. Der sich anschließende Nachverstärker besteht aus einem selbstjustierenden Differenzverstärker, der über einen weiten Gleichtaktbereich linear arbeitet, d.h. sein interner Arbeitspunkt wird durch eine Gegenkopplung auf $V_{DD}/2$ gehalten. Der Nachverstärker bringt die Spannungsdifferenz des Entscheiders über einen abschließenden Inverter auf Logikpegel.

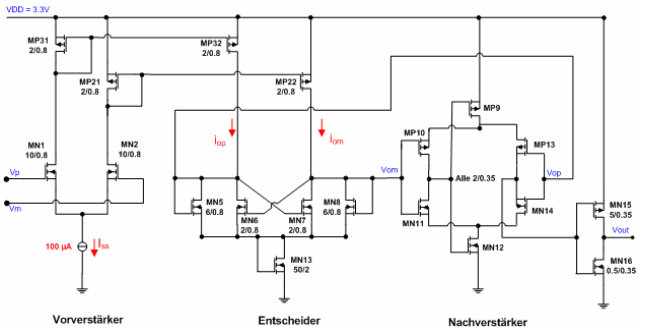


Bild 12: Dimensionierter High-Speed-Komparator

Beim Entwurf des Komparators galt es zunächst Offset- und Rauscheinflüsse gering zu halten. Hierzu wurden die Kanallängen der Analog-Transistoren auf 0,8 µm (statt 0,35 µm) gebracht. Zwischen Vorverstärker und Entscheider erhält man eine Spannungsverstärkung von ca. 50 dB. Der am Eingang überlagerte Rauschspannungseffektivwert beträgt 0,1 mV, was die Vernachlässigbarkeit gegenüber der kritischen ersten Stufe, dem Transimpedanzverstärker ($u_{r,eff} \sim 12$ mV), sichtbar werden lässt. Die Stromaufnahme des Komparators beträgt ca. 300 µA. Die absolute Verzögerungszeit, die allerdings keinen direkten Einfluss auf den Walk-Error hat, beläuft sich auf 1,4 ns. Durch geeignete Dimensionierung und Optimierung der Kanalweiten konnten die Anforderungen an den Komparator erfüllt werden. Besonders die Dimensionierung des Entscheiders war hier von Interesse. Durch Erhöhen der Kanalweiten der äußeren Transistoren M_5 und M_8 gegenüber den inneren Transistoren M_6 und M_7 verringert sich der differentielle Spannungshub V_{op} - V_{om}. Dies kommt dem Prinzip einer Clamping-Diode nahe, die bekanntlich den Spannungshub eines Gegentaktsignals reduziert (und stabilisiert). Der Walk-Error muss für Transimpedanzverstärker und Komparator gemeinsam betrachtet werden, da beide Komponenten Einfluss darauf nehmen. Auch Eingangskapazität C_k der Fotodiode beeinflusst den Walk-Error. Bild 13 stellt das Ergebnis für unterschiedliche Eingangskapazitäten dar:

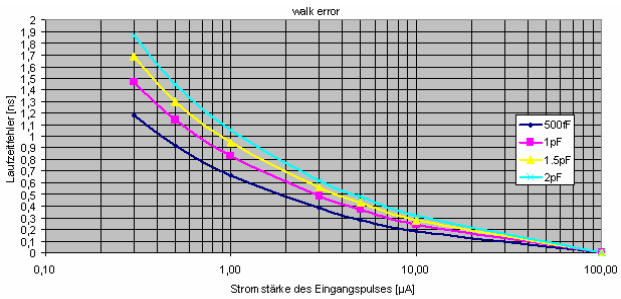


Bild 13: Walk-Error als Funktion des Eingangssignals bei unterschiedlichen Eingangskapazitäten

Interpretiert man die Kurven, so lässt sich die Aussage treffen, dass bei einer Eingangskapazität von 500 fF, was einer realistischen Größenordnung entspricht, der Walk-Error über einen Dynamikbereich von 50 dB (Faktor 330) ca. 1,2 ns aufweist. Dies gilt unter der Voraussetzung, dass das größte zu erwartende Signal 100 µA beträgt (Übersteuerung des Transimpedanzverstärkers) und das kleinste Signal, bei dem der Komparator nach schaltet, 300 nA. Die Auflösung verschlechtert sich damit im Fernbereich auf 30 cm.

3.3. Schwellenkontrolle

Der Laufzeitfehler aus Kapitel 3.2 lässt sich mit Hilfe einer zweiten, erhöhten Schaltschwelle nachträglich reduzieren. Hierzu wird ein zweiter Komparator in den Kanal eingebracht, der zur Gewinnung einer Amplitudeninformation verwendet wird. Die Amplitude darf nur in der Umgebung des vom ersten Komparator detektierten Pulses ausgewertet werden. Hierzu wird ein Verzögerungsglied von ca. 3 ns verwendet. Das Ausgangssignal walk_cor signalisiert damit mit einem HIGH das erfolgreiche Kreuzen der erhöhten Schwelle ctrlschwelle. Die Logik ist in Bild 14 dargestellt:

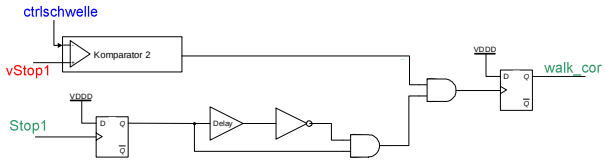


Bild 14: Logik der Schwellenkontrolle

Dabei stellt Stop1 den Digitalpuls des ersten Komparators dar, vstop1 hingegen den analogen Spannungspuls am Ausgang des Transimpedanzverstärkers. Mit dieser Information lässt sich die Fehlerkurve für 500 fF aus Bild 13 korrigieren. Die Art der Korrektur ist variabel. In diesem Beispiel wurde die ctrlschwelle so gelegt, dass walk_cor bei einem Eingangsstrompegel von 1 µA HIGH wird, unterhalb jedoch LOW bleibt. Unterhalb von 1 µA wird dann eine halbe Taktperiode (0,5/640 MHz) vom Ergebnis abgezogen. Somit erreicht man über den gesamten Dynamikbereich einen Walk-Error von ca. 620 ps. Bild 15 zeigt das korrigierte Ergebnis:

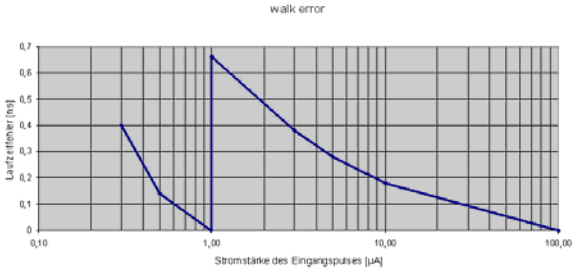


Bild 15: Korrigierter Walk-Error für $C_k = 500$ fF

3.4. Phasendetektion

Die Auflösung des LIDAR-Systems lässt sich durch Gewinnung einer Phaseninformation verdoppeln, indem festgestellt wird, ob Start- und Stop-Puls in der High- oder Low-Phase des Taktes beginnen. Da der Takt zunächst asynchron zum aus Start- und Stop-Puls generierten Fenster läuft, kann vor der Synchronisation auf HIGH- bzw. LOW-Phase des Taktes geprüft werden. So muss auch bei einer Auflösung von 15 cm auf eine Entfernung von 1 km kein zusätzliches Zählerbit eingeführt werden.

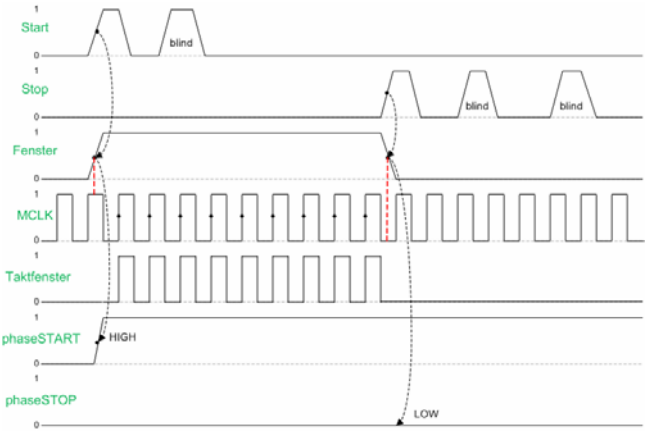


Bild 16: Prinzip der Phasendetektion

Mit den zwei zusätzlichen Signalen phaseSTART und phaseSTOP lässt sich die Auflösung mit der Korrekturvorschrift nach Tabelle 4 verbessern.

Tab. 4: Korrekturvorschrift

phaseSTART > phaseSTOP	Eine halbe Taktperiode addieren
phaseSTART < phaseSTOP	Eine halbe Taktperiode subtrahieren
phaseSTART = phaseSTOP	Keine Korrektur

Der Datenausgang eines Empfangskanals setzt sich somit aus den 12 Bits des Zählers, den beiden Ausgangssignalen der Phasendetektion und dem im vorherigen Kapitel 3.3 beschriebenen Kontrollsignal walk_cor zusammen (insgesamt 15 bit). Die gemessene Strecke d bestimmt sich im Korrekturfall nach der folgenden Beziehung:

$$d = \frac{T(N \pm 0{,}5) \cdot c}{2}$$

3.5. Taktgenerierung (DPLL)

Der interne Takt wird aus einem externen synthetisiert. Dabei wurde eine Transformation um den Faktor 32 realisiert. Die zu diesem Zweck verwendete Regelschleife (Bild 17) sieht einen digital realisierten Phasendetektor, ein Tiefpassfilter 2. Ordnung und einen spannungsgesteuerten Oszillator vor. In der Rückkopplung befindet sich ein 5-stufiger Binärteiler.

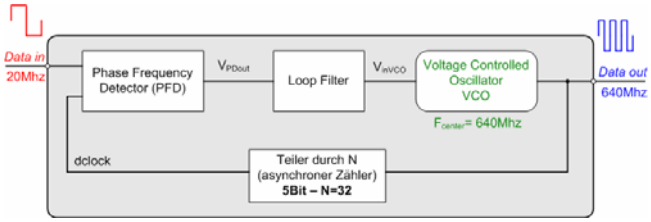


Bild 17: Frequenzsynthese mittels DPLL

Der VCO arbeitet in Source-Coupled-Architektur [5][6] nach dem Multivibrator-Prinzip (siehe Bild 18). Ausgangsseitig realisiert ein selbstjustierender Differenzverstärker eine differential-to-single-ended Wandlung. Die Mittenfrequenz des Oszillators entspricht der angestrebten Ausgangsfrequenz (640 MHz). Sie wird bei einer Eingangsspannung von $V_{DD}/2 = 1{,}65$ V erreicht. Durch die hier verwendete VCO-Architektur werden höhere Ausgangsfrequenzen bei akzeptabler Ruhestromaufnahme möglich.

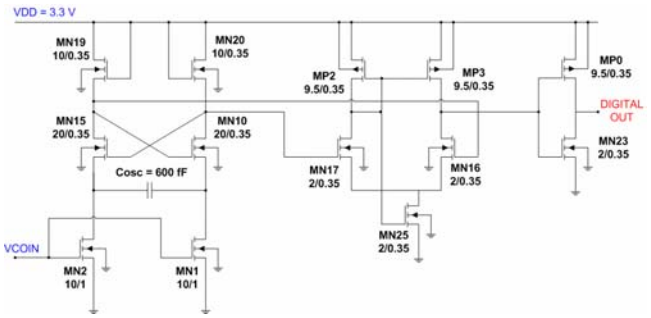


Bild 18: Integrierter VCO in source-coupled Architektur

Der frequenzbestimmende Kondensator C_{osc} wurde aus fertigungstechnischen Gründen mit 600 fF angesetzt. Die Frequenz des Rechtecks am Ausgang ist:

$$f_{osc} = \frac{I_d}{4 \cdot C_{osc} \cdot V_{th}}$$

V_{th} stellt dabei die Schwellenspannung der NMOS-Transistoren dar; sie beträgt im vorliegenden Prozess 0,6 V. I_D ist der über VCOIN eingestellte Strom in den Transistoren MN_1 und MN_2. Nach einem Einschwingvorgang von wenigen µs herrscht Phasengleichheit zwischen Ein- und Ausgangssignal. Um die Funktionalität der Regelschleife auch bei starken Streuungen der Transistorparameter gewährleisten zu können, wurden mehrere Einschwingvorgänge einer sog. Corner-Simulation unterzogen. Die in Bild 19 exemplarisch dargestellten Einschwingvorgänge sind bei 80 °C, 27 °C und -80 °C simuliert worden, wobei die Transistoren maximal schlecht beschaffen sind (worst case speed). Hier wurden verschiedene Fälle und Variationen betrachtet, z. B. die Parameterschwankungen der RC-Glieder im Schleifenfilter. Sie bestimmen die Dämpfung des Regelkreises. Spezielle Testeinrichtungen sind vorgesehen, mit welchen es möglich ist, die Funktionsfähigkeit und Robustheit der DPLL von außen zu überprüfen.

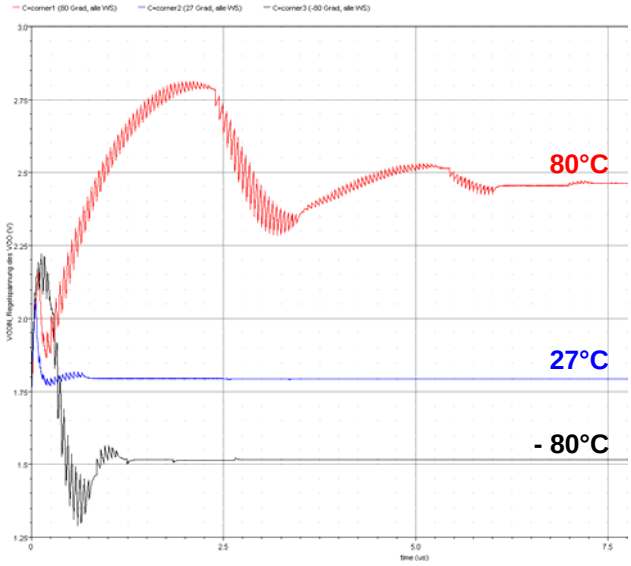


Bild 19: Corner-Simulation: DPLL-Einschwingvorgang

4. Layout

Bei der Integration kam das Cadence-Tool Virtuoso Layout und Virtuoso Layout XL zum Einsatz. Der Entwurf führte zu einer ausgeprägt länglichen Kanalstruktur. Dies erklärt sich aus der Pitchbreite von 25 µm der später einzusetzenden Photodioden. Beim Layout wurde aufgrund der relativ hohen Messfrequenz von 640 MHz auf eine strikte Trennung von Analog- und Digitalteil geachtet. Die Zusammenführung von Analog- und Digitalversorgung erfolgt off-chip. Als Checker/Extractor kamen sowohl die Tools DIVA als auch ASSURA zum Einsatz.

Der Testchip LR-1 verfügt über vier Einzelkanäle, einen Referenzkanal, einen Remote-Verstärker zur Schwelleneinstellung, eine Taktgenerierung, und eine

Ausleseschaltung samt Ausgangsbus (inkl. Pull-Down-Widerständen). Die Schaltungsstruktur gliedert sich hierarchisch, wie in Bild 20 graphisch veranschaulicht.

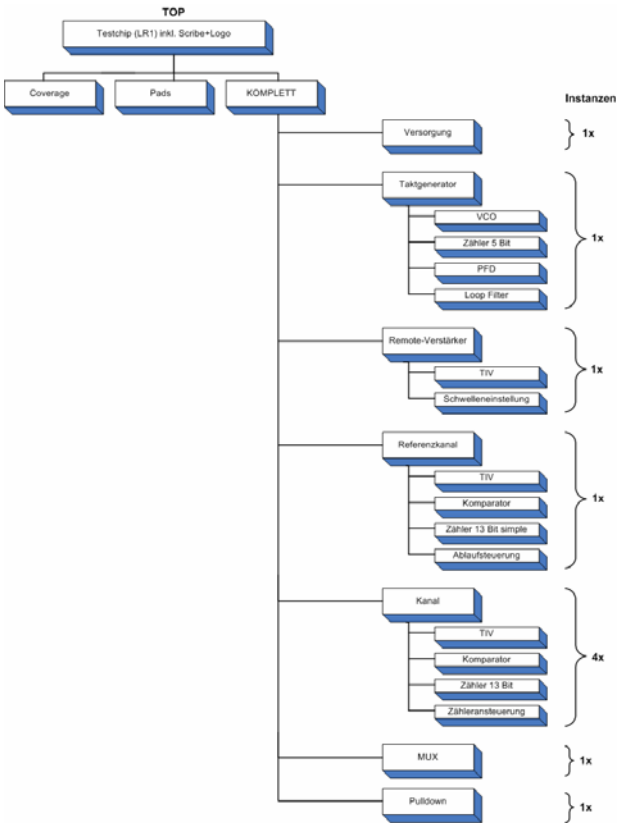


Bild 20: Hierarchische Baumstruktur des Testchips

Exemplarisch wird das fertige Zellen-Layout des in Abschnitt 3.2 präsentierten Komparators in Bild 21 dargestellt. Beim Design des Komparators wurde besonders auf Symmetrie und kurze Leitungslängen geachtet. Alle Transistoren wurden mit Guard-Ringen versehen, um das Einkoppeln von Störungen zu verhindern. Anschließend wurden auch die parasitären Kapazitäten der Leitungen gegen Substrat extrahiert, um so durch Postsimulationen die Performance sicherstellen zu können.

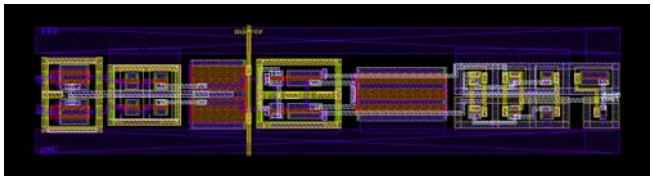


Bild 21: Layout des Komparators

In Bild 22 ist der komplette Chip dargestellt. Die Abmessungen des Testchips betragen 2150 µm * 1700 µm. Ein Empfangskanal im Core weist eine Länge von 1,2 mm auf, die Breite beträgt wie gefordert 25 µm. Das verwendete Plastikgehäuse verfügt über 48 Pins, die für eine Reihe von Mess- und Testmöglichkeiten genutzt werden. Trotz überschüssiger Fläche wurde auf den Einsatz von on-chip-Blockkondensatoren zur Glättung der Versorgung verzichtet, weil die Fläche später nicht mehr zur Verfügung stehen wird. Stattdessen wurden mehrere Power-Pads vorgesehen, um die Leitungsinduktivät der Versorgung und damit die Störsignale auf der Versorgungsspannung zu reduzieren.

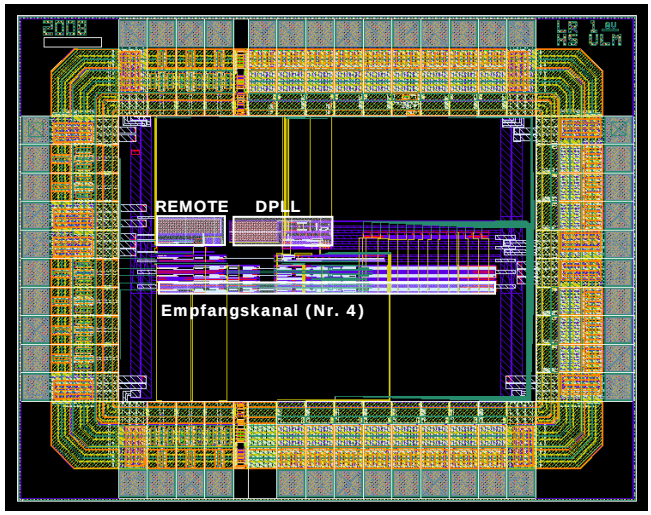


Bild 22: Layout des LR-1 Testchips (Core + Padring)

5. Zusammenfassung

Als Ergebnis dieser Arbeit wurde ein 4-kanaliger Testchip für ein integriertes Laser-Radar vorgestellt. Der Chip umfasst die komplette Empfangs- und Auswerteelektronik für ein mehrkanaliges LIDAR-System. Die Auflösung des Systems beträgt 15 cm auf eine Entfernung von 1 km. Mit dieser Machbarkeitsstudie soll die monolithische Integrierbarkeit von mindestens 64 Kanälen nachgewiesen werden. Der Testchip LR-1 wird je nach Betriebszustand eine mittlere Stromaufnahme von 15 bzw. 18 mA aufweisen. Dabei steuert der einmalig vorhandene Overhead (Taktgenerierung, Ausleseschaltung etc.) ca. 3,6 mA bei. Die Stromaufnahme eines Einzelkanals beträgt 2,1 mA. Ein als nächste Stufe des Entwicklungsvorhabens vorstellbares 64-kanaliges System müsste mit $I_{DD} = 175\ \mathrm{mA}$ bei $U_{DD} = 3{,}3\ \mathrm{V}$ realisierbar sein. Das Layout ermöglicht weiterhin eine leichte Erweiterbarkeit der Empfangskanäle. Zuvor sollen die Testchips elektrisch charakterisiert werden. Von besonderem Interesse sind die Parameter Auflösung, Rauschen, Offset und Arbeitspunkte des Systems. Ebenfalls sollen die Robustheit der internen DPLL und das Kanalübersprechen zwischen zwei benachbarten Kanälen untersucht werden. Zu diesem Zweck sind zahlreiche Mess- und Testpins vorgesehen.

6. Literatur

[1] P. Palojärvi, “Integrated Electronic and Opto-electronic Circuits and Devices for Pulsed Time-of-Flight Laser Rangefinding”, University of Oulu 2003

[2] T. Ruotsalainen, “Integrated Receiver Channel Circuits and Structures for a Pulsed Time-of-Flight Laser Radar”, University of Oulu 1999

[3] M. Ingels, M. Steyaert, “Integrated CMOS Circuits for Optical Communications”, Springer-Verlag, 2004 (ISBN: 3-540-20209-9)

[4] P. Gray, P. Hurst, S. Lewis, R. Meyer, “Analysis and Design of Analog Integrated Circuits”, Fourth Edition, John Wiley & Sons Inc. (ISBN: 9-780471-37752)

[5] R. Baker, “CMOS: Circuit, Layout and Simulation”, Second Edition, John Wiley & Sons, Inc. (ISBN: 0-471-70055-X)

Danksagung

Die Autoren bedanken sich bei Herrn Dipl.-Ing. (FH) F. Mrugalla, Herrn E. Ringwald und Herrn Dipl.-Ing. (FH) J. Schäfer für die Unterstützung bei der Anwendung der Tools. Weiterer Dank gilt Herrn Dr. rer. nat. K. Hofmann und Herrn Dipl.-Ing. (FH) A. Erni von der Firma AIM für nützliche Hinweise zu typischen Systemanforderungen.

Das Projekt wurde gefördert aus Mitteln der baden-württembergischen Multiprojekt-Chip-Gruppe.

Von funktionalen Programmen zur Hardwarebeschreibung Digitaler Filter

Tobias Häberlein, Matthias Brettschneider
Studiengang Kommunikations- und Softwaretechnik, HS Albstadt - Sigmaringen

23. April 2008

Abstract

Wir zeigen, dass digitale Filter sehr einfach und mit Hilfe der funktionalen Programmiersprache Haskell (bzw. ein auf den Entwurf von digitalen Schaltungen zugeschnittenes Haskell-Modul namens Lava [2]) spezifiziert und modelliert werden können. Als Beispiel wählen wir den Entwurf eines FIR-Filters.

1 Einführung

Anders als herkömmliche Hardwarebeschreibungssprachen, wie VHDL und Verilog, bietet Haskell das volle Repertoire an Abstraktionsmechanismen: Funktionen höherer Ordnung, Referentielle Transparenz und ein ausgereiftes statisches Typsystem. Funktionen höherer Ordnung dienen als Modelle für Schaltkreise, die referentielle Transparenz bietet die Möglichkeit mit Programmen bzw. Schaltkreisen "rechnen" zu können und sie von der TLM-Ebene automatisch (oder halb-automatisch) in die RTL- oder gar Gatter-Ebene zu transformieren und das ausgereifte statische Typsystem zwingt den Hardwaredesigner auch auf der TLM-Ebene das richtige Abstraktionsniveau zu halten.

Ein in gewissem Sinne paralleler Ansatz wurde schon in der Welt der prozeduralen Programmiersprachen durch die Entwicklung von System-C gegangen [8, 7]. Das, was System-C zu C++ ist, ist Lava zu Haskell; nämlich die Erweiterung einer bestehenden Software-Programmiersprache um Klassen, die die Hardwarebeschreibung ermöglichen.
Unterschied: der Weg einer funktionalen Programmiersprache zu einer Hardwarebeschreibungssprache ist sehr viel natürlicher als der Weg einer prozeduralen Sprache in Richtung des Hardwareentwurfs [11]. So benötigt System-C eine verhältnismäßig komplex programmierte Laufzeitumgebung, mit der nebenläufge Threads simuliert werden können, während Haskell bzw. dessen Modul Lava dies nicht benötigt: Dadurch, dass die Funktionale Programmierung von dem Kontrollfluss abstrahiert (erst der Compiler ordnet den Funktionsauswertungen eine Reihenfolge zu; der Programmierer muss sich darum nicht kümmern) ist Nebenläufigkeit nichts das simuliert zu werden braucht, sondern ist während der Programmierung (nicht aber unbedingt während der Ausführung) allgegenwärtig.

2 Funktionale Programmierung

Der Begriff funktionale Programmierung bezeichnet ein Programmierparadigma. Anders als bei prozedual/imperativen Sprachen besteht ein Programm nur aus Funktionen. Eine Funktion im funktionalem Sinne unterscheidet sich allerdings stark von der prozedualen Vorstellung einer Funktion. Denkt man prozedual so ist eine Funktion ja nichts anderes als ein Bezeichner für eine Reihe von Anweisungen sowie deren Reihenfolge. Außerdem hängen sie meist nicht nur von ihren Übergabewerten, sondern auch von dem Zustand der unterliegenden Maschine ab.

Die "funktionale" Funktion entspricht eher der rein mathematischen Funktion: Sie ist die bloße Abbildung von Eingabedaten auf Ausgabedaten; entsprechend hängt eine Funktion nur von ihren Übergabeparametern ab, nicht von Zuständen der aufrundenden Programme oder der Maschine. Das bringt eine Reihe von Vorteilen mit sich: Die Zeit, zu der eine Funktion ausgewertet wird, spielt keine Rolle. Egal wann die Funktion ausgeführt wird, sie liefert immer das selbe Ergebnis, da dieses ja per Definition nur von den Parametern abhängt. Eine Variante die hier möglich wird, ist die Bedarfsauswertung (lazy-evaluation). Ein Ausdruck wird immer nur

dann ausgewertet, wenn es auch notwendig wird. Das wiederum gibt uns die Möglichkeit unendliche Strukturen zu definieren:

```
fibs :: [Integer]
fibs = fibs' 0
   where fibs' n = (fib n):(fibs' $ n+1)
```

Listing 1: Unendliche Liste, die Fibonacci-Folge

Listing 1 zeigt, wie man eine solche unendliche Folge definieren kann. Hier die unendliche Reihe der Fibonaccizahlen. *fibs* wird nur bei Bedarf ausgewertet und dann auch nur soweit es nötig ist. Mit *take 10 fibs* würde man z.B. die Liste der ersten 10 Fibonaccizahlen erhalten.

Funktionalen Programmen fehlen Variablen. Dies ist die konsequente Fortführung des mathematischen Funktionsbegriffs. Eine mathematische Funktion wie beispielsweise das Polynom $f(x)_n = a_n x^n + a_{n-1}x^{n-1} + ... + a_1 x + a_0$ besitzt ebenfalls keine Variablen. Es gibt lediglich die Übergabeparameter von denen das Ergebnis abhängt, also x und n. Normalerweise sind beide Parameter zum Aufrufzeitpunkt bekannt. Falls nicht, lassen funktionale Sprachen auch eine Teilauswertung zu und das Ergebniss wird in Abhängigkeit von dem fehlenden Parameter angegeben, ist also auch wieder eine Funktion. Angenommen n sei 3 und x weiterhin offen, so ergibt sich $f(x)_3 = a_3x^3 + a_2x^2 + a_1x + a_0$ — eine neue Funktion.

Eine Funktion der man nicht alle erwarteten Parameter übergibt heißt *unterversorgt*. Den in C häufig verwendeten *++*–Operator könnte man sich in der funktionalen Programmierung als eine spezielle Version des "normalen" *+* vorstellen. *+* erwartet zwei Parameter. Füttert man *+* nun nur mit einem Parameter (mit *1*, um den *++*–Operator zu erhalten) so ist der Rückgabewert wieder eine Funktion; eine Funktion, die den Übergabewert um eins inkrementiert.

Der vorhergehende Abschnitt führt ein weiteres Feature der funktionalen Programmierung ein. Funktionen können höherer Ordnung ("higher order") sein [9]. Das bedeutet, Funktionen können sowohl als Parameter an andere Funktionen übergeben werden als auch der Rückgabewert einer Funktion sein.

3 Lava

3.1 Allgemeines

> Lava is a tool to assist circuit designers in specifying, designing, verifying and implementing hardware. [2]

Claessen selbst beschreibt Lava nur zu treffend. Es handelt sich dabei um ein Werkzeug für Hardwareentwickler, um Schaltkreise zu spezifizieren, zu designen, sie zu verifizieren und letztlich zu implementieren.

Lava selbst ist ein Modul für die Funktionale Programmiersprache Haskell (Siehe [2]). Lava besteht aus einer Ansammlung von Funktionen und Datentypen. Diese geben dem Anwender die Möglichkeit Schaltkreise zu definieren. Eine weitere Möglichkeit die Lava bietet ist es, Schaltkreise symbolisch auswerten zu lassen. Die Symbolische Auswertung benötigt man spätestens dann, wenn man seine Schaltungen von Lava interpretieren lassen möchte. Eine "Interpretation" macht es möglich, eine Lava-Schaltung in eine VHDL Beschreibung umzusetzen. Außerdem ist es mit Lava auch möglich, Schaltkreise mittels eines "theorem prover" verifizieren zu lassen.

3.2 Funktionen und Datentypen

Ein Schaltkreis in Lava entspricht einer Funktion in Haskell, die auf Signale definiert ist. Zum einen gibt es Funktionen, die keine Eingabewerte übergeben bekommen. Solche Funktionen stellen Konstanten dar. Ein Bauteil, dass konstant "True" ausgibt entspricht einer solchen konstanten Funktion. Zum anderen gibt es Funktion denen man ein oder mehrere Signale übergeben kann. Diese Signale werden dann bearbeitet und das Resultat wird als Ergebnis zurückgeliefert. Eine solche Funktion würde man typischerweise als Schaltkreis/Schaltung bezeichnen. Als Beispiel kann man sich einen Halbaddierer vorstellen. Wie in Abbildung 1 zu sehen ist, besteht ein

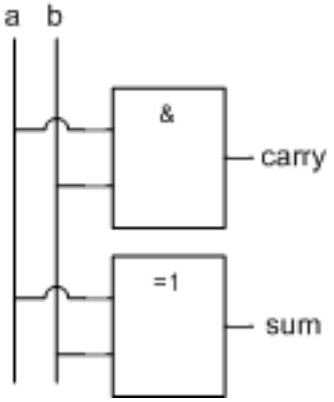


Abbildung 1: Halbaddierer

Halbaddierer aus zwei Gattern, nämlich einem "and"- und einem "xor"-Gatter. An beide Gatter werden jetzt jeweils beide Eingangssignale angelegt. Am Ausgang des "xor"-Gatters liegt die Summe (sum) an und am Ausgang des "and"-Gatters liegt der Übertrag (carry) an.

Die Implementierung des Halbaddierers in Lava ist im Listing 2 gezeigt [4, page 6].

```
halfAdd :: (Signal Bool, Signal Bool) ->
     (Signal Bool, Signal Bool)
halfAdd (a, b) = (sum, carry)
  where
  sum   = xor2 (a, b)
  carry = and2 (a, b)
```

Listing 2: Halbaddierer

Man sieht, dass der Halbaddierer tatsächlich eine Funktion ist, die Signale übergeben bekommt und Signale zurück liefert.

Die beiden Funktionen *xor2* und *and2* werden durch Lava bereitgestellt. Neben diesen zwei Funktionen bietet Lava noch eine Fülle von weiteren Funktionen (siehe [4, page 73 ff.]). Darunter findet man Logikfunktionen, arithmetische Funktionen, Verbindungsmuster wie Spalten-, Reihen- oder Netzverschaltungen oder Funktionen um den Code z.B. als VHDL interpretieren zu lassen.

Die Frage nach der Bedeutung des Suffix "2" an den Funktionsnamen bringt uns zu den Datentypen die Lava anbietet. Um es vorweg zu nehmen, die 2 an der Funktion sagt aus, dass es sich um ein (zweistelliges) binäres Gatter handelt. Es gibt auch nullstellige, sowie einstellige Gatter. Von den nullstelligen Gattern gibt es genau zwei in Lava, nämlich *low* und *high*, was den booleschen Werten "wahr" und "falsch" entspricht.

```
low, high :: Signal Bool
```

Listing 3: Typisierung von low und high

Nullstellige Funktionen sind Konstanten; im obigen Fall vom Typ *Signal Bool*.

Der Signal-Typ ist der wichtigste Datentyp in Lava. Lava selbst bringt zwei Ausprägungen dieses Typs mit, nämlich *Signal Bool* und *Signal Int*. Sämtliche Funktionen in Lava sind auf diesen Typ zugeschnitten. Ein *Signal Bool* kann man sich einfach als einen Draht vorstellen der zu einem Schaltkreis führt. So hat auch der Halbaddierer (siehe Abbildung 1) zwei Drähte die eingehen, und zwei Drähte die ausgehen. Genau das findet man in der Funktionsdefinition (siehe Listing 2) auch wieder.

Der Typ *Signal Int* ist nichts anderes, als die Zusammenfassung mehrerer boolescher Eingänge, also mehrerer *Signal Bool*-Eingänge. Einer Funktion die einen *Signal Int* Wert erwartet kann man einfach die Zahl 42 übergeben. Das wiederum bedeutet, dass die Zahl 42 den Typ: *42 :: Signal Int* hat.

Die Typen die Lava mitbringt lassen sich nach haskellüblicher Manier in Listen verpacken oder strukturieren. Hat man sie einmal in der bekannten Form, kann man die gesamte Eleganz funktionaler Sprachen verwenden, um Schaltungen zu beschreiben.

3.3 Interpretation

Allgemein

Eine der interessantesten Eigenschaften von Lava ist die Fähigkeit, das funktionale Modell der Schaltkreise auf verschiedene Weisen interpretieren zu können. Durch die Interpretation des Modells hat man die Möglichkeit, VHDL-Code zu erzeugen, das Modell zu simulieren oder es zu verifizieren. Wir wollen uns hier mit der Simulation funktionaler Modelle beschäftigen, sowie auch mit deren Verifiaktion und zuletzt kurz mit der Generierung von VHDL-Code.

Simulation

Eine Simulation der modellierten Schaltkreise ist wichtig, um das Modell gegen die eigenen Vorstellungen prüfen zu können. Auch hierfür liefert Lava die passenden Werkzeuge. Simulation bedeutet in Lava, dass man einen Schaltkreis mit bestimmten Eingabewerten aufruft. Wobei aufrufen hier nicht das richtige Wort ist. Ein Schaltkreis in Lava ist nämlich aus sich selbst nicht "lauffähig", so wie reiner VHDL Code auch aus sich selbst nicht lauffähig ist. Was bei VHDL der FPGA ist, ist in Lava die Simulationsfunktion. Erst mit ihrer Hilfe kann das funktionale Modell beobachtet werden. Die Funktion *simulateSeq* macht genau dies möglich [4].

simulateSeq bekommt den zu simulierenden Schaltkreis — der ja nichts anderes ist als eine Funktion — als ersten Übergabeparameter übergeben. Der zweite Parameter ist die Liste der Eingabewerte, mit denen die Funktion simuliert werden soll. Das Ergebnis ist dann wieder eine Liste, nämlich die Liste der Ausgabewerte, die der Schalkreis produziert hat.

Es gibt noch die Möglichkeit, den Schaltkreis nicht direkt mit Eingabwerten zu füttern, sondern indirekt. Lava bringt dazu eine besondere Liste mit, die Liste *domain*. Das Besondere an *domain* ist, dass diese Liste immer alle möglichen Eingabewerte beinhaltet. Der Schaltkreis der simuliert werden soll besitzt einen bestimmten Typ. In Abhängigkeit dieses Typs ist es möglich eine Liste zu erstellen, die alle möglichen Eingabewerte für den Schaltkreis beinhaltet. Man sollte sich im klaren darüber sein, dass diese Liste nicht endlich sein muss. So hat ein Schaltkreis der auf *Signal Int* definiert ist eine unendliche Liste als *domain*.

Verifikation

Ein Bonbon das Lava noch mitbringt ist, den Schaltkreis verifizieren zu können. Eine Verifikation setzt vorraus, dass man seinem Schaltkreis bestimmte Eigenschaften zuordnen kann. Diese Eigenschaften kann man in einer neuen Funktion verpacken. Ein Schaltkreis erfüllt die Eigenschaften genau dann, wenn die neue Funktion für alle Parameter *true* zurück gibt. Lava bietet nun die Möglichkeit an, genau das zu verifizieren. Dazu wird aus der Funktion eine mathematische Formel erstellt. Diese Formel wird dann an einen externen "theorem prover" übergeben. Der "theorem prover" bestätigt dann die Eigenschaft des Schaltkreises oder auch nicht.

Je komplexer eine zu verifizierende Funktion ist, umso komplexer wird auch die mathematische Beschreibung der selben. Es ist daher sinnvoll die Beweisfunktionen so speziell wie möglich zu halten. Dadurch hat man mehr Funktionen, die dann wiederum die Eigenschaften des Schaltkreises beschreiben. Der Vorteil liegt darin, dass man dem "theorem prover" seine Arbeit deutlich erleichtern kann.

Wir wollen hier kurz zeigen wie eine solche Verifikation funktioniert. Das hier gezeigte Beispiel ist wieder aus dem "Lava Tutorial" [4]

Angenommen wir wollen verifizieren dass der Halbaddierer (siehe Listing 2) das tut was er soll. Sollte er funktionieren, so dürfen — für alle Eingabewerte — niemals *carry* und *sum* gleichzeitig *true* sein. Diese Eigenschaft kann man so definieren: [4]

```
prop_HalfAddOutNeverBothTrue ::
  (Signal Bool, Signal Bool) -> Bool

prop_HalfAddOutNeverBothTrue (a, b) = ok
  where
  (sum, carry) = halfAdd (a, b)
  ok           = inv (and2 (sum, carry))
```

Listing 4: Halbaddiererausgänge nie beide wahr

Man sieht, dass die Funktion *and2* nur dann wahr werden kann, wenn der Halbaddierer nicht richtig funktioniert. Das heißt konkret, der Halbaddierer muss an beiden Ausgängen *true* ausgeben, damit unsere Beweisfunktion *false* ausgibt.

In Lava kann man diesen Beweis führen, indem man der Funktion *verify* unsere Beweisfunktion übergibt.
verify prop_HalfAddOutNeverBothTrue
Kann der "theorem prover" zeigen, dass die Formel immer richtig ist, hat er sie also bewiesen, so meldet er es Lava zurück, und wir bekommen von Lava die Ausgabe, dass unser Schaltkreis "valid" ist.

VHDL Generierung

Hat man seine Schaltkreise in Lava definiert, so kann man diese Definition als VHDL-Code interpretieren lassen. Für Lava ist das kein Mehraufwand. Die Verifikation und auch die symbolische Auswertung unterscheiden sich von der VHDL-Codeerzeugung aus der Sicht von Lava nicht. Lava stellt mehrere Funktionen zur Verfügung, um VHDL-Code zu erzeugen.

- *writeVhdl* bietet die Möglichkeit Schaltkreise in VHDL-Code zu überführen und diesen in einer Datei zu speichern
- *writeVhdlInput* arbeitet genau wie *writeVhdl*, allerdings kann man hier den Namen für die Eingabewerte festlegen
- *writeVhdlInputOutput* kann zusätzlich zur Funktionalität von *writeVhdlInput* auch noch Namen für Ausgabewerte vergeben

4 Digitale Filter

Unter einem Filter versteht man ein Bauteil, das mathematische Operationen auf Signale ausführt. Ein Filter ist nichts anderes als eine elektrische Schaltung. Diese elektrische Schaltung verändert ein Eingabesignal in ein Ausgabesignal. Es können so Frequenzbereiche ausgefiltert werden, oder andere verstärkt werden. Filter sind ein fundamentaler Bereich der Signalverarbeitung und ein fester Bestandteil unseres täglichen Lebens. Als Beispiele seien hier nur kurz die Stereoanlage, der Satelitenreceiver oder der DSL-Splitter genannt. Wir wollen exemplarisch zeigen wie man mit Hilfe von Lava, Filter mit endlicher Impulsantwort modelliert, kurz FIR-Filter.

4.1 Signalverarbeitung

Die digitale Signalverarbeitung ist das Gegenstück zur analogen Signalverarbeitung. Signalverarbeitung digital zu betreiben hat eine Riehe von Vorteilen gegenüber der analogen Variante. So müssen Filter nicht aus passiven analogen Bauteilen, wie z.B. Widerständen, Spulen oder Kondensatoren erstellt werden. Man kann seine Filter mit ICs oder FPGAs modellieren. Mit digitalen Filtern hat man die Möglichkeit, quasi jedes Filterverhalten zu erreichen, dass man als mathematische Gleichung oder als Algorithmus darstellen kann.

Bei einem digitalem Filter sind die Bauteile, die mathematischen Operationen sowie die Signale digital. Die digitalität eines Signales bedeutet, dass es:

- einen diskreten Zeitbereich hat, d.h. es liegen nur zu definierten Zeiten Werte vor. Aus einem kontinuierlichem Signal wird also nur zu bestimmten Zeitpunkten der Wert gemessen (z.B. alle 100 ms).
- einen diskreten Wertebereich hat, d.h. es gibt nur bestimmte Werte, z.B. die natürlichen Zahlen $\mathbb{N}$. Alle Werte die zwischen zwei natürlichen Zahlen liegen, werden z.B. gerundet.

Dass ein Signal im Werte-, sowie im Zeitbereich diskret ist, hat zur Folge, dass seine Kurve nicht mehr "schön rund", also nicht mehr kontinuierlich ist. Vielmehr ähnelt ein digitales Signal Treppenstufen (vgl. Abbildung 2). Will man aber ein Signal speichern oder in

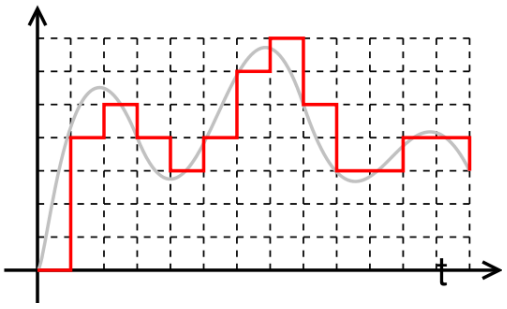


Abbildung 2: grau: kontinuierlich, rot: diskret

irgend einer Weise mit einem Computer erfassen, muss dieses Signal digitalisiert werden, da Computer ja bekanntlich nichts kontinuierliches verarbeiten können.

Natürlich hat die digitale Betrachtung von Signalen nicht nur Vorteile, sondern auch Nachteile. So geht bei der Wandelung eines analogen Signals in ein digitales Signal immer Information verloren. In Abbildung 2 sieht man das sehr schön. Natürlich enthält die graue Kurve mehr Informationen als die rote "Treppe". Man nennt diesen Informationsverlust den Quantisierungsfehler. Je ungenauer das kontinuierliche Signal in ein digitales Signal überführt wird, um so stärker ist das Quantisierungsrauschen. Der Signal-Rausch-Abstand sinkt.

4.2 Filter mit endlicher Impulsantwort

Allgemein

Ein "Filter mit endlicher Impulsantwort" oder auch FIR-Filter, ist ein nicht rekursiver Filter. Dieser Filter berechnet sein Ergebnis anhand eines Eingangswertes und einer endlichen Anzahl von vorhergehenden Eingangswerten. Die Ergebnisse selber fließen aber niemals in die Berechnung mit ein, was ihm zum nicht rekursiven Filter macht. Mathematisch kann man einen FIR-Filter mit folgender Gleichung beschreiben:

$$y(n) = \sum_{q=0}^{Q} b_q * x(n-q) \tag{1}$$

Jeder Eingangswert wird mit einem Faktor b_q gewichtet. Die Impulsantwort eines FIR-Filters hat immer eine endliche Länge. Hat der Filter die Ordnung Q, so hat seine Antwort auf einen Dirac-Impuls $\delta(n)$ die Länge $Q+1$ [6]. Ausserdem ist die Antwort des Systems auf einen Dirac-Impuls gleich den Filterkoeffizienten b_q.

Funktinales Modell eines FIR-Filters

Wir wollen jetzt den FIR-Filter in ein "Funktionales Modell" bringen. Das bedeutet, dass wir den FIR-Filter einmal mit Hilfe von Lava beschreiben müssen. Haben wir diesen Schritt geschafft, so kann man dieses Modell, nach den Möglichkeiten von Lava, interpretieren. Im weiteren wollen wir beschreiben, wie man diesen Schritt mit Hilfe von Lava geht. Der Begriff "Signal" bezeichnet im Folgenden ein Signal im Sinne von Lava. Ein Integersignal ist ein "Signal Int" im Sinne von Lava und unterscheidet sich von einem Int auf den ersten Blick nicht. [1]

Ein FIR-Filter ist in einem "Funktionalen Modell" nichts anderes, als eine Funktion die auf Daten arbeitet. Diese Daten sind zum einen die Filterkoeffizienten und zum anderen die Signale auf denen der Filter arbeitet. Wir gehen hier davon aus, dass es sich dabei jeweils um ganzzahlige Werte handelt.

```
type Values = [Signal Int]
type Coefficients = Values
```

Listing 5: Typdefinitionen

Wir definieren uns zwei neue Typen[2] *Values* und *Coefficients*.

- *Values* ist eine Liste, die aus Signalen besteht. Genauer, aus Signalen vom Typ Int. Dies soll das tatsächliche Signal darstellen, auf dem der FIR-Filter später arbeitet.
- *Coefficients* ist die Liste der Filterkoeffizienten. Diese sind nichts anderes als eine Liste, welche Integersignale beinhaltet. Koeffizienten sind also auch Werte (*Values*), die aber eine besondere Bedeutung haben. Dies rechtfertigt daher den eigenen Typ (*Coefficients*).

[1] Wer sich für Lavas Signal-Typ interesiert, findet dazu in Claessens Thesis mehr [3, Appendix C, Page 147]
[2] Streng genommen definiert man mit *type* ein Alias.

Als nächstes benötigen wir zwei Hilfsfunktionen. Anhand der Gleichung (1) sieht man, dass wir eine Funktion benötigen, die Signale miteinander multipliziert, sowie eine weitere Funktion, die eine Liste von Signalen aufsummiert.

```
ltimes :: Values -> Values -> Values
ltimes = zipWith (curry times)
```

Listing 6: Hilfsfunktion ltimes

Mit *ltimes* bilden wir die Multiplikation der Gleichung (1) ab. *ltimes* bekommt zwei *Values*, also zwei Listen von Integersignalen, übergeben. *zipWith*, welche übrigens eine Funktion ist, die im Haskell-Prelude vorhanden ist, verknüpft die beiden Listen mit Hilfe von *times*. *times* wiederum ist eine Funktion die von Lava bereitgestellt wird. Der Typ von *times* und der Typ, den *zipWith* anbietet stimmen nun nicht ganz überein. *times* erwartet die beiden Integersignale, die miteinander multipliziert werden, als strukturiertes Argument[3]. Die Funktion *curry* nimmt nun eine Funktion entgegen, die ein strukturiertes Argument erwartet und gibt eine Funktion zurück, die "gecurried" ist.

```
lsum :: Values -> Signal Int
lsum = foldl (curry plus) 0
```

Listing 7: Hilfsfunktion lsum

lsum ist die Funktion, die das Summenzeichen der Formel (1) abbildet. *lsum* bekommt eine Liste von Integersignalen übergeben, addiert diese zusammen und macht daraus genau ein Integersignal. Auch hier ist der Kern wieder eine Funktion aus dem Lava-Modul, nämlich *plus*. Und auch *plus* bringen wir in die curried Version damit sie mit *foldl* zusammenarbeiten kann. *foldl* ersetzt (anschaulich) die Trennzeichen einer Liste mit der übergeben Funktion sowie die leere Liste mit dem zweite Parameter (hier 0). So wird aus *[1, 2, 3, 4]*: *((((0 'plus' 1) 'plus' 2) 'plus' 3) 'plus' 4)*. Das *l* steht für "left" und besagt von wo geklammert wird.

Wir haben jetzt die Hilfsfunktionen definiert und können uns nun Gedanken darüber machen, wie wir die Faltungungsfunktion, die wir *conv* nennen, umsetzten. Als erstes überlegen wir uns den Typ, den *conv* haben wird. *conv* wird ein Signal mit Hilfe von Koeffizienten zu einem Ausgangangssignal "zusammenfalten". Dabei werden zusätzlich Signale betrachtet, die schon in der Vergangenheit liegen. Um diese Signale in die Berechnung mit einfließen zu lassen, müssen wir *conv* solange rekursiv aufrufen, bis alle vergangen Signale eingerechnet sind. Das Ende der Rekursion bestimmen wir mit Hilfe eines Zählers. Der Typ von *conv* sieht daher so aus: *conv :: Int →Coefficients →Values →Values*.

Im nächsten Schritt überlegen wir uns, wie *conv* intern arbeiten muss, um die Anforderungen zu erfüllen. Die Faltung lässt sich anschaulich mittels zwei Papierstreifen erklären. Die Papierstreifen werden aneinander vorbei geschoben, so dass die Zahlen immer übereinander stehen. Dann multipliziert man die jeweils untereinanderstehenden Zahlen zusammen. Die Produkte werden aufsummiert und ergeben den ersten Wert der Faltung (hier 3). Jetzt wird der untere Streifen eine Zahl weiter gerückt, wieder werden die Produkte gebildet und daraus die Summe berechnet. So fährt man solange fort, bis man alle Werte berechnet hat. Unsere Funktion muss das Multiplizieren und Summieren im obigen Beispiel genau 9 mal durchführen. Die Anzahl der Durchläufe ergibt sich aus der Anzahl der Koeffizienten plus der Anzahl der Werte minus eins. (*n = (length bs) + (length xs) - 1*) [4]

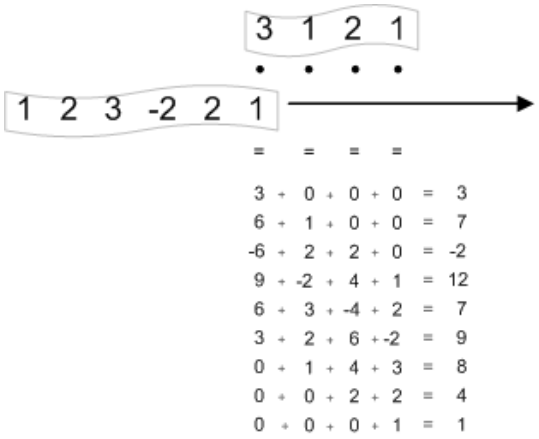


Abbildung 3: Faltung mittels Papierstreifenmethode [6]

Das erste Ergebnis der Faltung aus Abbildung 3 ist 3. Die 3 kommt daher, dass sich nur ein Zahlenpaar deckt. Alle anderen Werte werden also mit 0 multipliziert. Um dieses Verhalten in Lava nachzubilden, füllen wir einfach die Koeffizientenliste Vorne mit Nullen und die Werteliste Hinten mit Nullen auf. Dabei nehmen wir jeweils eine Null weniger als die jeweils andere Liste Elemente enthält. Die Werteliste wird außerdem noch umgedreht.
(*bs' = (replicate (length xs - 1) 0) ++ bs*)
(*xs' = (reverse xs) ++ (replicate (length bs - 1) 0)*)

Da wir jetzt zwei gleich große Listen haben können wir diese einfach mittels unserer Hilfsfunktion *ltimes* zusammenmultiplizieren und die Produkte durch *lsum* aufsummieren. Das Ergebnis ist der erste Ergebniswert unserer

[3] Von strukturierten Argumenten spricht man dann, wenn die Funktion einen Typ der Form *(a, b) →c* besitzt. Es müssen also beide Werte (a und b) vorhanden sein. Das Pendant dazu ist eine Curried-Funktion. Diese hat einen Typ der Form *a →b →c*

[4] *b* steht für einen Koeffizient und *x* steht für einen Wert. Das "s" symbolisiert dass es sich um mehrere b (oder x) handelt.

Faltungsfunktion *conv*.
(*y = lsum (ltimes bs' xs')*)

Jetzt fehlt nur noch ein Mechanismus, der uns das Weiterschieben des Papierstreifens nachbildet. Hierzu bedienen wir uns der Lavafunktion *delayInt*. Diese Funktion simuliert eine Verzögerung. Der erste Parameter von *delayInt* ist das Signal mit dem unsere Liste von Vorne aufgefüllt wird. Der zweite Parameter ist das erste Element unserer Liste. An das Ergebnis von *delayInt* hängen wir noch die restliche Liste an, wobei wir den letzten Wert vergessen. Damit ist sichergestellt, dass die Liste immer gleich lang bleibt. Die Restliste kann man so beschreiben: (*dxs'_rst = (reverse.tail.reverse) xs*)
Und das Weiterscheiben selber setzt man so um:
(*dxs' = (delayInt (int 0) (head xs)) : dxs'_rst*)

Jetzt haben wir alle Stücke zusammen und können die Funktion *conv* ein erstes mal definieren:

```
conv :: Int -> Coefficients -> Values -> Values
conv 0 _  _  = []
conv n bs xs = y : (conv (n-1) bs' dxs')
where
 bs'  = (replicate (length xs - 1) 0) ++ bs
 xs'  = (reverse xs) ++
        (replicate (length bs - 1) 0)
   y = lsum (ltimes bs' xs')
 dxs' = (delayInt (int 0) (head xs)) : dxs'_rst
 dxs'_rst = (reverse.tail.reverse) xs
```

Listing 8: Eine erste Version von conv

Das Ganze kann man aber noch verbessern. Der Aufruf unserer Faltungsfunktion sollte so funktionieren, dass man nur die Koeffizienten sowie die Liste der Signale übergibt z.B. so: *conv [3, -1, 2, 1] [1, 2, 3, -2, 2, 1].* Um dieses Verhalten zu erreichen müssen wir den Zähler *n* "verstecken". Wir definieren uns also zwei Versionen von *conv*. Die eine stellt das "Interface" zur Außenwelt dar. Unser "Interface" bekommt nur die Koeffizienten sowie die Werte übergeben. Die beiden Listen werden dann mit der richtigen Anzahl von Nullen gefüllt und die Anzahl der Iterationen wird bestimmt. All diese Werte werden jetzt an *conv'* übergeben, die dann die eigentliche Faltung durchführt. Das Ergebnis von *conv* ist das Ergebnis von *conv'*. Hier der Quellcode der überarbeiteten Variante von *conv*:

```
conv' ::Int -> Coefficients -> Values -> Values
conv' 0 _  _  = []
conv' n bs xs = y : (conv' (n-1) bs xs')
 where y = lsum (ltimes bs xs)
  xs'    = (delayInt (int 0) (head xs)):xs_rst
  xs_rst = (reverse.tail.reverse) xs
```

—

```
conv :: Coefficients -> Values -> Values
conv bs xs = conv' n bs' xs'
 where
  bs' = (replicate (length xs - 1) 0) ++ bs
  xs' = (reverse xs) ++
        (replicate (length bs - 1) 0)
  n   = length xs'
```

Listing 9: Verbesserte Version von conv

Diese wenigen Zeilen Quellcode stellen den gesamten FIR-Filter dar. Es ist möglich diesen Filter in jeder beliebigen Ordnung zu definieren. Dazu muss man lediglich die Funktion *conv* unterversorgen, ihr also nur die Koeffizienten mitgeben. Die Anzahl der Koeffizienten legt hierbei die Ordnung des Filters fest. Übrig bleibt eine Funktion, die vielleicht einen Hochpass, einen Tiefpass oder andere ein anderes Filterverhalten darstellt.

Wir haben nun die Möglichkeit diesen Filter auch auf seine Korrektheit zu überprüfen. Dazu müssen wir Eigenschaften finden, mit denen wir ziegen können, dass unser Filter so arbeitet wie erwartet. So sollte der Filter auf einen Dirac-Impuls ($\delta(n)$) seine Koeffizienten ausgeben. Wir können diese Eigenschaft in einer Funktion folgendermaßen darstellen:

```
prop_ConvByOneBringsCoefs ::
  (Coefficients -> Values -> Values) ->
  Coefficients -> Bool

prop_ConvByOneBringsCoefs f bs =
  bs == (f bs [1])
```

Listing 10: Faltung eines Dirac-Impulses liefert die Filterkoeffizienten zurück

Diese Funktion ist nur dann wahr, wenn unser Filter tatsächlich die erwartete Ergebnisse ausgibt. Mittels *verify prop_ConvByOneBringsCoefs* wird Lava dazu veranlasst die definierte Eigenschaft des Filters zu beweisen. Die Ausgabe *valid* besagt jetzt, dass sich der Filter auch tatsächlich so verhält, wie es unsere Funktion verlangt. An diesem Beispiel lässt sich sehr schön zeigen, wie einfach es ist, die Funktionalität von Schaltkreisen automatisch beweisen zu lassen.

Der FIR-Filter ist auf die Typen *Values* und *Coefficients* definiert, wobei letztere auch *Values* sind. Im Listing 5 haben wir festgelegt, dass *Values* Listen von Integersignalen sind. Möchte man den Filter jetzt erweitern, dass er mit Festkommazahlen oder Gleitkommazahlen arbeiten kann, so ist die Änderung minimal. Wir müssten nur die Definition des Typs *Values* auf z.B. *[Signal Float]* ändern. Mit dieser kleinen Änderung währe der Filter funktionsfähig. Natürlich hängt dies an dem Typ *Signal Float*, den man zunächst noch für Lava definieren müsste. Ein weiterer Schritt währe, den Filter auf eine Typklasse zu definieren, so dass er mit allen möglichen Zahlentypen zusammenarbeitet. Gerade anhand solcher Überlegungen kann man erahnen, welche Möglichkeiten ein "Funktionales Modell" dem Hardwaredesigner eröffnet.

5 Verwandte Ansätze

Um mit der rasanten Entwicklung der Halbleitertechnologie schritthalten zu können, müssen Hardwaredesigner in zunehmendem Maße in der Lage sein, Hardwarebeschreibungen abstrakter zu formulieren. Die klassischen Hardwarebeschreibungssprachen, wie VHDL und Verilog, können durch ihre sehr begrenzten Abstaktionsmöglichkeiten diesem Anspruch immer weniger gerecht werden. Zwar kann man mit VHDL und Verilog Hardware auf der Register-Transfer-Ebene modellieren (was im Vergleich zur Modellierung auf Gatter-Ebene sehr abstrakt ist); jedoch sind VHDL und Verilog weniger gut geeignet um mit noch abstrakteren Beschreibungen (oft Transaction-Level Modelling genannt) umzugehen.

Das ist der Grund, warum Hardwaredesigner in zunehmendem Maße Konzepte aus der Softwareprogrammierung übernehmen. So ist gerade SystemC [8, 7], eine speziell auf die Bedürfnisse des Hardwareentwurfs zugeschnittene C++-Klassenbbliothek, dabei sich zu etablieren. C++ ist eine prozedurale Programmiersprache in der der Programmierer typischerweise den Kontrollfluss explizit entwirft. Will der Programmierer Nebenläufigkeit modellieren — und beim Hardwaredesign ist Nebenläufigkeit allgegenwärtig — so muss er sich explizit darum kümmern. SystemC beinhaltet eine relativ komplexe Laufzeitumgebung, mit der diese Hardware-Nebenläufigkeit simuliert werden kann.

Funktionale Sprachen bieten in gewisser Weise einen natürlicheren Zugang zur Hardwarebeschreibung: sie abstrahieren vom Kontrollfluss, d.h. der Programmierer kümmert sich nicht darum, in welcher Reihenfolge sein Code ausgeführt wird; diese Entscheidung überlässt er dem Compiler. Die Möglichkeit zur nebenläufigen Ausführung ist daher latent vorhanden. Neben Lava gibt es noch einige andere Versuche, Funktionale Sprachen für die Modellierung von Hardware zu verwenden:

- Die an Python angelehnte Hardwarebeschreibungssprache MyHDL [5]. Python vereint prozedurale, objekt-orientierte und funktionale Konzepte, ist eine sehr einfach zu erlernende Programmiersprache. MyHDL erlaubt die Modellierung auf Register-Transfer-Level aber auch abstraktere Formen der Modellierung.
- Mit Hydra hat John O'Donnell [10] ebenfalls eine "Hardware Description Language" entworfen. Bei Hydra handelt es sich allerdings um "reinen" Haskell code. Auch in Hydra arbeitet man — ähnlich wie in Lava — mit einer Typklasse die Signale darstellt (*Signal*). Tatsächlich ist Hydra älter als Lava und hat die Entwickler von Lava sicherlich inspiriert, was sich auch in der Ähnlichkeit beider Ansätze wiederspiegelt.
- Atom [5] ist eine weitere in Haskell eingebettete Hardwarebeschreibungssprache die in neuster Zeit parallel zur Lava und Hydra entwickelt wurde.

6 Ausblick

Wir haben gesehen, wie man einfache Schaltungen in ein funktionales Modell überführt. Gerade das letzte Kapitel hat gezeigt, dass Lava noch lange nicht vollständigt ist. Eine Aufgabe wäre sicherlich, den Signal-Typ Lavas zu erweitern. Bislang umfasst dieser Typ nur boolesche Signale sowie Integersignale. Es ist denkbar diesen Typ auf Signale von Fest- und/oder Gleitkommazahlen zu erweitern. Man könnte auch, wie in [2] beschrieben, Signale komplexer Zahlen definieren. Gerade der Signaltyp ist ein Punkt, an dem man ansetzten kann, wenn man an einer Verfeinerung Lavas weiterarbeiten möchte.

Ein weiterer Gedanke ist, Hardwarebeschreibungen weiter zu abstrahieren, um sie damit von Lava selbst wieder zu lösen. So könnte man den FIR-Filter z.B. rein in Haskell beschreiben. Im Falle dass man sich die Features von Lava zunutze machen möchte konkretisiert man dann sein funktionales Modell wieder hin in Richtung Lava. Hier stellt sich dann die Frage, wie man die Typen in den Griff bekommt. Vielleicht ist es sogar möglich das funktionale Modell, bei der Konkretisierung, verschiedene Ziele anzugeben. So könnte man auch die Vorteile anderer "Hardware Description Languages" ausnutzen. Die eigentliche Beschreibung würde aber weiterhin nur in Haskell stattfinden.

Interessant ist für uns insbesondere die Modellierung rekonfigurierbarer Systeme, d.h. Systeme, deren Hardware zu Laufzeit nicht notwendigerweise konstant ist. Es hat sich gezeigt, dass Haskells (und insbesondere Lavas) Abstraktionsmechanismen gut geeignet sind rekonfigurierbare System zu entwerfen [12]. Wir überlegen, automobiltechnische und/oder signalverarbeitende Anwendungen in Lava als rekonfigurierbare Systeme zu entwerfen und durch Lavas VHDL Generierung in FPGAs zu implementieren.

Besonders bei Hardwareimplementierungen auf FPGAs sind auch viele kritische nicht-funktionale Eigenschafen (wie beispielsweise Platzverbrauch, Schnelligkeit, Energieeffizienz) gegeneinander abzuwägen. Auch hier hat sich gezeigt, dass eine abstrakte funktionale Herangehensweise bei der Abwägung von Entwurfsentscheidungen sehr hilfreich sein kann [1] und wir planen, entsprechende Techniken zu entwickeln und bestehende auszubauen.

[5] http://funhdl.org

Literatur

[1] Emil Axelsson, Koen Classen, and Mary Sheeran. Using lava and wired for design exploration. In *Designing Correct Circuits Workshop; A Satellite Event of ETAPS 2006*, Wien, Maerz 2006.

[2] Per Bjesse, Koen Claessen, Mary Sheeran, and Satnam Singh. Lava: Hardware design in haskell. *ACM Press*, 1998.

[3] Koen Claessen. An embedded language approach to hardware description and verification. Master's thesis, Göteborg University, 2000.

[4] Koen Claessen and Mary Sheeran. *A Tutorial on Lava*. Chalmers University of Technology.

[5] Jan Decaluwe. *The MyHDL manual*, 2006. http://myhdl.jandecaluwe.com/doku.php.

[6] Gunnar Eisenberg. *Implementierung von FIR-Filtern*. Technische Universität Berlin, 2000.

[7] Frank Ghenassia. *Transaction-Level Modeling with SystemC. TLM Concepts and Applications for Embedded Systems*. Springer Verlag, 2006.

[8] Thorsten Groetker, Stan Liao, Grant Martin, and Stuart Swan. *System Design with SystemC*. Kluwer Academic Publishers, 2002.

[9] John Hughes. Why functional programming matters. *Computer Journal*, 32(2):98–107, 1989.

[10] John T. O'Donnell. *The Hydra Computer Hardware Description Language*. University of Glasgow, 2003.

[11] Mary Sheeran. Hardware design and functional programming: a perfect match. *Journal on Universal Computer Science*, 11(7):1135–1158, 2005.

[12] Satnam Singh. Designing reconfigurable systems in lava. In *17th International Conference on VLSI Design*, Mumbai, India, January 2004. IEEE Computer Society.

Development of an Electro Physiological Front End with SPI-Bus Connection

Daniel Bau

ASIC Design Center der

University of Applied Sciences, Offenburg,
Badstrasse 24 daniel.bau@fh-offenburg.de

Im Rahmen einer Master Thesis wurde ausgehend von einem vorhandenen System On Chip Design, welches eingehende EKG-Datensignale verarbeitet, das bestehende System so erweitert dass es komplett über den standardisierten SPI-Bus steuerbar und auslesbar ist.

7. Einführung

Im Jahre 1999 entstand an der Hochschule Offenburg zur Erfassung von EKG-Datensignalen ein sogenannter DSWPC (Dual-Signal-Wavelet-Processing-Controller). Im Laufe von Jahren entstand damals ein ASIC, welcher ein komplettes Mikrocontrollersystem und die zur Signalverarbeitung entsprechenden Filterstufen enthält. Die über 2 Kanäle eingehende analoge Herzfrequenzen werden dabei von zwei Sigma-Delta-Wandlern digital aufbereitet und jeweils in einem Sinc3-Filter zur Signalaufbereitung für die nachgeschaltete IIR-FIR-Wavelet-Einheit weiterverarbeitet. Die IIR-FIR-Wavelet-Einheit beinhaltet eine 3-stufige Filterung der beiden zuvor aufbereiteten Kanaldaten und einen Wavelet-Algorithmus, welcher die Unmenge an anfallenden Filterdaten um den Faktor 4 komprimiert.

Neben diesem komplexen System verfügte der DSWPC-Chip über ein Mikrocontrollersystem, welches die Filtereinheiten steuerte und sich den Speicher mit der IIR-FIR-Wavelet-Einheit teilte. Der damals im DSWPC-System eingesetzte Prozessor FHOP (First Homemade Operational Processor) ist im Vergleich mit heutigen Prozessoren recht ineffizient. Die Programmierung von Treibern und Anwendungsprogrammen erfolgte in der Programmiersprache Assembler und die Verarbeitung eines Befehls des Prozessors FHOP erfolgte in durchschnittlich 8 Takten. Die mangelhafte Leistungsfähigkeit des Prozessors und die mit erheblichen Schwierigkeiten belastete gemeinsame Nutzung des Speichers gaben den Anstoß das vorhandene Systemkonzept zu überdenken und neu zu gestalten. Des Weiteren ist der DSWPC-Chip mit seinem integrierten Mikrocontrollersystem zu speziell und somit zu unflexible in seiner Anwendungsvielfalt.

Nach einigen Überlegungen kristallisierte sich das Konzept heraus, die zur Erfassung von EKG-Signalen benötigten Einheiten ohne jegliche Mikrocontrollersteuerung in einem ASIC zu platzieren. Die Filtereinheiten sollen über den standardisierten seriellen SPI-Bus zugänglich sein, was die Integration eines SPI-Controllers erfordert. Daraus ergeben sich die Vorteile, dass die gemeinsame Nutzung des Speichers sich auf einen minimalen Datenzugriff über einen SPI-Controller reduziert und der neu entwickelte ASIC je nach gewünschter Anzahl von Kanälen mehrfach in ein System integriert werden kann und somit ein flexibel einsetzbares System zur Verfügung steht. Jeder beliebige Mikrocontroller, welcher ein SPI-Interface besitzt, kann eingesetzt werden um den ASIC zu steuern. Eine Übersicht ist in Abbildung 1.1 zu sehen, welche auf Blockschaltbildebene das Funktionsprinzip wiedergibt.

Der im Jahre 1999 entstandene DSWPC-Chip wird in der 0,5 µm Technologie des Herstellers Alcatel-Mietec gefertigt und erfolgreich in der Medizintechnik eingesetzt. Die Neuentwicklung des ASICs soll auf der im ASIC-Labor der Hochschule Offenburg erfolgreich eingesetzten 0,35µm Technologie des Herstellers Alcatel-Mietec basieren und die Vielfältigkeit der Verwendung des Produkts erfolgreich erweitern.

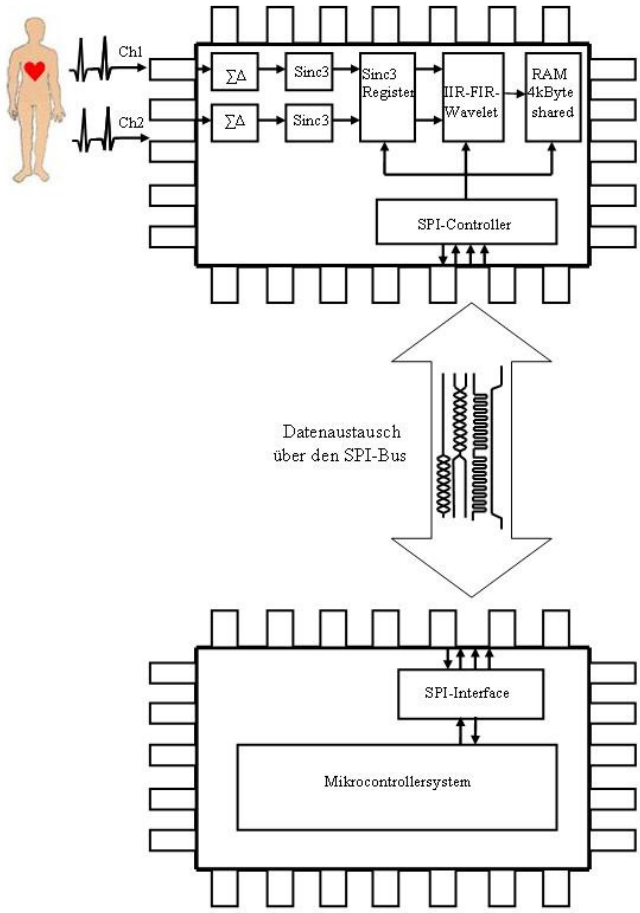


Abbildung 1.1: Blockschaltbild des Funktionsprinzips

7. Baugruppen des Systems

Das in Abbildung 1.1 dargestellte Systemkonzept des zur Erfassung der eingehenden EKG-Signale einzusetzenden ASICs, zeigt die intern verwendeten Baugruppen. Die Systemkomponenten bestehen hauptsächlich aus digitalen Baugruppen mit der Ausnahme der beiden analogen Sigma-Delta-Wandlern 2.Ordnung, welche die analogen EKG-Signale digitalisieren. Die Umsetzung der analogen Zellen lag bereits vor und musste lediglich nochmals verifiziert werden.

Die eingesetzten Baugruppen des Systems belaufen sich auf folgende:

- 2 analoge Sigma-Delta-Wandler 2.Ordnung
- 2 Sinc3-Filter und die dazugehörigen 20Bit-Sinc3-Register
- Taktteiler zur Einstellung der Abtastfrequenz
- IIR-FIR-Wavelet-Filtereinheit
- RAM-Sharing-Controller mit 4kByte Speicher
- SPI-Controller

Der SPI-Controller, die beiden Sinc3-Register mit Interruptgenerierung und der variabel einstellbare Taktteiler wurden in der Master Thesis komplett neuentwickelt. Die restlichen Einheiten wurden modifiziert und an die neu entstandenen Systembedingungen angepasst.

7. SPI-Controller des Systems

Die Schnittstelle findet ihren Einsatz in der seriellen Datenübertragung nach dem SPI (Serial Peripheral Interface) Standard und ist die zentrale Komponente im Datenaustausch der Filter-Einheit mit einem Mikrocontrollersystem. Über den SPI-Controller kann jedes Register und jeder Speicherbereich des 4 Kilobyte Speichers der Filter-Einheit gelesen und geschrieben werden. Um festzulegen welches Register oder welcher Speicherbereich angesteuert wird, ist ein 8Bit breites Kommando-Byte verantwortlich, welches bei der Kommunikation mit dem Controller als erstes Byte über den SPI-Bus von einem Mikrocontroller ausgehend gesendet wird und in der Steuereinheit entsprechend dekodiert wird. Für das Senden und Empfangen von Bits über den SPI-Bus stehen 2 Zustandsmaschinen zur Verfügung, die synchron zum SPI-Takt ein Byte einlesen oder auf dem Bus übertragen. Eine weitere Einheit generiert ein Enable-Signal solange ein SPI-Takt empfangen wird, um der Steuereinheit die Gültigkeit des empfangenen oder gesendeten Byte mitzuteilen. Zusätzlich zu den SPI-Signalen generiert der SPI-Controller ein Ready-Busy-Interrupt Signal, auf dass sich der Mikrocontroller synchronisieren muss. Über das sich im SPI-Controller befindliche Konfigurationsregister kann die Filter-Einheit in ihrem Filterumfang eingeschränkt und das Busprotokoll um ein Bestätigungsbyte erweitern werden. Abbildung 3.1 zeigt die Architektur des SPI-Controllers.

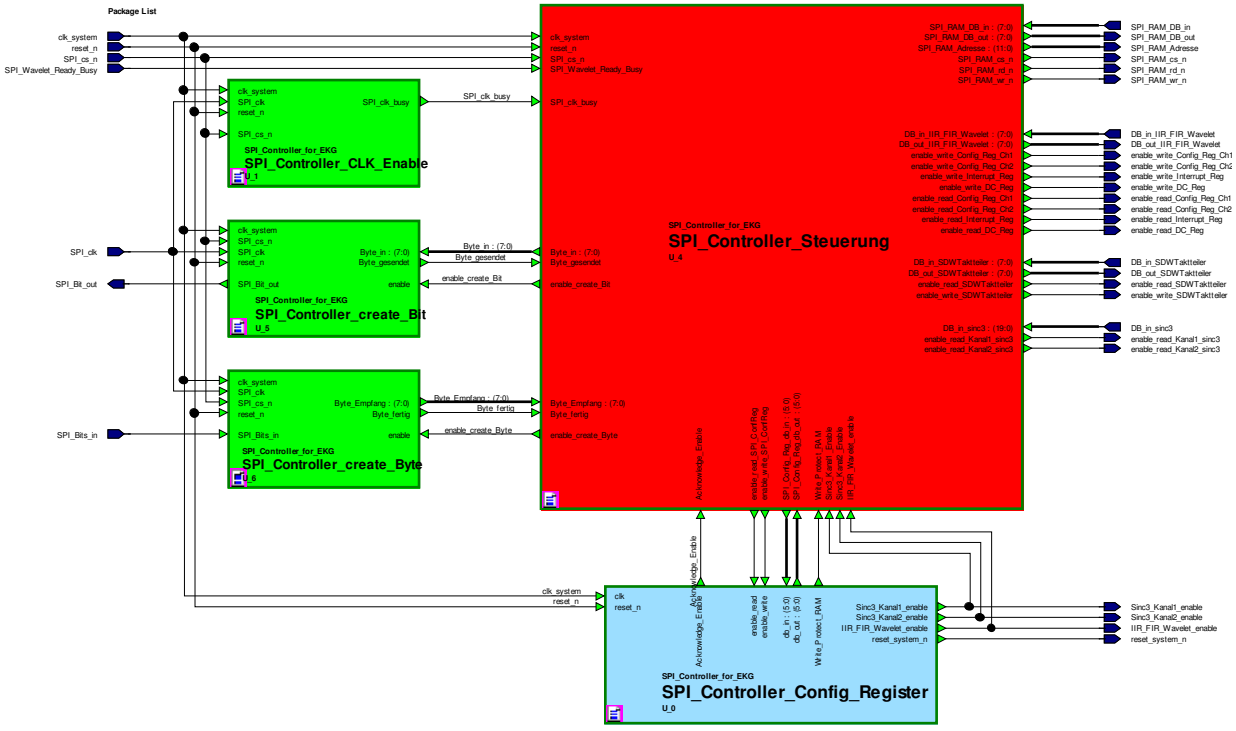


Abbildung 3.1: Aufbau des SPI-Controllers

Der SPI-Controller weist folgende Eigenschaften auf:

- Bis 4MHz SPI-Takt einsetzbar
- Übertragung nach dem SPI-Standard
- Bus Kommunikation nach SPI Mode 1 (Datenübertragung bei fallender Taktflanke)
- Zugriff auf den gesamten Speicherbereich des 4KB-RAMs
- Jedes Register der Filter-Einheiten steuerbar
- Protokoll-Handshake über ein implementiertes Acknowledge/Not-Acknowledge Byte

Das Kommando-Byte mit seinen 8 Bit, welches als erstes Byte bei einer beginnenden Kommunikation übertragen wird, ist folgend strukturiert. Die ersten 6 Bit des Bytes sind als reiner Befehlszähler kodiert und den letzten beiden Bits ihrer binären Wertigkeit entsprechend Funktionsprinzipien zugeordnet. Somit fungiert das 8. Bit als Lese-/Schreib-Bit und das 7. Bit als ein einzelner Register-/Speicherzugriff oder als eine Kombination von Lese- und Schreiboptionen auf verschiedene Register/Speicherbereiche.

Das Kommando-Byte weist in seiner Kodierung folgende Eigenschaften auf:

- 53 mögliche Befehlsoperationen
- 26 Schreib-Befehle
- 27 Lese-Befehle
- Davon 9 Block-Befehle, welche eine Kombination von Lese- und Schreibzugriffen ermöglichen

3.1 Befehlssequenzen des SPI-Controllers

Bei allen Befehlen und Daten welche der SPI-Controller über den SPI-Bus empfängt oder sendet, ist das MSB das Erste der Übertragung. Die ankommenden seriellen Daten am Dateneingang des SPI-Controllers werden bei der ersten steigenden Flanke des SPI-Taktes gespeichert sobald das Chip-Select ein low treibt. Gesendet werden die Bits über den seriellen Datenausgang bei steigender Flanke des SPI-Taktes. Als erstes Byte wird das Kommando-Byte eingelesen, welches am seriellen Dateneingang mit dem MSB als erstes Bit gesendet wird. Jedes Bit wird bei fallender Taktflanke des SPI-Taktes gespeichert. Der Befehlscode ist in dem Kommando-Byte kodiert.

Jede Befehlssequenz beginnt mit einem 1Byte Befehlscode. Abhängig von dem Befehl werden ein Register, mehrere Register, der Speicher oder eine

Kombination aus Register-Speicher-Zugriffen gelesen oder geschrieben. Je nach Einstellung des Konfigurationsregisters im SPI-Controller kann zur Bestätigung einer Befehlssequenz ein Acknowledge-Byte mit übertragen werden. Um eine Übertragung zu beenden muss das Chip-Select high getrieben werden, nachdem das letzte Bit einer Befehlssequenz bzw. das Acknowledge-Byte übertragen wurde.

Folgende Register und Speicherbereiche der Filter-Einheit können angesteuert werden:

- Register:
  - Taktteilerregister,
  - Konfig-Kanal1&2 Register,
  - Interrupt-Register,
  - DC-Register,
  - SPI-Konfigurationsregister
  - Sinc3-Register Kanal1&2
- Speicherbereiche:
  - IIR Filter-Koeffizienten mit je 2 Byte
  - Wavelet-Koeffizienten und Wavelet Zwischenspeicher mit je 8 Byte
  - 32. Ordnung FIR Filter-Koeffizienen mit 40 Byte
  - 64. Ordnung FIR Filter-Koeffizienten mit 80 Byte
  - IIR-Buffer und FIR-Buffer
  - 4 Filterdaten-Wechselbuffer mit je 200 Byte

Die entstandenen VHDL-Codes des SPI-Controllers wurden mit dem im HDL-Designer verknüpften Programm Modelsim von Mentor Graphics kompiliert und simuliert. Dabei wurden während der Entwicklungsphase mithilfe einer SPI-Testbench die entstandenen Funktionen durch Simulation nachgewiesen. Die Testbench generiert die auf dem SPI-Bus nötigen Signale, um über den SPI-Controller die Filter-Einheit zu steuern. Abbildung 3.2 zeigt die prinzipielle Kommunikation über den SPI-Bus eines Registerzugriffs.

Wird eines der beiden 20 Bit breiten Sinc3-Register über das Kommando-Byte angesprochen, sendet der SPI-Controller im ersten Byte Bit7-0, im zweiten Byte Bit15-8 und im letzten Byte Bit19-16 plus 4 Nullen. Je nach Einstellung des SPI-Konfigurationsregisters, wird ein Acknowledge-Byte zur Bestätigung des Lesevorgangs übertragen oder die Kommunikation direkt mit einem hoch treibenden Chip-Select beendet.

4. Verifikation auf einem FPGA

Nach erfolgreicher Simulation wurde der VHDL-Code mit dem Syntheseprogramm „Synopsys“ auf die 0,35µm ASIC Technologie synthetisiert. Durch eine modifizierte allgemeine Beschreibung der in der 0,35µm eingesetzten Mietec-Standardzellen, ist die daraus resultierende Netzliste in Kombination mit der VHDL-Beschreibung der Standardzellen technologieunabhängig. Somit ist die von dem Synthesetool Synopsys ausgehende Netzliste universell einsetzbar und kann auf jede beliebige Zieltechnologie, ob ASIC oder FPGA umgesetzt werden. Bei der weiteren Umsetzung auf FPGA-Ebene, werden lediglich die Standardzellen der Mietec-Technologie mit den Standardzellen der FPGA-Technologie ersetzt.

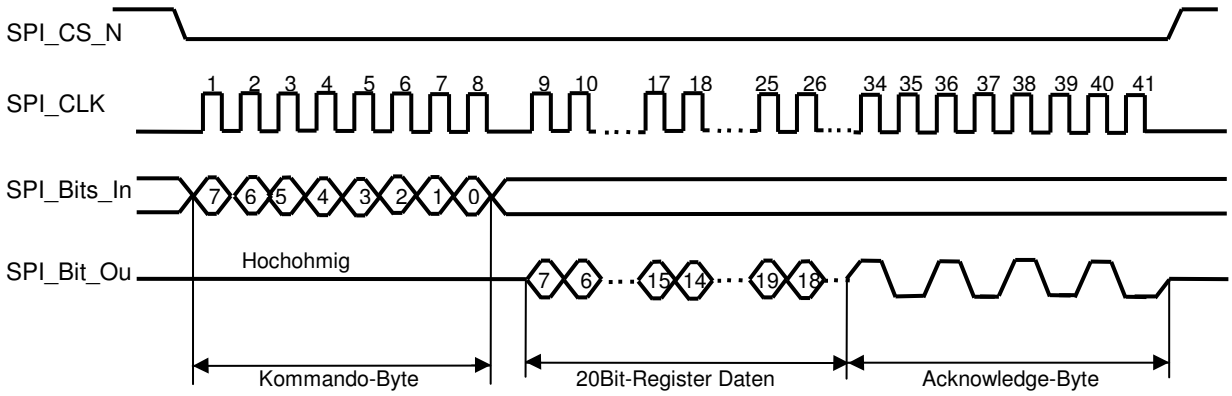


Abbildung 3.2: 20Bit Sinc3-Register lesen

Dadurch wird ein realitätsnahes Abbild des ASIC-Designs auf dem FPGA gewährleistet. Des Weiteren verfügt das Synthesetool Synopsys über leistungsstarke Optimierungsalgorithmen und über spezielle Bibliotheken, welche auf Geschwindigkeit optimierte Komponenten wie z.B. Addierer oder Multiplizierer einbinden kann. Der Speicher, welcher im ASIC zum Einsatz kommt, ist in einem FPGA nicht umsetzbar, da dieser seine eigenen vordefinierten Speicherzellen besitzt, welche man sich mithilfe der Megafunctions von Altera generieren kann. Aufgrund nicht vollständig vergleichbarer Ansteuerungssignalen der beiden Speichertechnologien, ist eine geringfügige Änderung der Speicheransteuerung notwendig.

Die von Synopsys generierte Netzliste und die VHDL-Codes der Standardzellen wurden zur Umsetzung auf FPGA-Ebene im Synthese- und Routingtool „Quartus“ weiterverarbeitet. Als Zieltechnologie wurde hier der FPGA-Chip vom Typ CycloneIII-EP3C25E144C8 der Firma Altera eingesetzt, welcher mit 24.624 Logic Cells, 24.624 Registern und 608.256 Speicher-Bits ausgestattet ist. Der FPGA befindet sich auf einem in der Hochschule entwickeltem FPGA-Board, welches im Jahre 2007 speziell für das Mikrocontrollersystem SIRIUS designed wurde. Um das Wavelet-System effizient zu testen, wird ein weiteres FPGA-Board eingesetzt, welches das Mikrocontrollersystem SIRIUS enthält. Abbildung 4.1 zeigt den prinzipiellen Testaufbau der beiden eingesetzten FPGA-Boards.

Jedes Testboard ist mit einem frei programmierbaren CycloneIII-FPGA ausgestattet. In dem FPGA-Board worin sich die Speicherkarte befindet, ist das SIRIUS Mikrocontrollersystem aufgespielt und über das USB-Kabel mit einem PC verbunden. Die auf dem PC installierte USB-Software ermöglicht den Zugriff auf die SD-Karte und kann die dort abgelegten Programmfiles, welche mit der SIRIUS IDE erstellt und kompiliert wurden, ausführen. Das Hinterlegen der erstellten Programmfiles kann über einen entsprechenden Softwareaufruf über die USB-Verbindung gesteuert werden, oder man entnimmt die SD-Karte und speichert die Programmfiles mit einem handelsüblichen Kartenlesegerät ab.

Das zweite FPGA-Board enthält das Wavelet-System und wird über das USB-Kabel lediglich mit Energie versorgt. Die einzige Verbindung der beiden FPGA-Designs besteht über den SPI-Bus. Der digitale Part des Wavelet-Systems mit seinen neu entwickelten Komponenten wie z.B. SPI-Controller und die vorhandenen modifizierten Komponenten, wurde mithilfe der erstellten Treiberroutinen auf seine Funktion hin überprüft und dessen Funktionsfähigkeit erfolgreich nachgewiesen. Alle im SPI-Controller kodierten 53 verschiedenen Befehlssequenzen wurden über das Mikrocontrollersystem SIRIUS angesteuert und erfolgreich verifiziert.

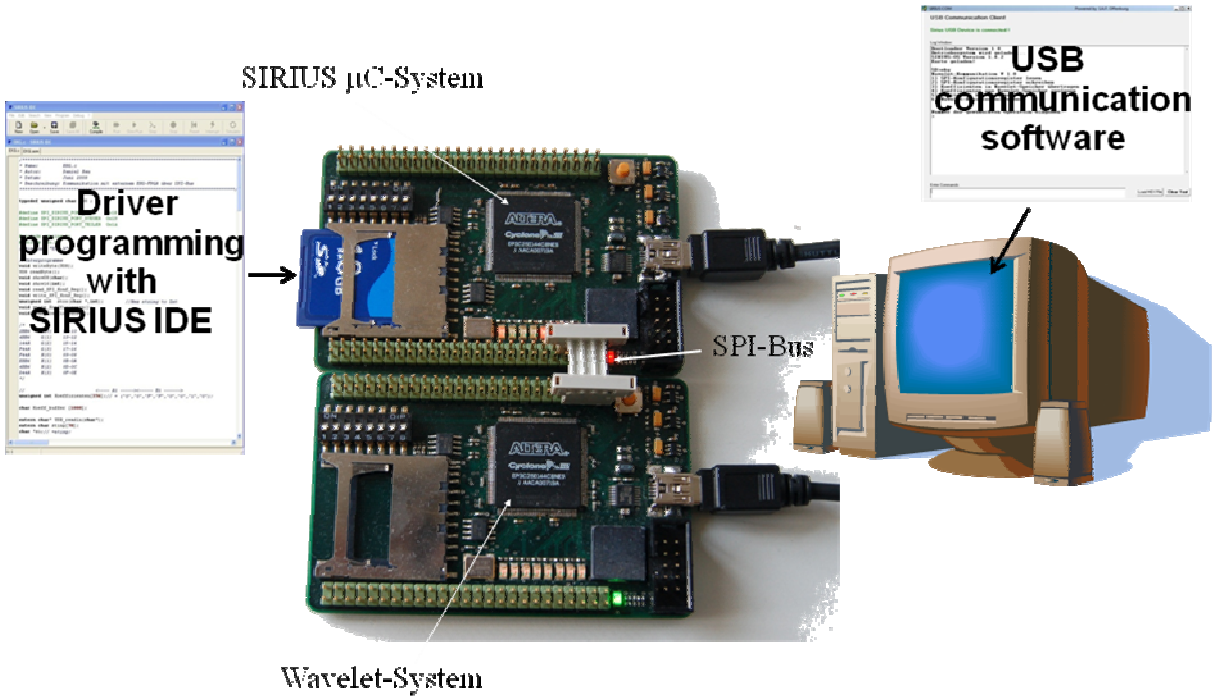


Abbildung 4.1: Testumgebung

Das Design wurde mit dem Place&Route-Tool Quartus auf dem FPGA platziert und geroutet. Die beiden folgenden Tabellen zeigen die Quartus-Ergebnisse des kompletten digitalen Schaltungsparts der jeweiligen FPGA-Designs.

Wavelet Synthese-Ergebnis			
Beschreibung	Verwendet	Verfügbar	Ausnutzung
IOs (IN/OUT-Ports)	9	83	11%
LCs (Logic Cells)	3.429	24.624	14%
Memory Bits	32.768	608.256	5%

Tabelle 4.1: Synthese-Ergebnis der digitalen Wavelet-Einheit

SIRIUS µC-System Synthese-Ergebnis			
Beschreibung	Verwendet	Verfügbar	Ausnutzung
IOs (IN/OUT-Ports)	48	83	58%
LCs (Logic Cells)	8.923	24.624	36%
Memory Bits	525.312	608.256	86%

Tabelle 4.2: Synthese-Ergebnis des SIRIUS µC-System

7. ASIC-Layout

Nach erfolgreicher FGA-Verifikation wurde das digitale System in die 0,35µm CMOS Mietec-Technologie übersetzt. Die von dem verwendeten Synthesetool Synopsys gewonnene Verilog-Netzliste der digitalen Schaltungen im Wavelet-System wurde in dem von der Firma Cadence verfügbarem Layout-Tool Encounter eingelesen und gelayoutet. Dieses Tool ist im digitalen platzieren, routen, erneute Taktsynthese und layouten einfach zu bedienen und bietet im Vergleich mit dem Layout-Tool IC-Station der Firma Mentor Graphics einen besseren Platzier- und Routingalgorithmus. Nach Platzieren und Beenden des Routens kann man sich die über die Taktsynthese optimierte Netzliste herausschreiben lassen und ein SDF-File (Standard Delay Format) generieren, mit welchem die Möglichkeit besteht, unter realen Verzögerungszeiten den digitalen Schaltungspart des Designs erneut zu simulieren. Für analoge Schaltungsentwicklung ist das Layout-Tool Encounter nicht geeignet.

Bei einem rein digitalen ASIC könnte man die beim Layout entstehenden GDS-Files direkt dem Hersteller zur Fertigung übermitteln. Das Programm arbeitet mit sogenannten LEF-Files (LEF=Layout Exchange Format) welche die geometrische Lage von Pins, die Dimension und die verschiedenen Layer einer Zelle beschreibt. Die in der Netzliste beschriebenen Standardzellen, Padzellen und der Speicher sind vom Hersteller über die Beschreibung im LEF-Format gegeben. Die beiden analogen Sigma-Delta-Wandler wurden in der IC-Station gelayoutet und in das LEF-Format konvertiert. Somit liegt jede im ASIC verwendete Einheit als geometrische Beschreibung vor, wodurch die Komponenten untereinander über die entsprechenden Routingalgorithmen miteinander verdrahtet werden können. Nach erfolgreichem layouten wird das entstandene Design in das GDSII-Format exportiert und dann in die IC-Station eingelesen. In der IC-Station erhält man über das GDSII-Format eine Einsicht in die komplette Struktur des ASICs. Jeder Layer ist sichtbar und der geometrische Platzhalter des analogen Schaltungsparts wird mit dem Orginal-Layout ersetzt. Zuletzt erfolgt dann der abschließende Design Rule Check, welcher das entstandene ASIC-Layout auf die vom Hersteller geforderten geometrischen Bestimmungen überprüft.

Im Rahmen der zeitlichen Begrenzung der Master Thesis war es nicht möglich, den Wavelet-ASIC komplett fertig zu stellen. Die erstellte Vorabversion des Chips in der 0,35µm CMOS Technologie weist eine Fläche von 3350µm x 4700µm auf.

7. Zusammenfassung

In der vorliegenden Master Thesis spiegeln sich die jahrelangen Entwicklungen und Erfahrungen des ASIC Design Centers der Hochschule Offenburg wieder. Ausgangspunkt der

Arbeit war das Design des DSWPC-ASICs, mit dessen Hilfe es möglich ist, EKG-Datensignale zu verarbeiten. Zunächst einmal galt es einen Überblick über die Komplexität des erfolgreich in der Medizintechnik eingesetzten DSWPC zu bekommen.

Nach den ersten Betrachtungen zeigte sich, dass das Gesamtkonzept des vorliegenden ASICs schlecht zu vermarkten ist. Die Kombination der zur Datenerfassung eingebauten Filterstufen mit einem ebenfalls integrierten Mikrocontrollersystem war zu speziell und unflexibel. Daher war Sinn und Zweck dieser Master Thesis, das System so zu Optimieren, dass der entstehende ASIC universell einsetzbar ist. Über den in der Industrie weit verbreiteten und einfach zu kontrollierenden SPI-Bus kann das entstandene „Electro Physiological Front End“ (EPFE) vollständig gesteuert werden. Um dies zu ermöglichen wurde das im DSWPC integrierte Mikrocontrollersystem FHOP durch einen SPI-Controller ersetzt, welcher als Slave fungiert und über jedes beliebige Mikrocontrollersystem mit verfügbarer Master-SPI-Schnittstelle angesprochen werden kann. Dadurch ist das EPFE-System, welches jeweils über 2 Kanäle verfügt, modular auf ein Vielfaches von 2 Kanälen erweiterbar. Durch den universellen Einsatz und Erweiterbarkeit des Grundsystems ist die Grundlage gegeben für eine zukünftige Produktfamilie von EPFE-Systemen.

Der neu entwickelte digitale Schaltungsteil wurde komplett in einem FPGA verifiziert und mit entsprechenden Testverfahren eine volle Funktionsfähigkeit nachgewiesen. Zur Ansteuerung des EPFE-Systems wurde das im ASIC Design Center zur Verfügung stehende und relativ einfach zu implementierende Mikrocontrollersystem SIRIUS verwendet. Die zur Ansteuerung notwendigen Treiber wurden in C erstellt und erfolgreich in die Testumgebung miteinbezogen.

Der analoge Sigma Delta Wandler im EPFE-System wurde in vorangegangenen Arbeiten auf die 0,35µm Zieltechnologie umgesetzt und lediglich das Layout wurde erneut auf DRC-Fehler hin überprüft und gegebenenfalls korrigiert. Aus zeitlichen Gründen konnte im Rahmen der Master Thesis nur eine Vorabversion des ASIC-Layouts erstellt werden, welches ein Flächenverbrauch von 3350µm x 4700µm aufweist. Der Chip wird voraussichtlich im November 2008 beim Hersteller EUROPRACTICE in die Produktion gegeben und steht dann im Februar 2009 der Hochschule zur Verfügung.

7. Referenzen:

[1] Carsten Störk, Wolfgang Vollmer: Dual Signal Wavelet Processing Controller (DSWPC), Projektbericht, HS-Offenburg, 1999

[2] Markus Striebel: Technologie-Umsetzung DSWPC, Diplomarbeit, HS-Offenburg, 2001

[3] Carsten Störk: Entwicklung einer integrierten Sigma-Delta-Wandler-Zelle, Projektbericht, HS-Offenburg, 1999

[4] Dipl.-Ing. M. Rezaeian, Prof. Dr. D. Jansen: Wavelet-Kompression von EKG-Signalen, Technischer Bericht, Projekt MINELOG, HS- Ulm, HS- Offenburg, Dec.1998

[5] Barber Zamin Khan: Redesign, Evaluation in VHDL-AMS and Layout of a High Performance 16-bit Second-Order Sigma Delta Converter in Full Custom Style to a Deep Sub Micron CMOS technology, Master Thesis, HS-Offenburg, 2004

[6] Dipl.-Ing. Marc Durrenberger, Dipl.-Ing. Daniel Bau, Prof. Dr. D. Jansen: SIRIUS Mikrocontrollersystem Design Kit, Technischer Bericht, HS-Offenburg, 2007

[7] Datasheet MIETEC 0,35µm Generators Design Data Book: „SPS2 synchronous Single Port RAM“ , 1999

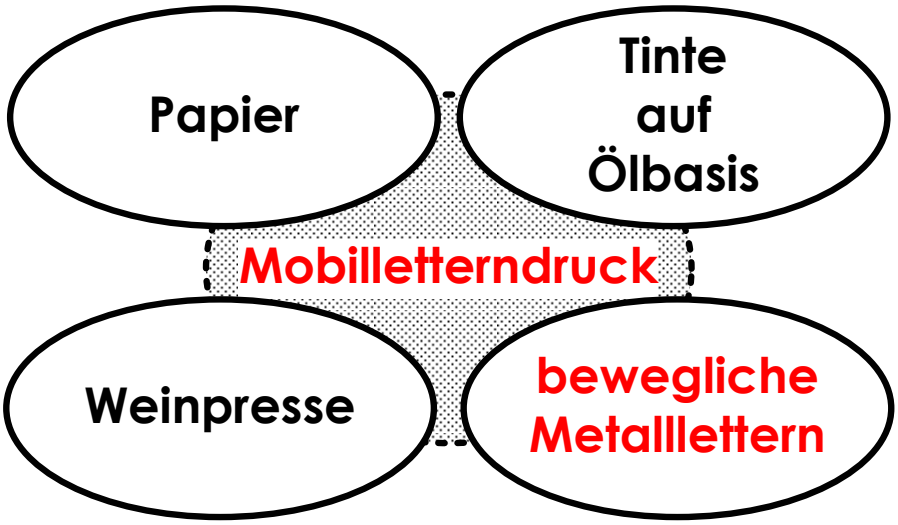


Abbildung 1 Innovationsprozess um 1450 – Mobilletterndruck, Johannes Gensfleisch, genannt Gutenberg, , Papier – Tinte auf Ölbasis und die Weinpresse waren die Grundlage

Organic/Paper Substrate RFID

e-Paper – Intelligent Paper - Inkjet-Printing - Printable Electronic - Printed RFID Tags

Prof. Dr.-Ing. M. Bartel

Hochschule Aalen, 73430 Aalen, Anton-Huber-Str. 25

Tel. 07361.576.4182, Fax. 07361.44.4247, Email manfred.bartel@htw-aalen.de

1. Zusammenfassung

Die Fähigkeit eines Paradigmenwechsels charakterisiert innovative Denkprozesse. Die Nutzung der 550 Jahre alten Drucktechnologie für den vollkommen neuen Anwendungsbereich „Auf Papier gedruckter Elektronik“, stellt solch einen Vorgang dar. Die Fähigkeit menschliche Innovationsprozesse modellieren zu können (die Repräsentation einer Repräsentation), stellt einen weiteren wichtigen Schritt in der menschlichen Entwicklung dar.

Es werden am Beispiel gedruckter Elektronik auf Papiersubstrat Hinweise herausgearbeitet, wie innovative Denkprozesse strukturiert sind und zu welchen konkreten Ergebnissen sie aktuell geführt haben.

2. Einführung

Sobald eine grundlegende, revolutionäre Innovation die breite Öffentlichkeit erreicht, fragen sich viele Menschen, warum sie nicht selber auf diese Idee gekommen sind, da doch fast alle Einzelbestandteile schon lange bekannt waren und nur noch ein kleiner Schritt vollbracht werden musste, um diese neue Idee Wirklichkeit werden zu lassen.

Exakt vor solch einem Problem stand Johannes Gensfleisch, genannt Gutenberg, im Jahr 1450 (siehe Abbildung 1). Als Innovationsgrundlage waren Papier, Tinte auf Ölbasis und die Weinpresse lange bekannt, der letzte Schritt war die Idee mobile Metalllettern einzusetzen und der Buchdruck war erfunden.

Im Jahr 1905 veröffentlichte Albert Einstein den Artikel „Zur Elektrodynamik bewegter Körper“ und legte so den Kern der speziellen Relativitätstheorie, mit Aussagen über die Raumzeit. Bis zu diesem Zeitpunkt fiel es Menschen schwer, ihre eigenen Sinnesprozesse zu abstrahieren. Diese generelle Erkenntnisgrenze ergab sich aus der Tatsache, dass evolutionär kein Selektionsdruck bestand eine Raumzeitsensorik zu entwickeln, da Menschen nicht in diesem physikalischen Umfeld überleben mussten.

Als weiterführende Schlussfolgerung ergibt sich, dass das Gehirn-Geist-System daher bei allen Sinnesprozessen die eingehenden Signalströme zwanghaft in struktur- und verhaltensbezogene Anteile zerlegt, die

separat verarbeitet werden (siehe Abbildung 2). Die innere Verwobenheit, in der vierdimensionalen Raumzeit, bleibt somit verborgen.

Mit dieser Erkenntnis lassen sich schlüssig weitere Phänomene erklären, wie z.B. die Notwendigkeit, in vorwissenschaftlicher Zeit, nicht erklärliche Prozesse einer Göttlichkeit zuzuordnen.

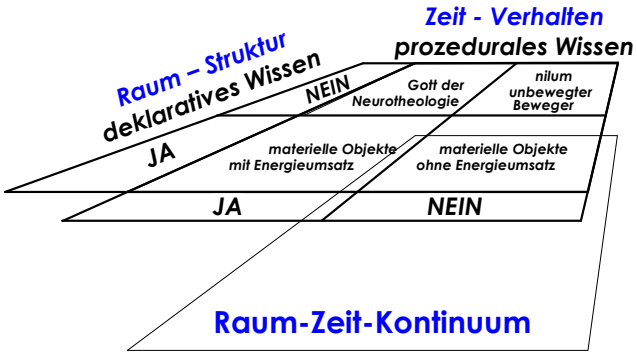


Abbildung 2 Menschliche Wahrnehmung - Wissensstand seit 1905 [EINS22]

Der Innovationsprozess, den Albert Einstein angestoßen hat, ist somit die Abstraktion der menschlichen Erkenntnisprozesse auf ein höheres abstraktes Niveau (definierender Algorithmus: Repräsentation einer Repräsentation) und stellt einen Abstraktionsprozess auf einer höheren Ebene dar, als der, den Gutenberg vollzogen hat (siehe Abbildung 3).

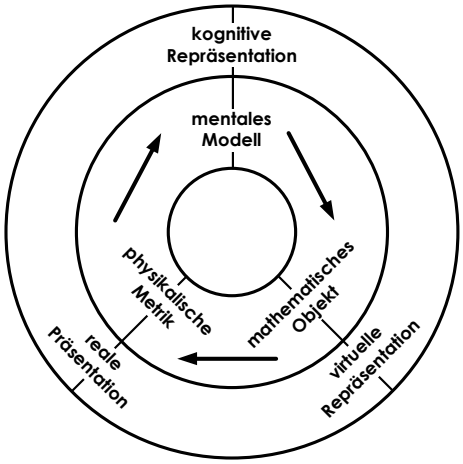


Abbildung 3 Dreiphasiger Entwicklungskreislauf

Im 21. Jahrhundert, haben der Akademisierungsgrad und die Informationsverfügbarkeit so stark zugenommen, dass täglich über neue Innovationen berichtet wird. In diesem Papier wird eine aktuelle Zusammenfassung über organische, elektronisch nutzbare Substrate geliefert, wie z.B. Papier, deren Entwicklung seit ca. zehn Jahren betrieben wird.

Es wird versucht, an diesem Beispiel weitere generelle Ingredienzen innovativer Prozesse sichtbar zu machen und den Stand der Technik einer aktuellen Innovation zu dokumentieren.

2.1. Ziel

Alle Systementwickler sind es gewohnt sich in der dualen Welt von Raum und Zeit oder Struktur und Verhalten (Abbildung 2) zu bewegen und erfolgreich Systeme zu denken, zu implementieren und zu realisieren (siehe Abbildung 3, siehe auch [DONA93]).

Wenn sie ein neues System denken, beschäftigen sie sich zuerst mit dem gewünschten Verhalten, dass sie mental repräsentieren. Sie erzeugen danach virtuelle Repräsentationen, die automatisch in fertigbare Strukturen transformiert werden. Die Nano-Mikro-Elektronik, in Kombination mit der Informatik, beherrscht dieses Verfahren und perfektioniert es täglich. Untersucht man unter diesem Gesichtspunkt die Relation „Struktur versus Verhalten“, ergibt sich folgendes Postulat: „Verhalten ist alles! Struktur ist nichts!!!“. Da dieses Postulat zu dogmatisch ist, schwächen wir es ab und formulieren zielgerichteter: „Das Verhalten ist das Ziel! Die Struktur ist austauschbar!!!“. Dies ist die grundlegende Idee aller HDL´s (hardware description language).

Seit der Erfindung des Germanium-Transistors in den Bell Laboratories am 23. Dezember 1947 wurde das generelle Verfahren, Transistoren in einem Halbleiter zu realisieren, enorm perfektioniert.

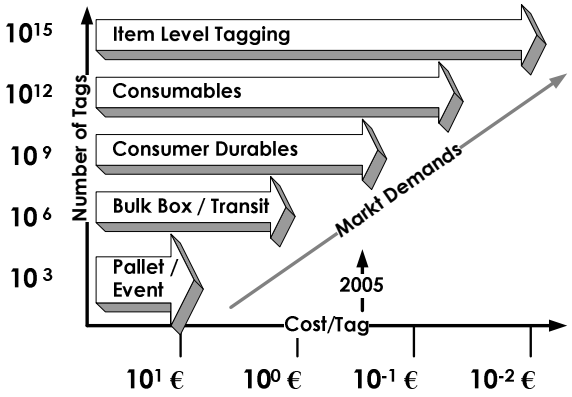


Abbildung 4 Die RFID Herausforderung – die Herstellungskosten müssen soweit reduziert werden, dass „Item Level Tagging“ möglich wird [NORB05]

Unter Kostengesichtspunkten lassen sich mit diesem Verfahren aber nicht alle Produkte realisieren (siehe

Abbildung 4), die man sich ausdenken kann. Z.B. der Bereich der kontaktlosen Übertragung kleiner Datenpakete, die gezielte datentechnische Hinweise über kurze Entfernungen, über jedes ausgerüstete reale Objekt, senden können, stagniert. RFID Tags sind mit Hilfe von Halbleiterprozessen im Cent oder Millicent Bereich nicht zu fertigen. Auf Papier gedruckte Elektronik kann diese Anforderung lösen.

2.2. Funktionsanalyse des Papier-Tinte-Systems

Generell startet jede Analyse mit **W-Fragen**. Wie bei allen innovativ-kreativen Prozessen stellt die Frage den intelligenten Teilprozess dar und nicht die Antwort.

- **Welches** Ziel haben die schwarzen Symbole auf dem Papier? Sie sollen eine Verhaltensveränderung bewirken.
- **Wessen** Verhalten soll verändert werden? Das mentale Verhalten des Lesers.

Durch diese beiden Fragen ist die Schnittstelle zwischen dem realen Objekt Papier-Tinte und den sensorisch-kognitiven Prozessen ausreichend charakterisiert.

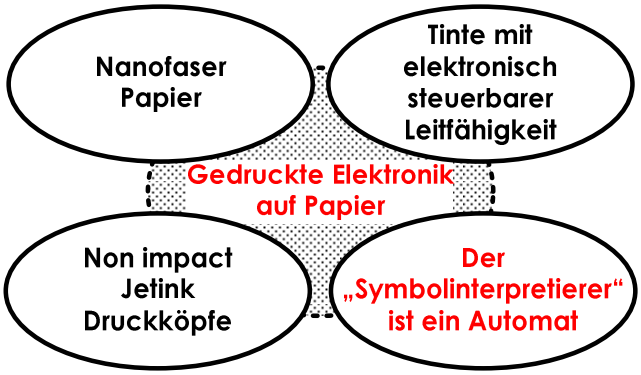


Abbildung 5 Das neue Druckparadigma

Der notwendige Paradigmenwechsel, um einen weiterführenden innovativ-kreativen Prozess zu generieren, ist eine dritte Frage, die über das vorhandene System hinausweist. Seit 550 Jahren war der Frageprozess an dieser Grenze abgeschlossen und alle Beteiligten waren mit dem Ergebnis zufrieden. Diese Frage könnte z.B. lauten:

- **Welche** anderen Ziele gibt es, deren Verhalten man mit gedruckten Symbolen verändern kann? Da die bekannten kognitiv-mentalen Systeme seit 550 Jahren bedient werden, verbleiben nur signal-, daten- oder informationsverarbeitende Systeme.
- **Wie** muss das System gestaltet sein, das auf Papier gedruckte Tintensymbole interpretieren kann? Sobald die Tintensymbole eine elektrotechnisch-elektronische Struktur bilden, die zusätzlich ein sensorisches und aktuatorisches Verhalten zeigen, ist eine Verhaltensveränderung möglich. Durch aktuatorische Reaktionen ist die Verhaltensänderung extern zusätzlich messbar und kann von weiteren Systemen genutzt werden.

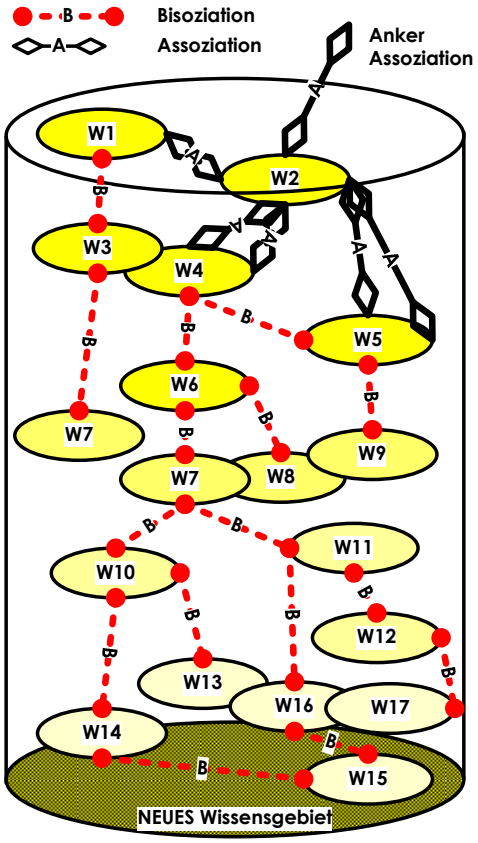


Abbildung 6 Entstehung eines neuen Wissensgebietes in einem mentalen Wissensnetzwerkes durch Transformationen von Bisoziationen in Assoziationen.

Abbildung 5 zeigt, vergleichen sie sie mit Abbildung 1, die schon vorhandenen Technologien. Sogar der Begriff „gedruckte Schaltungen / PCB – printed circuit board“ war schon lange, im Zusammenhang mit Epoxidsubstraten, bekannt. Trotzdem blieb der letzte konsequente Schritt der Erkenntnis lange verschlossen, bis auch auf Papier gedruckte Tintensymbole von einem Automaten interpretiert werden konnten.

Diese rückblickend durchgeführte Systemanalyse gibt Hinweise, mit Hilfe welcher Analyseprozesse man aus einem abgegrenzten und fixierten mentalen Modell (synonym: „Blick über den Tellerrand“) in einen vollkommen neuen Sinneszusammenhang gelangen kann und aus Bisoziationen mental aktiv nutzbare Assoziationen gewinnt. Mit Hilfe des dreiphasigen Entwicklungskreislaufs in Abbildung 3 überführen Menschen ihre Ideen in konkret fertigbare und messbare Produkte.

In Abbildung 6 wird schematisch dargestellt, wie der „Blick über den Tellerrand“ zu neuen Denkmustern führt und neue Assoziationen entstehen. Nachdem dieser erste, elementare Denkprozess erfolgt ist, können Standardverfahren wie TRIZ/TIPS (Teorija Rezhenija Jzobretatel´skich Zadach/Theory of Inventive Problem Solving) für die Implementierung und Realisierung der neuen Idee eingesetzt werden. Die TRIZ/-TIPS Prozesse sind mit dem Gutenberg´schen Innovationsprozess vergleichbar.

2.3. Fertigungstechnologien

Wie in der Einführung erwähnt, kann die Silizium Halbleitertechnologie unter Kostengesichtspunkten nicht alle denk- und wünschbaren Produkte realisieren. Alternative Herstellungstechnologien werden daher häufig relativ zur Silizium Halbleitertechnologie beurteilt (siehe Abbildung 7). Durch diesen Vergleich wird sofort ersichtlich, dass beide Fertigungstechnologien vollkommen unterschiedliche Produktbereiche bedienen und sich daher ergänzen und nicht in Konkurrenz zu einander stehen.

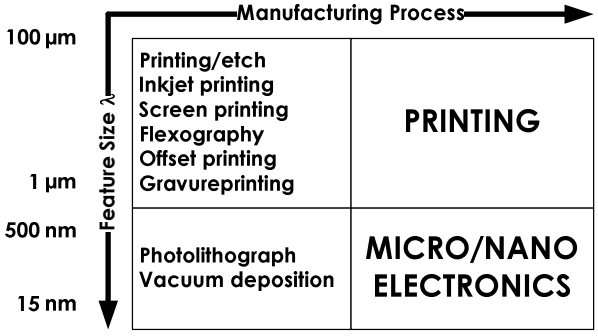


Abbildung 7 Fertigungstechnologien und ihre geometrischen Auflösungen λ in der Druckmedien- und Silizium-Halbleiter-Industrie [CHAS05]

Betrachtet man einschränkender den Bereich der RFID Tags, können die Paper RFID Tags mittelfristig die aktuellen Fertigungstechnologien in den niederen Frequenzbereichen ablösen, d.h. RFID Hersteller, die die aktuell eingesetzten Fertigungsprozesse beherrschen, müssen aufmerksam diese neu entstehenden Möglichkeiten beobachten [HENN05].

In Abbildung 7 werden die realisierbaren Auflösungsbereiche für die Druck- und die Halbleiterindustrie schematisch dargestellt. Folgende Eigenschaften charakterisieren drei der Druckprozesse:

- Tintenstrahl (inkjet printing): individuell fertigbare Komponenten (one-lot production)
- Siebdruck (screen printing): geringe Auflösung und Kosten
- Tiefdruck (gravure printing): hoher Durchsatz

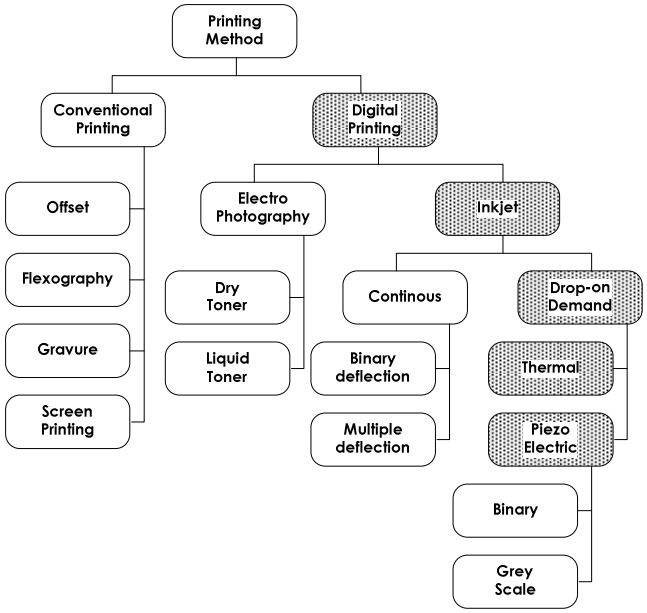


Abbildung 8 Verfügbare konventionelle und digitale Drucktechnologien; grau unterlegt, die hier diskutierten Verfahren [HAKO06]

Halbleiterfertigungsprozesse mit $\lambda = 500$ nm sind nur noch sehr beschränkt, für spezielle Anwendungen, im Einsatz. Die geometrische Auflösung $\lambda = 100 - 5$ µm für gedruckte Produkte, stellt den aktuellen Stand der Technik dar. Die Silizium Halbleitertechnologie hat sich innerhalb von 60 Jahren enorm weiterentwickelt. Vermutlich können die technisch wissenschaftlichen Erkenntnisse der Mikro/Nano-Technologien teilweise auf die unterschiedlichen sich entwickelnden Drucktechnologien übertragen werden, womit sie sich im Vergleich beschleunigt entwickeln werden.

Abbildung 8 demonstriert die technologischen Zusammenhänge der aktuell verfügbaren Drucktechnologien. Die hohe Diversität ermöglicht einen durchgängigen Abtausch der Kosten-Leistungsfähigkeit-Anforderungen bei unterschiedlichen Anwendungen. In Abbildung

7 und Abbildung 8 sind die Fertigungstechniken notiert, die aktuell für die Herstellung von gedruckter Elektronik einsetzbar sind. In diesem Artikel wird einschränkend nur die Inkjet Technik diskutiert.

2.4. Drop-on-Demand (DoD) Inkjet Drucktechnologie

Eine aktuell sehr wichtige Drucktechnologie ist das Drop-on-Demand Inkjet Verfahren. Folgende Attribute sind charakteristisch:

- substratunabhängig
- präzise, hohe Auflösung
- hohe Geschwindigkeit (1 m/s)
- eine Massenproduktion ist möglich
- digitale, anschlagfreie Druckmethode
- geringer Materialverbrauch, da additiv
- Tinten für alle Anwendungsfälle sind verfügbar, die Tintenentwicklung ist aber nicht trivial
- die Inkjet Technologie kann einfach mit existierenden Produktionslinien kombiniert werden

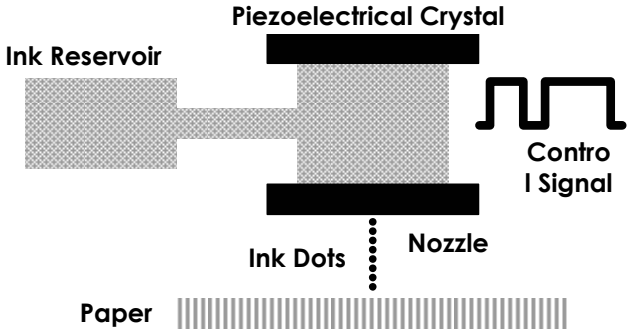


Abbildung 9 Drop-on-demand Piezoelektrische Inkjet Technik der Tintenaufbringung [GUPT07]

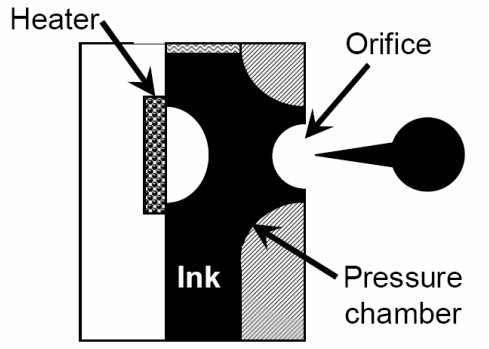


Abbildung 10 Drop-on-demand Inkjet [HAKO06]

In Abbildung 9 ist das Drop-on-demand Inkjet Grundprinzip schematisch dargestellt. Bei dieser Prozessführung dient ein piezoelektrischer Kristall als Aktuator. Der Kristall wird mit digitalen Signalen angesteuert und beschleunigt eine sehr kleine Tintenmenge von ca. 10 pl (Picoliter) in Richtung auf das Papiersubstrat. In Abbildung 10 wird die thermische „drop-on-demand" Technik mit Hilfe eines Heizelements vorgenommen, das dafür sorgt, das ein Tintentropfen aus der Öffnung (orifice) einer Druckkammer herausgepresst wird.

2.5. Inkjet System - Prozessoptimierung

Für die Entwicklung der Inkjet Fertigungstechnologie müssen die Prozess bestimmenden Strukturen und ihr Verhalten bekannt sein und Schritt für Schritt optimiert werden. Dieses Vorgehen wurde schon in der einführenden Betrachtung diskutiert und taucht auf dieser Betrachtungsebene wiederum auf.

In Abbildung 11 werden die konkreten und die generalisierten, Prozess bestimmenden Strukturen, gegenübergestellt. Struktur und Verhalten der vier Komponenten müssen kooperativ optimiert werden, um ein globales Optimum zu erreichen.

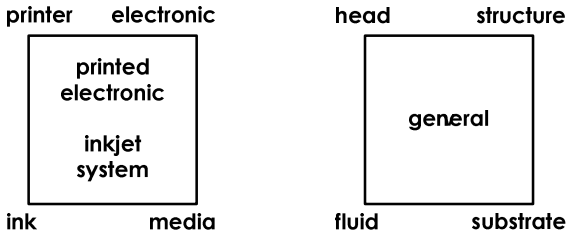


Abbildung 11 Inkjet Prozess; links: die vier wesentlichen konkreten Prozessobjekte; rechts: die vier generellen Prozessobjekte [HODG07]

Der Druckkopf (head), die Tinte (fluid), das Papiermedium (substrate) und die zu realisierende elektronische Schaltung (structure) stellen ein zusammenhängendes System dar. Um eine Kosten- und Verbrauchsvorstellung zu haben, werden der konventionelle und der Inkjet Druckprozess verglichen [HAKO05]:

Konventionelle Drucktechnik

- leitendes Polymer ~ 400 €/kg
- Polymerverbrauch ~ 0.8 €/m^2
- Metallpartikeltinte ~ 2.000 – 20.000 €/kg
- Tintenverbrauch ~ 2 g/m^2
- Metallpartikeltinte 4 - 40 €/m^2
- Zusätzliche Kosten: Druckplatte

Inkjet Drucktechnik

- leitendes Polymer ~ 400 €/kg
- Polymerverbrauch ~ 0.4 €/m^2
- Metallpartikeltinte ~ 5.000 – 10.000 €/kg
- Tintenverbrauch ~ 1 g/m^2, eine Tintenebene
- Tintenverbrauch 5 - 10 €/m^2

Schlussfolgerung, die Inkjet ist kostengünstiger als die konventionelle Drucktechnik. In den folgenden Kapiteln werden die Systembestandteile einzeln betrachtet. Abschließend werden die erreichten elektrotechnischen und elektronischen Parameter diskutiert.

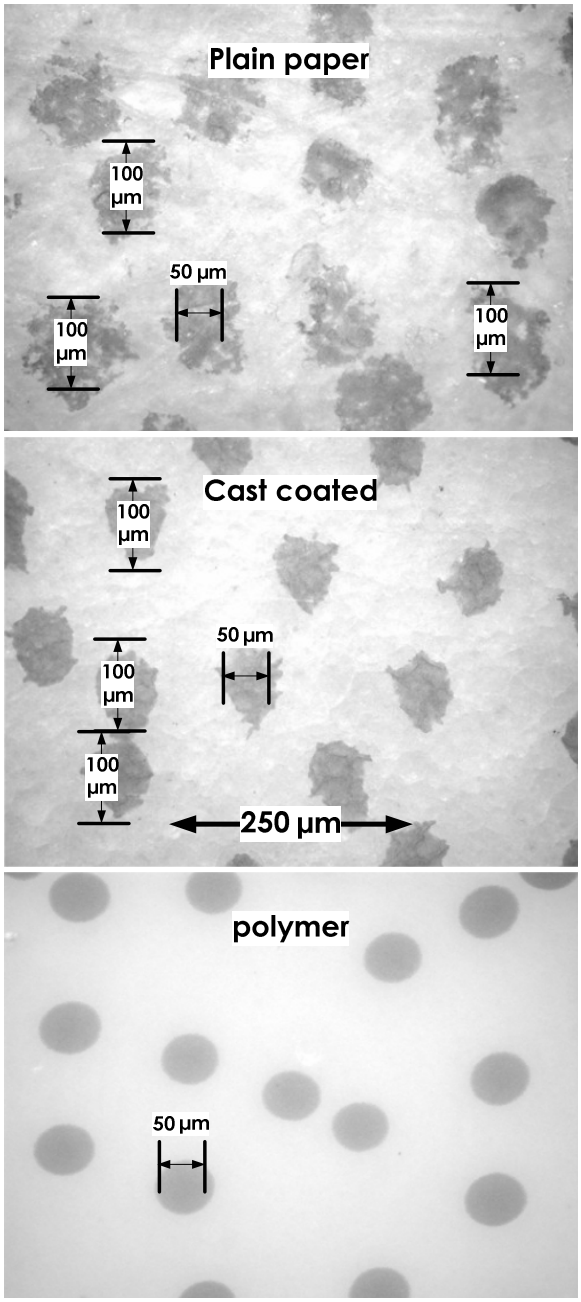


Abbildung 12 Unterschiedliche Papierqualitäten – oben: unbehandeltes Papier; mittig: Papier mit einer Oberflächenvergütung; unten: Polymer [HODG07]

2.6. Eigenschaften des Papiersubstrats

Die Druckqualität kann sehr gut visuell charakterisiert werden, wie man in Abbildung 12 sehen kann. Die drei dargestellten unterschiedlichen Substratrauhigkeiten (unbehandeltetes Papier, oberflächenvergütetes Papier, Polymer) transformieren die aufgetroffenen Tintentropfen zu unterschiedlichen geometrischen Formen, die wiederum die Güte der so realisierten elektrischen oder elektronischen Komponenten stark beeinflussen.

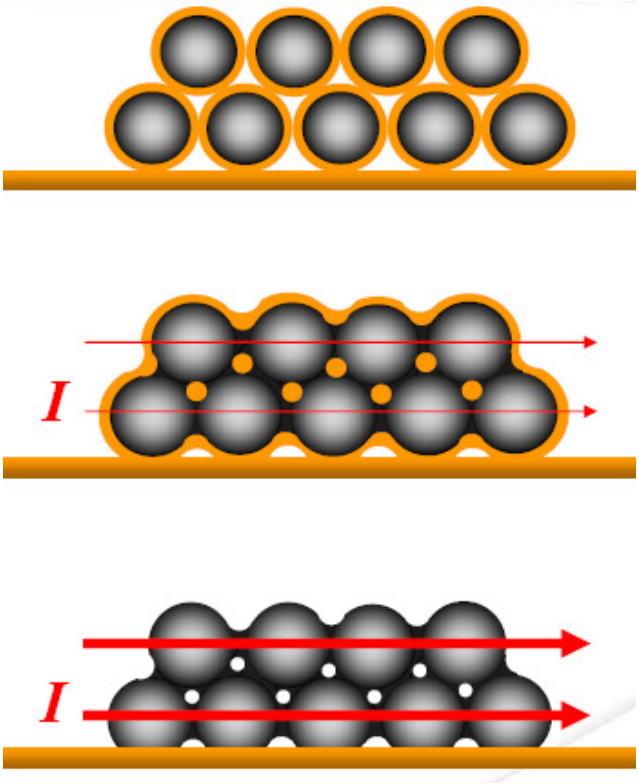


Abbildung 13 [HAKO05] Abnehmender Leitungswiderstand mit ansteigender Tempertemperatur - oben: Raumtemperatur, kein Stromfluss möglich; mittig: bei 125° getempert; unten: bei 550° getempert

Je fraktaler die Tintentropfenrandstruktur ausgeprägt ist, verursacht durch die Substratrauhigkeit und die Faserstruktur, umso schlechter sind die resultierenden elektrischen oder elektronischen Parameter, was wiederum dazu führt, dass man beim Schaltungsentwurf mit größeren Parameterstreuungen rechnen muss.

Die elektrischen oder elektronischen Parameter können durch nachfolgende Temperprozesse noch gezielt beeinflusst werden, wie in Abbildung 13 dargestellt.

Je kürzer die an der Substratoberfläche auftauchenden Faserstrukturen sind, umso mehr nähert sich die

Tintentropfenrandstruktur einer ganzzahligen Dimension mit optimalen elektrischen oder elektronischen Parametern. D.h. die Papierhersteller werden für diesen Anwendungsbereich in den Bereich nanometrischer Faserdimensionen gehen, um den unerwünschten Transformationsprozess zu minimieren.

Bei dieser Entwicklung ist natürlich permanent zu untersuchen, wie das Konkurrenzsubstrat Polymer weiterentwickelt wird.

2.7. Druckkopf

Bei der Halbleiter Fertigungstechnik spielt bei den fotolithografischen Prozessen, die verfügbare Lichtquelle, die eingesetzten Optiken und Masken eine hervorragende Rolle. Bei gedruckter Elektronik ist dies der Druckkopf und seinen Fähigkeiten möglichst kleine Tintentropfen zu erzeugen und so präzise wie möglich auf ein Substrat zu spritzen.

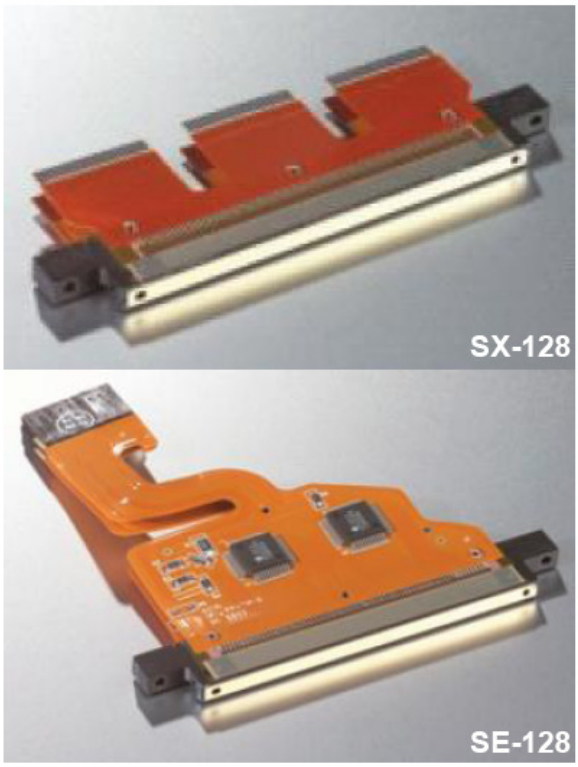


Abbildung 14 [HAKO05] Dimatix Inkjet Druckköpfe SY-128 und SE-128

Zwei Dimatix Druckköpfe für Inkjet gedruckte Elektronik sind in Abbildung 14 abgebildet. Die SX-128 Köpfe weisen folgende Prozessparameter auf:

- Tropfenparameter: 10 Picoliter bei 27 µm Durchmesser
- Nutzbar für Tintenflüssigkeiten mit einem pH Wert von bis zu 1,5, wie z.B: bei Polymeren
- Jede Düse kann individuell gesteuert werden

2.8. Gedruckte FETs und ihre elektronischen Eigenschaften

In [MOLE05] werden gedruckte organische Transistoren vorgestellt, die im sub 10V V_{DD} Bereich arbeiten. Als Fertigungstechnologie werden leitfähige Nanopartikel, Polymerdielektrika und Pentacen Halbleiter mit einem Inkjetprozess aufgebracht.

Die auf Plastiksubstrat realisierten Komponenten weisen eine Beweglichkeit von > 0.05cm^2/V-s und ein on/off Widerstandsverhältnis von 10^5 auf.

Man erkennt in Abbildung 15, dass das Substrat mit PVP (poly-4-vinylphenol) beschichtet wurde. Dies entspricht der mittleren Struktur in Abbildung 12 und gibt Hinweise über die Abbildungsqualität der Tintentropfen. Das Gate Dielektrikum basiert auf einem durch Infineon entwickelten Prozess [KLAU02]. Der eingesetzte Halbleiter ist ein lösliches Pentacen, das von IBM entwickelt wurde [AFZA02].

Mit den in [MOLE05] entwickelten Fertigungstechniken erzielt man geringere Gate-Source und Gate-Drain Überlappungen, womit sich eine verbesserte FET AC Leistungsfähigkeit realisieren lässt und so einen RFID Betrieb mit Betriebsfrequenzen von > 200 kHz ermöglicht.

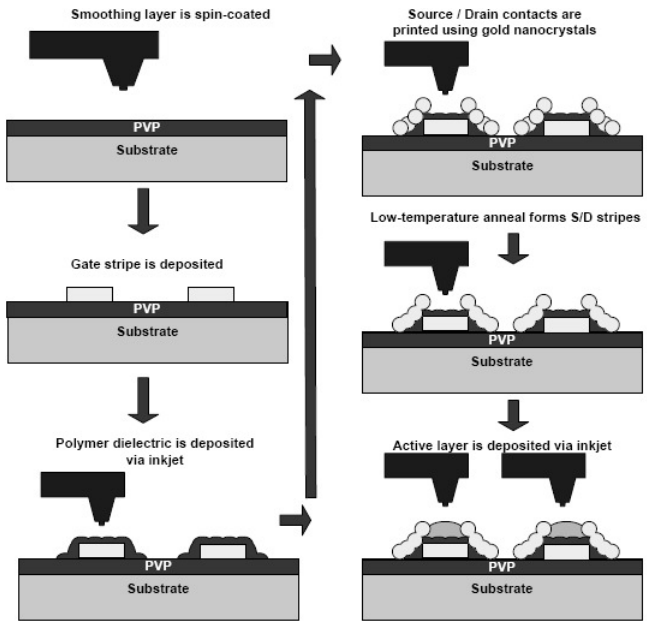


Abbildung 15 Fertigungsprozessfluss für die Realisierung eines gedruckten FET Transistors [MOLE05]

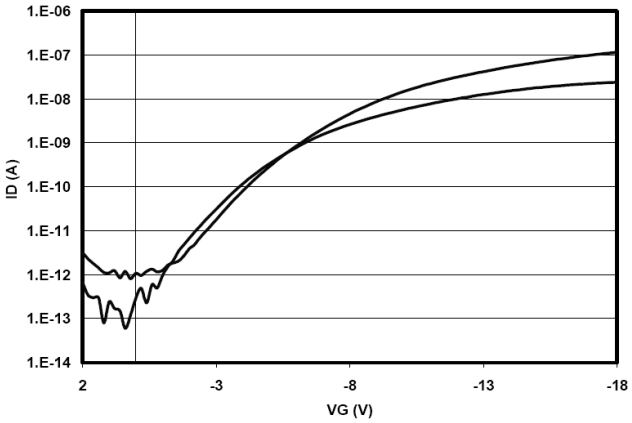


Abbildung 16 Übertragungscharakteristik für ein 26µm/27µm Transistor mit t_{ox} ~ 45nm, µ=0.05cm²/V-s und I_{on}/I_{off} ~ 10^5 [MOLE05]

Die resultierenden FET Übertragungs- und Ausgangscharakteristiken in Abbildung 16 und Abbildung 17 und die realisierte Beweglichkeit lassen sich für neue RFID Produkte, in dem kostensensitiven Bereich einsetzen (siehe Abbildung 4).

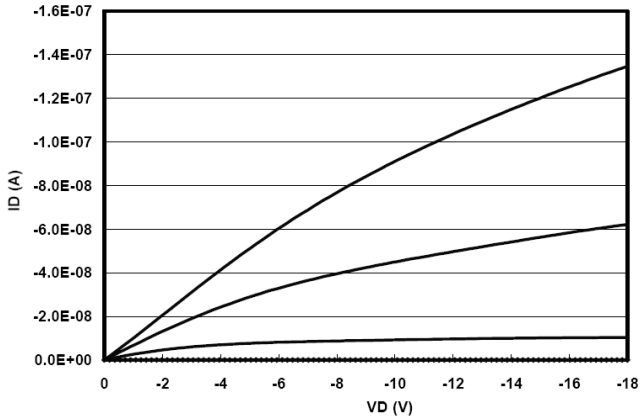


Abbildung 17 Ausgangscharakteristik für ein 26µm / 27µm FET mit einer Gateoxiddicke von t_{ox} ~ 45nm und einer Beweglichkeit von µ =0.05cm²/V-s [MOLE05], [LÖGD05]

2.9. RFID-Tags - Komponentenintegration

Der letzte Entwicklungsschritt ist die Integration der gedruckten elektrotechnischen Komponenten, wie niederohmige Leitungen, Widerstände, Kapazitäten, Induktivitäten, Antennen, Batterien und Akkumulatoren und den elektronischen Komponenten wie Dioden, OLED, Transistoren zu einem Gesamtsystem, wie z.B. zu einem RFID Tag (siehe Abbildung 18 und Abbildung 19) [KAHN03] [HARV04] [MOLE05] [NORB05] [SHAH05] [SIDE05] [DUPO06] [FERR06] [TENT06] [YANG06] [LIRO07] [RIDA07] [GEOR08] [MÄKE08] [NILS08].

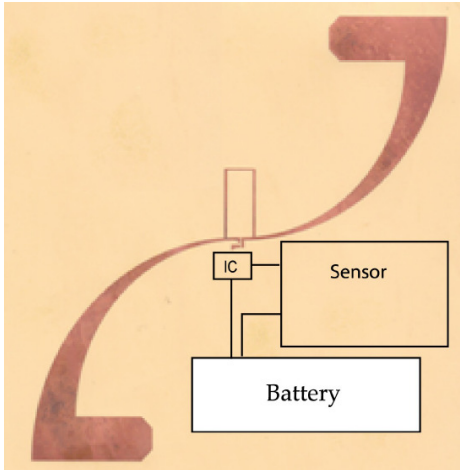


Abbildung 18 Papier RFID Tag mit IC, Antenne, Sensor und Batterie [TENT06] [DUPO06]

Besonders die Ausgestaltung der Antenne stellt ein wichtiges konstruktives Merkmal dar [NORD05]. Einige Autoren untersuchen mittlerweile auf Papier gedruckte UHF RFID Tags.

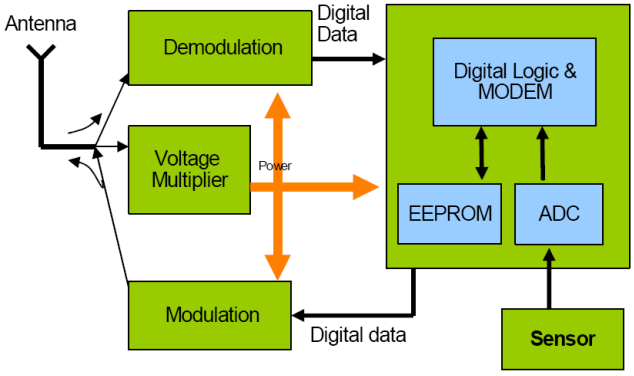


Abbildung 19 Blockplan eines RFID Tags

3. Schlussfolgerungen

Die Forschung und Entwicklung gedruckter Elektronik auf Papiersubstrat wird weltweit betrieben [HARV04] [BERG05] [TENT06] [HODG07] [GEOR08]. Alle für einen Fertigungsprozess benötigten Komponenten stehen aus industrieller Fertigung zur Verfügung. Die realisierbaren elektrotechnischen und elektronischen Komponenten weisen Parameter auf, die den Einsatz in RFID Tags ermöglichen. Es können FETs, Batterien und Antennen realisiert werden, die zu extrem kostengünstigen Tags führen. Es sollte generell untersucht

werden, ob diese Technologie auch für den Bereich der kostengünstigen Prototypenentwicklung eingesetzt werden kann, um z.B. ohne gedruckte Schaltungen auf Expoxidbasis auszukommen und kürzere Entwicklungszeiten realisieren zu können.

4. Literatur

[AFZA02] Afzali, et al, JACS Comm., 2002, IBM

[BERG05] Magnus Berggren, Michael Lögdlund, COIN - Center for Organic INformatics, Aperturen 2/2005, Aperturen2005-02.pdf

[CHAS05] Chason, M., Brazis, P.W.,Jr, Jie Zhang, Kalyanasundaram, K. and Gamota, D.R., Printed organic semiconducting devices, Proc. the IEEE, 93(2005)1348-1356

[DONA93] Merlin Donald, Origins of the Modern Mind – Three Stages in the Evolution of Culture and Cognition, Harvard, 1993, ISBN 0-674-64484-0

[DUPO06] 5033 Screen-Printable Conductive Ink for RFID Antennas, DuPont Microcircuit Materials, Application Profile, K-10797 03/06, ASK_profile.pdf

[EINS22] Albert Einstein, Space-Time, Encyclopædia Britannica in the Twelfth Edition, 1922, http://preview.britannica.co.kr/-spotlights/classic/eins1.html

[FERR06] Antonio Ferrer-Vidal, Amin Rida, Serkan Basat, Li Yang, and Manos M. Tentzeris, Integration of Sensors and RFID's on Ultra-low-cost Paper-based Substrates for Wireless Sensor Networks Applications, 2006 IEEE, WiMeshNet06.pdf

[GEOR08] Georgia Institut of Technology, 2008, printedelectronics.pdf

[GUPT07] Deepak Gupta, Organic Electronics, Material & Metallurgical, 2007, Engineering, MFT_02_DeepakGupta.ppt.pdf

[HAKO05] Liisa Hakola, Benefits of inkjet printing for printed electronics, Pira Printed Electronics 14.15.9.2005,London, UK, VTT Information Technology, PIRA Printed Electronics 2005 Hakola.pdf

[HAKO06] Liisa Hakola, Inkjet Printing For Making Fine Conductors and MultiLayer Electronics, VTT –Technical Research Centre of Finland, 2006, RT2006_Hakola.pdf

[HARV04] James E. Harvey, Printing Electronics Applications of Nanotechnology & Microtechnology in Printing and the Graphic Arts, 2004, Printing_Electronics_-13April04.pdf

[HENN05] Lars-Olov Hennerdal, Michael Lögdlund, Electronic Paper Printing House, Aperturen 2/2005, Aperturen2005-02.pdf

[HODG07] Alan Hodgson, The role of paper in the future of printed electronics, 2007, 3.pdf

[KAHN03] Bruce E. Kahn, Printing Radio Frequency Identification (RFID) Tag Antennas Using Inks Containing Metal Nanoparticles, 2003, Particles 2003.pdf

[KALL08] Elja Kallberg, Intelligent Papers, Master's Thesis University of Jyväskylä, 2008, URN_NBN_fi_jyu-2006462.pdf

[KLAU02] H. Klauk, et al, Journal of Applied Physics, Vol. 92, pp. 5259, 2002, Infineon

[LIRO07] Li-Rong Zheng, - Ubiquitous Intelligence in Paper and Packaging - A Short Introduction, VINN Excellence iPack Center, 2007, iPack_short_Introduction-_200708.pdf

[LÖGD05] Michael Lögdlund, Tommi Remonen, Organic Electronics, Aperturen 2/2005, Aperturen2005-02.pdf

[MÄKE08] Tapio Mäkelä, Towards printed electronic devices Large-scale processing methods for conducting polyaniline, ESPOO 2008, VTT PUBLICATIONS 674, P674.pdf

[MOLE05] Steven E. Molesa, Alejandro de la Fuente Vornbrock, Paul C. Chang, and Vivek Subramanian, Low-voltage inkjetted organic transistors for printed RFID and display applications, 2005, IEDM05.pdf

[NILS08] H-E. Nilsson, C.G. Mattsson, J. Sidén, U. Geyer, Printing of UHF modulators on embedded silicon in semi-passive RFID, 2008, 050-Nilsson.pdf

[NORB05] Petronella Norberg, RFID - Radio Frequency Identification, Aperturen 2/2005, Aperturen2005-02.pdf

[NORD05] Staffan Nordlinder, Dry Phase Patterning - New concepts or the patterning of flexible substrates, Aperturen 2/2005, Aperturen2005-02.pdf

[OLED__] OLED History, www.oled-info.com/history

[RIDA07] Amin Rida, Li Yang, Rushi Vyas, Swapan Bhattacharya, and Manos M. Tentzeris, Design and Integration of Inkjet-printed Paper-Based UHF Components for RFID and Ubiquitous Sensing Applications, Proceedings of the 37th European Microwave Conference, 2007 EuMA, rida5.pdf

[SHAH05] Jayesh Shah, Bo Xia, Stone Cheng, Wanda O'Hara, and Vito Buffa, Low Temperature, Spot Cure Conductive Adhesive Technologies For Rfid Tags Assembly, 2005, 2005SMTALowtempspotcureconductiveadhesives.pdf

[SIDE05] Johan Sidén, Torbjörn Olsson, Andrei Koptioug, Hans-Erik Nilsson, Reduced Amount of Conductive Ink with Gridded Printed Antennas, 2005, reduced_-amount_of_conductive_ink.pdf

[TENT06] Manos M. Tentzeris, GEDC RFID ACTIVITY, June 16, 2006, Manos.pdf

[YANG06] Li Yang, Serkan Basat, Amin Rida, and Manos M. Tentzeris, Design and Development of Novel Miniaturized UHF RFID Tags on Ultra-low-cost Paper-based Substrates, Proceedings of Asia-Pacific Microwave Conference 2006, APMC06_Li_Final.pdf

Sicherer Entwurf von Hochfrequenzschaltungen durch Verwendung eines echten statistischen Designkits

Hans-Joachim Wassener

Director R&D Atmel RFA

Übersicht

- Einleitung
- Was ist TRADICA?
- Allgemeine Arten von Prozeßschwankungen
- Arbeiten im Förderprojekt DETAILS
- Das TRADICA-Prinzip
- TRADICA im Design-Flow
- Arten der statistischen Simulation
- Beispiel für Optimierung per statistischer Simulation
- Messung der Prozeßschwankungen
- Loop Closure
- Weitere Beispiele
- Ausblick

Einleitung

- Ein Schaltkreis sollte so entworfen werden können, daß er unter Berücksichtigung aller Prozeßschwankungen wirtschaftlich entwickelt und produziert werden kann.
- Alle dazu notwendigen Tools und Informationen sollten optimal und anwenderfreundlich im Design-Flow eingebunden sein.
- Die Tools müssen in der Lage sein, die Schwankungen des Prozesses abzubilden.

Workshop der MPC-Gruppe Juli 2008

Was ist TRADICA?

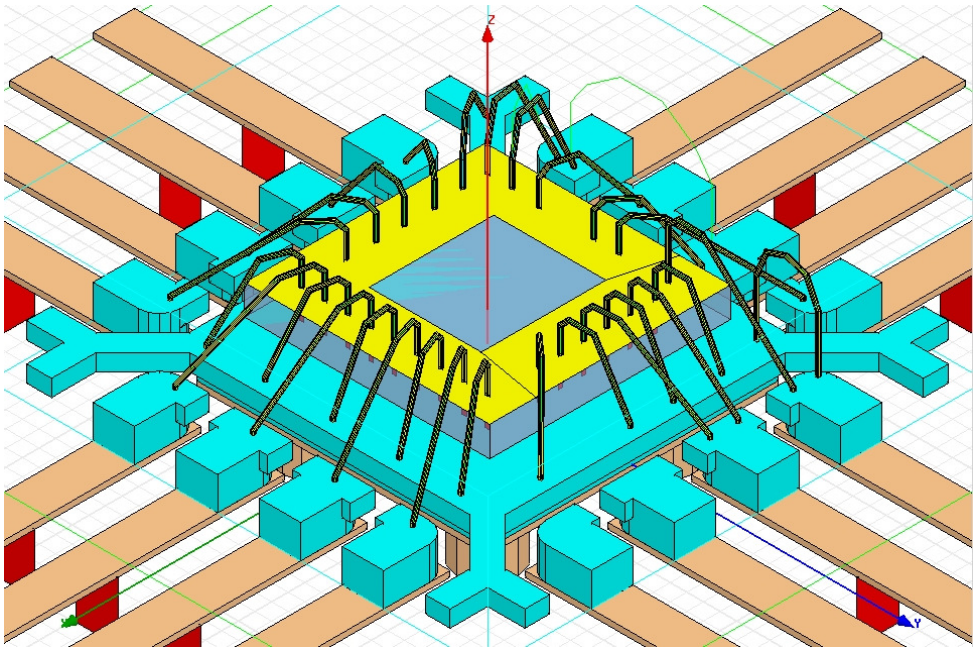
- Ursprünglich ein kleines Programm zur Dimensionierung und Berechnung von Transistor-Parametern für einen Analogrechner und Simulationen (ca. ab 1970 bei der AEG-Telefunken in Heilbronn).
- Einsatz auch bei der Entwicklung des weltweit ersten Frequenzteilers mit monolithisch integriertem Vorverstärker (1979).
- Seitdem kontinuierlich weiterentwickelt zu einem Expertensystem zum
  - Dimensionieren von Transistoren
  - Support verschiedener Modell-Level von BJT (SGP und HICUM)
  - Erzeugen von Modellen für statistische Simulationen
  - Support auch der passiven Bauteile einer Technologie

Allgemeine Arten von Prozeßschwankungen (1)

- Für die
  - Technologie: Abweichungen vom typischen Wert für Prozesse in der Wafer-Fabrik (z.B. ein Schichtwiderstand)
  - Schaltung: Streuungen von elektrischen Parametern (z.B. der Wert eines Widerstandes)
  - Layout: Einfluß auf elektrische Parameter (z.B. Präzision, Matching, parasitäre Effekte, Substratverkopplung)
  - Bonddraht: parasitäre Effekte durch Variation z.B. der Länge, Loop-Form
  - Gehäuse: parasitäre Effekte durch Variation z.B. der Kantenabrundung, Dielektrizitätskonstante der Moldmasse
  - Applikation: Schwankung der Parameter einer Platine, von Bauteilen

Workshop der MPC-Gruppe Juli 2008

Allgemeine Arten von Prozeßschwankungen (2)



Arbeiten im Förderprojekt DETAILS

- Berücksichtigung von Prozeßschwankungen:
  - Prozeßparameterschwankungen
  - Verbesserung der Ausbeute
  - BISC
  - Systematische Empfindlichkeitsanalyse

- Berücksichtigung von Prozessen:
  - Parasitäre Effekte
  - System-in-Package
  - Isolationsmaßnahmen
  - Elektro-thermische Kopplungen

Workshop der MPC-Gruppe Juli 2008

Das TRADICA-Prinzip (1)

- Es gibt 2 verschiedene Methoden, statistische Daten zu gewinnen:
  - aus elektrischen Parametern
  - aus Prozeßdaten

- Verwendung elektrischer Parameter
  - offensichtliche Methode für Schaltungsentwickler
  - einfache Gewinnung aus den PCM-Messungen
  - viele Parameter für > 100 Modellparameter
  - komplizierte statistische Abhängigkeit zwischen Parametern
  - Gefahr von „mathematischem Zahlenschrott“
  - langer Beobachtungszeitraum notwendig

Das TRADICA-Prinzip (2)

- Verwendung von Prozeßdaten
  - geht nur mit Expertenwissen -> TRADICA
  - Prozeßparameter sind per se statistisch unabhängig
  - nur geringe Zahl von Input-Parametern
  - Mittelwerte sind schon ohne Silizium bestimmbar
  - alle Bauelemente eines Prozesses sind korrekt statistisch verkoppelt
  - Prozeßdaten werden in der Produktion nicht gemonitort -> „closing the loop“

- Beim Vergleich beider Methoden erweist sich die Verwendung von Prozeßdaten als überlegen. Das TRADICA-Prinzip muß allerdings in den Design-Flow integriert werden.

Workshop der MPC-Gruppe Juli 2008

Das TRADICA-Prinzip (3)

Fixed variable	Included in device	Remark	Mean value	± sdev [1]
d_rsbi	NPN		1	0.1
d_rspm	NPN		1	0.1
d_re	NPN		1	0.12
d_rsbl	NPN		1	0.08
d_cjei	NPN		1	0.06
d_cjci	NPN, ESD1-3	Specific Cjc bottom	1	0.07
d_roci	NPN		1	0.15
d_rosu	NPN		1	0.05
d_cjep	Not used.		1	0
d_cjcb	Not used.		1	0
d_cjcp	Not used.		1	0

Variable	Included in device	Remark	Mean value	± sdev [1]
d_cjsi	ESD1-3,DYTSNI1	Specific Cjs bottom	1	0
d_cjsp	DYTSNI1,DISUNL 1	Specific Cjs pheripheral	1	0
d_coxb	CNA1TS1,RNYM1, RPYH1,RPYL1, PA2SU20801	Parasitic substrate capacitance	1	0
d_cnitb	CNA1TS1	Specific cap.	1	0
d_rspyl	RPYL1	Sheet res.	1	0.29 [2]
d_rspyh	RPYH1	Sheet res.	1	0.111 [2]
d_rsnym	RNYM1	Sheet res.	1	0.065 [2]
d_bflpnp	LPNP	DC Beta	1	0.3
d_bflpnpb	LPNP_NEW	DC Beta	1	0.3

TRADICA im Design-Flow (1)

- TRADICA erzeugt kein skalierbares einzelnes BJT-Modell, sondern eine definierbare Vielzahl von „fertig skalierten“ Modellen.
- Zwei unterschiedliche Wege sind möglich:
  - statisch: statistische Informationen werden im Modell durch weitere Gleichungen zur Verfügung gestellt. -> Atmel
  - dynamisch: pro Simulationslauf erzeugt TRADICA ein temporäres Modellfile -> Cadence

Workshop der MPC-Gruppe Juli 2008

TRADICA im Design-Flow (2)

- Die Implementierung bei Atmel umfaßt folgende Maßnahmen
  - Normierung der Parameter (z.B. Mittelwert eines Widerstandes 1, Schwankungsbreite von 0.8 bis 1.2, also +- 20%).
  - Einfügen von Schaltern, um selektiv die statistische Simulation aktivieren zu können.
  - Einfügen von Gleichungen, um die Abhängigkeit elektrischer Parameter von Prozeßparametern abzubilden.
  - Alle o.g. Maßnahmen wurden in einer speziellen TRADICA-Version implementiert.
  - Mit dieser Spezialversion wird ein Modell-File erzeugt und in das Design-Kit eingebaut.
  - Anpassung des Design-Kits (GUI), um die verschiedenen Arten der statistischen Simulation durchführen zu können.

TRADICA im Design-Flow (3)

Implementierungsbeispiel Widerstand

```
subckt RPYH11 d s sub

parameters \
R=2000 W=1.6u WMAT=0 NS=1 NP=1

xmac d s sub macro r=R w=W nser=NS npar=NP

subckt macro d s sub

parameters
+ r      = 2000          w      = 1.6u    npar = 1    nser = 1
+
+ wcmin  = 1.6u    wc_s = 0.8u
```


TRADICA im Design-Flow (4)

Implementierungsbeispiel Widerstand

```
+ wrw    = 0             wrc    = 0.4u
+ lc_s   = 0.8u          lrw    = 0       lrc  = 0.4u
+
+ cox_sa = 48a
+ rcell  = r*npar/nser
+ um     = 1u        um2 = 1p
+
+ cox    = cox_sa
+
+ wct    = (w>=wcmin)? w : wcmin
+ wc = wct - 2*wrc
+
//
// Calculating length from nominal parameters only.
//
+ rc      = rcs_rpyh1_n/wc*um
+ l       = (rcell-2*rc)*(w+2*dw_rpyh1_n)/rsh_rpyh1_n
+ error   = sqrt(l)
//
// Calculating Mismatch
//
+ sig_rsh = ARSH_rpyh1/sqrt(w*l)
+ sig_dw  = ADW_rpyh1/sqrt(l)
+ sig_rcs = ARCS_rpyh1
//
// Calculating Local Variations
//
+ rsh_lrdev = 3*sig_rsh*rsh_rpyh1_n01_mm
+ rcs_lrdev = 3*sig_rcs*rcs_rpyh1_n01_mm
+ dw_ladev  = 3*sig_dw*dw_rpyh1_n01_mm
//
// Calculating Model
//
+ rvalt   = (rsh_rpyh1*(1+rsh_lrdev)*l)/(w+2*(dw_rpyh1+dw_ladev))*nser/npar
+
```

TRADICA im Design-Flow (5)

Implementierungsbeispiel Widerstand

```
+ rcon    = rcs_rpyh1*(1+rcs_lrdev)/wc*nser/npar*um
+
//
// Calculation of Capacitance
//
+ lr   = l+2*(-lrc)
+ lct  = lc_s+2*lrc+dw_rpyh1_n
+ arc  = lr*(w+2*dw_rpyh1_n)
+ act  = 2*(wct+2*dw_rpyh1_n)*lct
+ ccox = cox*(arc+act)*nser*npar/um2

rcd d a resistor r=rcon      tc1=-138u   tc2=1u
rnd a c resistor r=rvalt/2 tc1=-1.886m   tc2=7.576u
rns b c resistor r=rvalt/2 tc1=-1.886m   tc2=7.576u
rcs s b resistor r=rcon      tc1=-138u   tc2=1u

cd  a sub capacitor c=ccox/6
cn  c sub capacitor c=2*ccox/3
cs  b sub capacitor c=ccox/6

ends macro

ends RPYH11
```


Workshop der MPC-Gruppe Juli 2008

Arten der statistischen Simulation (1)

- Simulation mit den typischen Werten
- Cornersimulation
- Parametersweep
- Statistische MC-Simulation „Process only“
- Statistische MC-Simulation „Mismatch only“
- Statistische MC-Simulation „Process & Mismatch“
- Beschränkung der Statistik auf selektierte Instanzen

Arten der statistischen Simulation (2)

	Mittelwert μ		Standardabw. σ	
Sicherheit:	90 %	98 %	90 %	98 %
Runs n				
3	±2,5 s	±5,7 s	-42 / +342 %	-53 / +897 %
5	±1,2 s	±2,1 s	-35 / +137 %	-45 / +267 %
10	±0,72 s	±1,0 s	-27 / +65 %	-36 / +108 %
30	±0,37 s	±0,50 s	-17 / +28%	-24 / +43 %
100	±0,20 s	±0,26 s	-10 / +13 %	-14 / +20 %
300	±0,11 s	±0,15 s	-6,3 / +7,2 %	-8,7 / +10,5 %
1000	±0,06 s	±0,08 s	-3,5 / +3,8 %	-5,0 / +5,5 %

s = Std-Abw. der Stichprobe
m = Mittelwert der Stichprobe

$$-t_{\gamma,n-1}\frac{s}{\sqrt{n}} \le \mu \le \bar{x} + t_{\gamma,n-1}\frac{s}{\sqrt{n}}$$

$$\frac{(n-1)s^2}{\chi^2_{\frac{\alpha}{2};n-1}} \le \sigma^2 \le \frac{(n-1)s^2}{\chi^2_{\frac{1-\alpha}{2};n-1}}$$

Quantile der Student-Verteilung

- **Beispiel: Nach 5 Runs liegt**
  - **der Mittelwert μ der Grundgesamtheit mit einer Sicherheit von 90 % im Intervall m - 1,2 s < μ < m + 1,2 s**
  - **die Standardabweichung σ der Grundgesamtheit mit einer Sicherheit von 90 % im Intervall 0,65 s < σ < 2,37 s**

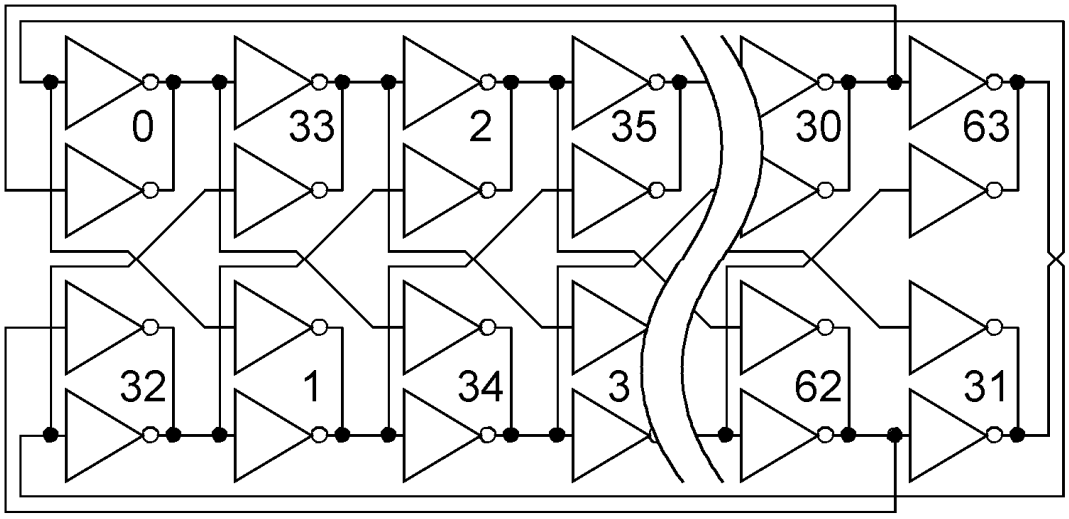
Workshop der MPC-Gruppe Juli 2008

Beispiel für die Optimierung per Statistik (1)

- **Ein als kritisch erkannter Block wird nur mit typischen Werten entwickelt.**
- **Der Block wird dann mit dem statistischen DK erneut simuliert, um die kritischen Bauteile zu ermitteln.**
- **Modifikation derselben, um den Einfluß der Prozeßschwankungen zu reduzieren.**
- **Resimulation des Ergebnisses und Vergleich mit dem ursprünglichen Design.**

Beispiel für die Optimierung per Statistik (2)

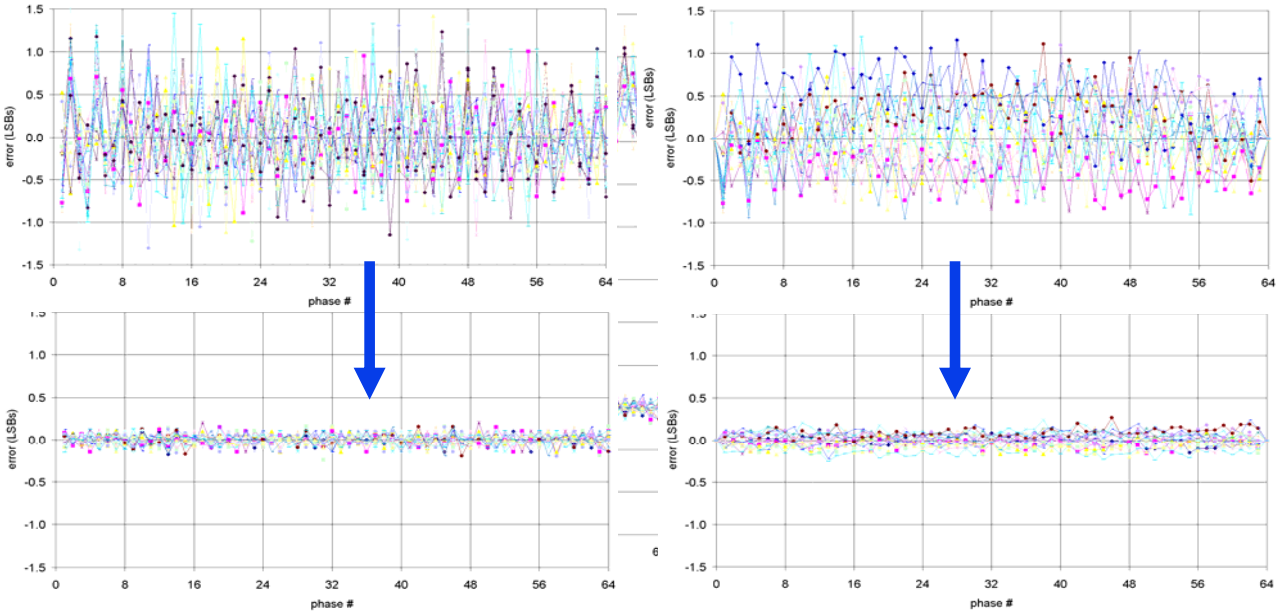
Beispiel PLL und Ring-VCO mit 64 Phasen



Beispiel für die Optimierung per Statistik (3)

Differentieller Fehler

Integraler Fehler



Messungen der Prozeßschwankungen

- Hier werden prinzipiell 3 Phasen unterschieden:
  - Messung an TRADICA-Strukturen. Diese speziellen Strukturen dienen zum Ermitteln von Prozeßparametern als Input für TRADICA.
  - Messungen der (elektrischen) PCM-Parameter. Hiermit werden (indirekt) sowohl die Prozeßstreuungen als auch der Mismatch erfaßt.
  - Messung an Mismatch-Strukturen. Diese speziellen Strukturen dienen zum Ermitteln des Mismatch-Anteils.
- Die Daten können per TRADICA auf einen gewünschten Mittelwert „verschoben“ werden. Somit ist kein goldener Wafer notwendig.
- Ebenso können mathematisch die noch notwendigen Prozeßparameter erzeugt werden.

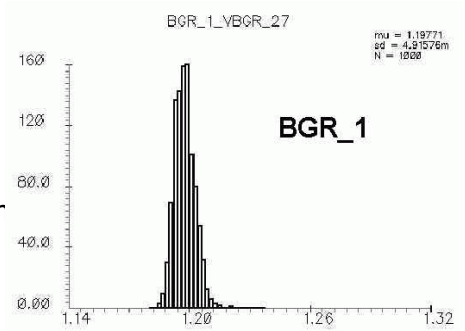
Workshop der MPC-Gruppe Juli 2008

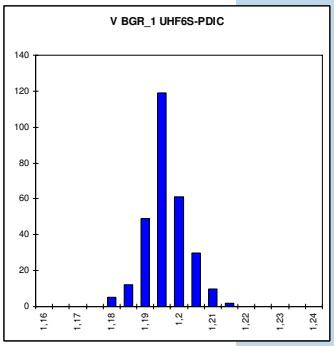
Loop Closure

- Die statistischen Basisdaten werden am Ende des Wafer-Durchlaufs durch Messung von PCM-Strukturen ermittelt.
- Aus ihnen wird auf die Schwankungen der technologischen Prozesse rückgeschlossen.
- Diese statistisch unabhängigen Prozesse gehen mit Mittelwert und Streuung als Parametersatz in TRADICA ein.
- TRADICA erzeugt die Simulationsmodelle.
- Werden jetzt mit den statistischen Modellen die PCM-Strukturen so simuliert, wie sie auch gemessen werden, und stimmen dann gemessene und simulierte Ergebnisse inklusive der Verteilung überein, ist das Verfahren verifiziert.

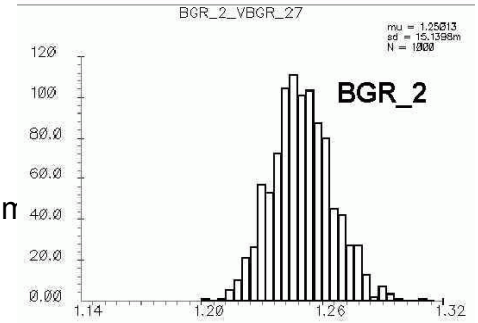
Weitere Beispiele Bandgap

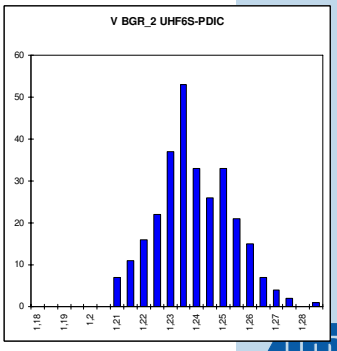
BGR_1
NPN
Emitter 0.7 x 0.5um





BGR_2
NPN
Emitter 4.7 x 0.5um





Weitere Beispiele Op-Amp

	relative Gain	Simulation 1-sigma	Measurement 1-sigma	Difference Meas-Sim
		mV	mV	mV
TIA1	40	1.65	1.88	0.23
	13.3	1.28	1.49	0.21
	3	2.88	2.74	-0.14
	1	1.28	1.17	-0.11
TIA2	160	1.85	3.66	1.81
	53.2	1.37	2.30	0.93
	12	2.42	3.05	0.63
	4	1.12	1.95	0.83

Ausblick

- TRADICA erweist sich als vorteilhaft für die
  - Erstellung erster Modell-Bibliotheken ohne Silizium
  - statistische Simulation (per se unabhängige Input-Parameter)
- Zwei verschiedene Arten der Einbindung in den Design-Flow sind möglich.
- Es muß ein Paradigma-Wechsel eingeleitet werden, an Modell-Parametern darf nicht „einfach so gedreht werden“.
- Ähnlich wie bei Krylov und der Harmonic-Balance-Methode muß die statistische Simulation mit modernen mathematischen Methoden auf die notwendige Geschwindigkeit gebracht werden.

VLSI-basierte Hardwarearchitekturen für Hodgkin-Huxley-Neuronenmodelle

Marcel Beuler, Werner Bonath
University of Applied Sciences Giessen-Friedberg
bonath@ieee.org

1. Das Hodgkin-Huxley Modell
2. Systemsimulation und VHDL-AMS-Modelle
3. Datenfluß-Architekturen
4. Zusammenfassung

1. Das Hodgkin-Huxley Modell

A.L.HODGKIN, A.F. HUXLEY:
"*A quantitative description of membrane current and its application to conduction and excitation in nerve*"
J. Physiol. (1952) 117, 500-544....

Modell zur Simulation des biologisch/elektrischen Verhaltens von Neuronen

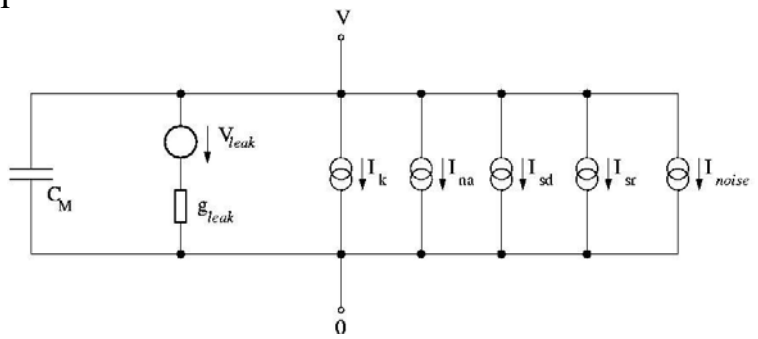
Ionenströme (Stromquellen)
Zellmembran (Membrankapzität)
variable Widerstände (nichtlinear)

Sehr genaue Modellierungsform

Komplexität: nur zur Simulation einzelner/weniger Neuronen geeignet

1. Das Hodgkin-Huxley Modell

Um Temperaturverhalten und Rauschen erweitertes Modell nach Huber/Braun[1,2]



$$C_M \cdot \frac{dV}{dt} = -I_{leak} - I_{na} - I_k - I_{sd} - I_{sr} + I_{noise} + c_i$$

C_M: Membrane Capacity

$I_{leak} = g_{leak} \cdot (V - V_{leak})$ Leak Conductance and Potential

I_{na}, I_k, I_{sd}, I_{sr}: Currents through Ion Channels

I_{noise}: white noise

C_i: Coupling Term (external)

[1] : S.Bahar, 2001
[2] : Huber,Braun, 2003

1. Das Hodgkin-Huxley Modell

Beschreibung der Ionenströme

$$I_{na} = \rho \cdot g_{na} \cdot a_{na} \cdot (V - V_{na})$$

$$\rho = 1.3^{(T - T_0)/10}$$ Temperature dependent Na^+-Current

$$\frac{da_{na}}{dt} = \Phi \cdot \left(\frac{a_{na,\infty} - a_{na}}{\tau_{na}} \right)$$ Temperature Scaling Factor

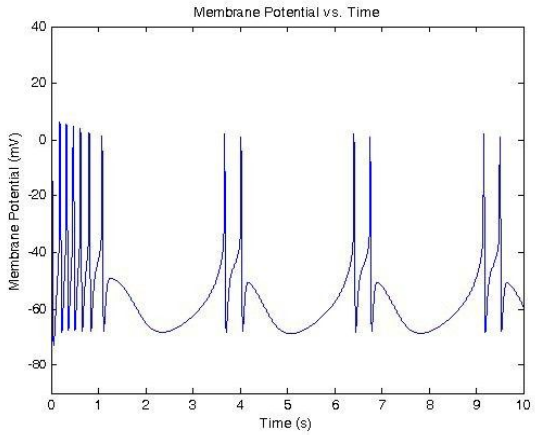
$$\Phi = 3.0^{(T - T0)/10}$$ Activation Variable

$$a_{na,\infty} = \left(1 + e^{-s_{na} \cdot (V - V_{0,na})}\right)^{-1}$$ Temperature Scaling Factor

Similar Definition of I_k, I_{sd}, I_{sr}:

- k^+ - channel current I_k
- Slow Depolarization Ca^{++} Current I_{sr}
- Slow Repolarization K^+ Current I_{sd}

1. Das Hodgkin-Huxley Modell: Verhalten



Beispiel: Membranspannung bei 25° C

- Einzelne Neuronen schwingen und erzeugen Bursts
- Überlagerung verschiedener Zeitkonstanten, starke Nichtlinearitäten
- Starke Temperaturabhängigkeiten
- Weißes Rauschen beeinfluß Verhalten
- Synchronisationseffekte bei Feldern gekoppelter Neuronen

2. Systemsimulation und VHDL-AMS-Modelle

Software-Simulationsmodelle in

- JAVA
- C
- MATLAB
- VHDL-AMS

+++ Reproduzierbarkeit, systematische Parametervariation

--- lange bis sehr lange Rechenzeiten,
Genauigkeit und numerische Stabilität

2. Systemsimulation und VHDL-AMS-Modelle: Hardware-Simulation

Modellierung des HH-Neurons
durch elektronische Hardware

Ziel: exakte Nachbildung einzelner
Neuronen oder große neuronale
Netze aus stark vereinfachten
Neuronenfunktionen

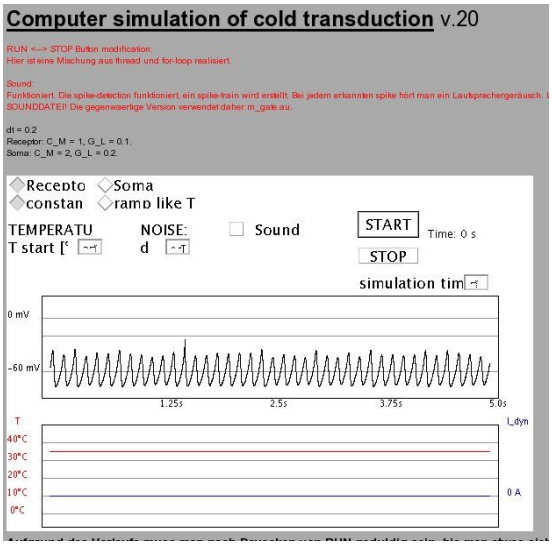
einfache Transistorschaltungenin
diskreter Elektronik

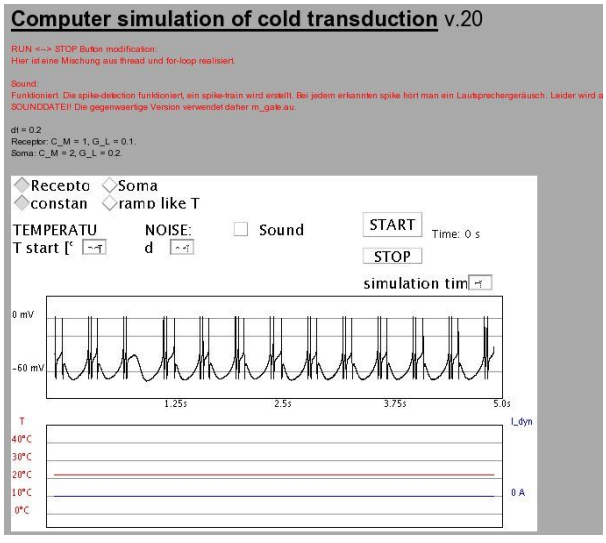
analoge integrierte Schaltungen

hochintegrierte digitale Schaltungen

2. Systemsimulation und VHDL-AMS-Modelle

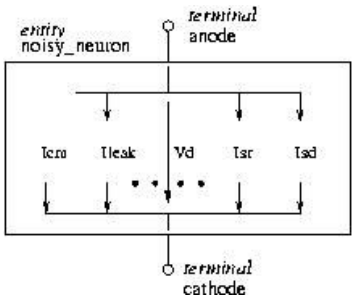
Simulation Results: Java based Simulator[1]





[1]:Huber, Braun, Uni Marburg

2. Systemsimulation und VHDL-AMS-Modelle: Implementierung mit VHDL-AMS



Two-Terminal Entity

```
neuron.vhd [Geändert] - KWrite
Datei Bearbeiten Ansicht Lesezeichen Extras Einstellungen Hilfe
library ieee_proposed;
use ieee_proposed.electrical_systems.all;
library ieee, Disciplines;
use IEEE.math_real.all;

entity noisy is
   port (terminal anode, cathode: electrical);
end entity noisy;

architecture behavior of noisy is

   quantity vd across
            i_g, i_l, i_d, i_r, i_sd, i_sr through
            anode to cathode;
 ............

end architecture behavior;
```

Branch Quantities for Electrical Domain

2. Systemsimulation und VHDL-AMS-Modelle: Implementierung mit VHDL-AMS

Branch Quantities for all Currents

Absolute Quantities for the Activation Variables

Capacitor Equation for the Membrane Capacitance

One Equation for the leakage current

Differential Equations for Activation Variables

```
neuron.vhd - KWrite
Datei Bearbeiten Ansicht Lesezeichen Extras Einstellungen Hilfe
   constant v_sd: REAL := 50.0E-3; -- equilibrium potential
   constant tau_sd: REAL := 10.0E-3; --activation time constant
   constant s_sd: REAL := 0.25;
   constant v_0sd: REAL := -40.0E-3; -- half activation potential
   constant g_sr: REAL := 2.0E-3; -- ionic conductance
   constant v_sr: REAL := -90.0E-3; -- equilibrium potential
   constant s_sr: REAL := 0.25;
   constant v_0sr: REAL := -40.0E-3; -- half activation potential

begin
   -- 1. membrane equation (rest of...)
   i_g == Cm*vd'dot;
   i_l == g1*(vd - v1);
   -- 2. Fast ionic currents for action potentials
   i_d == rho * g_d * a_d * (vd - v_d);
   a_d == 1.0/(1.0+exp(-s_d*(vd-v_0d)));
   i_r == rho * g_r * a_r * (vd - v_r);
   a_r'dot == phi * (1.0/(1.0 + exp(-s_r*(vd - v_0r)))- a_r)/tau_r;
   -- 3. slow ionic currents for oscillatory activity
   i_sd == - rho * g_sd * a_sd * (vd - v_sd);
   a_sd'dot == phi * (1.0/(1.0 + exp(-s_sd*(vd - v_0sd)))- a_sd)/tau_sd;
   i_sr == - rho * g_sr * a_sr * (vd - v_sr);
   a_sr'dot == phi * (1.0/(1.0 + exp(-s_sr*(vd - v_0sr)))- a_sd)/tau_sd;
end architecture behavior;
```

Temperature Model

- similar to temperature dependent resistor
- temperature modelled by an absolute quantity

neuron.vhd [Geändert] - KWrite

Datei Bearbeiten Ansicht Lesezeichen Extras Einstellungen Hilfe

```
library ieee_proposed;
use ieee_proposed.electrical_systems.all;
use ieee_proposed.thermal_systems.all;
library ieee, Disciplines;
use IEEE.math_real.all;

entity noisy is
   port (terminal anode, cathode: electrical;
         quantity temp in: temperature);
end entity noisy;
```

```
   quantity phi: REAL := 1.0; -- time constant
   quantity rho: REAL := 1.0;  -- temperature scaling

begin
   -- temperature dependencies
   rho == 1.3**((temp-temp_0)/10.0);
   phi == 3.0**((temp-temp_0)/10.0);
   -- 1. membrane equation (rest of...)
   i_g == Cm*vd'dot;
```

- temperature variation: testbench

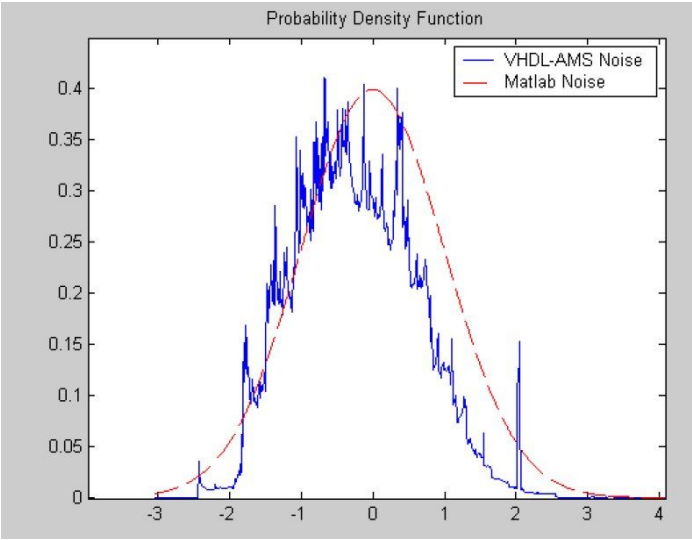
Noise Model

White Noise Source: Pseudo-Random Number from *math_real →UNIFORM*

Box-Muller-Transformation:

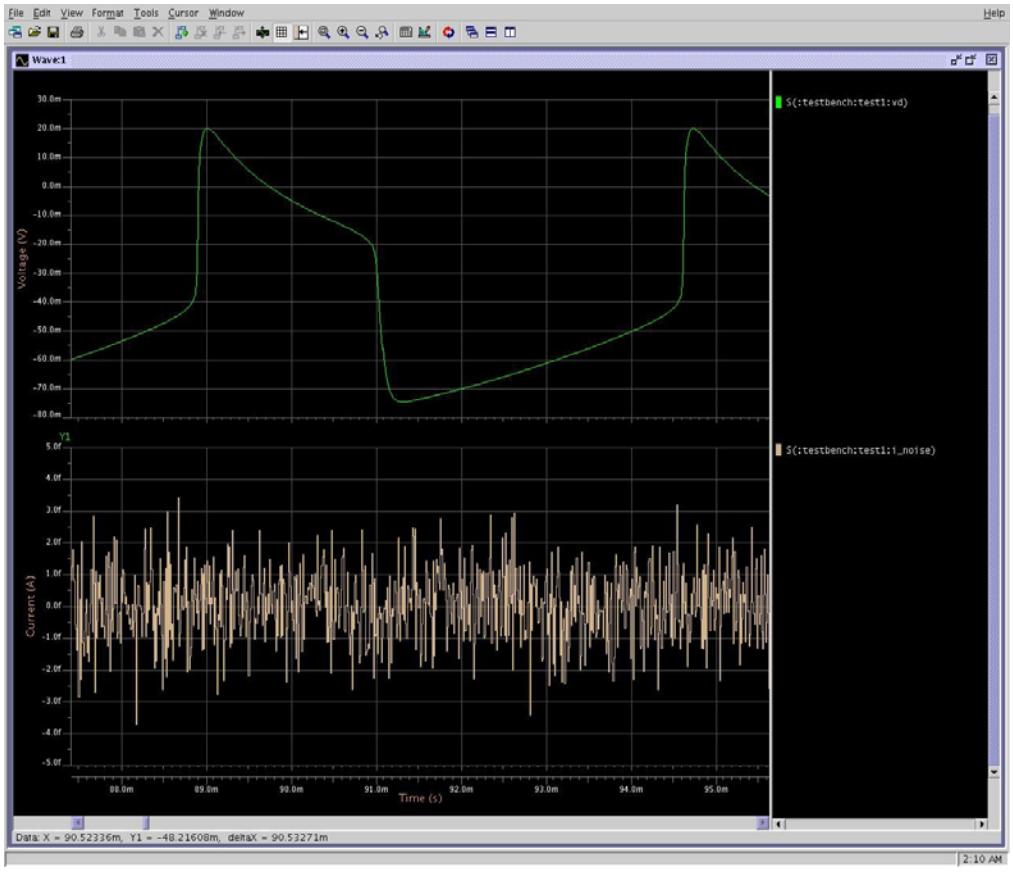
$$I_{noise} = e \cdot \left(\sqrt{-2 \cdot d \cdot dt \cdot \ln(a)} \right) \cdot \cos(2 \cdot \pi \cdot b)$$

with random numbers *a*,*b* and constants *d* and *e*



[1]from:*VHDL-AMS Modeling of VCSEL including Noise,* Mohamed KARRAY , Patricia DESGREYS and Jean-Jacques CHARLOT, TELECOM Paris

2. Systemsimulation und VHDL-AMS-Modelle: Implementierung mit VHDL-AMS



2. Systemsimulation und VHDL-AMS-Modelle: Implementierung als VHDL-Prozess

```
eqn: process is
begin

      wait until clk = 1.0;
      -- Leakage Current
      i_l <= g_l * (V - v_l);

      -- Fast ionic currents for action potentials
      -- Na - Sodium
      a_d_inf <= 1.0 / (1.0 + exp(-s_d * (V - v_0d)));
      i_d <= rho * g_d * a_d_inf * (V - v_d);

      -- K - Potassium
      a_r_inf <= 1.0 / (1.0 + exp(-s_r * (V - v_0r)));
      a_r_pres <= a_r_prev + dt*((a_r_inf - a_r_prev) / tau_r);
      i_r <= rho * g_r * a_r_pres * (V - v_r);
      a_r_prev <= a_r_pres;

      -- Currents
      i_spike <= i_d + i_r;

      -- Total current
      i_total <= i_l + i_spike - i_inj;

      -- Total voltage
      V <= V_prev - (i_total * dt)/C_M;
      V_prev <= V;

end process eqn;
```

- „pseudodigitale Modellierung“
- Differentiation durch Differenzenbildung
- fester Zeitschritt
- Zeitkonstanten vorgegeben

2. Systemsimulation und VHDL-AMS-Modelle: Implementierung als VHDL-Prozess

Simulationsergebnisse (Hamster-Simulator, FH Gießen 2008)

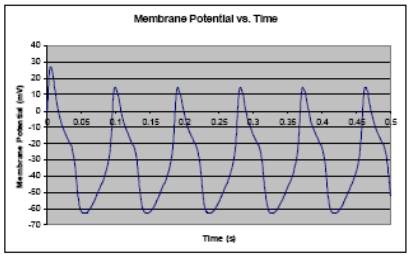
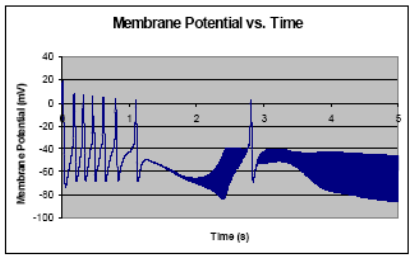


Figure 15 – Output response of hAMSter simulator (0.2331ms)



Übereinstimmung mit Meßwerten vergleichbar C/Java/Matlab

Numerische Probleme, Oszillationen, Schwingungsabbruch, kein Anschwingen

2. Systemsimulation und VHDL-AMS-Modelle: Implementierung mit VHDL-AMS

Behavioral (high level) simulation with Java/C++/Matlab:
- a large number of simulation programs published
- „behavioral“ means fast computation **without** realistic time constants
- results compatibel to measurements
- often current density based

Simulation with Simplorer/Mentor-Tools:
- realistic and dynamic time constants and amplitudes
- start value problems
- accuracy vs. Simulation time vs. Stability
- compatibility to measurements to be verified

Structural Approach has many advantages:
- observability and changeability of internals
- top-down substitution by hardware models
- strongly typed language as interface to neuronal scientists

3. Elementare Funktionen für digitaler Architekturen

Gleichungen des Huber/Braun-Neuronenmodells (Ausschnitt):

$$\frac{da_i}{dt} = \frac{\phi(a_{ij} - a_i)}{\tau_i} \qquad a_{iM} = \frac{1}{1 + e^{-s_i(V - V_{0i})}}$$

$$g_w = \sqrt{-4d \cdot \ln(a)} \cdot \cos(2\pi b)$$

- Ansatz: Realisierung eines Huber/Braun-Neuronenmodells mit rein digitaler Hardware
- Elementare mathematische Funktionen sind zu berechnen
- Verwendung des CORDIC-Algorithmus

3. Elementare Funktionen für digitaler Architekturen :

Der CORDIC-Algorithmus

- **CO**ordinate **R**otation **DI**gital **C**omputer
- 1959 von J.E. Volder entwickelt und 1971 von J.S. Walther auf hyperbolische Funktionen erweitert
- Der CORDIC ist ein iteratives Verfahren
- Es sind (bis auf Skalierung am Ende) nur ADD- und SHIFT-Operationen auszuführen
- Alle für das Huber/Braun-Neuronenmodell erforderlichen Funktionen lassen sich berechnen: *cos, exp, ln, sqrt*

3. Elementare Funktionen für digitaler Architekturen:

CORDIC-Iterationsgleichungen

Allgemeine Form:

$$\begin{pmatrix} x_{i+1} \\ y_{i+1} \\ z_{i+1} \end{pmatrix} = \begin{pmatrix} 1 & -m\sigma_i\delta_{m,i} & 0 \\ \sigma_i\delta_{m,i} & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_i \\ y_i \\ z_i \end{pmatrix} \quad \begin{matrix} 0 \\ 0 \\ -\sigma_i\alpha_{m,i} \end{matrix}$$

$m \in \{1, 0, -1\}$ Parameter für den Modus

$\delta_{m,i} = 2^{-i}$ Schrittweite des i-ten Iterationsschrittes

K_m^{-1} Skalierungsfaktor

$\alpha_{m,i}$ Dreh- bzw. Teilwinkel im i-ten Iterationsschritt

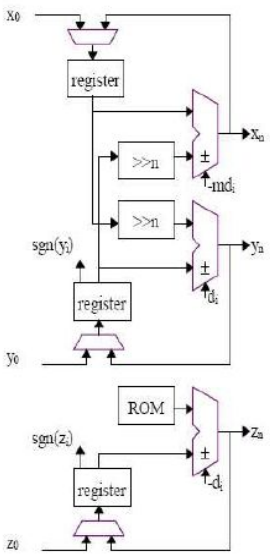
3. Elementare Funktionen für digitaler Architekturen:

CORDIC-Iterationsgleichungen

Mode m	Rotating $x_0 = 1\,,\, y_0 = 0\,,\, z_0 = \varphi$ $z_n \to 0\,,\, \sigma_i = \mathrm{sgn}(z_i)$	Vectoring $z_0 = 0\,,\, y_n \to 0$ $\sigma_i = -\mathrm{sgn}(y_i)$
$m = 1$ $K_1 = 0.607252935$ $S(1,i) = \{0,1,...,n-1\}$	$x_{i+1} = x_i - \sigma_i \cdot 2^{-S(1,i)} \cdot y_i$ $y_{i+1} = y_i + \sigma_i \cdot 2^{-S(1,i)} \cdot x_i$ $0 \leftarrow z_{i+1} = z_i - \sigma_i \cdot \arctan\left(2^{-S(1,i)}\right)$ $\Rightarrow \quad x_n = K_1^{-1} \cdot \cos(\varphi)$ $y_n = K_1^{-1} \cdot \sin(\varphi)$	$x_{i+1} = x_i - \sigma_i \cdot 2^{-S(1,i)} \cdot y_i$ $0 \leftarrow y_{i+1} = y_i + \sigma_i \cdot 2^{-S(1,i)} \cdot x_i$ $z_{i+1} = z_i - \sigma_i \cdot \arctan\left(2^{-S(1,i)}\right)$ $\Rightarrow \quad x_n = K_1^{-1} \cdot \sqrt{x_0^2 + y_0^2}$ $z_n = \arctan(y_0 / x_0)$
$m = -1$ $K_{-1} = 1.207497068$ $S(-1,i) = \{1,2,3,4,4,5,..,13,13,14,...,n\}$	$x_{i+1} = x_i + \sigma_i \cdot 2^{-S(-1,i)} \cdot y_i$ $y_{i+1} = y_i + \sigma_i \cdot 2^{-S(-1,i)} \cdot x_i$ $0 \leftarrow z_{i+1} = z_i - \sigma_i \cdot \mathrm{artanh}\left(2^{-S(-1,i)}\right)$ $\Rightarrow \quad x_n = K_{-1}^{-1} \cdot \cosh(\varphi)$ $y_n = K_{-1}^{-1} \cdot \sinh(\varphi)$	$x_{i+1} = x_i + \sigma_i \cdot 2^{-S(-1,i)} \cdot y_i$ $0 \leftarrow y_{i+1} = y_i + \sigma_i \cdot 2^{-S(-1,i)} \cdot x_i$ $z_{i+1} = z_i - \sigma_i \cdot \mathrm{artanh}\left(2^{-S(-1,i)}\right)$ $\Rightarrow \quad x_n = K_{-1}^{-1} \cdot \sqrt{x_0^2 - y_0^2}$ $z_n = \mathrm{artanh}(y_0 / x_0)$

3. Elementare Funktionen für digitaler Architekturen :

Implementierung des CORDIC Algorithmus



Zunächst bitparallele iterative CORDIC-Struktur

Datenformat völlig offen (n-Bit-Festkomma bzw. Gleitkomma)

- Kompromiss zwischen Genauigkeit und Chipfläche
- Numerische Probleme bei VHDL-AMS-Simulation deuten auf Gleitkommaformat hin

3. Elementare Funktionen für digitaler Architekturen:

Zahlenformate

Ansatz:

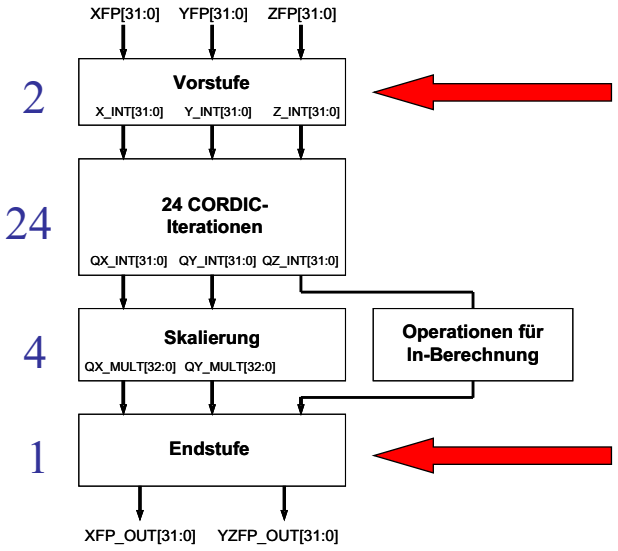
Externes 32-Bit ANSI/IEEE-Gleitkommaformat

- Einfache Argumentbereichs-Erweiterung für ln und sqrt, da Numerus und Radikand bereits durch Mantisse und Exponent dargestellt

Internes 32-Bit-Festkommaformat für alle 3 Datenpfade (Zweierkomplement)

3. Elementare Funktionen für digitaler Architekturen

Blockdiagramm Hardware-Umsetzung:



Umwandlung Gleit- in Festkommaformat
Argumentbereichs-Erweiterung

$$\exp(x) = \exp(z + m \cdot \ln 2) = 2^m \cdot [\cosh(z) + \sinh(z)]$$

$$\ln(M \cdot 2^E) = \ln(M) + E \cdot \ln 2$$

$$\sqrt{r} = 2^{E/2} \cdot \sqrt{M} \quad \text{für: } E \bmod 2 = 0 \text{ (gerade)}$$

$$\sqrt{r} = 2^{(E+1)/2} \cdot \sqrt{M/2} \quad \text{für: } E \bmod 2 = 1 \text{ (ungerade)}$$

Umwandlung Fest- in Gleitkommaformat

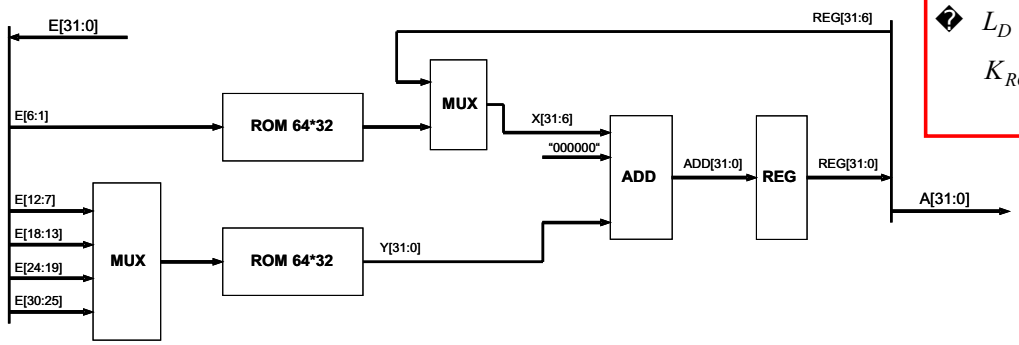
31 Taktzyklen pro Berechnung

3. Elementare Funktionen für digitaler Architekturen

Kompensation des Skalierungsfaktors:

- Abhängig von der Shiftfolge (zirkular bzw. hyperbolisch)
- Notwendig für die Datenpfade x und y
- Einfache Realisierung über ROM-Speicher (Konstanten-Multiplizierer)

Sequentielle Realisierung (4 Taktzyklen):



$$L_D = 30; \quad L_K = 26$$

$$\Rightarrow \quad L_D = 30 = N_{ROM} \cdot A_{ROM} = 5 \cdot$$

$$K_{ROM} = 2^{A_{ROM}} * (L_K + A_{ROM})$$

$$= 64 * 32$$

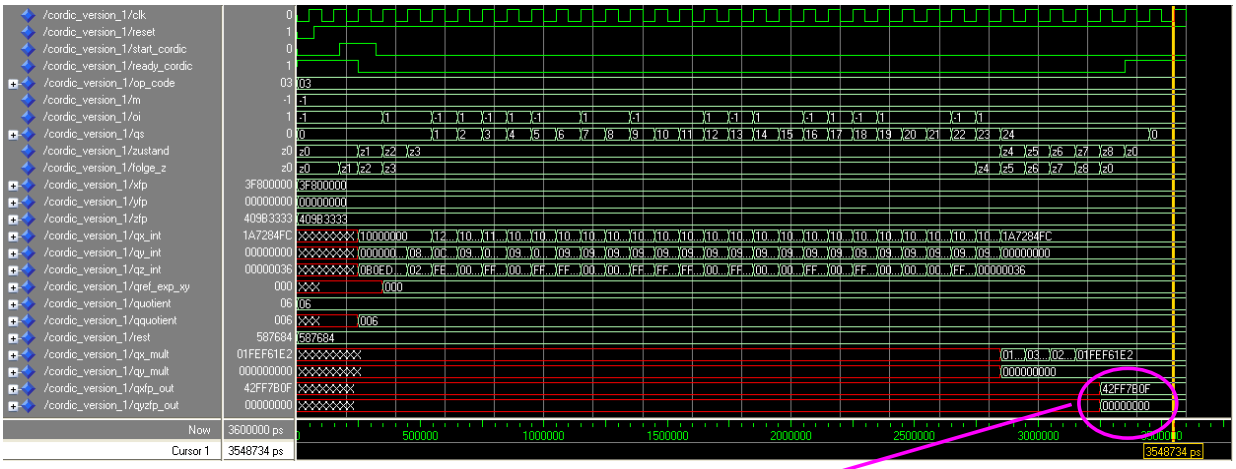
Umgesetzte Funktionen + Wertebereich der Eingabedaten:

OP_CODE[7:0]	Eingabe XFP	YFP	ZFP	Ergebnis XFP_OUT	YZFP_OUT	Wertebereich der Eingabedaten
X“01“	1	0	z	cos(z)	sin(z)	$0 \le z \le 2\pi$
X“03“	1	0	z	exp(z)	0	$\lvert z \rvert < 16$
X“04“	r	r	0	0	ln(r)	$1.084E\text{-}19 \le r \le 1.845E19$
X“05“	r	r	0	sqrt(r)	0	$9.537E\text{-}7 \le r \le 2.097E6$

(Wertebereich der Eingabedaten für Simulation ausreichend)

3. Elementare Funktionen für digitaler Architekturen

Simulation des strukturellen digitalen VHDL-Modells:



Berechnung exp(4.85):

```
QXFP_OUT[31:0] = 42FF7B0Fh  = 127.74035d   (Simulation)
                 exp(4.85)  = 127.7403898… (exakt)
```

4. Zusammenfassung/Ausblick

Hardware-Implementierungen der exakten Hudgley Hodgkins Neuronenmodelle erscheinen möglich

VHDL/VHDL-AMS erlaubt leistungsfähige Modellierung der Huber-Braun HH Neuronengleichungen

Systemebene: Untersuchung bzgl. Genauigkeit, Stabilität und Skalierbarkeit

weitere Arbeiten: Partitionierung, Kopplungsmechanismen

Bottom Up: Realisierung elementarer Funktionen mit bitparallelen iterativen CORDIC-Strukturen

weitere Arbeiten: Pipeline-Strukturen, FPGA-Pilotrealisierung, Datenfluß-Architekturen, Auslastungsoptimierung

Conclusion

VHDL-AMS model of a Noisy Neuronal Oscillator was presented

Structural and hardware oriented approach

Creation of small Neuronal Nets based on Analogue HH-Models including noise is possible

Comparison between VHDL-AMS Simulation, Functional High Level Simulation and Measurement Results

MULTI PROJEKT CHIP GRUPPE

Hochschule Aalen
Prof. Dr. Bartel, (07361) 576-4182
manfred.bartel@htw-aalen.de

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

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

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

Hochschule Heilbronn
Prof. Dr. Schröder, (07131) 504-400
jschroeder@hs-heilbronn.de

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

Hochschule Konstanz
Prof. Dr. Freudenberger, (07531) 206-647
juergen.freudenberger@htwg-konstanz.de

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

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

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

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

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

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

www.mpc.belwue.de