MPC

MULTI PROJEKT CHIP GRUPPE
BADEN-WÜRTTEMBERG

Herausgeber: Hochschule Ulm **Ausgabe:** 43 **ISSN** 1862-7102 **Workshop:** Göppingen Februar 2010

Cooperating Organisation
Solid-State Circuit Society Chapter
IEEE German Section

**Tagungsband zum Workshop der Multiprojekt-Chip-Gruppe Baden-Württemberg,
Göppingen, 5. Februar 2010**

Inhaltsverzeichnis

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

Der FlexRay™ Physical-Layer[1)]
Aspekte der
Spezifikation, Simulation und Implementierung

Jürgen Minuth

Hochschule Esslingen, 73037 Göppingen, Robert-Bosch-Straße 1

juergen.minuth@hs-esslingen.de

Das FlexRay™-Konsortium bestehend aus Kfz-Herstellern und –Zulieferern wurde vor über 10 Jahren mit dem Ziel gegründet ein Kfz-taugliches Kommunikationssystem zu spezifizieren, das die prognostizierten Kernanforderungen zukünftiger Anwendungen erfüllt: deterministische Echtzeitfähigkeit, hohe Verfügbarkeit bei deutlich höherer Geschwindigkeit als CAN. Die momentan am Markt verfügbaren Produkte sind kompatibel zu der 2005 veröffentlichenden Spezifikation Version 2.1. Die Veröffentlichung der Version 3.0 ist für 2010 angekündigt – die Halbleiterhersteller arbeiten bereits an kompatiblen Produkten.

1. FlexRay™

1.1. Begriffe

Der Beitrag verwendet folgende FlexRay™-üblichen Abkürzungen:

BD	Bus Driver (vergleichbar mit dem CAN Transceiver) Pegelwandler digital ⇔ Busleitungen
SG	(elektronisches) Steuergerät
CC	Communication Controller Hardware, die das FlexRay™-Protokoll umsetzt
Host	Rechner zur Abarbeitung der Funktions-Software
AS	Active Star oder Aktiver Stern bidirektionaler Signalverstärker ohne Protokoll-Maschine
CMD	Common Mode Drossel, Gleichtaktdrossel
ESD	Bauteil zum Schutz gegen elektrostatische Entladung

[1)]Quellen: http://www…flexray…

1.2. Übersicht

FlexRay™ unterstützt zwei grundsätzlich unterschiedliche Kommunikationsverfahren im zum Kompilierzeitpunkt festgelegten Kommunikationszyklus:

- synchrone Botschaften in vordefinierten Slots → "static part"
  unbelegte Slots stehen für Erweiterungen zur Verfügung, 100% Buslast ist zulässig
- asynchrone Botschaften über Zugriffsprioritäten gesteuert (vergleichbar mit der CAN-Arbitrierung) → "dynamic part"

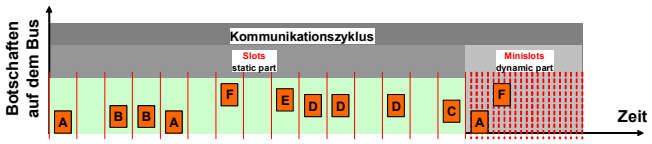


Bild 1: Botschaft auf FlexRay™

Protokollmechanismen synchronisieren die verteilten lokalen Uhren der CC. Zur Erhörung der Verfügbarkeit oder der Nettodatenrate stehen optional zwei Kanäle zur Verfügung.

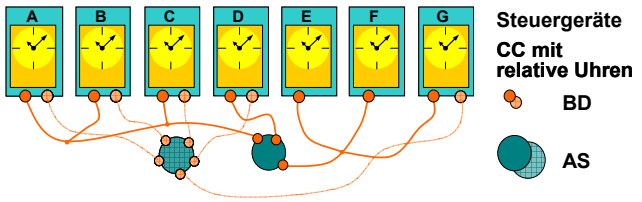


Bild 2: Architektur einer FlexRay™ Vernetzung
sieben SG A bis G mit 2 AS
zwei (physikalische) Kanäle

Datenraten von 2.5, 5 oder 10 MBit/s stehen zur Vernetzung von bis zu 64 SG über Punkt-zu-Punkt-Verbindungen, passive Busse und Sterne sowie AS bereit.

1.3. Komponenten

Die FlexRay™ Physical Layer Spezifikation berücksichtigt zur Übertragung einer Botschaft vom Sende-SG zum Empfangs-SG unterschiedliche Komponenten:

- Treiber des sendenden CC
- sendender BD
- Anschluss-Netzwerk mit CMD, ESD-Schutz und Leitungsterminierung
- Kabelbaum in verschiedenen Leitungstopologien unter Verwendung einer Zwei-Draht Leitung Schirm: optional
  Wellenwiderstand: 80Ω bis 110Ω
- AS mit je zwei Anschluss-Netzwerken und BD optional darf ein CC angeschlossen sein

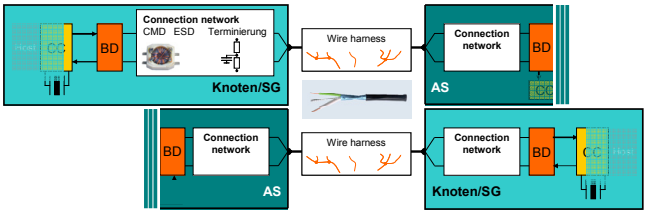


Bild 3: Komponenten auf dem Kommunikationspfad

- Kabelbaum in verschiedenen Leitungstopologien
- Anschluss-Netzwerk mit CMD, ESD-Schutz und Leitungsterminierung
- empfangender BD
- Empfangsstufe des empfangenden CC

1.4. Protokoll, Kodierung und Abtastung

Der FlexRay™ Physical Layer und das Protokoll beeinflussen sich aufgrund ihrer Eigenschaften gegenseitig:

- der Beginn jeder Botschaft wird durch die BD während der Übertragung gekürzt (TSS: Transmission Start Sequence)
- am Ende einer Botschaft können zusätzliche Bits entstehen
  (nach der FES: Frame End Sequence)

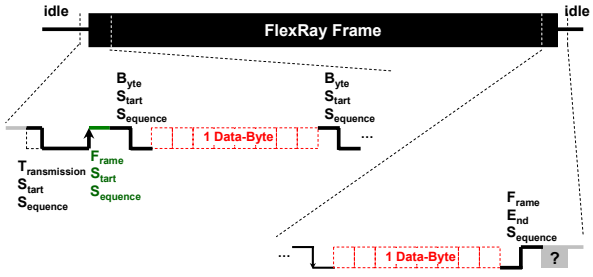


Bild 4: Anfang und Ende einer FlexRay™-Botschaft (Frame)

- der Abtastvorgang beginnt mit der 1. steigenden Flanke der Botschaft
  (FSS: Frame Start Sequence)
- jedes Byte wird synchronisiert auf die fallende Flanke des vorangestellten 2 Bit-Musters
  (BSS: Byte Start Sequence)

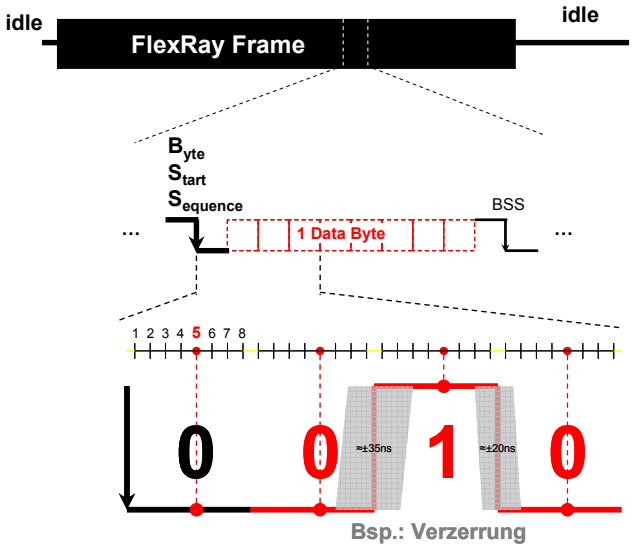


Bild 5: Datenbyte innerhalb einer FlexRay™-Botschaft (Frame)

- jedes Byte wird 8-fach abgetastet, der jeweils 5. Abtastwert wird als Bitwert interpretiert

Die aufgezeigten Eigenschaften wirken sich an unterschiedlichen Stellen des Physical Layers aus. Anhand zweier Beispiele lässt sich dies einfach aufzeigen:

- Die AS müssen Botschaften in Echtzeit erkennen und ihr Weiterleiten vom Empfangsarm auf alle restlichen Arme freischalten. Die hierfür benötigte Zeit führt zum Abschneiden der TSS.
- 10 aufeinanderfolgende Bits müssen ohne Nachsynchronisation fehlerfrei bei 62.5% der Empfängerbitzeit abgetastet werden. Zur fehlerfreien Funktion dürfen dabei steigende und fallende Flanken nur in einem bestimmten Bereich statisch oder dynamisch zeitlich abweichen.

2. Physikalische Datenübertragung

2.1. Bus-Driver (BD) und Bus-Signale

Der FlexRay™ BD wandelt den Datenstrom bi-direktional zwischen den Bus-Signalen und den logischen Signalen. Er unterscheidet drei Bus- bzw. Betriebszustände:

- **idle**
  Busleitungen hochohm´sch an 2.5V
  Differenz-Busspannung 0V
- **busy "0"**
  niederohm´sch erzeugte negative Differenz-Bus-spannung
- **busy “1”**
  niederohm´sch erzeugte positive Differenz-Bus-spannung

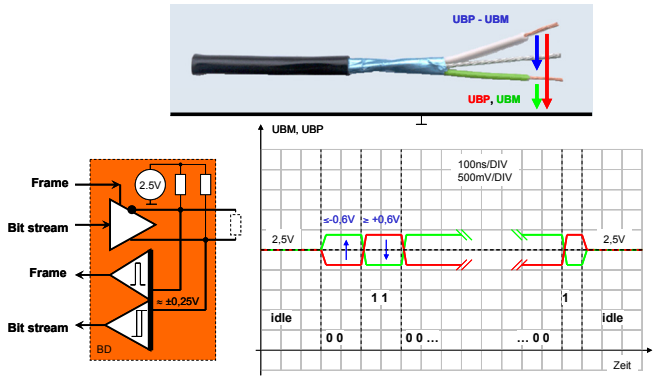


Bild 6: Blockschaltbild eines BDs mit Leitung und Signalen

Der Datenstrom wird über eine aktivierbare “push/pull” Endstufe auf den Bus gelegt. Zum Empfang der Daten ist ein Differenz-Schmitt-Trigger vorgesehen. Zum Erkennen der Botschaft ist ein Differenzfenster-komparator vorgesehen.

2.2. Aktiver Stern (AS)

Der AS koppelt mehrere passive Netze als seine Arme miteinander. Er unterscheidet unterschiedliche Betriebszustände kompatibel zu den Betriebs-zuständen des BD:

- Ist eine Botschaft weiterzuleiten?
  Aktivierung über “00..0”-Erkennung (TSS) gefiltert über den " activity time-out "
- Ist die Botschaft beendet?
  Deaktivierung über "idle"-Erkennung gefiltert über "idle time-out"

Beide Funktionen werden über einen "langsamen" Differenzfensterkomparator realisiert.

Die Daten werden über einen "schnellen" Differenz-Schmitt-Trigger auf die "push-pull"-Endstufen weiter-geleitet.

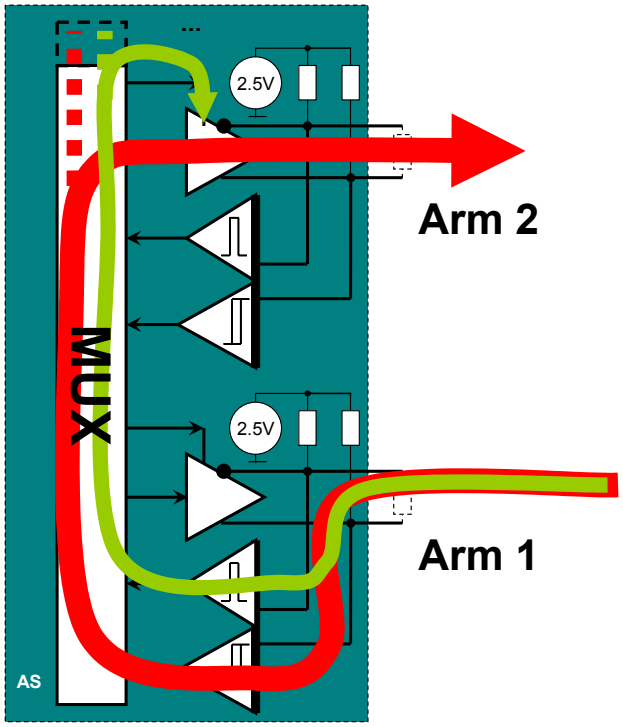


Bild 7: Blockschaltbild eines AS

2.3. Glitches

Im EMV-Lastenheft des Fahrzeugherstellers ist ein Test der Busübertragungen unter Störimpulsen vorgesehen. Dabei können unter ungünstigsten Bedingungen vom BD nicht mehr unterdrückbare Störungen eingekoppelt werden, die sich als Gliches auf dem Datenstrom in den CC äußern können.

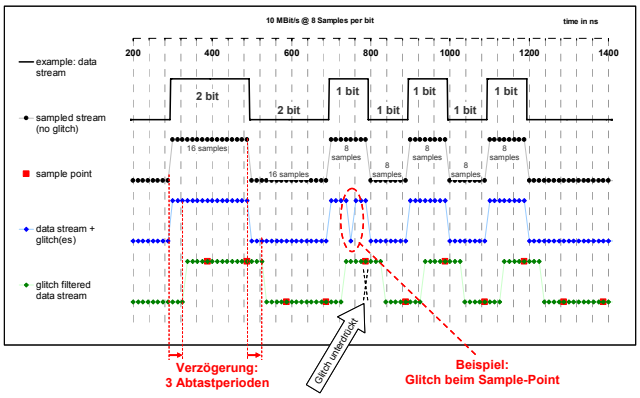


Bild 8: Abtastung und Glitch-Filterung

Das FlexRay™ Protokoll spezifiziert einen Filter zur Unterdrückung der Glitches:

- 3 aus 5 Filter
  Gefilterter Wert = Durchschnitt der 5 vorherigen Abtastwerte
- der Datenstrom wird um 3 Abtastperioden verzögert
- 1 falscher Abtastwert wird ausgefiltert

- 2 falsche Abtastwerte unmittelbar nacheinander werden ausgefiltert

3. Messung und Simulation

3.1. Signalbewertung

FlexRay™ lässt große Freiheiten in der Auswahl der Topologie bzw. der darin enthaltenen passiven Leitungstopologien zu. Die Datenkommunikation funktioniert auch dann noch problemlos, wenn beispielweise aufgrund von Reflexionen die üblichen Augendiagrammtests einen Fehler melden.

Zur Begutachtung der Bussignale ist ein Signalbewertungsverfahren (SI-Voting) spezifiziert, das in seinem Kern die spezifizierten Schmitt-Trigger-Eigenschaften des Empfängers berücksichtigt.

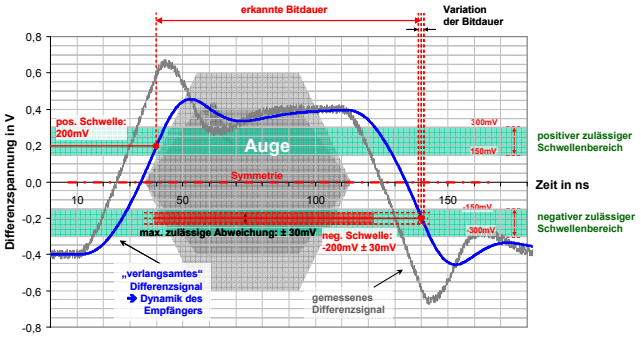


Bild 9: Biterkennung ⇔ Augendiagramm

Das in seiner Dynamik reduzierte Bussignal wird über 16 zulässige Schwellwertkombinationen vermessen.

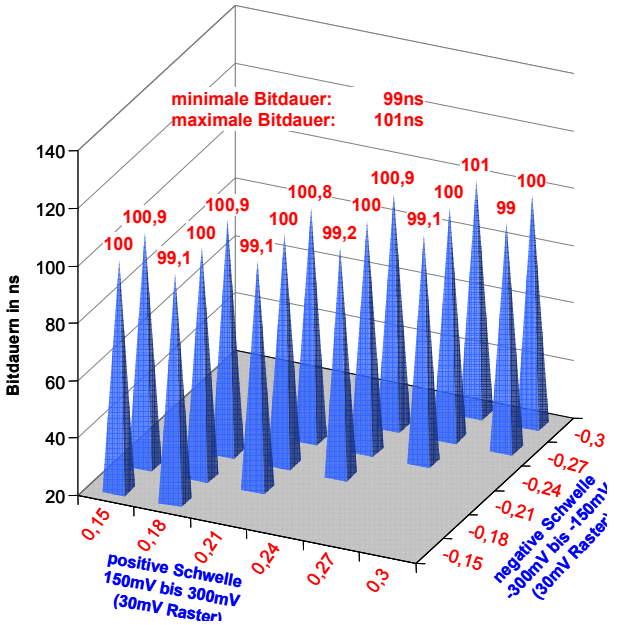


Bild 10: Biterkennung mit Toleranzen

Die ermittelte Bitlängenvariation wird gegen die Spezifikation geprüft.

3.2. Generischer Bus-Driver (BD)

3.2.1 Sendestufe

Simulationsmodelle von Halbleiterherstellern spiegeln das Produktdatenblatt wieder; viele Werte nutzen dabei nicht den ganzen spezifizierten Freiraum aus. Der generische BD lässt sich auf den ganzen Spezifikationsraum parametrisieren. Er ist geeignet Konzeptsimulationen an beliebigen Topologien schnell und effizient durchzuführen.

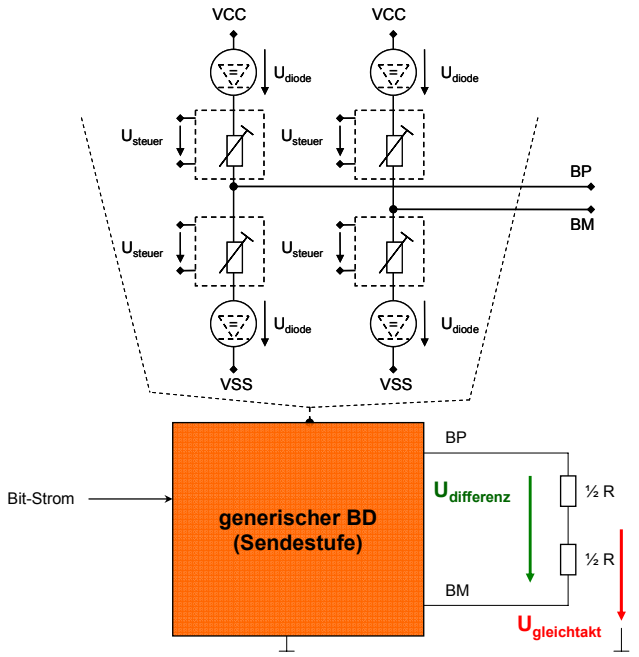


Bild 11: Sendestufe des generischen BD

Die beiden "push-pull"-Stufen werden über vier spannungsgesteuerte Widerstände modelliert. Die üblichen Dioden zur Sicherstellung der Rückstromfreiheit werden über Spannungsquellen berücksichtigt. Aus Sicht des Anwenders ergeben sich bestimmte Eigenschaften:

- Timing- und Pegel-Asymmetrien einstellbar
- Flankensteilheiten einstellbar
- Versorgungsstrom wird erfasst
- Gleich- und Gegen-Takt wird erfasst
- nahezu lineare Flanke bei ohm´scher Last

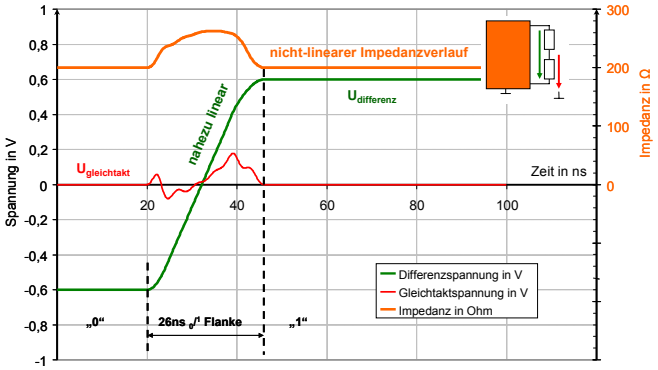


Bild 12: Beispiel einer 0➔1-Flanke mit geringen Asymmetrien im Differenz- und Gleichtakt

3.2.2 Empfangsstufe

Die Empfangstufe des generischen BD enthält die Komponenten der Signalbewertung (SI-voting) zzgl. eines Fensterkomparators.

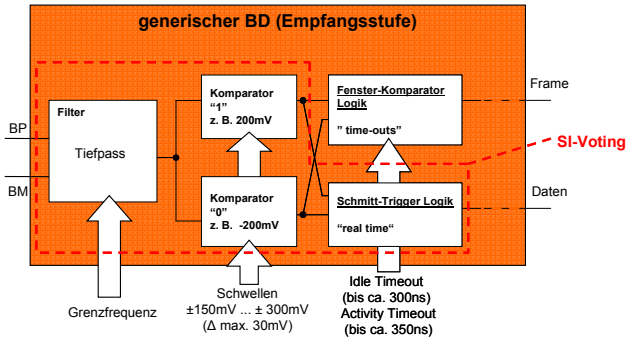


Bild 13: Empfangsstufe des generischen BD

3.3. Simulationsbeispiel

Am Beispiel eines passiven Busses für zwei SG und vier zusätzlich über kurze Stichleitungen angeschlossenen SG lassen sich die typische Effekte bei Übertragen einer vereinfachten Botschaft illustrieren.

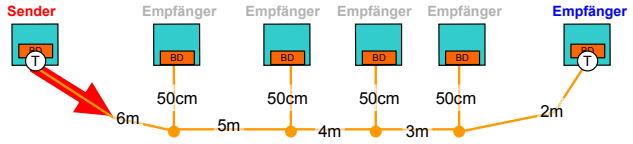


Bild 14: FlexRay™ in der Topologie "Passiver Bus"

Tab. 1: Stimuli der 1. Simulation

Treiberstufe	±800mV an Normlast
Flanken (0% ⇔ 100%)	20ns
Asymmetrie	keine
Empfänger-schwellen	±200mV
Grenzfrequenz Empfänger	ca. 20 MHz
Terminierung Anfang und Ende	4% zu hochohmig
Terminierung Stichleitungen	hochohmig
Streufaktor der CMD	<< 1%
Botschaft	10 MBit/s `idle00100001101111idle`

Die Simulation zeigt die erwarteten Ergebnisse:

- Bits auf dem Bus praktisch unverzerrt
- ± 900mV Buspegel
- alle empfangenen Datenströme stimmen mit dem gesendeten Datenstrom überein
- Reflexionen aufgrund einer Bitflanke können um mehr als eine Bitdauer verzögert werden

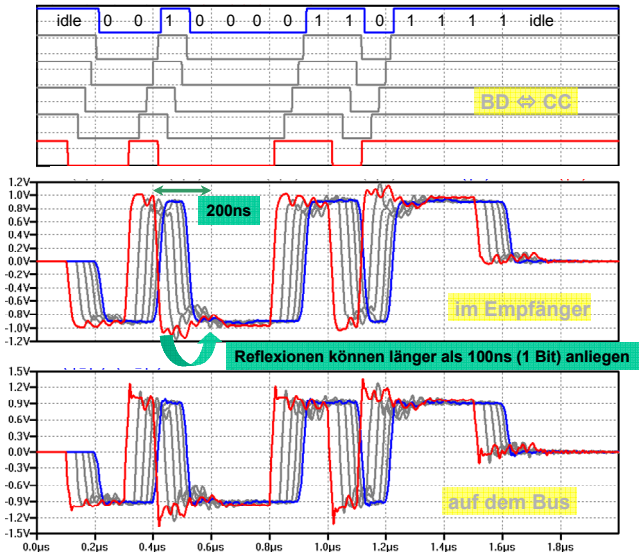


Bild 15: Datenstrom Beispiel 1

In der abschließenden Simulation werden 3 Stimuli-Parameter verändert.

Tab. 2: Stimuli der 2. Simulation

Treiberstufe	±800mV an Normlast
Flanken (0% ⇔ 100%)	20ns
Asymmetrie	keine
Empfängerschwellen	±150mV
Grenzfrequenz Empfänger	>> 20 MHz
Terminierung Anfang und Ende	4% zu hochohmig
Terminierung Stichleitungen	hochohmig
Streufaktor der CMD	1%
Botschaft	10 MBit/s idle00100001101111idle

Die Simulation zeigt ein paar auf den ersten Blick überraschend erscheinende Ergebnisse:

- am Ende der Botschaft treten Schwingungen auf, die bei den Empfängern zu "Ringing" führen
- Anfang und Ende der Botschaft unterscheiden sich auf dem Bus deutlich voneinander
- Bits auf dem Bus innerhalb der Botschaft sind nur wenig verzerrt
- alle empfangenen Datenströme stimmen bis zum Botschaftsende mit dem gesendeten Datenstrom überein

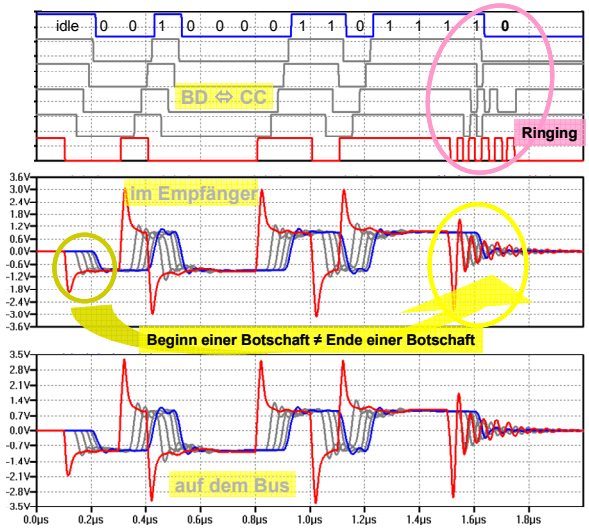


Bild 16: Datenstrom Beispiel 2

Das “Ringing” am Botschafts-Ende kann dazu führen, dass der Empfänger auf dem inversen Bitwert endet, der verzögert über den "idle Time-out" zurückgenommen werden wird. Die dabei auftretende Gesamtverzögerung lässt sich bei den einstellbaren Protokollparametern berücksichtigen. "Ringing" ist bei entsprechender Komponenten- und Topologiewahl zuverlässig vermeidbar.

4. Serieneinsatz

Seit 2006 ist das erste Fahrzeug mit FlexRay in Serie am Markt. Der X5 von BMW kann mit der Sonderausstattung "Aktives Fahrwerk" geordert werden. Fünf über FlexRay™ vernetzte SG regeln die Dämpfung Rad-individuell u. a. unter Nutzung der gemessenen Rad- und der Chassis-beschleunigungen.

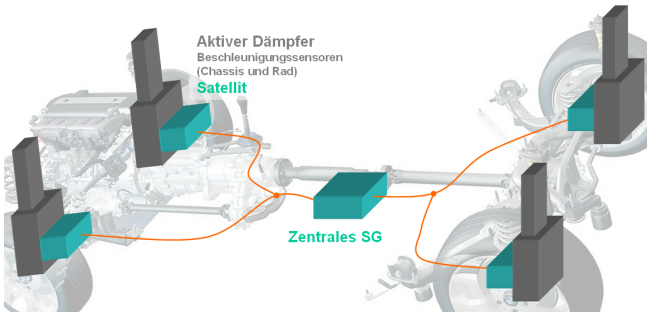


Bild 17: FlexRay™-Vernetzung BMW X5 Modell 2006
“Aktives Fahrwerk”
fünf Steuergeräte (SG) mit der standardisierten Basis-Software OSEK
Quelle: http://www.tomshardware.com

Weitere Serieneinsätze sind angekündigt. Wir sind gespannt die nächsten FlexRay™-Anwendung zu sehen.

Design eines Quarzoszillators in 0,35 µm CMOS-Technologie

Goran Bratek, Jürgen Giehl, Bernd Vettermann
Hochschule Mannheim, Paul-Wittsack-Straße 10, 68163 Mannheim
Fakultät für Informationstechnik – Institut für Entwurf integrierter Schaltkreise
Bratekg@aol.com

Um Oszillationen mit hoher Frequenzstabilität zu erzielen, werden oft Quarzoszillatoren eingesetzt. Quarze nutzen den piezoelektrischen Effekt aus, um aus Verformungen eine Spannung zu erzeugen und umgekehrt. In der Regel sind sie kostengünstig für alle Frequenzen zu erwerben. Als Grundlage für diesen Artikel liegt eine Diplomarbeit zu Grunde, die ab September 2009 entstand.

Ziel ist es, einen Quarzoszillator vom Pierce-Typ zu realisieren, der trotz unterschiedlicher Temperatur- und Spannungsbedingungen die gewünschte Frequenz liefert. Außerdem ist die Lastkapazität für den eingesetzten Quarz zu bestimmen. Der Quarzoszillator benötigt keine Spannungs- oder Stromreferenz. Ein Aufstarten der Versorgungsspannung wird simuliert, um ein Anschwingen des Quarzes sicherzustellen. Ebenfalls wird ein Layout der Oszillatorschaltung verlangt, welches sich am Padrand befindet. Der Hintergrund dieses Vorgehens ist, dass der Quarzoszillator nur ein Bestandteil eines PLL-Bausteins (Phasenregelkreis) sein wird. Dabei agiert der Quarzoszillator als Referenzgeber für die PLL.

1. Einleitung

1.1 Einführung

In der Informationstechnik gibt es eine große Anzahl von Anwendungen, bei denen bei selbstständigen Systemen die Taktfrequenz exakt gleich oder in einer festen Beziehung zu einander stehen. Aufgrund von nicht vermeidbaren Störeinflüssen kann die exakte Frequenzgleichheit bzw. -differenz nicht bleibend eingestellt werden. Um dies jedoch so exakt wie möglich zu erzielen, werden Synchronisationsverfahren verwendet.

Dieses Problem lässt sich auch mit der Phasenregelschleife (PLL, Phase-Locked Loop) lösen. Mit der PLL kann man ein quarzstabiles und höherfrequentes Signal erzeugen, indem man die Frequenz von Taktsignalen ohne Beeinträchtigungen vervielfachen kann.

Diese Taktsignale werden oft von Quarzen erzeugt, die im Vergleich zu Schwingkreisen eine höhere Güte bieten, welche für große Flankensteilheit, hohe Frequenzgenauigkeit und Frequenzstabilität erforderlich ist. Außerdem sind Quarze im Vergleich zu elektrischen Schwingkreisen deutlich unabhängiger von Temperatureinflüssen.

1.2. Ersatzschaltbild der Quarzes

Ein Quarz, auch als Schwingquarz (engl.: Crystal, Quartz) bezeichnet, ist ein herausgeschnittenes Plättchen eines in der Natur vorkommenden Quarzkristalls. Heutzutage werden sie jedoch meist künstlich hergestellt. Wird nun ein elektrisches Wechselfeld an das Plättchen gelegt, so erzeugt der Quarz mechanische Schwingungen, die bei seiner Resonanzfrequenz liegen. Jedoch braucht ein natürlicher Quarz eine gewisse Zeit, um sich einzuschwingen und um einen vollen Amplitudenausschlag zu erreichen. Die Aufstartzeit von Quarzen liegt im Bereich von 10ms.

Das elektronische Ersatzschaltbild für den Quarz besteht aus einer dynamischen Induktivität L_1, dynamische Kapazität C_1 und einen Widerstand R_1. Die Hauptresonanz wird genau durch diesen Serienresonanzkreis dargestellt. Nebenwellen und Obertöne werden durch zusätzliche parallele Schwingkreise erzeugt.

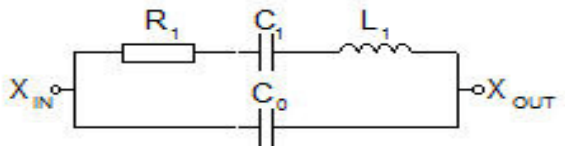


Abbildung 1: Einfaches Ersatzschaltbild des Quarzes nach Butterworth/Van Dyke

Die dynamische Induktivität L_1 charakterisiert hierbei die schwingende Masse des Quarzresonators. Die dynamische Kapazität C_1 erfasst hingegen die Elastizität und den piezoelektrischen Effekt des Quarzes. Die Dämpfung, die durch die viskosen Eigenschaften des

Quarzes erzeugt wird, wird durch den Widerstand R_1 repräsentiert. Dieser begrenzt auch die Qualität des Schwingquarzes.

Statische Kapazitäten die durch Anschlussstifte und Elektroden erzeugt werden, sind in der Koppelkapazität C_0 zusammengefasst. Diese wird noch genauer in C_{10}, C_{20} und C_{12} aufgeteilt, welche für die Schaltungssimulationen nicht vernachlässigbar sind. Sie repräsentieren den Plattenkondensator, der unbedingt geerdet sein soll.

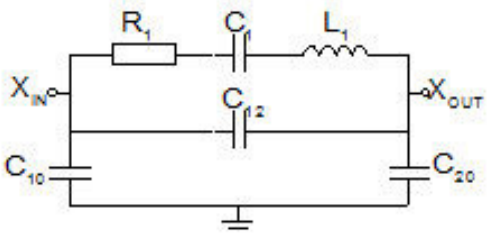


Abbildung 2: Erweitertes Ersatzschaltbild des Quarzes nach Butterworth/Van Dyke

C_{12} stellt die statische Kapazität dar, die durch die Elektrodenfläche bestimmt wird und die man durch deren Fläche und Abstand auch berechnen könnte. Die Kapazitäten, die sich zwischen dem Gehäuse und Resonator befinden, werden durch C_{10} und C_{20} verkörpert und werden Streukapazitäten genannt. Sie liegen im Bereich von 1 pF.

Der Quarz besitzt zwei Resonanzfrequenzen. Bei der sogenannten Serienresonanz weist er eine minimale Impedanz, bei der Parallelresonanz eine maximale Impedanz auf. Die frequenzabhängige Impedanz des Quarzes wird wie folgt beschrieben:

$$Z(j\omega) = (\mathrm{R}_1 + \mathrm{j}\omega \mathrm{L}_1 + \frac{1}{j\omega C_1}) \| \frac{1}{\mathrm{j}\omega \mathrm{C}_0}$$

$$= \frac{(\mathrm{j}\omega)^2 L_1 C_1 + j\omega R_1 C_1 + 1}{(j\omega)^3 L_1 C_1 C_0 + (j\omega)^2 R_1 C_1 C_0 + j\omega(C_1 + C_0)}$$

Ist R_1 vernachlässigbar klein gegenüber der Impedanz (jωL) an der Resonanz so gilt:

$$\approx \frac{(\mathrm{j}\omega)^2 \mathrm{L}_1 \mathrm{C}_1 + 1}{(\mathrm{j}\omega)^3 \mathrm{L}_1 \mathrm{C}_1 \mathrm{C}_0 + \mathrm{j}\omega(C_1 + C_0)}$$

Setzt man nun den Zähler bzw. den Nenner gleich „0“ so erhält man die Serienresonanz fs bzw. die Parallelresonanz fp.

$$\rightarrow f_s = \frac{1}{2\pi\sqrt{L_1 C_1}}$$

$$f_p = \frac{1}{2\pi\sqrt{L_1 C_1}} \sqrt{1 + \frac{C_1}{C_0}} = \frac{1}{2\pi} \sqrt{\frac{C_1 C_0}{L_1 C_1 C_0}}$$

$$\rightarrow f_p = f_s \sqrt{1 + \frac{C_1}{C_0}}$$

Da die Koppelkapazität C_0 sehr viel größer ist als die dynamische Kapazität C_1, ist der erwartete Wert vom Term $\sqrt{1 + \frac{C_1}{C_0}}$ kaum größer als 1, so liegen Parallel- und Serienresonanz nah aneinander.

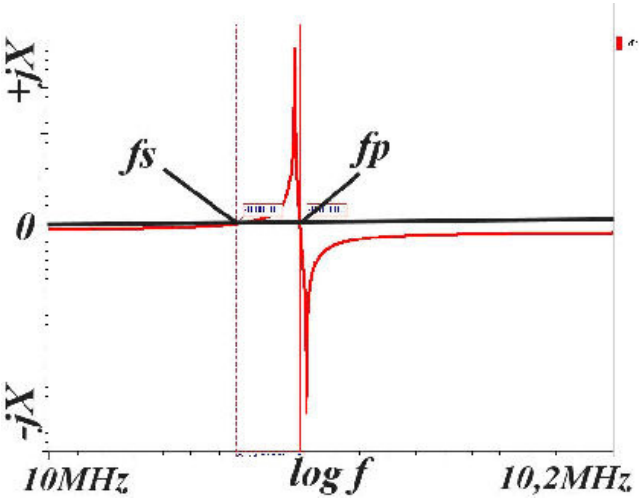


Abbildung 3: Blindwiderstandsdiagramm

Wird der Quarz in einer so genannten Pi- Schaltung betrieben, bildet dieser mit den Kapazitäten C_{in} und C_{out} eine Lastresonanzfrequenz f_L mit deren Frequenz der Quarz schwingt.

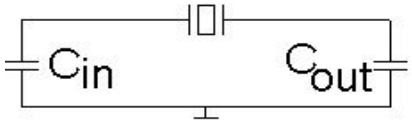


Abbildung 4: Quarz in der Pi-Schaltung

$$\frac{1}{C_L} = \frac{1}{C_{IN}} + \frac{1}{C_{OUT}}$$

$$f_L = f_s + f_s \frac{C_1}{2(C_0 + C_L)}$$

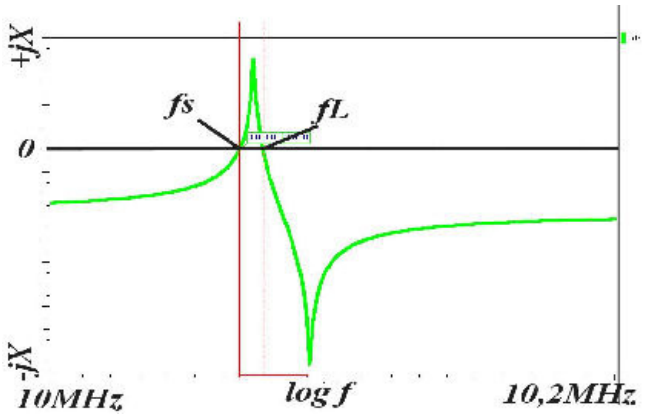


Abbildung 5: Blindwiderstandsdiagramm mit Lastkapazität

Die für den eingesetzten Quarz mit einer Resonanzfrequenz von ca. 10 MHz im Quarzersatzschaltbild verwendeten Bauteilwerte sind wie folgt angegeben:

Resonanzwiderstand R_1 = 50 Ω

Induktivität L_1 = 10 mH

Dynamische Kapazität C_1 = 25 fF

Koppelkapazität C_0 = 5,5 pf

dabei

Streukapazität $C_{10} = C_{20}$ = 1 pf

statische Kapazität C_{12} = 5 pF

Serienresonanz fs = 10,066 MHz

Parallelresonanz fp = 10,089 MHz.

Die Güte Q des eingesetzten Quarzes:

$$Q = \frac{1}{R_1}\sqrt{\frac{L_1}{C_1}} \approx 12650$$

Um ein Aufstarten des Quarzes sicherzustellen, muss jeweils die Lastkapazität des Quarzes bestimmt werden. Außerdem muss die gesamte Schaltung auf Versorgungs- und Temperaturschwankungen überprüft werden. Dabei muss auch auf Technologie- Corner, die u. a. durch den Fertigungsprozess entstehen können, eingegangen werden. Wichtige Kriterien zur Beurteilung von Quarzen sind deren Güte, Aufstartzeit, Frequenzgenauigkeit und Temperaturdrift.

1.3. Der Pierce-Oszillator

Der Pierce-Oszillator wird auch Pierce–Gate– Oszillator genannt.

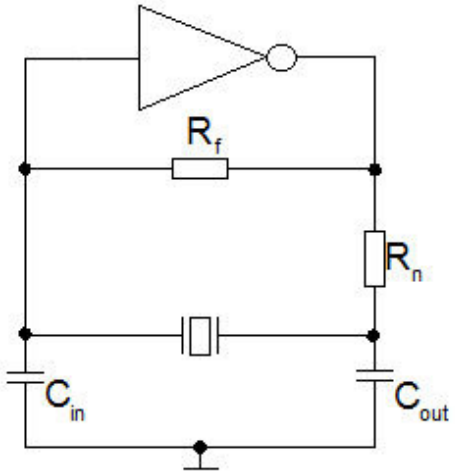


Abbildung 6: Aufbau der Pierce-Schaltung mit Logik-Gattern

Der Pierce-Oszillator ist einer der zuverlässigsten Oszillatorschaltungen. Als Vorteil dieser Schaltung kann man die große Ausgangsamplitude nennen. Jedoch verursacht diese hohe Ausgangsamplitude eine hohe Quarzbelastung, welche die Lebenszeit des Quarzes reduziert.

Beim Pierce-Oszillator wird in der Rückkopplung eines Inverters ein hochohmiger Widerstand Rf eingesetzt. Dieser bildet die einzige DC-Verbindung zwischen Inverter Ein- und Ausgang und bestimmt den Arbeitspunkt der Schaltung, der sich dann bei halber Versorgungsspannung einstellt. Parallel zum Widerstand R_n befindet sich der Quarz. Soll der Oszillator in einer CMOS Schaltung unter 5 MHz betrieben werden, so muss hier insbesondere noch ein Widerstand R_n eingesetzt werden. Dieser bildet mit C_{out} ein Verzögerungsnetzwerk. Diese zusätzliche Phasenverschiebung, die unter 5 MHz notwendig ist, reduziert entweder den Jitter im Zeitbereich oder das Phasenrauschen im Frequenzbereich. Er setzt die nichtlineare und sehr niedrige Ausgangsimpedanz des Gatters hoch, um parasitäre Effekte zu vermeiden. Manchmal, insbesondere bei Frequenzen über 20 MHz, wird dieser Widerstand nicht benötigt, da der Ausgangswiderstand des Inverters in Verbindung mit C_{out} eine ausreichende Verzögerung mit sich bringt. Außerdem bietet der Widerstand R_n zusätzliche Optimierungsmöglichkeiten für den Entwickler z. B. um den Drive Level (Leistung) am Quarz zu reduzieren und somit auch die Quarzbelastung. Bei der Reduzierung der Quarzbelastung wird zwar die Lebenszeit des Quarzes verlängert, jedoch reduziert sich auch die Schleifenverstärkung, dies wiederum erschwert das Anschwingen. Die

Stromaufnahme wird durch R_n reduziert, da der Auf- und Entladestrom von C_{out} reduziert wird. Er isoliert zusätzlich den Ausgangstreiber des Inverters vom komplexen Scheinwiderstand, der aus C_{in}, C_{out} und dem Quarz gebildet wird. Allgemein gilt: je größer die Frequenz desto kleiner wird der Wert von R_n.

Dank des Rauschen und des Einschaltvorgangs (Hochfahren der Versorgungsspannung) wird der Quarz bei der Serienresonanz betrieben und der Anstieg der Amplitude erfolgt. Wird keine Amplitudenregelung eingesetzt, so begrenzt der Aussteuerbereich des Inverters die Amplitude, wodurch das Ausgangssignal kein spektral reiner Sinus mehr ist.

Dient der Pierce-Oszillator als Zeitgeber in digitalen Schaltungen so spielt dies keine Rolle, da das Ausgangssignal mit einem weiteren Inverter verstärkt wird. In dieser Konstellation erhalten wir dann ein zweiwertiges Taktsignal mit hohen Anstiegs- und Abfallszeiten.

Je nach Modell werden C_{in} und C_{out} benötigt. Jedoch sollte man darauf achten, sie nahe an den Quarzanschlüssen zu platzieren, um keine störenden parasitäre Kapazitäten zu verursachen.

2. Schaltungsaufbau

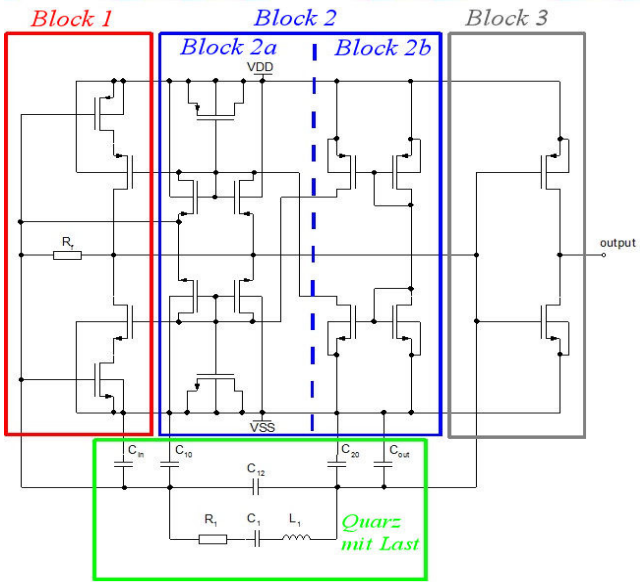


Abbildung 7: Schaltungsaufbau

Der komplette Schaltungsaufbau des Pierce-Quarzoszillators ist in Abbildung 7 dargestellt. Der untere Block ist dabei der Quarz mit Lastkapazitäten, der zwischen der eigentlichen Oszillatorschaltung geschaltet ist. Block 1 besteht aus einem Inverter und einer Kaskade, die ausschließlich zur Stromverstärkung dient. Zwischen Invertereingang und -ausgang ist noch der hochohmige Widerstand Rf geschaltet.

Um das Aufschwingen des Quarzes zu erleichtern, wurde noch eine Struktur aus pmos und nmos Transistoren in der Rückkopplung des Inverters eingebaut (Block 2). Je nach Pegelzustand leitet immer nur ein Transistor eines Paars der über Kreuz liegenden Transistoren in Block 2a. Das andere über Kreuz liegende Paar sperrt. Die in Block 2b bestehenden Stromspiegeln erzeugen einen nahezu konstanten Strom und versuchen diesen aufrecht zu erhalten, welcher dann über das leitende Transistorpaar fließt. In dem Moment, indem die Transistoren anfangen zu leiten, erzeugt sie eine kleine Mitkopplung, die ausreicht, um das Aufschwingen des Quarzes zu erleichtern. Die Kondensatoren dienen dabei zur Erzeugung einer nahezu konstanten Gatespannung dieser Transistoren. Außerdem Liefert Block 2 den Biasstrom für die Kaskoden in Block 1. Ein weiterer Inverter ist im Block 3 abgebildet. Dieser agiert als Buffer, um die Ausgangsstufe zu treiben. Deutlich wird, dass dieser Pierce-Oszillator keine Strom- oder Spannungsreferenz braucht.

3. Simulationen

Um den eingesetzten Quarz genauer zu spezifizieren und die Schaltung auf ihre Funktion zu prüfen, wurde zunächst ein Schaltbild mit dem Mentor IC Studio erstellt. Um bei der Simulation realistische Ergebnisse zu erzielen, ist auf die richtige Wahl des eingesetzten mathematischen Algorithmus für den Simulator zu achten. Außerdem muss zunächst der Arbeitspunkt der Schaltung bestimmt werden um eine Initial Condition, eine Anfangsbedingung, für den Quarz bei der Transientenanalyse zu setzen. Diese ist auch nur bei der Transientenanalyse gültig und gewährleistet das Aufstarten des Quarzes bei der Simulation. In der Realität übernimmt das Aufstarten des Quarzes das Hochfahren der Versorgungsspannung und das vorkommende Rauschen. Hierbei ist zu erwähnen, dass verschiedene Initial Conditions verschiedene Aufstartzeiten zur Folge haben.

Um sicher zu stellen, dass der Quarz auch ohne Anfangsbedingungen anschwingt, wird zunächst das Hochfahren der Versorgungsspannung simuliert. Hierbei wurde auf die Initial Conditions verzichtet.

Für die weiteren Simulationen wurde nun eine Anfangsbedingung gesetzt. Diese liegt im Arbeitspunkt der Schaltung für den Nominallauf. Um die Lastkapazität des Quarzes zu bestimmen wurden Simulationen mit folgenden Arbeitsbedingungen durchgeführt:

- Schwankungen der Versorgungsspannung mit VDD ± 0,2 V
- Temperaturbereich -40 °C – 120 °C
- Prozessschwankungen bei der Herstellung der Transistoren mit „worst speed“ und „worst power“ Simulationen
- statische Prozessschwankungen bei der Herstellung der Transistoren mit der Monte-Carlo-Simulation

Bevor die Schaltung simuliert wurde, ist im Hinblick auf das Layout die Schaltung noch modifiziert worden.

4. Simulationen

Das Ziel beim Layout bestand darin, es so kompakt und so klein wie möglich zu halten. Um dies zu erreichten, wurde wie erwähnt im Vergleich zum Schaltplan noch eine Veränderung vollzogen: Die idealen Kapazitäten wurden durch Transistoren ersetzt. Dabei wird die Gatekapazität der Transistoren genutzt, welche den größten Kapazitätsbelag hat. Die Layouts wurden mit dem Mentor IC Studio erstellt.

Alle PMOS-Transistoren wurden in eine Wanne gelegt. Die Breite der Verdrahtungsleitungen wurde nach den durchfließenden Strömen gewählt. Versorgungs- und Masseleitungen wurden mit 2 µm, Verbindungen von und zu den Inverter mit 1 µm und alle weiteren Leitungen mit 0,8 µm ausgelegt. Für die Verbindungen wurden größtenteils nur zwei Metallisierungsebenen, Metall 1 waagrecht und Metall 2 senkrecht, benutzt. Versorgungs- und Masse-Pins sind oben bzw. unten platziert. Der Ausgang der Oszillatorschaltung ist nach rechts geführt. Pinanschlüsse für den Quarz (X_{in} und X_{out}) sind nach links gelegt, um später beim Toplayout alle Pins untereinander anordnen zu können.

Um die Schaltung von Stress-, Prozess- und Temperaturgradienten weitestgehend zu entkoppeln, muss bei Block 3 auf Matching der Transistoren geachtet werden. Dabei wurden die Transistoren gefaltet, aufgespaltet und dann in einer Common Centroid Anordnug wieder zusammengefügt. Um das Matching zu vervollständigen, wurden die Längen der Zuleitungen gleich ausgelegt.

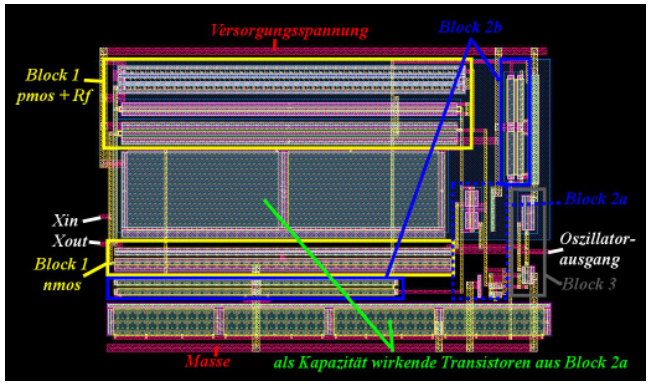


Abbildung 8: Layout der Oszillatorschaltung

Um das gesteckte Ziel zu erreichen, ein Layout zu erstellen, bei dem die Oszillatorschaltung nur eine Baugruppe der PLL ist, ist das Layout nun so angelegt worden, dass sich die Oszillatorschaltung am Padrand befindet.

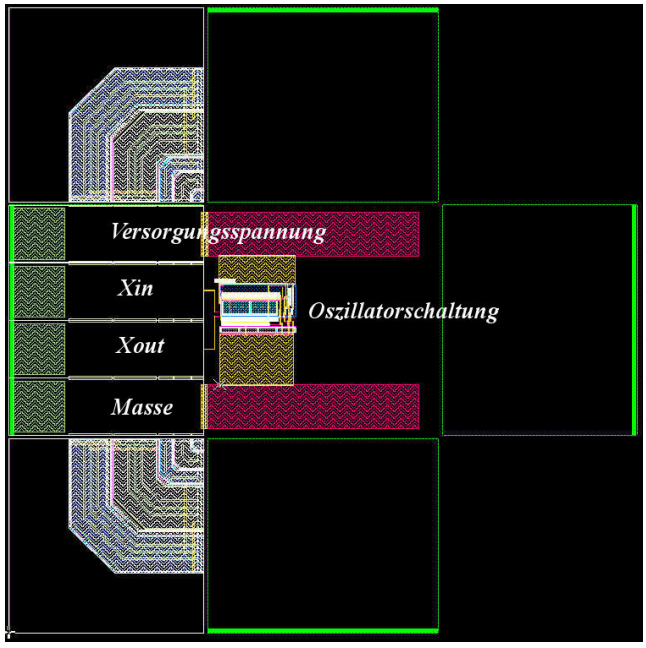


Abbildung 9: Layout der Oszillatorschaltung am Padrand

In Abbildung 9 ist zu erkennen, dass sich alle vier Pads untereinander befinden. Um Später die verschiedenen Baugruppen der PLL, z. B. den VCO an die Versorgungsspannung zu klemmen, ist außerdem eine Spannungsversorgungs- und eine Potentialschiene hinzugefügt worden.

5. Zusammenfassung

Anhand der Simulationen konnten für den Quarz folgende Spezifikationen ermittelt werden, darunter u. a. auch die minimale Lastkapazität:

Tab. 1: Spezifikationen des eingesetzten Quarzes

Frequenzbereich in MHz	10,065-10,070
Betriebsart	Grundton
Maximale mittlere Frequenzabweichung in ppm	300
Maximale Frequenzabweichung in ppm	50
ESR (R_1) in Ω	70 max.
Shunt Capacitance (C_0) in pF	6 max
Load Capacitance (C_1) in pF	16 min
Drive Level in µW	220 typisch
Arbeitstemperaturbereich	-40 °C ~ +120 °C

Ebenfalls wurden Spezifikationen des gesamten Quarzoszillators erstellt:

Tab. 2: Spezifikationen des Quarzoszillators

Nennfrequenz in MHz	10
Arbeitsfrequenz in MHz	10,068
Frequenzbereich in MHz	10,065-10,070
Betriebsspannung in V ± 0,2V	3,3
Arbeitstemperaturbereich	-40 °C ~ +120 °C
Mittlere Frequenztoleranz in ppm, maximale mittlere zulässige Abweichung von der Arbeitsfrequenz	300 max.
Mittlere Frequenztoleranz in ppm bei der Bezugstemperatur (25 °C)	50 max.
Mittlere temperaturbedingte Frequenzänderung in ppm im Arbeitstemperaturbereich	50
Mittlere Frequenzänderung in ppm in Abhängigkeit von der Betriebsspannung	50
Maximale Frequenzänderung in ppm	1300

Die Funktion des Quarzes und des Quarzoszillators sind anhand der Simulationen bestätigt worden. Der Quarz liefert wie erhofft eine Frequenz von ca. 10 MHz und ist gegenüber Temperatur- und Betriebsspannungen relativ unempfindlich. Die ermittelte minimale erforderliche Lastkapazität von 16 pF ist für einen Quarz relativ hoch, jedoch stellt sie kein Problem dar.

6. Quellen

[1] Neubig, B. und Briese, W.: „Das große Quarzkochbuch. Quarze, Quarzoszillatoren, Quarz- und Oberflächenwellenfilter (SAW), Meßtechnik“, Franzis Verlag GmbH, 1997

[2] Sansen, Willy. M.C.: „Analog Design Essentials“, Springer Verlag, 2006

[3] Kielkowski, Ron: „Inside Spice. Overcoming the obstacles of circuit simulation“, McGraw-Hill, 1994

[4] Fachverband Bauelemente der Elektronik, Tagungs-Dokumentation: „Schwingquarze ein unverzichtbares Bauelement in der Elektronik. Wirkungsweise, Eigenschaften und Anwendungen von Schwingquarzen“; VISTIS Verlag Berlin, 1985

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

[6] Hartl, H., Krasser, E., Pribyl, W., Söser, P. Und Winkler, G.: „Elektronische Schaltungstechnik. Mit Beispielen in PSpice“, Pearson Studium, 2008

Implementation of an IO-Link Interface Library Component for SoC Applications

Debayan Paul, Thomas Deuble, Cor Scherjon, Dr. B. Hoppe

Hochschule Darmstadt, 64295 Darmstadt & IMS Chips, 70569 Stuttgart

Telefon : +49-176-2471-3680, E-Mail : debayanpaul@yahoo.com, info@ims-chips.de, hoppe@eit.h-da.de

IO-Link is a new point-to-point communication and connectivity protocol between a master and a sensor/actuator (device), using which it is possible to transfer serial data by way of switching output, using only 3 wires. It is possible to develop an IO-Link (IOL) environment using a generic µC and programming it or using development kits available in the market. But here a custom approach has been taken to model the data link layer of a device, ignoring the above availabilities.

First the IO-Link (IOL) protocol was studied and information regarding the data link layer of the device was investigated. As a first time and totally independent approach, only the most basic functions were taken up for implementation. These include wakeup, receiving & sending back proper data (simplest frame-type) after analysis, fallback and time-out functions [1]. Behavioral modelling of the device was then done using VHDL which implemented the above functions. ModelSim® simulation was then carried out using proper test-benches. Then hardware synthesis was done using IMS Chips GateForest® CMOS 0.8 µm technology. The design was optimized using Design Analyzer®, after which the netlist was extracted and simulated.

Netlist simulation results indicate that this model is capable of delivering some basic functions (Parameter Data handling). Other than the advantages of less area and liberty of defining the clock-rate, this model can be used as a library component for future single-chip solutions of IOL devices.

1. Introduction

1.1. Background

Previously sensors and actuators have generally been connected to the peripheral assemblies of the control units via digital and analog voltage or current signals. For some time now, modern field devices have been unable to provide pure measuring or switching units. The new devices do not only receive measured values but are also equipped to output and configure data relating to processes, diagnostics, and parameters. Unfortunately, this data is not accessible through the conventional interface or, if it is, then only with great difficulty. The density of information provided is continually on the rise and the communication requirement is also increasing. In a search for ways of achieving an improvement, IOL has come to the forefront as a system that aims to revolutionize the sensor and actuator interface.

The process of signal connection should therefore become more intelligent without needing to change the topology and wiring technology involved. Furthermore, this interface should be able to connect to all common fieldbus systems. As a rule, it also means that individual distances of up to 20m must be bridged to enable the decentralized connection of process signals using conventional, unshielded signal cables and plugging and clamping technologies. This is precisely where the new communications standard of IOL shows its value. As a pure point-to-point connection between the master and the sensor, it is possible in IOL mode to transfer serial data additionally by way of the switching output. This is without the described complexity involving special cables, connectors, inputs and outputs, and without having to rely on proprietary solutions that often involve high cost. In this way, primarily parameter data are transmitted from the master to the sensor. In

the other direction, process data and, if necessary, even service and diagnostic data are relayed to the master.

It may apparently seem that these additional features and information also mean increased complexity and cost on the hardware side. But this is not so. The prices that tend to be higher, are more than offset by the time saved using IOL. Special emphasis should be given here to the fact that mixed operation of IO-Link enabled sensors and conventional sensors is possible.

The IO-Link protocol aims to eliminate any final bottleneck in the field of automation. It is a new and open standard being adopted by leading manufacturers in the fields of sensor and automation.

1.2. Motivation

25 renowned automation technology manufacturers joined together behind the doors of Profibus and Profinet International to form a working group aimed at defining this new standardized interface called IO-Link [3]. Among the IO-Link Consortium there are companies which offer a complete development package (kit) for building IOL enabled products. Still others provide a single chip solution only for the Physical Layer of an IOL device.

To obtain the knowledge required to build a custom IOL product or at least its electronics, one needs to be a member of the IO-Link Consortium. All technical specifications related to the protocol are described in the IO-Link Communication Specification v1.0 [1]. This is freely available, but it does not guide the reader as to how one should proceed to build the actual hardware or software and implement it in real life.

To make an existing sensor IO-Link enabled, one of the solutions is to use IOL development kits available in the market. This approach may lead to wastage of kit-resources, space, and clock & timing mismatches may arise. These maybe the main problems when one wants to integrate the sensor hardware and the IOL hardware, together on a SoC.

Miniature sensors require that a single ASIC will deliver all functionalities. So it is desirable here to develop an IOL interface library that can be integrated with the sensor ASIC, giving a single chip solution. This is the main motivation of the work presented here. As an independent approach, I propose here a working model of the data link layer of an IO-Link device which carries out the most basic functions.

1.3. Scope of This Work

The IO-Link protocol specification document [1] contains the definitions for the physical layer, the data link layer and the application layer for IO-Link master and IO-Link device according to the ISO/OSI reference model. In my work focus is given to IOL device development. To be more specific, modeling the data-link layer of the device is the prime focus of this work. The physical layer and the application layer are ignored.

Initially, after literature study of the IOL specification, information related to the device data-link layer was sorted out. Importance was given on how the device receives data, what operations have to be done with the received data and finally how and which type of data has to be transferred back to the master. All relevant timing requirements were also taken into account. In the second step, the hardware blocks for the data-link layer were modeled using VHDL and then simulated using ModelSim®. In the third step, the model was fed into Synopsys Design Analyzer® and synthesized using 0.8 µm CMOS GateForest® technology (Standard Cell Library from IMS Chips [2]). An analysis on the complexity for implementation in an ASIC was realized. Thus the model of the IO-Link device data-link has been developed here and will serve as a VHDL library component.

2. Theoretical Concepts

2.1. Topology and Connection

An IO-Link master can be connected to several devices at the same time. But only one device can be connected to each port of the master signifying a point-to-point connection. The master and the device follow a layered structure as per the ISO/OSI reference model. The most important layers here are the Application Layer, the Data Link Layer and the Physical Layer. This is illustrated in fig. 1.

The distinguishing features of the IO-Link system are:

• Single-master/single-device communication in half-duplex mode.

• 3 supported baud rates: 4,800 bauds (COM1), 38,400 bauds(COM2) and 230,400 bauds (COM3).

But COM3 is optional for the device.

• The physical interface or connection is a 3-wire system, named PHY2.

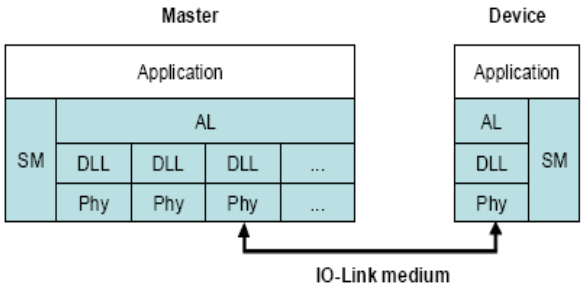


Fig. 1: Layered structure of the IO-Link protocol

The master mandatorily supports all 3 baud rates and the device supports any one. As a master has many ports for connection to different devices, so various devices can be communicating with one master at different baud rates.

The Physics2 (PHY2) system defines the connectivity of the Physical Layer between the master and the device. Such three wire connectivity is shown in fig. 2. Here the communication and power supply run via separate lines. L+ defines the power supply connection which is 24V, L- is the ground connection and C/Q is the connection for communication (C) or switching (Q) signal.

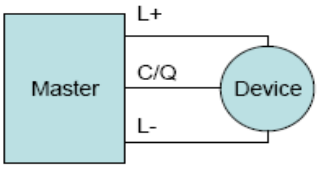


Fig. 2: Physics2, the 3-wire connectivity system

As a pure point-to-point connection, it is possible to either transfer serial data (IO-Link mode of operation), or by way of switching output (SIO mode). Fig. 3 shows the two possible states of the C/Q line. This is done by 24V signal modulation over the C/Q wire. *In my model the device supports only the IO-Link mode of data transfer.*

For data communication the “Non Return to Zero” (NRZ) code is used for bit-by-bit coding. The UART format is used for character-by-character encoding.

2.2. Use of Frames for Data Communication

The master and device exchange data in frames. A complete frame consists of the master telegram/frame and the device telegram/frame. These telegrams are in turn composed of one or many frame bytes, fig. 4, and one frame byte is one complete UART character.

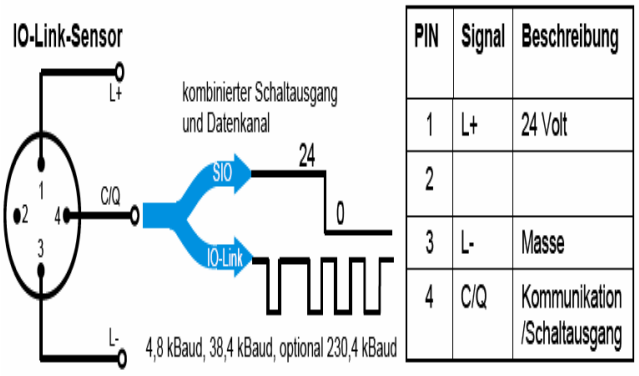


Fig. 3: States of the C/Q line

Various frame types can be selected to meet the particular needs of an actuator or sensor (scan rate, process data volume, etc.). The length of master and device telegrams may vary depending on the type of telegram and the data transmission direction.

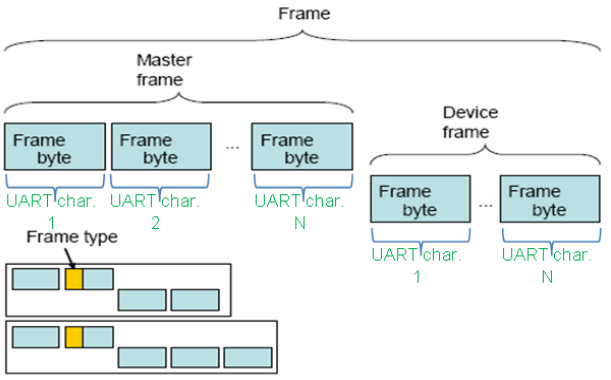


Fig. 4: Frame structure

In my work only frame Type0 has been implemented for data exchange, fig. 5. It only transmits “on-request” data [1]. 1 byte of user data is read from the device ROM for a Type0 read frame (frame P0_IS1R) per cycle. For a Type0 write frame (frame P0_IS1W), 1 byte of data is written to the device per cycle.

Communication between a master and an associated device takes place in a fixed schedule called the frame time, Tframe.

$$Tframe = (m+n)*11*Tbit + tA + (m-1)*t1 + (n-1)*t2$$

where, m is the number of UART characters sent by the master to the device and n is the number of UART characters sent by the device to the master within a frame. The bit time, Tbit is the time it takes to transmit a single bit.

$$Tbit = 1/(\text{data transmission rate})$$

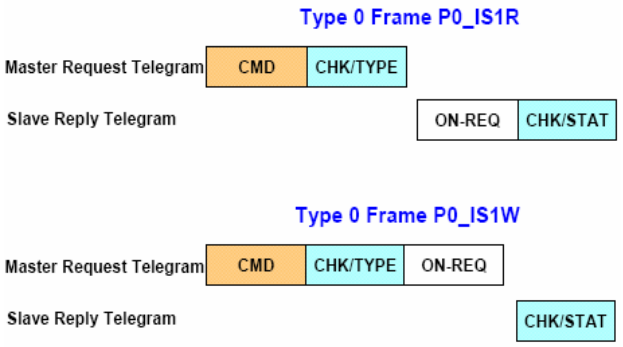


Fig. 5: Structure of frame type 0

In my model 38,400 bauds (COM2) is the data transmission rate and so the Tbit = 26.04 µs [2].

t1 is the master's character transmission delay, 0 ≤ t1 ≤ 1Tbit. t2 is the device's character transmission delay, 0 ≤ t2 ≤ 3Tbit. In my model t2 is not observed. Theoretically there is no pause between the transmissions of successive UART characters from the device to the master.

tA is the device's response time; the duration of the pause between the stop bit of the last UART character sent by the master being received and the start bit of the first UART character being sent by the device, 1Tbit ≤ tA ≤ 10 Tbit. *The tA observed in this model is approximately 54 µs [2]*. The position of the timings with respect to the frame bytes is shown in fig. 6.

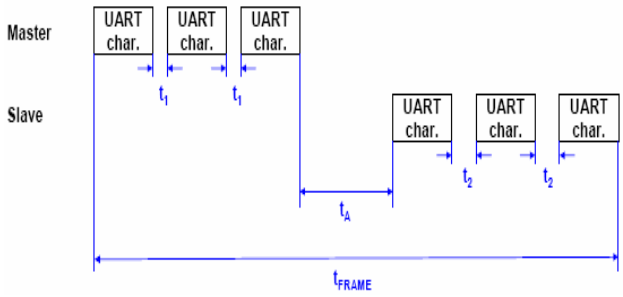


Fig. 6: Frame timings

2.3. States of Device Data-Link Layer & its Communication Behavior

As per the IOL specification [1] the functioning of the device data link layer can be explained by referring to a state diagram and it is briefly described below, fig. 7.

After **Power ON** the DLL of the device goes to the SIO state.

SIO: In the Standard Input/Output state, the port operates as a standard switching signal and is considered to be a process data bit.

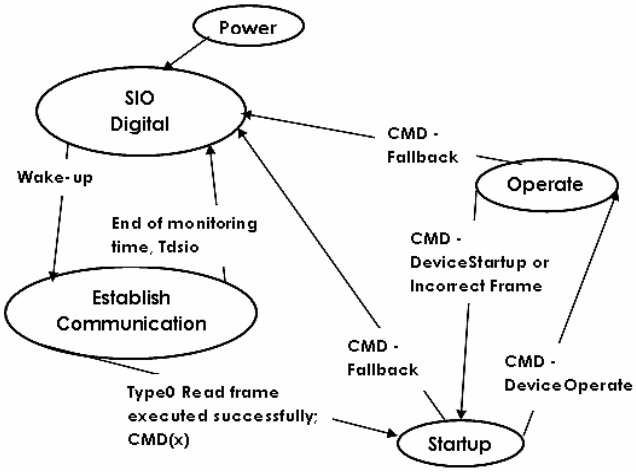


Fig. 7: State diagram of device data link layer

EstablishComm: The device switches to this state when a wake-up request (WURQ) is received. After this if a Type0 read-frame is executed successfully (i.e. parameter = 'Min Cycle Time' read-out), the device shall switch to the Startup state. Otherwise, once the monitoring time Tdsio has elapsed, it shall switch back to the SIO state.

Startup: Data exchange takes place in the Startup state. The device uses the DeviceOperate command to initialize its communication with the set parameters. In this state if a 'Fallback' command is received from the master, then the device switches to the SIO state.

Operate: In this state, the on-request and process data handlers are activated in order for data exchange to take place. Transition from this state to others is as depicted in fig. 7.

My model does not implement all the above states. After PowerON it remains in a wait state which is theoretically not the SIO state, and proceeds on to the next state upon WURQ receipt. As only frame Type0 is implemented here, so realization of the Operate state is not possible (it requires higher order frames types).

With the above information on device states, it is time to describe how exactly the master communicates with the device using frames and associated timings.

After 'PowerON' the device goes to the 'SIO' state and waits there for a WURQ from the master. The WURQ signifies that the master is trying to communicate with the device. As per the IOL protocol, the WURQ is current event on the C/Q line. But for this model, <1> the WURQ is a high pulse of duration 75 us, and <2> instead of the C/Q, a separate connection (wake_up) is used. After receipt of WURQ, the device becomes ready to receive frames from the

master. Consequently the state now shifts to the 'EstablishComm'.

After this the master executes a Type0 read frame (master command to read out 'Min Cycle Time' value from the device ROM [1]). Since all 3 baud rates are supported by the master, so the above data is first sent to the device with COM3, then with COM2 and finally with COM1. Depending on which baud rate the device supports, a valid response is obtained after COM1, COM2 or COM3. The waiting time of the master before switching to the next lower baud rate is Tdmt of the next baud rate [1]. If a valid response is received from the device, 'Startup' commences. In this state more communication parameters are first read out by the master from the device memory and then parameters are written to the device memory (read cycle followed by write cycle). After this shift takes place to the 'Operate' state which is not implemented here. This process is shown in fig. 8 and the significance of the numericals listed is as follows:

1> Master telegram in COM3

2> Master telegram in COM2

3> Master telegram in COM1

4> Response from device in COM1 (assumption that device supports COM1)

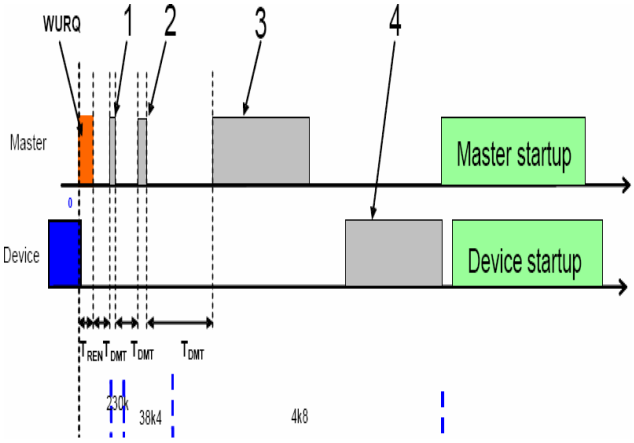


Fig. 8: Successful establishment of communication

3. Device Modeling

3.1. Overview

In this work, only the function of an IO-Link data link layer has been taken up for study and other features, electrical properties or connections involving the physical layer have been ignored. Accordingly the top level interconnections along with the VHDL units are as shown in fig. 9. Notice that L+, L- power lines are missing and for WURQ there is a dedicated connection.

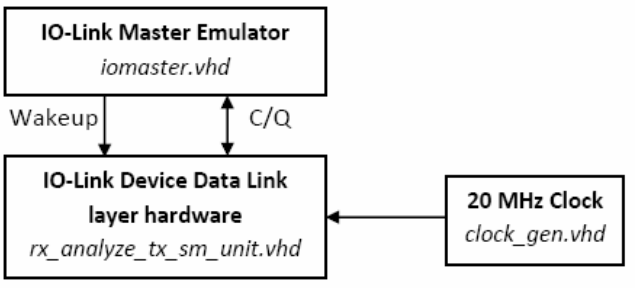


Fig. 9: Top-level interconnections

3.2. IO-Link Master Emulator

For simulation level analysis, a real IOL master cannot be used. So a VHDL unit has been developed, "IO-Link Master Emulator" which performs a few basic functions of the real IOL master. The emulator behaves as per the commands it encounters while reading sequentially through a text file, 'sequence.txt' [2]. This text file can be edited as per the various test conditions a user wants to execute. Type0 frames (read and write) are used to communicate with the device. Upon reception of reply from the device the master emulator generates pass or failure messages in the ModelSim® Transcript window (visual indication of a successful communication without having to inspect the ModelSim® Wave window).

The commands which are listed in the text file are as follows:

1> PAUSE <time>: Waits for an amount of time indicated in <time>. VHDL syntax is used for the time units, e.g. 1 µs or 100 ns.

2> WAKEUP <time>: Generates a wakeup pulse signal for <time>.

3> TX <data_byte_1> <data_byte_2> … <data_byte_N>: Sends data bytes (the text file contains integer entries) via the serial interface using the current baud-rate.

4> SETBAUD <baud-rate>: Sets the baud-rate (integer value) for transmission/reception.

5> CMDREAD <address> <cmpdata>: Transmits the frame Type0 command P0_IS1R and compares the received data with <cmpdata>.

6> CMDWRITE <address> <data>: Transmits the frame type 0 command P0_IS1W with ON-REQ data <data>.

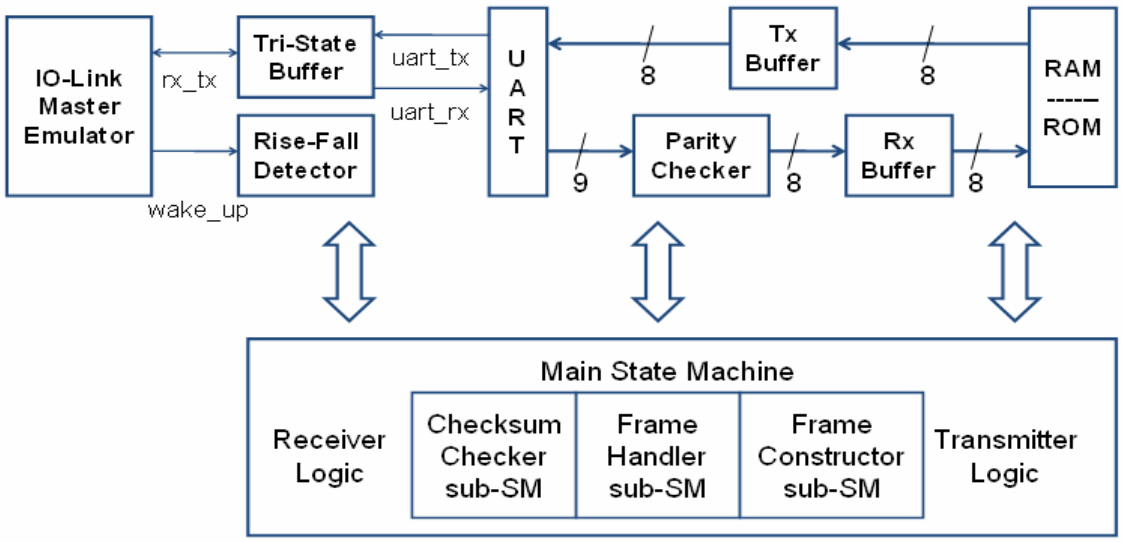


Fig. 10: Block diagram of the data link layer model

7> WAKEUP_SEQ: First performs a WAKEUP sequence, then tries auto-bauding to communicate data. Stops functioning after getting a valid data for device 'Min Cycle Time' read out.

3.3. Data Link Layer Model of the Device

The block level depiction of the data link layer model is as shown in fig. 10. Only the most important blocks, namely UART, Receive Buffer Memory, Transmit Buffer Memory, RAM/ROM module, sub-state machines and the main-state machines have been shown. The main and sub state machines control the UART behavior (receive or transmit mode), flow of data in and out of the receive & transmit buffers and also reading from ROM or writing to the RAM. The working of the above units gives us an understanding as to how this model works and is listed below.

1> The device after power up always remains idle (theoretically this is the SIO state). Under this condition, the receiver SM is active and it waits for the WURQ pulse. (SM = State Machine)

2> If the master sends a WURQ, the Receiver SM changes state after the WURQ request dies out. The Tri-State Buffer allows incoming data and the UART is ready to receive frames from the master.

3> The Receiver SM now continuously processes incoming serial data, and after checking for parity (Parity Checker block used), stores the 8 bit data into the Rx Buffer sequentially. If parity error occurs then the SM just waits for more data.

4> If no more data is sent by the master (timeout occurs) then the Checksum Checker sub-SM takes over control. It verifies the checksum for the received set of frames. If checksum error occurs, no more data is received and the state machine waits for a new WURQ.

5> If all goes well, then control is now transferred to the Frame Handler sub-SM. It analyzes the CMD byte and decides whether to read or write, from or to the device RAM/ROM. Also if a 'Fallback' command is received, then the SM switches to the waiting (SIO state) and remains idle.

6> For a valid command received, an acknowledgement needs to be sent back to the master in the form of reply frame/s. So the Frame Constructor sub-SM creates the necessary bytes and stores them in the Tx Buffer.

7> Control now shifts to the Transmitter SM (main SM) again. It reads out the bytes stored in the Tx Buffer and transfers them to the UART's internal buffer. All the while during the transfer process, the UART is in the transmit mode and the Tri-State Buffer allows sending out of data.

8> When sending of all data has been over then the controller switches back to the receive mode, i.e. the Receiver SM again becomes active.

9> A Timer unit (not shown in fig. 10) keeps track of the Tdsio whenever the main SM enters the data-receiving state. The minimum Tdsio value has been considered (as per [1]), and once 60 ms elapses, the main SM becomes idle (waits for WURQ).

4. Simulation Results

The behavioral simulation has been done using ModelSim® SE-64 6.0b. A test-bench provides the necessary stimulus to rx_analyze_tx_sm_unit.vhd along with the unit clock_gen.vhd and iomaster.vhd. The test-bench has to be run for 9 ms in order to execute all the commands listed in the sequence.txt file. The simulation sequence is shown in fig. 11. It has been broken into 7 parts and each part signifies the execution of a specific command (as per listings in the sequence.txt file).

The 1st command executed is WAKEUP_SEQ. The wakeup pulse is first generated which remains high for 75 µs (marked in yellow circle). Then the master emulator send data to the device using the highest defined baud rate (i.e. COM1). This is rejected (device doesn't reply back) as the device only supports COM2.

Next data is sent to the device using COM2 baud rate, which is accepted. A Type0 read frame is send by the master for which the 'Min Cycle Time' is read out from the device ROM. It is clearly seen that 2 bytes are received and then 2 bytes are transmitted back to the master.

The 2nd command is CMDWRITE 9 160. It writes the binary equivalent of 160 to memory location 9 of the device RAM. As a consequence, 1 byte of checksum data is sent back from the device to the master (acknowledgement).

The 3rd command, CMD 10 170 has the same function, the only difference is that here the data and memory location is different.

The 4th comm As a consequence, 1 byte checksum data is sent back from the device to the master (acknowledgement).
and is CMDREAD 9 160. The master emulator reads from device RAM location 9 and compares whether the value sent back from the device is 160. Apart from being a memory read command, it serves as a check for the 2nd command. The 5th command CMDREAD 10 170 has the same function as 4th, and it also verifies the 3rd command.

The 6th command is TX 90 34 and it signifies a 'Fallback' condition. After execution of this command the device goes back to idle condition and cannot receive any data till it is woken up again.

The 7th command is WAKEUP 1 µs which changes the idle condition of the device and enables it to receive data (marked in yellow circle). This command is used here to test the device recovery (re-wakeup).

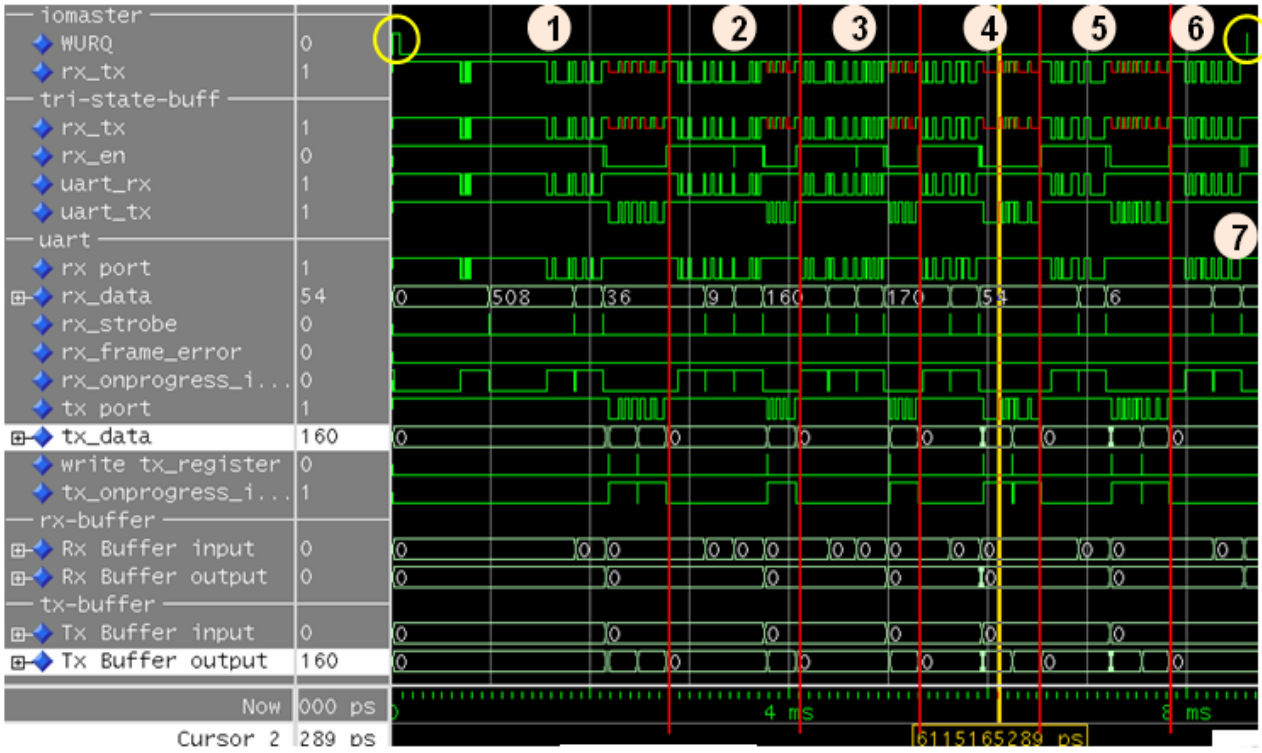


Fig. 11: Simulation result screenshot

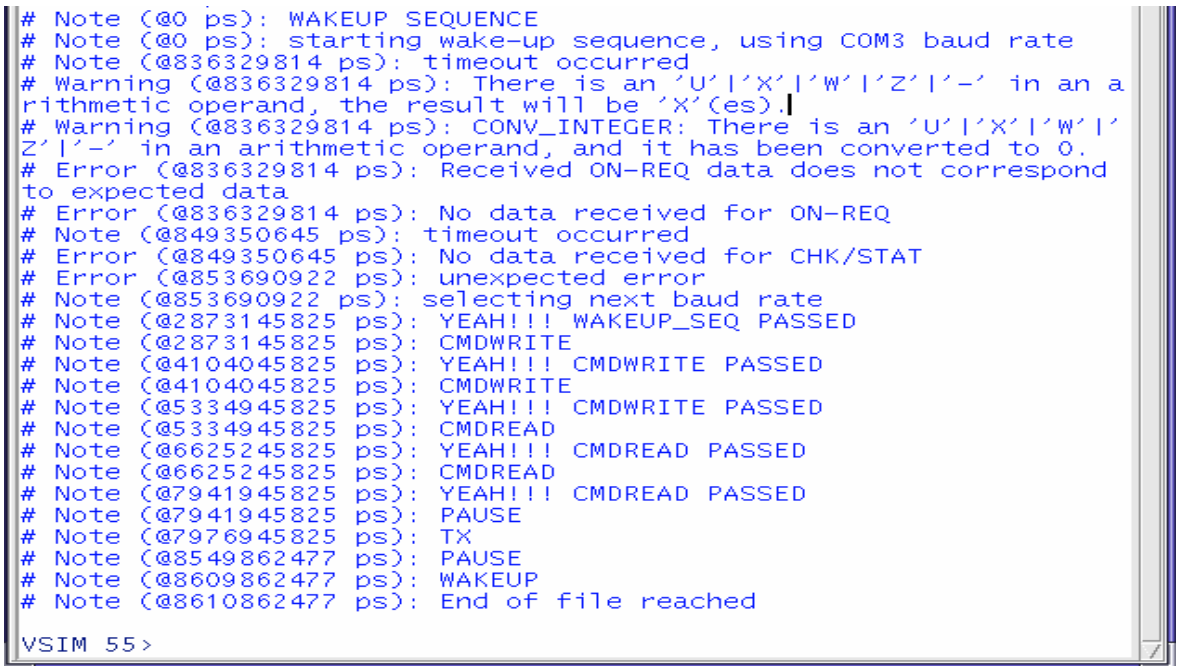


Fig. 12: Screenshot of the ModelSim® transcript window

The result of the execution of each command is listed in the ModelSim® transcript window as shown in fig. 12. Pass or failure messages are listed as command execution sequence progresses and when all commands have run, an 'End of file reached' message is displayed. The above simulation results indicate that the device can communicate with master properly using Type0 frames and Parameter data exchange can take place. Transmission error conditions are difficult to simulate and have not been attempted.

5. Synthesis Results

Hardware synthesis has been done using Synopsys Design Analyzer®, V-2003.12 for sparcOS5. This tool uses the IMS GateForest® Library GFN120. This library is implemented in a 0.8 µm CMOS technology using two metal wiring. So when any digital design is fed into this tool and synthesized, the resulting hardware is said to be realized using IMS GateForest® technology [2].

Design optimization is first done using medium map effort. Timing analysis was done and then incremental compilation was used to remove violations. Area analysis was later performed and the netlist was extracted. The netlist was simulated under ModelSim® and it behaves almost exactly as the behavioral model.

This model of the data link layer had no violated constraints and its maximum operating frequency is 73.53 MHz (system clock = 20 MHz). The total cell area is 6923 (out of these 320 flip-flops are used for the buffer memories and the RAM/ROM module combined), which includes combinational as well as non-combinational areas (net interconnect areas have not been taken into consideration).

6. Limitations

This model supports only Type0 frames, both read and write. Higher order frames need to be implemented to enable complex data exchange. This will enable the complete modelling of the device data link layer. Here the WURQ and frames (data) are handled by two different connections. But as per the specification, all communications should take place over the C/Q line only. Also, in the ideal case, if a parity error occurs, the device state-machine should quit receiving data and switch to the 'SIO' mode. This is so because it would be pointless to receive other UART characters if one is damaged. But in this model, the state machine continues to receive data. If some unrecognized data/command is received the state machine logic simply does nothing and later the time-out function Tdsio takes over.

7. Conclusion & Summary

State-machines were used to do the complex task of overall controlling as well as frame-handling of the data. Data transfer is done by the UART, buffer memories are used for intermittent data storage whose final destination is the RAM/ROM. The main advantage here was that the complexity of the state-machines and the amount of memory and registers needed could be designed as per specific needs. Liberty to choose the clock frequency was also there and it was shown that this model cannot operate over 73 MHz. This study also reveals that IO-Link is a rather slow protocol with no harsh timing requirements and we can have the luxury of multiple data fetch cycles.

The IO-Link Master Emulator does an excellent job of a master. Additionally, it also serves the purpose of an intelligent verification module and is flexible enough to test this model of the device. So all possible communication methods (within the scope of Type0 frame & 'Startup') were tested and verified.

A basic working model of the IO-Link device data link layer has thus been conceived here. This device model can communicate with the master and simplest form of data (parameter data) exchange is possible. It conforms to the protocol and obeys most of the constraints.

The accommodation of higher order frame types will enable implementation of the 'Operate' state of fig.7. Also if the 'SIO' mode is implemented, then the complete modeling of the data link layer will be realized. This model, coupled with the IOL physical layer can be integrated with existing sensor electronics. Thus the realization of a sensor ASIC delivering IO-Link capabilities can be conceived.

Acknowledgements

I sincerely thank Institut für Mikroelektronik Stuttgart (IMS Chips), for coming up with the idea of this investigative project on IO-Link device and allowing me to work on it.

A lot of thanks to Thomas Deuble, for guiding me through the entire process of planning and coming up with an idea. Special thanks to Cor Scherjon, for helping me out in fine tuning my work.

Many thanks to Professor B. Hoppe and Hochschule Darmstadt for always being there and helping me out in all possible ways.

References

[1] IO-Link Communication Specification, Version – 1.0, January 2009 (Document identification: IOL-09-0001). www.io-link.com

[2] "Implementation of an IO-Link Interface Library Component for SoC Applications". M.Sc. Thesis by Debayan Paul done at IMS Chips, Stuttgart in co-operation with Hochschule Darmstadt, Darmstadt.

[3] http://www.io-link.com/en/index.php

Charakterisierung von Hallplatten für integrierte, intelligente Mikrosysteme und Entwurf einer spannungs- und temperaturstabilen Stromquelle in der IMS 0,5 µm GATE-FOREST-Technologie

Torsten Schmitz (HS Esslingen), Johannes Thielmann (IMS)

Institut für Mikroelektronik Stuttgart, Allmandring 30a, 70569 Stuttgart

Telefon 0 711 / 21 855 -10, Fax -100, E-Mail info@ims-chips.de

Hallsensoren finden in vielen Anwendungen in der Industrie ihren Einsatz. Dazu gehören beispielsweise die Lagemessung des Stators in Synchronmotoren oder die Winkelmessung in Drosselklappen und elektronischen Lenksystemen im Kraftfahrzeug. Die Drehzahl- und Geschwindigkeitsmessung bis hin zu Strommessung in Fehlerstromschutzschaltern (RCD) sind ebenfalls Anwendungen. Dabei sind die Sensoren wartungsarm, widerstandsfähig gegen Umwelteinflüsse und haben eine hohe Lebensdauer.

Für die IMS 0,5 µm Gate-Array-Technologie wurden Hallplatten vermessen, die als n-Wannen auf einem Chip vorliegen. Diese Hallplatten sollen in Zukunft für ein vom IMS entwickeltes Hallsensorsystem eingesetzt werden. Daher war es nötig, das Hallsignal, den Widerstand und die Offset-Spannung ohne integrierte Auswerteschaltung zu untersuchen.

Da die Hallplatten einen konstanten Strom benötigen, der über Temperatur und Versorgungsspannung stabil ist, wurde eine neue Referenzstromquelle entworfen. Des Weiteren wurden zwei Methoden entwickelt, um den Temperaturgang zu kompensieren.

1. IMS 0,5 µm Gate-Array

Die IMS GATE-FOREST-Technologie ist eine Gate-Array, beziehungsweise Sea-of-Gates Architektur. Das Gate-Array Prinzip besteht darin, dass die für die Schaltung verfügbaren Bauelemente (T, R, C), vordefiniert und vorgefertigt sind(Masterstruktur) und die jeweilige Schaltung durch die Verschaltung dieser Elemente realisiert wird. Die Masterstruktur beinhaltet einen kleineren Analogteil und einen größeren Digitalteil (Abbildung 1).

Die Grundzelle des Analogteils beinhaltet Widerstandsbereiche mit einem Schichtwiderstand von 10 Ω/□ und 80 Ω/□. Des Weiteren befinden sich N-Kanal und P-Kanal-Transistoren mit unterschiedlichen W/L-Verhältnissen und 16 Kapazitätseinheiten mit je 90fF in einer Zelle. Die Grundzelle des Digitalteils besteht aus je zwei NMOS- und PMOS-Transistoren mit vorgegebener Transistorlänge und Weite. Diese Elemente können je nach Schaltung in einem Layout- Prozess in der IMS internen Technologie verbunden werden. Durch den Digital- und Analogteil ist eine Mixed-Signal Anwendung möglich [6].

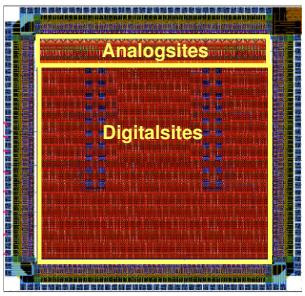


Abbildung 1: Prinzip des Gate-Array Masters

1.1. Master mit Sensorfeld

Für spezielle Anwendungen wurde in den Digitalteil ein Sensorfeld integriert (Abbildung 2). Dieses Sensorfeld beinhaltet ausschließlich n-Wannen. Diese sind für verschiedene Anwendungen einsetzbar. In dieser Arbeit wurden die n-Wannen als Hallplatten eingesetzt und ohne integrierte Auswerteschaltung vermessen. Die Hallplatten sollen in Zukunft in einem Hallsensorsystem eingesetzt werden und wurden aus diesem Grund charakterisiert.

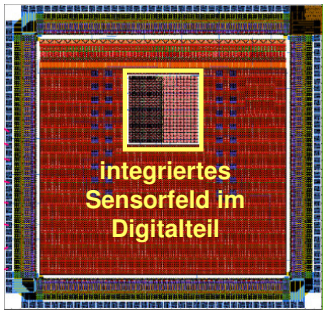


Abbildung 2: Master mit Sensorfeld

2. Der Halleffekt

Steht ein Magnetfeld B senkrecht auf einer Hallplatte und fließt ein konstanter Strom I_{REF} durch diese, entsteht eine Spannung U_H senkrecht zur Stromrichtung. Der Grund ist, dass durch die Lorentzkraft die Ladungsträger in der Hallplatte verschoben werden, wodurch ein Potentialunterschied senkrecht zur Stromrichtung entsteht (Abbildung 3) [1] [4].

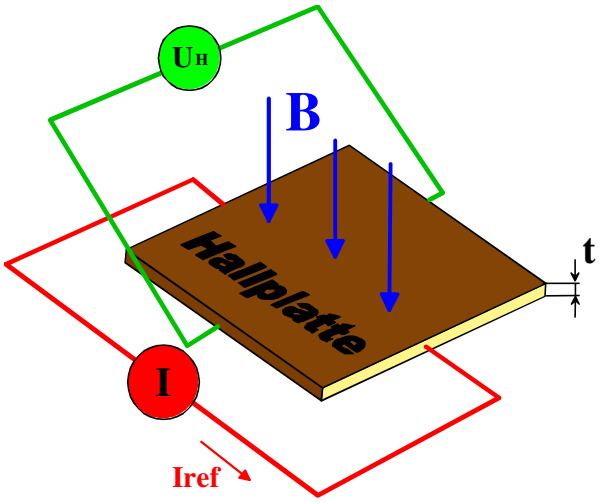


Abbildung 3: Halleffekt an Hallplatte

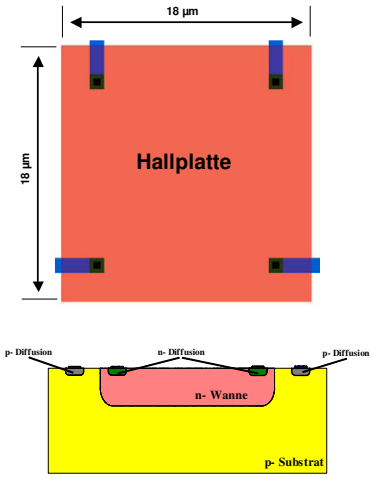
$$U_{H\_MAX} = R_H * I_{REF} * B_{MAX} * \frac{1}{t} \qquad \text{mit} \qquad R_H = \frac{1}{ne}$$

Die resultierende Spannung wird durch das Magnetfeld B und dem Referenzstrom I_{REF} bestimmt. Außerdem geht die Volumendichte *n* und die Tiefe *t* antiproportional in das Hallsignal mit ein. Das bedeutet, dass bei sinkender Ladungsträgerdichte und geringerer Tiefe der Hallplatte die Sensitivität der Hallplatte steigt.

Die Mobilität der Ladungsträger kann auch einen Einfluss auf die Hallspannung haben. Jedoch ist diese nur signifikant, wenn beide Ladungsträgerarten vorhanden sind. (Elektronen und Löcher). Da die Hallplatten n dotiert sind und deshalb nur Elektronen vorhanden sind, kann die Mobilität vernachlässigt werden und der vereinfachte Hallkoeffizient R_H ist korrekt [4].

3. Hallplatten im IMS Gate-Array

Es wurden zwei unterschiedlich große Hallplatten vermessen, eine Geometrie mit einer Kantenlänge von 40 µm x 40 µm und die etwas kleinere 18 µm x 18 µm. Die Hallplatte kann symmetrisch angesteuert werden, was für die Offsetkompensation benötigt wird. Die Tiefe *t* und die Dotierung *n* sind durch die Technologie vorgegeben und können nicht variiert werden. Aus diesem Grund muss ein ausreichend großer Strom I_{REF} durch die Hallplatte getrieben werden, damit eine ausreichende Sensitivität von mindestens 100 mV/T gegeben ist.



n-Wanne als Hallplatte

Abbildung 4: n-Wanne als Hallplatte

3.1. Forderungen an die Hallplatte

Folgende Forderungen werden an die Hallplatten gestellt:

- Referenzstrom I_{REF} von 1 mA
- Spannungsfall an der Hallplatte maximal 3.5 V
- Signalhub von 4,0 mV bei 40 mT (100 mV/T)
- geringe Offset-Spannung

Der Spannungsfall über der Hallplatte darf nicht mehr als 3,5 V betragen, da sonst die Stromquelle des Hallsensorsystems nicht mehr korrekt arbeiten kann. Diese Forderung kann durch die kleinere Hallplatte erreicht werden, da diese einen kleineren Widerstand als die große Hallplatte aufweist. Der Signalhub sollte bei 40 mT mindestens 4 mV betragen (100 mV/T), damit dem Hallsensorsystem ein ausreichendes Signal für die Signalverarbeitung zur Verfügung steht. Diese Forderung kann mit beiden Hallplattengeometrien erreicht werden. Die Offset-Spannung ist ohne Kompensation jedoch noch zu hoch und liegt zwischen 3 mV und 5 mV. Daher wurde eine Kompensation nach der „Spinning-Current-Methode“ durchgeführt.

3.2. Ursachen für einen Offset-Spannung

Die Hallplatte kann als Wheatston'sche Brückenschaltung betrachtet werden (Abbildung 5).

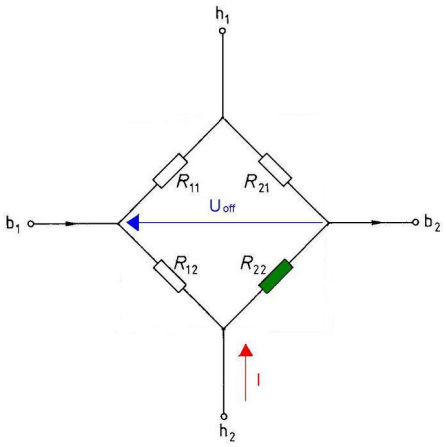


Abbildung 5: Ersatzschaltbild der Hallplatte

Folgende Effekte verursachen eine Offset-Spannung, indem die Brückenschaltung verstimmt wird. Das bedeutet, dass auch ohne anliegendes Magnetfeld eine Spannung an der Hallplatte gemessen wird [1].

Ursachen für eine Offset-Spannung können sein:

- piezoresistive Effekte
- Metall-Silizium Kontakte
- Inhomogene Temperaturverteilung (z.B. durch Umgebung, Strom)
- variierendes Dotierprofil an der Hallplatte
- Ungenauigkeiten in den Maskenausrichtung

4. Spinning-Current-Methode

Durch die symmetrische Ansteuerung der Hallplatte, sind die Beträge der Offset-Spannungen in allen diagonalen Stromrichtungen ähnlich. Diese Eigenschaft wird bei der Spinning-Current-Methode genutzt

4.1. Prinzip

Der Hallstrom wird, jeweils um 90° versetzt, in 4 Richtungen durch die Hallplatte getrieben. Dadurch fällt der Offset 2-mal positiv und 2-mal negativ aus. Durch die symmetrische Anordnung der Kontakte wird sich die Offset-Spannung bei der Addition der 4 Werte kompensieren.

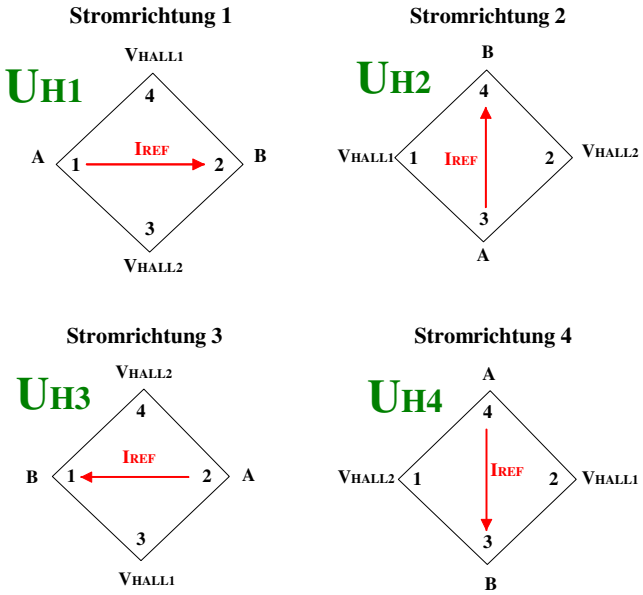


Abbildung 6: Spinning Current

$$U_{HALL\_gesamt} = U_{H1} + U_{H2} + U_{H3} + U_{H4}$$

4.2. Ergebnis

Folgendes Schaubild zeigt die kompensierte Offset-Spannung bei einem Hallstrom von 0 mA bis 2.5 mA.

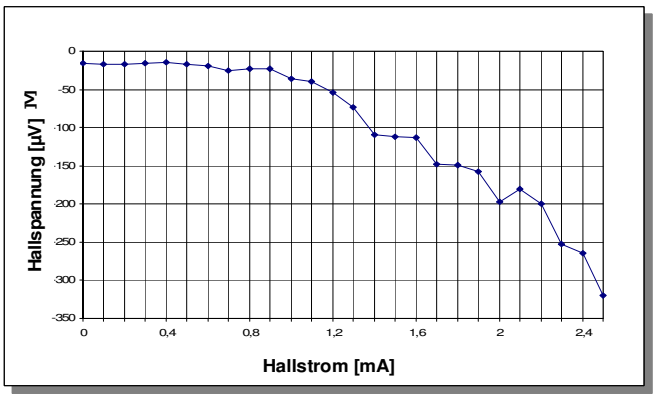


Abbildung 7: Kompensierte Offset- Spannung

Bis zu einem Strom von 1,2 mA liegt der Restoffset unter 50 µV. Bei höheren Strömen machen sich verstärkt Nichtlinearitäten im Silizium bemerkbar. Im Durchschnitt kann die Offset-Spannung um den Faktor 22 (96%) verringert werden.

5. Messaufbau

Für die Messung der Hallspannung ist ein definiertes Magnetfeld notwendig. Dafür wurden 2 Arten von Spulen vermessen, wobei die für die Messung ab besten geeignete Spule für den Messplatz im Labor genutzt wurde.

5.1. Spulen

Es wurde eine Helmholtzspule und eine Trafoblechspule vermessen. Eine Helmholtzspule zeichnet sich durch ihr sehr homogenes Magnetfeld und das große Experimentiervolumen aus. Durch die Trafobleche der Trafoblechspule kann dagegen ein höheres Magnetfeld erreicht werden, da diese eine hohe Permeabilität besitzen. Jedoch muss hier auf eine Remanenzfeldstärke der Trafobleche geachtet werden.

Beide Spulen funktionieren nach demselben Prinzip. Es sind 2 Spulenteile vorhanden, durch die jeweils ein Strom in dieselbe Richtung getrieben wird (Abbildung 8). Dadurch entsteht im Inneren des Spulenpaars ein homogenes Magnetfeld [2]. Abbildung 9 stellt die Feldstärke im Inneren beider Spulen gegenüber.

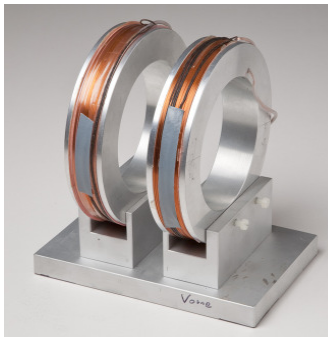


Abbildung 8: Beispiel Helmholtzspule

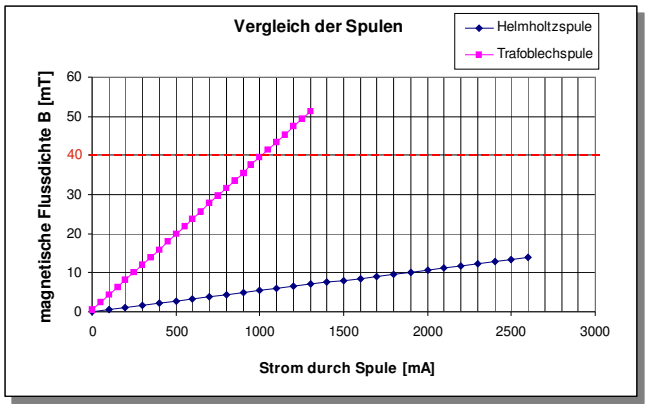


Abbildung 9: Spulenvergleich

Das Schaubild zeigt, dass die Helmholtzspule nicht die geforderte Magnetfeldstärke von 40 mT erreichen kann, ohne eine zu hohe Leistung zu beanspruchen. Daher wurde für den Messplatz die Trafoblechspule gewählt. Diese hat zwischen 0 A und 1 A einen Magnetfeldhub von 40 mT.

6. Messung des Hallsignals

Abbildung 10 zeigt das Hallsignal an einer Hallplatte, in Abhängigkeit des Magnetfeldes der Trafoblechspule. Dabei sind alle 4 Stromrichtungen aufgetragen.

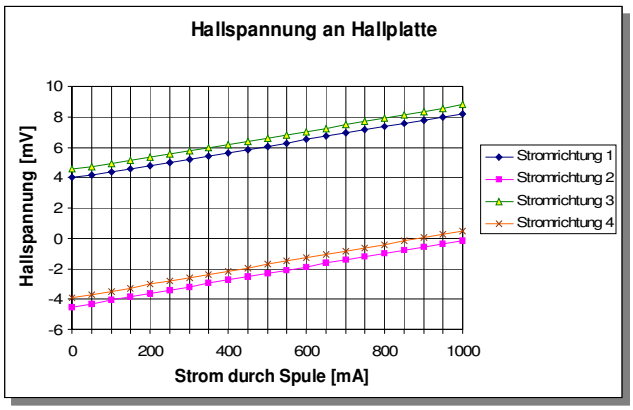


Abbildung 10: Hallsignal in 4 Stromrichtungen

Der Signalhub jeder Stromrichtung beträgt ~4,3 mV, wodurch die geforderte Sensitivität erreicht wird. Die Offset-Spannung liegt in diesem Beispiel bei ~4 mV.

Durch die Spinning-Current-Methode kann nun der Offset kompensiert werden (Abbildung 11). Der Restoffset liegt bei ca. 30µV. Der 4-fache Spannungshub kommt durch die numerische Addition der 4 Stromrichtungen zustande.

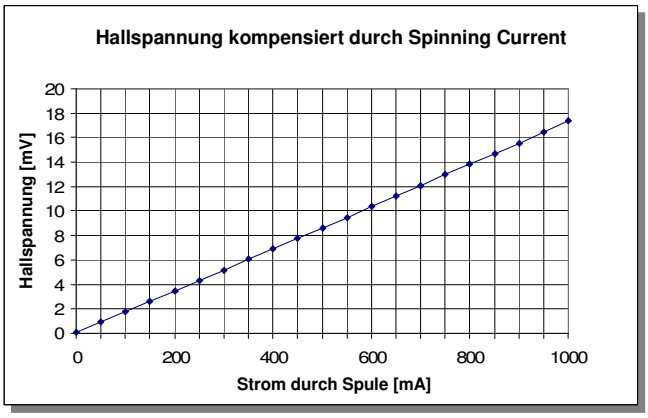


Abbildung 11: Kompensierte Hallspannung

7. Referenzstromquelle

Das vorhandene Hallsensorsystem besitzt bereits eine Stromquelle zur Generierung eines Referenzstromes I_{REF} für die Hallplatten [3]. Jedoch entspricht diese Stromquelle noch nicht den Anforderungen. Es zeigen sich starke Abhängigkeiten von der Temperatur und der Versorgungsspannung. Aus diesem Grund wurde eine neue Stromquelle entworfen, die diesen Anforderungen gerecht wird. Dazu wurden zwei Methoden für die Stromquelle entwickelt, die die Temperaturabhängigkeit des Regelwiderstandes kompensieren.

Anforderungen an die Stromquelle:

- Referenzstrom I_{REF} von 1 mA
- Betrieb an 5 V Versorgungsspannung
- geringe Abhängigkeit von der Versorgung sspannung (4,75 V bis 5,25 V) und Temperatur (-40 °C bis 85 °C)
- Abweichung von maximal 2 %
- Hallplatte soll direkt an VDD anliegen

7.1. Stromregelung mit Bandgap

Die Grundidee der Stromquelle ist eine Stromregelung mittels einer Bandgap-Referenz zu verwirklichen (Abbildung 12). Diese zeichnet sich durch ihre spannungs- und temperaturstabile Referenzspannung aus [5].

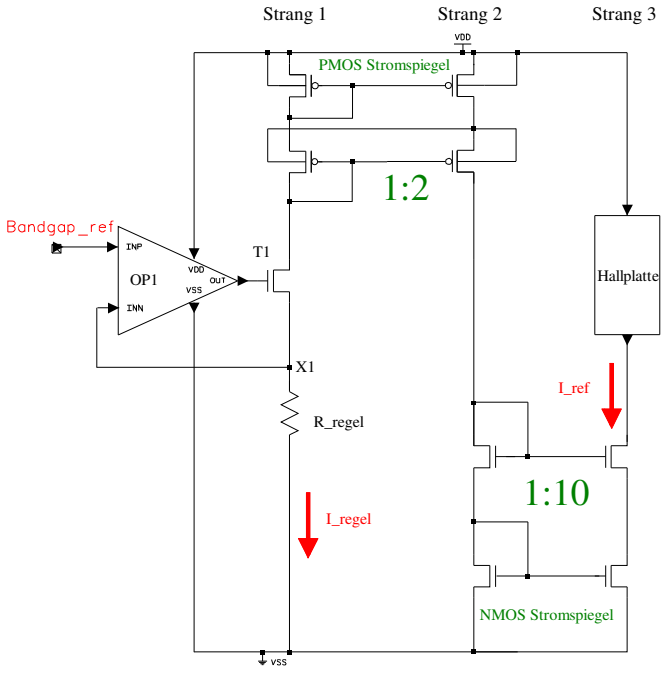


Abbildung 12: Einfache Stromregelung

Die Spannung bandgap_ref von 1,25 V wird am Knoten x1 des Widerstands R_regel durch den OP1 geregelt. Durch den Ohm'schen Zusammenhang wird demnach ein Strom ($I_{\_regel} = U_{bandgap\_ref} / R_{\_regel}$) durch den Strang 1 fließen. Dieser liegt bei 50 µA. Durch das Verhältnis der Transistoren im Stromspiegel wird der Strom aus Strang 1 mit 20 multipliziert, wodurch durch die Hallplatte in Strang 3 ein Strom von 1 mA fließen wird. Der resultierende Strom über der Temperatur ist in Abbildung 13 zu erkennen. Die 2-fache Spiegelung ermöglicht es, dass die Hallplatte direkt an VDD liegen kann, was für das Hallsensorsystem nötig ist. Die Kaskadierung erhöht den Ausgangswiderstand der Stromquelle um ein Vielfaches, was die Variation des Stromes durch eine geringe Widerstandsänderung bei der Spinning-Current-Methode verhindert.

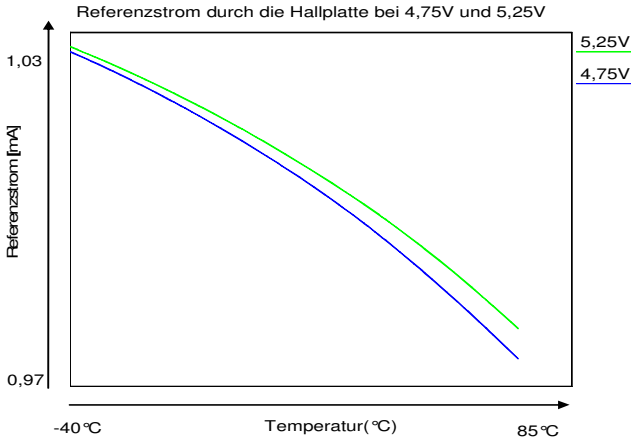


Abbildung 13: Referenzstrom

Die Spannungsabhängigkeit kann durch die Bandgap stark reduziert werden und zeigt nur noch eine Abweichung von 0,5 %. Der positive Temperaturkoeffizient des Regelwiderstandes macht sich jedoch stark im Referenzstrom bemerkbar. Die Abweichung über der Temperatur und Versorgung liegt bei typischen Parametern bei 4,5 %. Daher wurden durch zwei Methoden der positive Temperaturkoeffizient des Regelwiderstandes kompensiert.

7.2. Kompensationsmethoden

Für beide Kompensationsmethoden wird ein temperaturabhängiges Signal aus der Bandgap-Schaltung benötigt. Dieses ist antiproportional zur Temperatur und hat einen Spannungshub von 200 mV zwischen -40 °C und 85 °C. Aus Abbildung 14 kann das Signal v_temp entnommen werden.

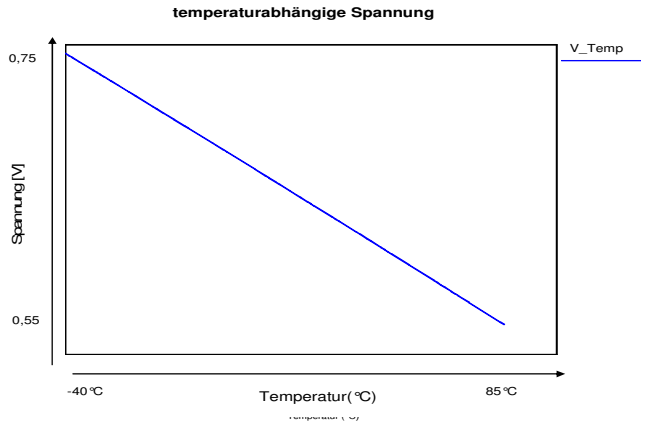


Abbildung 14: Signal v_temp

7.2.1 Methode 1: Komparatoren

In der ersten Kompensationsmethode wird der Regelwiderstand in Abhängigkeit der Temperatur direkt beeinflusst.

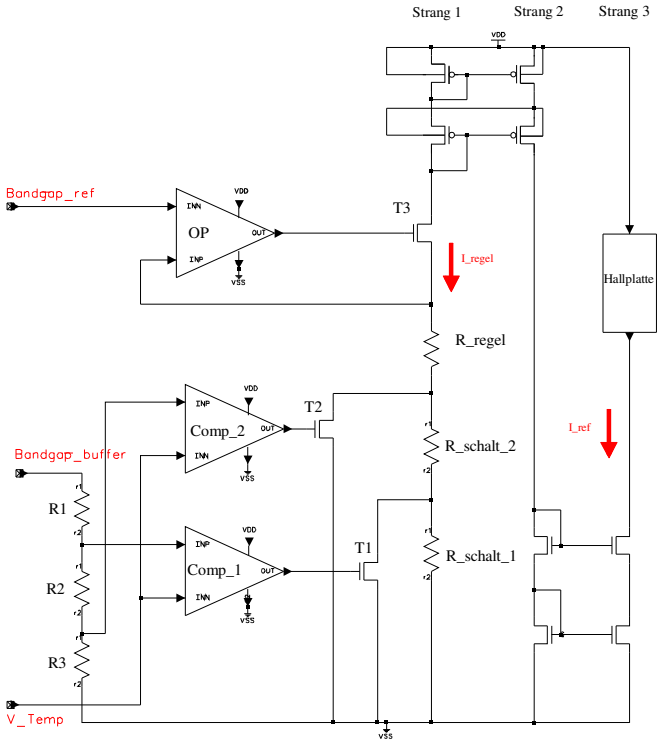


Abbildung 15: Komparatorkompensation

Bei Temperaturen unter 15°C sind die Komparatoren ausgeschaltet und die Widerstände R_regel, R_schalt_1 und R_schalt_2 bilden den Gesamtregelwiderstand. Die Schaltschwellen für die Komparatoren werden durch den Spannungsteiler mit den Widerständen R1, R2 und R3 erzeugt. Diese Schaltschwellen werden durch die Komparatoren mit dem temperaturabhängigen Signal v_temp verglichen.

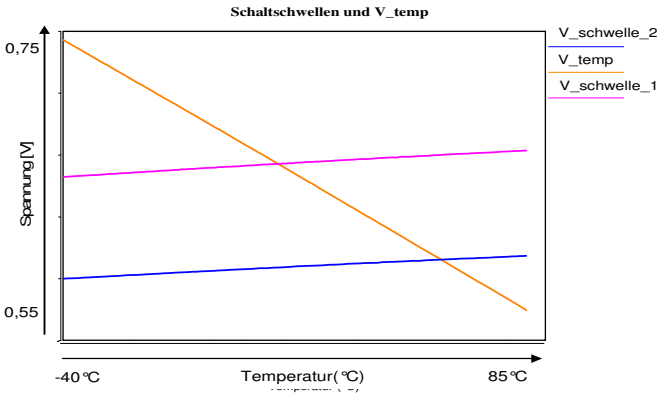


Abbildung 16: Schwellen an den Komparatoren

Abbildung 16 zeigt die Spannungen, die an den Komparatoren anliegen, sowie das temperaturabhängige Signal v_temp. Steigt die Temperatur bis ca. 15°C an, wird durch den Komparator Comp_1 und Transistor T1 der Widerstand R_schalt_1 kurzgeschlossen, wodurch sich der Gesamtregelwiderstand verkleinert. Steigt die Temperatur weiter an, wird bei ca. 60°C auch der Widerstand R_schalt_2 kurzgeschlossen, wodurch nur noch R_regel an der Bandgap-Spannung anliegt und den Strom durch Strang 1 bestimmt.

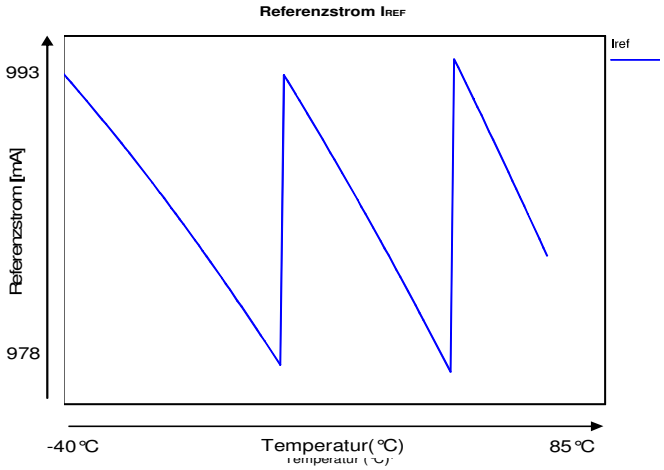


Abbildung 17: Kompensierter Referenzstrom

Abbildung 17 zeigt den resultierenden Referenzstrom. Durch geeignete Dimensionierung des Spannungsteilers und der Schaltwiderstände, kann der Temperaturgang minimiert werden. Durch diese Methode konnte die Abhängigkeit bei typischen Parametern auf 2% reduziert werden. Durch die Komparatoren könnte sich

bei kritischen Temperaturen jedoch ein Schwingen an den Komparatoren einstellen, was durch eine Hysterese an den Komparatoren verhindert werden kann.

7.2.2 Methode 2: Subtrahierverstärker

In der zweiten Kompensationsmethode wird der Regelwiderstand nicht direkt beeinflusst, sondern ein Strom im Strang 2 subtrahiert.

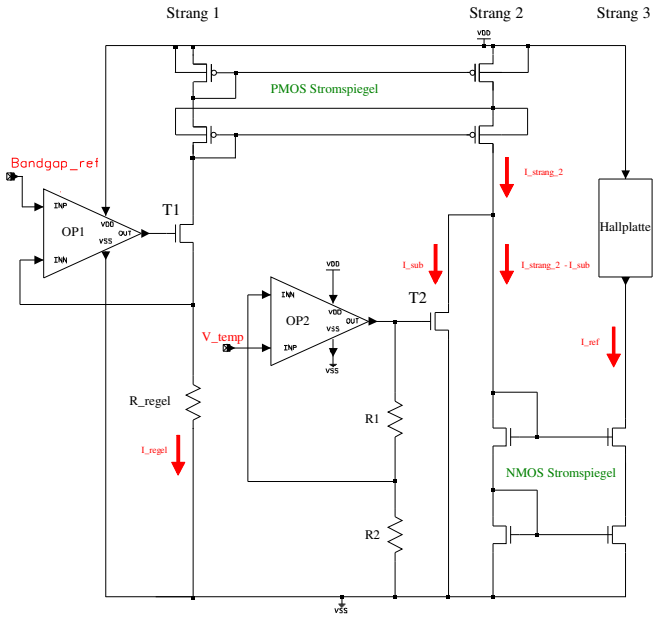


Abbildung 18: Subtrahierkompensation

Abbildung 18 zeigt die Kompensation mittels eines nicht-invertierenden Verstärkers mit OP2. Die Regelung in Strang 1 ist gleich wie bei der einfachen Regelung. Jedoch wird durch den Verstärker mit OP2 in Abhängigkeit der Temperatur ein Teilstrom aus den Strang 2 subtrahiert, so dass der Referenzstrom I_{REF} in Strang 3 kompensiert wird.

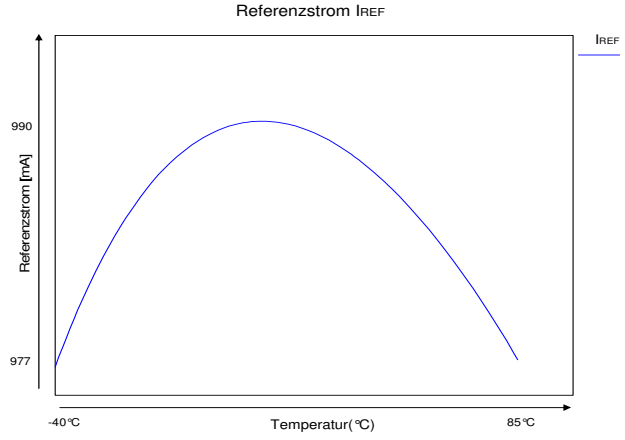


Abbildung 19: Kompensierter Referenzstrom

Die Kompensation kann durch das Verhältnis von R1 und R2 eingestellt werden, bis der Strom bei -40°C und 85°C denselben Betrag hat.

Abbildung 19 zeigt den resultierenden Strom I_{REF}. Durch die stetige Kompensation sind keine Schwellen zu erkennen, wodurch auch Schwingungen verhindert werden. Die Abweichung unter typischen Bedingungen konnte auf 1,6 % reduziert werden.

7.3. Zusammenfassung

Die Hallplatten entsprechen den Anforderungen für das Hallsensorsystem aus [3] und die starke Offset-Spannung der Hallplatten kann mit der Spinning-Current-Methode zuverlässig kompensiert werden. Weiterhin wurde eine neue Stromquelle für das vorhandene Hallsensorsystem in der IMS 0,5 µm Gate-Array-Technologie entwickelt.

Die Abhängigkeit des Stromwertes von der Versorgungsspannung kann in beiden Kompensationsmethoden, wie auch schon bei der einfachen Stromquelle, nahezu verhindert werden und liegt bei ~0,5 %. Bei der Stromquelle mit den Komparatoren konnte die geforderte Gesamtabweichung von 2 % unter typischen Parametern eingehalten werden. Mit der Subtrahiermethode wurde dieser Wert sogar noch unterschritten und liegt bei 1,6 %. Ein eventuelles Schwingen an den Komparatoren der ersten Methode kann durch eine Hysterese verhindert werden. Dies ist bei der Subtrahierlösung nicht nötig, da kein Schwingen auftreten kann. Jedoch werden sich technologiebedingte Toleranzen des Subtrahiertransistors stärker auf den Referenzstrom auswirken, als bei den Schalttransistoren an der Komparatorkompensation, da diese direkt den zu subtrahierenden Strom beeinflussen. Die Schalttransistoren fungieren lediglich als Schalter, um die jeweiligen Widerstände kurzzuschließen. Eine Cornersimulation hat dies bestätigt. Bei der kritischsten Cornersimulation kann die Abweichung der Komparatormethode auf 3,3 % und bei der Subtrahiermethode auf 4 % ansteigen.

Da beide Versionen der Stromquelle unter typischen Bedingungen gute Simulationsergebnisse liefern, ist das Layout beider Stromquellen für das IMS 0,5 µm Gate-Array erstellt worden. Durch zukünftige Messungen kann schlussendlich festgestellt werden, wie gut die Kompensationsmethoden funktionieren.

8. Danksagung

Der Autor, Torsten Schmitz, möchte sich bei Prof. Dr. Ing. H. Töpfer für die hochschulseitige Betreuung seiner Arbeit am IMS bedanken. Des Weiteren möchte er allen Mitarbeiten am IMS danken, die ihn während seiner Zeit am IMS tatkräftig unterstützt haben.

9. Literatur

[1] Srdjan Kordic., Titel: “Offset Reduction And Three- dimensional Field Sensing with Magnetotransistors”, 1987

[2] Wolfgang Berger, Hans Jürgen Butterweck, Titel: „ Die Berechnung von Spulen zur Erzeugung homogener Felder und konstanter feldgradienten“: Verlag: Archiv für Elektrotechnik“ 1955

[3] Diplomarbeit von Dipl. Ing. (FH) Johannes Thielmann. Titel:„ Entwicklung von Ansteuer- und Auswerteschaltungen für integrierte Hall-Platten“, 2005

[4] W. Göpel, J. Hesse, J. N. Zemel, Titel: „SENSORS A Comprehensive Survey, Magnetic Sensors“ ,Volume 5, VHC Verlag, 1989, ISBN: 3-527-26771-9

[5] Tietze Schenk, Titel: „Halbleiter- Schaltungstechnik“, 7. Auflage
Verlag: Springer, ISBN: 3-540-15134-6

[6] Dr. Christian Burwick, Ir. Cor Scherjon
Titel:“ Structured ASICs für Mixed-Signal Anwendungen“ MCP- Workshop 2009
Ausgabe: 41 ISSN 1862-7102

Aufbau- und Verbindungstechnik für ein mikrotechnisch hergestelltes Bauelement

Kögler, Daniela; Osterwinter, Heinz

Hochschule Esslingen, Robert-Bosch-Straße 1, 73630 Göppingen

Telefon: +49(0)7161-679-1227, E-Mail: Daniela.Koegler@hs-esslingen.de

Vorgestellt wird die Aufbau- und Verbindungstechnik (AVT) für ein mikrotechnisch hergestelltes Bauelement für Fokussieraufgaben. Das Bauelement besteht aus einer mit SOI-Technologie hergestellten Silizium-Membran, die als Spiegeloberfläche verwendet wird. Mit Hilfe einer Gegenelektrode kann die Membran elektrostatisch verformt und somit zur dynamischen Fokussierung eingesetzt werden.
Im Rahmen dieser Arbeit werden die einzelnen Prozessschritte zur Realisierung der Gegenelektrode sowie der Einbau aller Komponenten in das Gehäuse detailiert vorgestellt. Ein fertig gestellter Prototyp wird präsentiert.

1. Mikrooptisches Bauelement

Die hier vorgestellten Arbeiten sind Teil einer Systementwicklung, in deren Kern eine mikrotechnisch strukturierte Membran steht. Die Membran kann als aktiv-optisches Modul für eine Vielzahl von Fokussieraufgaben in der optischen Industrie eingesetzt werden und als Ersatz konventioneller, linsenbasierter Lösungen durch MOEMS-Systeme dienen. Die Membran aus kristallinem Silizium wird als Spiegeloberfläche verwendet und stellt eine auslenkbare Elektrode dar. Durch Anlegen einer Spannung zwischen Membran und einer fixierten Gegenelektrode kann die Membran elektrostatisch ausgelenkt und die Fokuslänge eines einfallenden Lichtstrahls variiert werden. Das System ist in Bild 1 schematisch dargestellt. Durch Verwendung einer ringförmigen Gegenelektrode und einer optimierten Membranaufhänung kann eine perfekte Parabolizität der verformten Membran mit einer angemessen niedrigen Betriebsspannung erzielt werden.

Das mikrooptische System - insbesondere der Membranspiegel - wurde in [1,2] ausführlich vorgestellt. Im Folgenden wird ein Konzept für die spannungsarme Verbindung der Membran auf einem Träger vorgestellt. Anforderungen sind hier u.a. der exakt einstellbare Abstand zwischen einer Gegenelektrode, die sich auf einem Träger befindet, und der Membran, sowie die thermomechanische Entkopplung der Membran.

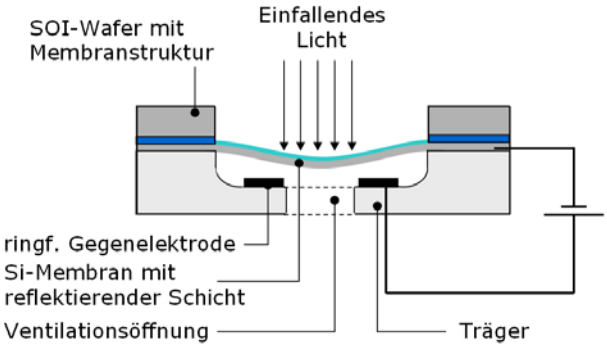


Bild 1: Schematische Darstellung des optischen Systems

Ziel des Projekts war der Aufbau eines Gesamtsystems zum Einbau in unterschiedliche Anwendungen. Zu diesem Gesamtsystem gehören:

- die mit der SOI-Technologie mikromechanisch hergestellte, reflektierende Silizium-Membran,
- die starre Gegenelektrode für die Erzeugung des elektrostatischen Feldes zur Auslenkung der Membran,
- die Elektronik zur Erzeugung der Hochspannung und Ansteuerung der Membran,
- sowie ein Gehäuse mit optischem Fenster zur Kapselung von äußeren Einflüssen und der Möglichkeit der elektrischen Kontaktierung.

In dieser Arbeit werden die einzelnen Prozessschritte zur Realisierung der Gegenelektrode und der Einbau aller Komponenten in das Gehäuse detailliert vorgestellt. Der verwendete SOI-Membran-Chip wurde von der Hochschule Furtwangen hergestellt. Ein fertig gestellter Prototyp wird präsentiert.

Die Entwicklung des mikrooptischen Systems wurde im Rahmen des Verbundprojekts MiaBoSs durchgeführt.

2. Aufbau- und Verbindungstechnik

Zur Entwicklung eines geeigneten Konzepts für die Aufbau- und Verbindungstechnik wurden im ersten Schritt die Anforderungen an das Gesamtsystem erarbeitet [3]:

Durch Finite Elemente Simulationen mit dem Programm ANSYS wurden an der Hochschule Furtwangen die optimierten Parameter für das Design der Gegenelektrode ermittelt:

Innenradius Gegenelektrode:	2,1 mm
Membrandurchmesser:	7 mm
Elektrodenabstand:	60 µm
Toleranz Elektrodenabstand:	0,5 µm

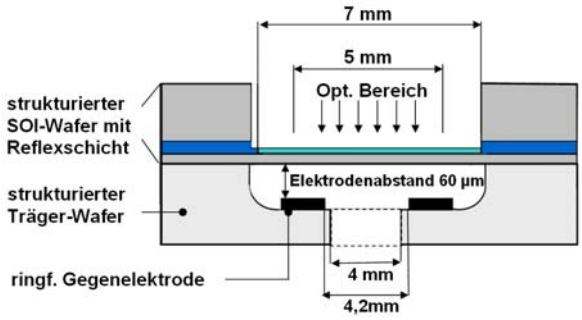


Bild 2: Schematische Veranschaulichung der Geometrien des Gesamtsystems

Die benötigte Betriebsspannung für die Membranauslenkung steht in direktem Zusammenhang mit dem Abstand der Elektroden sowie der benötigten Auslenkung der Membran. Bereits eine geringe Änderung des Elektrodenabstands erfordert einen manuellen Abgleich der einzelnen Systeme und sollte deshalb so weit wie möglich vermieden werden. Demzufolge ist eine exakte Abstandsdefinition der Gegenelektrode zur Silizium-Membran ein wichtiger Designaspekt.

Ebenso wichtig ist die thermomechanische Entkopplung der Silizium-Membran vom Gesamtsystem. Diese ist notwendig, um über einen großen Temperaturbereich die hohe erforderliche Genauigkeit der optischen Elemente gewährleisten zu können. Bei Verwendung von Materialien mit nicht angepassten Wärmeausdehnungskoeffizienten können bei einer Temperaturänderung mechanische Spannungen auf den Chip und damit auch auf die Membran einwirken. Dies hätte einen Verlust der optischen Eigenschaften zur Folge.

Weitere Aspekte sind die Langzeitstabilität der Materialen und Verbindungsstellen des Gesamtsystems sowie eine ausreichende Isolation zwischen Gegenelektrode und Membran bis Spannungen von 300 V [4,5]. Um Aufwand und Kosten zu minimieren, empfiehlt sich ein Fügeprozess auf Waferebene.

3. Konzept

Für die Aufbau- und Verbindungstechnik der Gegenelektrode mit der Membran und für das Gehäuse wurde ein Konzept erstellt, das im Weiteren erläutert wird.

3.1. Gegenelektrode

Der Prozessablauf für die Herstellung der Gegenelektrode ist in Bild 3 dargestellt. Als Trägermaterial für die Gegenelektrode wurde ein Borosilikatglas-Wafer verwendet. In diesen werden mittels nasschemischem Ätzen mit Flusssäure Kavitäten in der benötigten Tiefe (≈ 60 µm) eingebracht (a). Im nächsten Schritt wurde eine metallische Schicht für die Gegenelektrode in der Kavität abgeschieden und strukturiert (b). Ventilationslöcher zur Vermeidung eines Druckaufbaus in der Kavität beim Auslenken der Membran wurden mit einem Kurzpulslaser eingebracht (c). Der fertig bearbeitete Borosilikatglaswafer wurde mit dem vorbereiteten jedoch noch nicht geätzten SOI-Wafer anodisch gebondet (d). Der Ätzprozess zur Definition der Membranen erfolgte nach dem Fügen. Die Systeme wurden mittels Wafersägen vereinzelt.

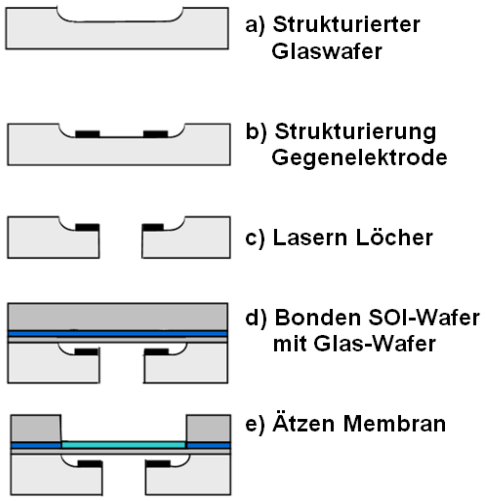


Bild 3: Prozessablauf zur Herstellung der Gegenelektrode und deren Verbindung mit der Membran

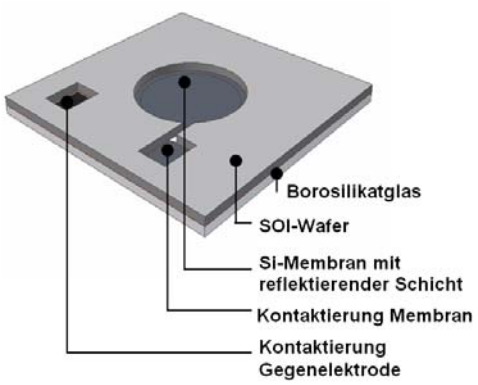


Bild 4: Schematische 3D-Darstellung des Chips

Durch die Verwendung von Borosilikatglas in Kombination mit dem Verbinden des Borosilikatglas-Wafers mit dem SOI-Wafer durch anodisches Bonden konnten die gestellten Anforderungen an die geringe Toleranzen der Elektrodenabständen und der Langzeitstabilität des Systems erfüllt werden. Da die Wärmeausdehnungskoeffizienten von Borosilikatglas und Silizium über einen weiten Temperaturbereich fast gleich sind, werden thermomechanische Spannungen vermieden. Des Weiteren ist durch die Verwendung von Borosilikatglas die Isolation der Elektroden gesichert.

Im Folgenden werden die in Bild 3 vorgestellten Prozessschritte zur Herstellung der Gegenelektrode und deren Verbindung mit der Membran genauer beschrieben. Die Strukturierung des SOI-Wafers zur Herstellung der Membran wurde bereits in [1,2] vorgestellt und ist nicht Teil der Arbeit.

3.1.1 Strukturierung Glaswafer

Die Strukturierung der benötigten Kavitäten im Glaswafers kann durch nasschemisches Ätzen mit Flusssäure erfolgen. Dazu muss eine geeignete Maskierungsschicht abgeschieden werden, die über die benötigte Ätzdauer stabil gegen das Ätzmedium ist. Laut Literatur kommt eine Maskierung mit einer Cr-Au-Schicht, einer Cr-Cu-Schicht bzw. einer Poly-Si-Schicht in Frage [6,7].

Mit Hilfe von Versuchen wurde die Machbarkeit ermittelt. Hierzu wurden mehrere Substrate mit den unterschiedlichen Maskierungsschichten hergestellt. Die Maskierung mit der Cr-Au-Schicht und der Cr-Cu-Schicht sowie die Strukturierung mittels Lift-Off konnten an der Hochschule Esslingen hergestellt werden. Die Beschichtung mit Poly-Silizium wurde extern durchgeführt.

Die Messergebnisse zeigten, dass nur die Beschichtung mit Poly-Si zum gewünschten Ergebnis führte. Bei einer Maskierung mit Cr-Au sowie mit Cr-Cu ist die Oberfläche nach dem Ätzprozess zu rau, ein anodisches Bonden war hier nicht mehr möglich. Aufgrund des hohen Prozessaufwands und der hohen Kosten durch die externe Beschichtung mit Poly-Si wurde beschlossen, die Borosikikatglaswafer mit bereits vorstrukturierten Kavitäten extern zu beziehen.

3.1.2 Abscheidung und Strukturierung der metallischen Schicht für die Gegenelektrode

Für die Abscheidung der Gegenelektrode in der Kavität muss eine NiCr-Au-Schicht verwendet werden. Die NiCr-Au-Schicht wurde aus zwei Gründen verwendet: zum einen ist sie unempfindlich gegenüber eines BHF-Bads, dem die Schicht im weiteren Prozessablauf standhalten muss; zum zweiten kann auf der NiCr-Au-Schicht eine Verbindung zur Elektronik mittels Drahtbonden realisiert werden. Für die Strukturierung der NiCr-Au-Schicht muss ein Lift-Off-Prozess angewendet werden, da Gold schwer ätzbar ist.

Um ein optimales Ergebnis für den Lithographieprozess in der 60 µm tiefen Kavität zu erzielen, wurde die Belackung mittels Sprühen, Tauchen und Aufschleudern untersucht. Das beste Ergebnis wurde mittels Aufschleudern mit dem Resist 1420 bei 500 U/min für 30 Sekunden erzielt. Anschließend wurde eine Ruhephase von 1 min vor dem Ausbacken auf der Hotplate in den Prozess integriert. Auf den strukturierten Resist wird die NiCr-Au-Schicht aufgesputtert. Die Schichtdicke beträgt für die NiCr-Schicht 50 nm und für die Au-Schicht 300 nm. Nach dem Aufsputtern kann der Fotoresist mittels Aceton in einem Ultraschallbad entfernt werden.

Die Haftung der Metallschicht der Gegenelektrode auf dem Glas sowie die prinzipielle Drahtbondbarkeit auf der Metallschicht für eine elektrische Kontaktierung konnte durch Versuche auf Proben bestätigt werden.

3.1.3 Lasern der Ventilationslöcher

Das Ventilationsloch ist notwendig, um beim Auslenken der Membran einen Druckaufbau in der Kavität zu vermeiden. Das Einbringen der Löcher in den Borosilikatglaswafer erfolgt mittels Kurzpulslasern. Der Prozess wurde bei einem Projektpartner im Applikationslabor durchgeführt. Die Laserbearbeitung muss als letzter Prozessschritt vor dem anodischen Bonden durchgeführt werden. Durch die Löcher entstehen ansonsten Schwierigkeiten bei Handling der Wafer für die weiteren Prozessschritte, da die meisten Laboreinrichtungen zum Justieren und Fixieren der Wafer mit Vakuum arbeiten.

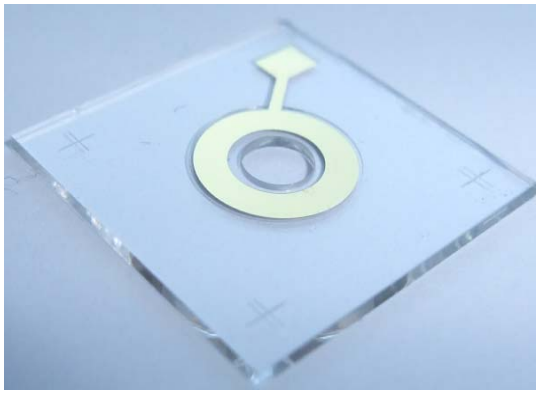


Bild 5: Foto der Gegenelektrode auf dem Borosilikatglaswafer mit den gelaserten Ventilationslöchern

3.1.4 Fügen durch anodisches Bonden

Der Bondprozess muss vor dem Ätzprozess durchgeführt werden, da die sehr hohen Spannungen beim Bonden zu einem Snap-In der Membran und somit zu ihrer Zerstörung führen würden. Der Bondprozess erfolgte bei 430°C und 1kV unter Vakuum [8].

Das Ergebnis des anodischen Bonds des strukturierten Glaswafers mit dem SOI-Wafer ist in Bild 6 zu sehen. Bei dem Waferpackage konnte ein anodischer Bond auf einem Großteil der Fläche des Wafers erzielt werden, jedoch fand im Randbereich des Wafers kein Bond statt. Messungen ergaben einen sehr hohen Bow des Waferpackages von über 200 µm so dass eine Weiterbearbeitung praktisch unmöglich wurde. Zudem wurden hohe mechanische Spannungen in das System eingebracht, die bei der Vereinzelung im Trennprozess zu hohen Ausbeuteverlusten führen können.

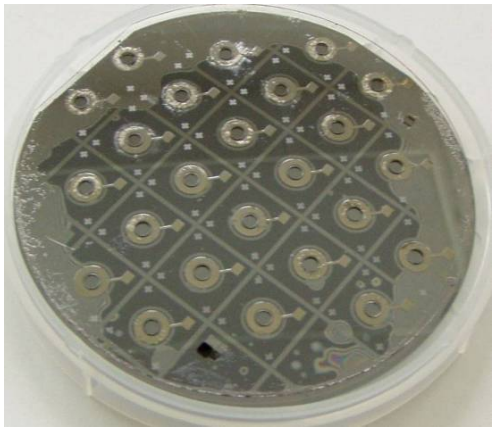


Bild 6: anodisch gebondeter SOI –und Borosilikatglaswafer nach Schritt d) in Bild 3

Um den Bow während des gesamten Prozesses zu beobachten, wurde der Glaswafer nach dem Ätzen der Kavitäten sowie nach der Beschichtung der Gegenelektrode und der Laserbearbeitung vermessen.

Hierbei wurde festgestellt, dass die strukturierten Glaswafer bereits vor dem anodischen Bonden einen unerwartet hohen Bow aufweisen. Dieser hohe Bow ist auf den Ätzprozess für die Kavitäten zurückzuführen. Weiterhin wurde eine Änderung des Bows nach jedem weiteren Prozessschritt beobachtet. Laut Hersteller haben die unbearbeiteten Borosilikatglaswafer einen Bow von max. 30 µm.

Um den hohen Bow zu vermeiden, wurde der Einfluss der Temperatur auf den Bondprozess untersucht. Es wurde eine Versuchsreihe in 50 °- Schritten von 250 °C – 400°C durchgeführt. Für die Versuche wurden im ersten Schritt unbearbeitete Glas- und Siliziumwafer verwendet. Während bei 400 °C der Bow des Waferpackage bei über 200 µm liegt, konnten bei 250 °C sehr gute Ergebnisse mit einem Warp zwischen 14µm und 25 µm erzielt werden.

Auf Basis der Ergebnisse mit unbearbeiteten Wafern mit einem Bow < 10 µm vor dem Bonden müssen im nächsten Schritt die Ergebnisse auf Wafer mit einem Bow von ca. 200 µm vor dem Bonden angepasst werden. Es muss untersucht werden, ob ein Ausgleich des hohen Bows des strukturierten Glaswafers mit optimierten Bond-Parametern möglich ist.

3.2. Gesamtkonzept gehäust

Zur Inbetriebnahme des Chips wurde dieser auf den keramischen Träger der Ansteuerelektronik integriert. Die Ansteuerelektronik wird benötigt, da das Gesamtsystem mit 5V betrieben werden soll, zur Spiegelauslenkung jedoch Spannungen zwischen 0 -240 V erforderlich sind.

Das Trägermaterial Keramik wurde für eine optimale thermomechanische Entkopplung verwendet. Um auch eine thermomechanische Entkopplung des Gesamtsystems zu erreichen, wurde als Gehäusematerial die Legierung Kovar ausgewählt. Für die elektrischen Kontaktierungen wurden Glas-Metall-Durchführungen eingesetzt. In den Deckel wurde ein optisches Interface im Winkel von zehn Grad integriert. Die Verbindung zwischen Chip und Platine, sowie zwischen Platine und Gehäuse erfolgte mittels Kleben, die elektrische Verbindung zwischen Chip und Keramik sowie zwischen der Keramik und der Elektronik erfolgte mittels Drahtbonden. Als Integrationstechnik für die Elektronik wurde die Dickschicht-Hybridtechnik gewählt.

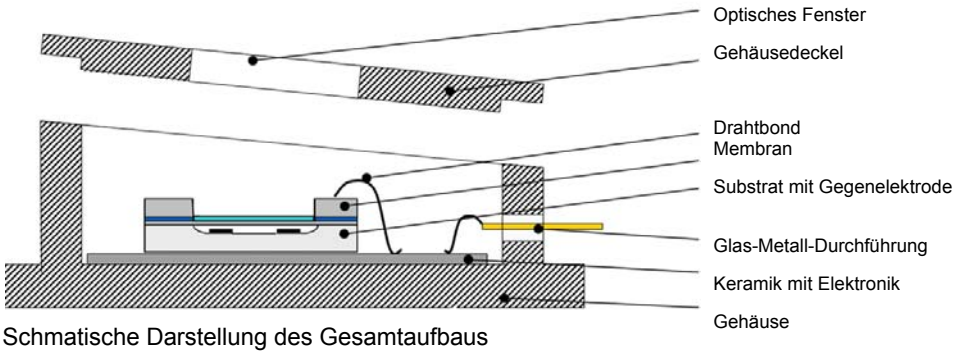


Bild 7: Schmatische Darstellung des Gesamtaufbaus

4. Aufbau eines Prototypen

Aufgrund der Problematik des hohen Ausgangsbows des Borosilikatglaswafers war zum Aufbau eines Prototyps der Umstieg auf den Fügeprozess Kleben notwendig. D.h. die Vereinzelung des Borosilikatglaswafers und des SOI-Wafers erfolgte getrennt. Anstelle des anodischen Bondens wurde ein Membranchip dann auf eine vereinzelte Gegenelektrode geklebt. Der Borosilikatglaswafer wurde durch Sägen an der Hochschule Göppingen und der SOI-Wafer wurde mit einem konventionellen Laser an der Hochschule Furtwangen vereinzelt.

Der fertige Membranchip wurde mit der Elektronik in das Gehäuse geklebt. Die Kontaktierung zwischen Chip, Elektronik und Gehäuse erfolgte mit Drahtbonden. Der Prototyp, bestehend aus Gehäuse, Keramik mit Elektronik und Chip, ist in den Bildern 8 und 9 dargestellt.

In ersten Messungen konnte die grundsätzliche Funktionalität des Prototyps nachgewiesen werden.

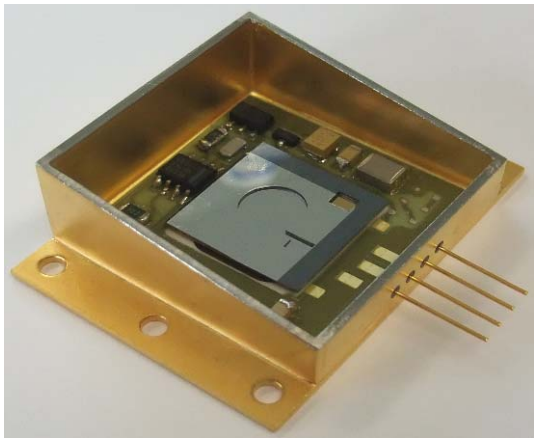


Bild 8: Gesamtsystem mit offenem Deckel

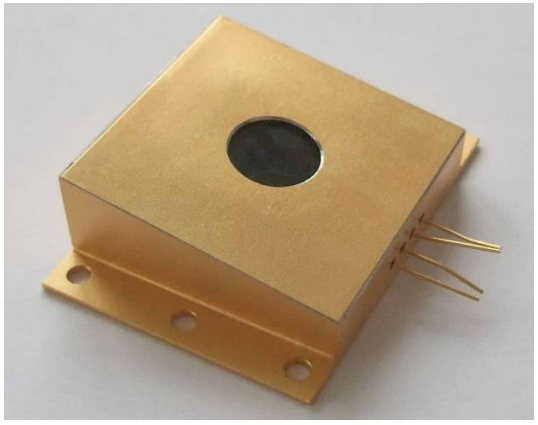


Bild 9: Gesamtsystem mit geschlossenem Deckel

5 Literatur

[1] U. Mescheder, Z. Torok, W. Kronast: *Active focusing device based on MOEMS technology*; Proceedings of SPIE Vol.6186, 618601 (April 2006), pp.1-12

[2] U. Mescheder, R. Huster, W. Kronast, T. Hellmuth, K. Khrennikov: *Optimierung und Charakterisierung eines Membranspiegels zur dynamischen Fokussierung*: VDI/VDE Mikrosystemtechnik Kongress 2007; Dresden, Oktober 2007, ISBN 9-783800-730612

[3] D. Kögler, R. Huster, W. Kronast, U. Mescheder, H. Osterwinter: *Aufbau- und Verbindungstechnik für ein mikrooptisches System*; VDI/VDE Mikrosystemtechnik Kongress 2007; Dresden, Oktober 2007, ISBN 9-783800-730612, S. 55-58

[4] L.H. Germer: *Electrical Breakdown between Close Electrodes in Air*, Jornal of Applied Physics, Vol. 30 (1959) No. 1, S. 46f

[5] J-M Torres, R S Dhariwal: *Electric field breakdown at micrometre separations*, Nanotechnology (1999) A 10: 102-107

[6] Francis E.H. Tay, Ciprian Iliescu, Ji Jing, Jianmin Miao: *Defect free wet etching through pyrex glass using Cr/Au mask*, Microsyst Technol (2006) 12:935-939

[7] Ciprian Illiescu, Jianmin Miao, Francis E.H. Tay: *Stress control in masking layers for deep wet micromachining of Pyres glass*, Sensors and Actuators (2005) A117:286-292

[8] M. Elwenspoek, H. Jansen: *Silicon Micromachining*, Cambridge University Press (1998) Kap. 4.3: S. 111-122, ISBN 0-521-59054-X

Architektur für die Anbindung eines CMOS-Kameramoduls mit Low-Level-Interface an ein FPGA

F. Schumacher, R. Sessler, S. Nothacker, T. Greiner, F. Kesel

Hochschule Pforzheim, MERSES - Zentrum für Angewandte Forschung

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

Die ständig steigende Datenrate moderner Bildsensoren stellt hohe Anforderungen an das gesamte Bildaufnahmesystem. Zur schnellen und effizienten Umsetzung bieten sich FPGAs an. Von Vorteil ist hier auch eine mögliche parallele Vorverarbeitung der Bilddaten. Schwierig gestaltet sich oft die elektronische Anbindung an das Gesamtsystem. Die Schnittstelle der Kameramodule ist proprietär, so dass für die Anbindung der FPGAs eine spezielle Lösung entworfen und realisiert werden muss.

Dieser Beitrag beschreibt Architektur und Funktionsweise einer entsprechend ausgearbeiteten Schnittstelle. Im Detail werden die Bilddaten über die realisierte Schnittstelle vom FPGA ausgelesen, in FIFOs zwischengespeichert, mittels eines Debayer-Algorithmus vorverarbeitet und im RAM gespeichert. Auf das RAM greift ein On-Board DVI-Controller mittels des Xilinx TFT-IP-Cores zu und stellt das Bild auf einem TFT-Display dar.

Im Rahmen der Arbeit wurden verschiedene FPGA-Architekturen zum Datentransfer und Debayering evaluiert und verglichen. Als Hardware kam ein Xilinx ML510 Evaluationsboard mit einem Virtex5-FPGA zum Einsatz sowie ein Kameramodul mit 5 Megapixel CMOS-Chip zum Einsatz.

1. Einleitung

Aus Kostengründen werden in digitalen Bildaufnahmegeräten wie Digitalkameras oder Camcorder meist CMOS-Bildsensoren eingesetzt. Durch die VLSI-Integration sind in diesen Bildsensoren neben der eigentlichen Bildaufnahmematrix auch weitere Funktionsblöcke realisiert, wie beispielsweise die Analog-Digitalwandlung, Belichtungssteuerung, PLLs zur internen Takterzeugung und digitale Schnittstellen. Oft können diese Bildsensoren über eine digitale Schnittstelle konfiguriert werden, beispielsweise mittels I^2C-Protokoll. Damit können die Bildsensoren auf unterschiedliche Aufnahmemodi wie Video- oder Einzelbild sowie unterschiedliche Auflösungen eingestellt werden. Auch verschiedene Kalibrierungen werden unterstützt.

CMOS-Bildsensoren erreichen heutzutage typischerweise eine Auflösung von über 10 Megapixeln. Durch diese hohe Datenrate sowie der Integration kompletter digitaler Verarbeitungsblöcke auf dem Sensor, ergeben sich hohe Anforderungen an die Ansteuerung. Für eine Auflösung von 5 Megapixeln bei einer Bildwiederholrate von 20 Bildern pro Sekunde ist beispielsweise eine Datenrate von 100 Megapixeln pro Sekunde notwendig. Bei 8-Bit Auflösung pro Pixel entspricht dies einer Datenrate von 100 MB/s. Um Daten bei solchen hohen Geschwindigkeiten verarbeiten zu können, sind spezielle FPGA-Architekturen erforderlich.

Der Beitrag gliedert sich wie folgt: In Abschnitt 2 werden die verwendeten Hardwarekomponenten näher vorgestellt. Abschnitt 3 beschreibt zunächst die einzelnen Verarbeitungsblöcke und stellt dann die Ansätze für die Architekturen vor. In Abschnitt 4 wird auf das Debayering eingegangen, eine Bildverarbeitungsoperation, um RGB-Farben aus den aufgenommenen Pixeldaten zu berechnen. In Abschnitt 5 werden erste Ergebnisse präsentiert und Abschnitt 6 fasst den Bericht zusammen und gibt einen Ausblick.

2. Hardwarekomponenten

2.1. CMOS-Kameramodul

Das CMOS-Kameramodul besteht aus dem eigentlichen Bildsensor sowie einer Adapterplatine. Der Bildsensor erlaubt eine Auflösung von bis zu 5 Megapixeln. Auf der Adapterplatine befinden sich verschiedene Bauteile zur Spannungsversorgung sowie eine Steckerleiste.

Die wichtigsten Signale für die Ansteuerung sind an der Pinschnittstelle zugänglich

- Takteingang und Taktausgang
- 12 Datenpins
- Horizontales und vertikales Synchronisationssignal
- I^2C-Schnittstelle
- Masse und Versorgungsspannung

In der Standardkonfiguration sind die horizontalen und vertikalen Synchronisationssignale notwendig, um gültige Bilddaten zu erkennen, da der Bildsensor neben Pixeldaten auch so genannte Blank-Daten ausgibt. Diese befinden sich rechts und unterhalb der relevanten Bildinformation, siehe Bild 1.

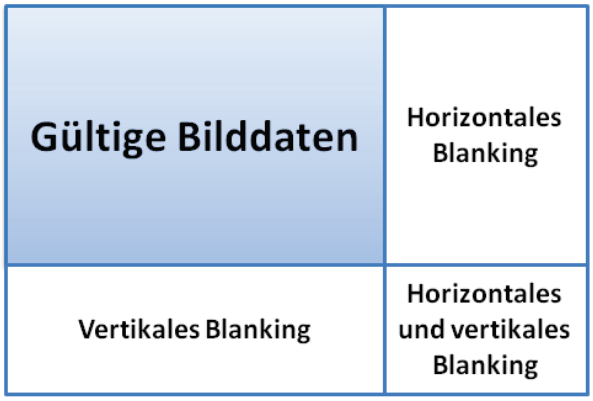


Bild 1: Gültige und ungültige Bilddatenbereiche

2.2. ML510 FPGA-Evaluationsboard

Als FPGA-Evaluationsboard wurde ein ML510 von Xilinx eingesetzt. Die wichtigsten Eigenschaften im Überblick:

- Xilinx Virtex5 FX130T FPGA
- 2 PowerPC-Kerne als Hardmakros im FPGA
- 2 x DDR2-RAM Speicherbänke
- Eine Vielzahl User I/O-Pins über externe Stecker zugänglich
- Chrontel DVI-Controller zur Ansteuerung von TFT-Displays

Zur Kommunikation von IP-Cores innerhalb Virtex5-FPGAs können Processor Local Busse PLBs eingesetzt werden. Ein PLB ist ebenfalls ein konfigurierbarer IP-Core von Xilinx. Er hat eine Bitbreite von 128 Bit und kann mit maximal 125 MHz betrieben werden. Es können mehrere Master und Slaves angeschlossen werden, die Slaveschnittstellen können 32 Bit, 64 Bit oder 128 Bit breit sein.

Weitere Informationen können der entsprechenden Xilinx-Homepage entnommen werden [1]

2.3. TFT-IP-Core und Chrontel DVI-Controller

Ein wichtiges Kriterium bei der Entwicklung von Hardwarearchitekturen ist der funktionale Test. Eine in der Hardware-Entwicklung übliche Verifizierung der Funktionalität anhand von aufgezeichneten internen digitalen Signalen, kann in diesem Fall nur sehr schwer Aufschluss darüber geben, ob ein Kamerabild korrekt vom FPGA aufgezeichnet wurde. Der einfachste Test der Verarbeitungskette erfolgt dabei durch visuelle Überprüfung eines aufgenommenen Bildes. Dazu kam der auf dem ML510 verbaute DVI-Controller der Firma Chrontel zum Einsatz. Dieser steuert die DVI-Schnittstelle des Boards an, an die ein handelsüblicher TFT-Monitor angeschlossen ist. Der DVI-Controller ist außerdem mit dem FPGA verbunden und bekommt von diesem Bilddaten. Als Grafikspeicher kann das DDR2-RAM des Boards genutzt werden. Die Ansteuerung und Konfiguration des DVI-Controllers vom FPGA erfolgt mit dem freien XPS-TFT-IP-Core von Xilinx. Während der Chrontel-Chip eine Auflösung von bis zu 1600 x 1200 Pixel bei 24 Bit pro Pixel Farbtiefe erreicht, lässt sich mit dem XPS-Core lediglich eine Auflösung von 640 x 480 Pixel erzielen. Einen leistungsfähigeren IP-Core zur Ansteuerung des Chrontel-Chips bietet Xilinx derzeit nicht an.

3. Architekturansätze

Verschiedene Architekturansätze wurden hinsichtlich Komplexität, Datendurchsatz und Chipflächenverbrauch im FPGA evaluiert. Im Wesentlichen unterscheiden sich die Ansätze in der Art der Datenspeicherung, sowie der Strategie, wie die Daten von der Bildaufnahme bis zum DVI-Controller fließen. Weitere Unterschiede stellt der Debayer-Algorithmus dar, siehe Abschnitt 4.

3.1. Prinzip der Bildverarbeitungskette

Die prinzipiellen Komponenten sind

- Bildaufnahmesensor
- Datenaufnahme- und Speicherung
- Bildverarbeitung
- Bildanzeige

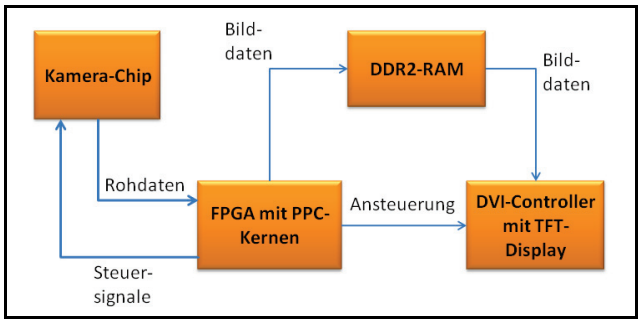


Bild 2: Prinzipschaltbild Bildverarbeitungskette

Diese Bestandteile wurden im FPGA umgesetzt. Für die Ansteuerung und Datenaufnahme des Bildsensors wurde ein IP-Core in VHDL entwickelt. Die Konfiguration des Bildsensors und des XPS TFT-Cores erfolgt mit Hilfe eines I^2C-Controllers, der als freier IP-Core von Xilinx verfügbar ist. Die Anbindung an das DDR2-RAM wurde mit dem Multi Point Memory Controller hergestellt, ebenfalls ein Xilinx IP-Core. Für die Steuerung des Datenstroms sowie die Konfiguration der einzelnen Blöcke wurde einer der beiden PowerPC-Kerne des Virtex5-FPGA genutzt. Die Verbindung aller Bestandteile des Systems wurde mit PLB Master- oder Slave-Busschnittstellen hergestellt. Je nach Architekturansatz wurden verschiedene Anbindungsstrategien und mehrere PLBs eingesetzt.

Da der Bildsensor und das restliche System mit unterschiedlichen Taktraten betrieben werden, müssen mit einem oder mehreren FIFO-Speichern Daten gepuffert und Taktdomänen entkoppelt werden.

3.2. Architektur mit einem PLB

Im ersten Architekturansatz wurde ein PLB genutzt, um alle Systemblöcke miteinander zu verbinden. Der PowerPC-Kern, sowie der XPS-TFT-Core besitzen jeweils eine Busmasterschnittstelle. Der Kamera-Core (Cam-Core), der I^2C-Controller für die Konfiguration des Bildsensors sowie der DDR2-RAM MPMC sind jeweils mit einer Busslave-Schnittstelle mit dem PLB verbunden. Weiterhin hat der TFT-Core eine PLB-Slaveschnittstelle. Während die Masterschnittstelle zum Transfer der Daten vom DDR2-RAM zum DVI-Controller benötigt wird, ist die Konfiguration per I^2C-Befehlen nur über die Slaveschnittstelle möglich. Über die Masterschnittstelle des TFT-Core kann nicht schreibend zugegriffen werden.

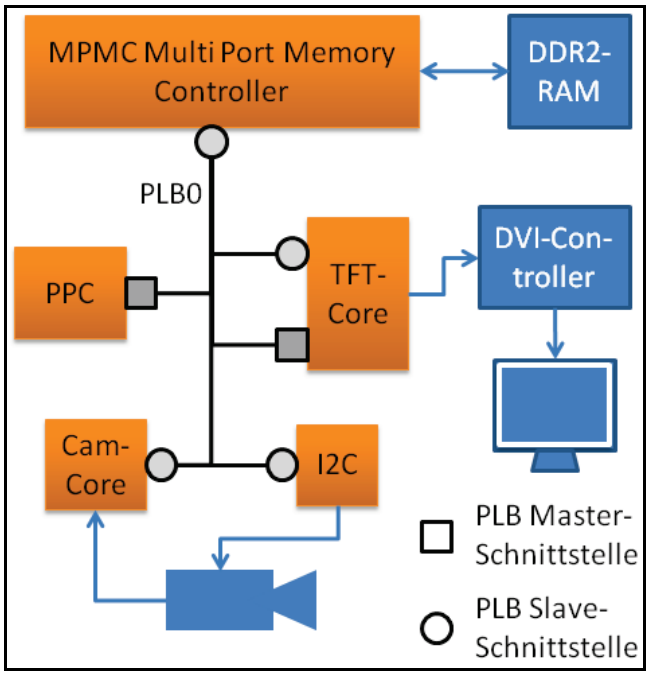


Bild 3: Architektur mit einem PLB

Nach dem Einschalten des Systems konfiguriert der PPC die Kamera und den TFT-Core über den I^2C-Controller. Der TFT-Core wiederum konfiguriert den DVI-Controller. Danach aktiviert der PPC den Cam-Core. Dieser liest Daten aus der Kamera aus und speichert sie zum Debayering in einen oder mehrere FIFO-Speicher, näheres dazu in Abschnitt 4. Der PPC arbeitet im Polling-Modus, das heißt, sobald das Not-Empty-Flag des FIFO gesetzt ist, werden die Daten in das DDR2-RAM geschrieben. Sobald ein komplettes Bild aufgenommen wurde, veranlasst der PPC den TFT-Core die Bilddaten aus dem Speicher auszulesen und an den DVI-Controller zu schicken. Dieser stellt dann die Bilddaten über die DVI-Schnittstelle auf dem TFT-Monitor dar.

In dieser Architektur werden sämtliche Daten über einen einzigen PLB transferiert. Dadurch ist die gesamte Datenrate des Systems relativ gering. Mit diesem Ansatz konnte der Kamerachip nur mit 6 MHz betrieben werden und es wurde eine Bildwiederholfrequenz von etwa 2 Bildern pro Sekunde (Frames per Second Fps) erzielt.

3.3. TFT-Core mit eigener Speicheranbindung

Um den Datendurchsatz zu steigern, wurde ein zweiter PLB in die Architektur eingebaut. Damit konnte der TFT-Core mit seiner Masterschnittstelle über einen dedizierten PLB an das DDR2-RAM angeschlossen werden Die restlichen Anschlüsse an PLB0 blieben gleich.

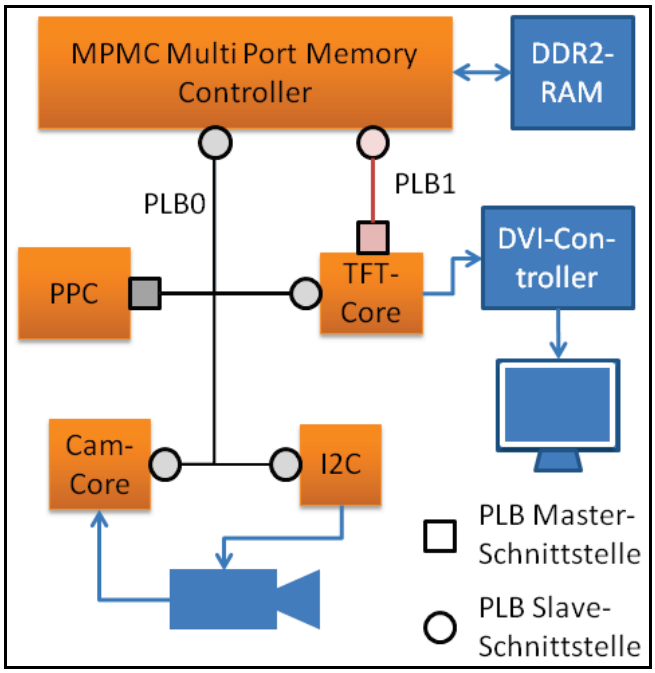


Bild 4: TFT-Core mit eigener Speicheranbindung

Mit dieser Maßnahme konnte die Datenrate des Systems deutlich gesteigert werden und das Kameramodul konnte mit einer Geschwindigkeit von 12 MHz angesteuert werden, resultierend in einer Verdopplung der Bildwiederholfrequenz auf etwa 4 Fps.

4. Debayering

Aus Kostengründen besitzen die meisten CMOS-Bildsensoren pro Ortskoordinate keine getrennten Farbpixel für die Primärfarben RGB. Stattdessen wird meist eine Mosaikfilterschicht verwendet, die das Bild in Subpixel unterteilt, siehe [2]. Üblicherweise lehnt man sich an die Physiologie des menschlichen Auges an, indem die Farbe Grün doppelt so häufig codiert wird als die Farben Rot und Blau.

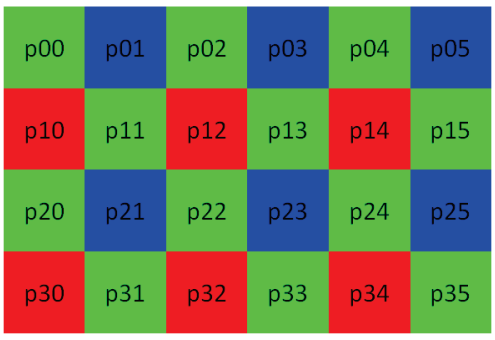


Bild 5: Bayer-Mosaikmuster

Jeder Pixelwert ist dabei mit jeweils 12 Bit codiert. Um aus dieser Darstellung die drei RGB-Werte eines Zielpixels zu ermitteln, gibt es verschiedene Verfahren, die sich in ihrer Komplexität und Qualität unterscheiden.

RGB 00	RGB 01	RGB 02
RGB 10	RGB 11	RGB 12
RGB 20	RGB 21	RGB 22

Bild 6: Typische Aufteilung eines RGB-Bilds

Ein einfaches Verfahren verwirft „überschüssige“ Bayer-Pixel und interpretiert die übrigen Pixel direkt als RGB-Pixel. Dieser Ansatz verringert die Auflösung des Zielbilds und verfälscht die Farben, kann aber sehr schnell durchgeführt werden. Aufwändigere Verfahren berechnen ein RGB-Pixel durch Interpolation von benachbarten Bayer-Pixeln und behalten die Auflösung bei. Weiterhin gibt es verschiedene geeignete Verfahren für Einzelbilder oder Videos. Je nach Vorgehen, ist ein bestimmter FPGA-Architekturansatz besser geeignet. Es bietet sich beispielsweise die Gruppierung der jeweiligen Farben bereits beim Auslesen an, so dass zusammengehörige Pixel gemeinsam gespeichert werden.

In jedem Fall muss das Problem umgangen werden, dass beim Auslesen einer Bildzeile nur zwei Farbinformationen zur Verfügung stehen, Grün und entweder Rot oder Blau. Für die Berechnung eines RGB-Pixel bei aufwändigeren Debayer-Algorithmen werden jedoch alle Farbinformationen und somit alle umliegenden Mosaikpixel benötigt. Dafür müssen im FPGA mindestens zwei oder mehr Bildzeilen zwischengespeichert werden. Die gängigsten Methoden hierfür sind Schieberegisterpipelines, FIFO-Speicher, die sich im Block-RAM des FPGA befinden oder eine Kombination aus beidem.

Für das Debayering wurden zwei verschiedene Ansätze evaluiert. Beide Ansätze unterscheiden sich in der Art und Weise, wie die einzelnen RGB-Farbpixel aus den Bayer-Mosaikpixeln berechnet werden. Im zweiten Ansatz wird eine Schieberegisterpipeline zur Zwischenspeicherung von Bildzeilen genutzt. Dort werden die aus dem Bildsensor ausgelesenen Bayer-Pixel eingetaktet.

Die Länge der Pipeline hängt davon ab, mit wie vielen Bildzeilen der Debayer-Algorithmus arbeitet. Wenn N die Anzahl der Pixel einer Zeile sind und Z die benötigten Zeilen für das Debayering, dann ist die Pixellänge L der Pipeline

$$L = (Z-1)\cdot N + X$$

wobei X ist die Breite des Bayer-Operators ist. Bei einem 2x2 Operator ist X = 2, bei einem 3x3 Operator ist X = 3.

Bild 7: Zeilenpipeline der Länge L = 2*N+2

Die RGB-Pixeldaten werden nach dem Debayering in drei FIFO-Speicher, jeweils für R, G und B abgelegt. Von dort werden sie in den DDR2-Grafikspeicher transferiert.

4.1. Einfache Debayer-Methode

Die einfachste Methode zum Debayering ist die direkte Interpretation der Bayer-Pixel als RGB-Pixel. Dazu wird ein 2x2 Operator benötigt, der über das Bild geschoben wird. Bezogen auf Bild 5 und Bild 6 werden die einzelnen RGB-Pixel wie folgt berechnet:

R11 = p12

B11 = p21

G11 = p11 oder (p11+p22)/2

…

R12 = p12

G12 = p22 oder (p13+p22)/2

B12 = p23

Diese Vorgehensweise nutzt Bayer-Pixel mehrfach und hat zur Folge, dass im Ergebnis immer jeweils vier benachbarte RGB-Pixel denselben Farbwert haben. Vorteilhaft hierbei ist, dass ein kleiner FIFO-Speicher ausreicht und relativ wenige Zugriffe auf diesen erforderlich sind, was das System insgesamt beschleunigt. Durch die nicht benötigte Pixelpipeline wird weiterhin Chipfläche im FPGA eingespart.

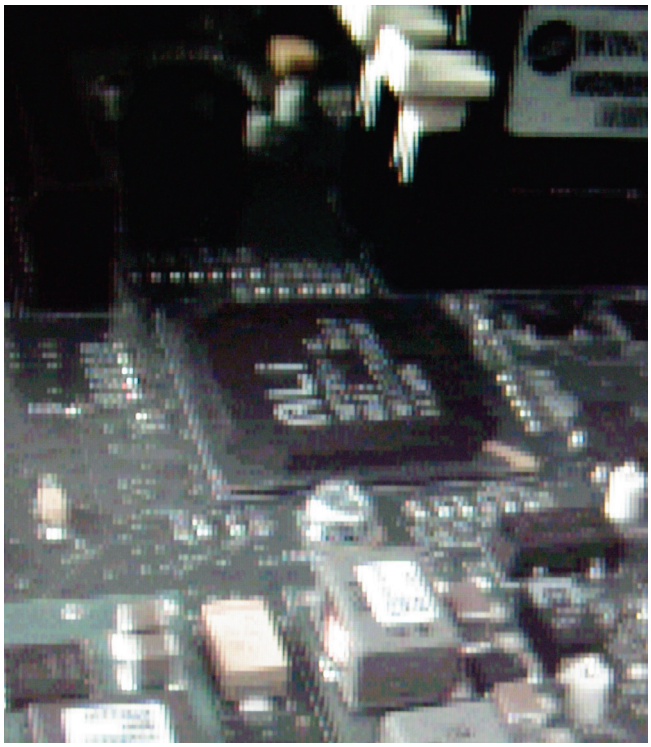


Bild 8: Testbild mit erstem Debayering-Ansatz

In diesem Bild sind insbesonders an den Objektkanten die Debayer-Artefakte gut zu erkennen. Das Bild wirkt insgesamt auch sehr pixelig.

4.2. Debayering mit bilinearer Interpolation

Bessere Bildqualität wird mit einem Debayer-Algorithmus erzielt, der mit bilinearer Interpolation arbeitet. Dieser benötigt eine längere Pixelpipeline und ist aufwändiger zu berechnen. Der Ansatz wird in [3] vorgestellt. Hier werden Informationen aus drei Bildzeilen benötigt und somit beträgt die Länge der Pixelpipeline 2*N+3. Für N = 480 beträgt die Länge 963 Pixel.

Die einzelnen Komponenten der RGB-Werte werden wie folgt berechnet: Die RGB-Farbkomponente, die dem aktuellen Bayer-Pixel entspricht, wird direkt übernommen, im Fall von beispielsweise p12 ist dies der Rot-Wert. Die Berechnung der beiden fehlenden Farben bezieht alle benachbarten Bayer-Pixel mit ein. Für RGB12 beispielsweise werden der Grün- und Blauwert nach folgender Vorschrift berechnet:

G12 = (p11 + p22 + p13 + p02) / 4

B12 = (p01 + p03 + p21 + p23) / 4

R12 entspricht p12, da p12 ein rotes Bayer-Pixel darstellt. In diesem Fall werden alle 8 benachbarten Bayer-Pixel verwendet.

Falls dem RGB-Pixel ein grünes Bayer-Pixel entspricht, werden nur 4 benachbarte Werte benötigt, beispielsweise für RGB13:

R13 = (p12 + p14) / 2

B13 = (p03 + p23) / 2

G13 = p13

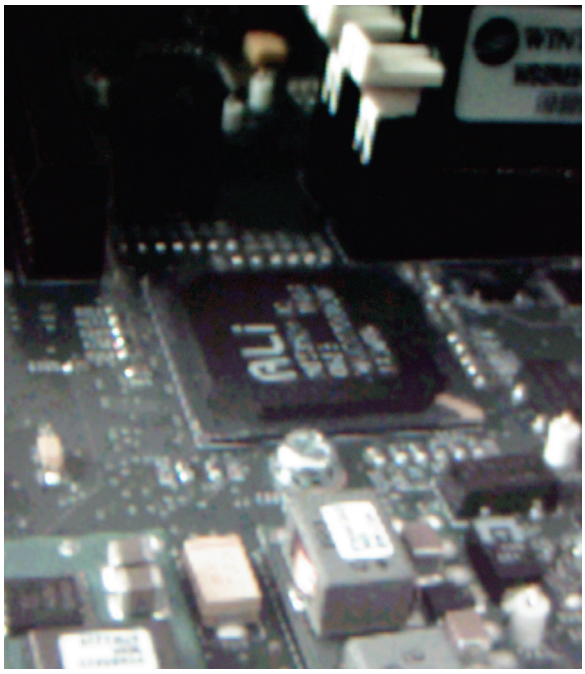


Bild 9: Debayering mit bilinearer Interpolation

Im Vergleich zu Bild 8 ist hier bei gleich gebliebener Auflösung eine deutliche Qualitätssteigerung zu beobachten. Die leichte Unschärfe kommt unter anderem daher, dass das 640 x 480 große Bild vom TFT-Monitor auf 1280 x 1024 hochskaliert wird.

5. Ergebnisse

Um die einzelnen Architekturen vergleichen zu können, wurden verschiedene Kennzahlen ermittelt.

Es zeigte sich, dass mit der ersten Architektur eine Ansteuerungsgeschwindigkeit des Kameramoduls von 6 MHz erzielt werden konnte. Als Debayering-Methode kam nur der einfache Ansatz in Frage.

Tab. 1: Vergleich der unterschiedlichen Ansätze

	Architektur 1	**Architektur 2**
Kamera-ansteuerung	6 MHz	12 MHz
Fps	2	4
Debayering	einfach	Bilineare Interpolation (nur 6 MHz)

Mit dem dedizierten PLB für den TFT-Core in der zweiten Architektur konnte die Ansteuergeschwindigkeit des Bildsensors auf 12 MHz verdoppelt werden, allerdings unter Beibehaltung der einfachen Debayer-Methode. Mit der bilinearen Interpolation verringerte sich die Ansteuerung wieder auf 6 MHz. Ursache hierfür sind die größeren FIFO-Speicher, die benötigt werden, um im Gegensatz zum einfachen Ansatz jeden RGB-Wert zu berechnen.

Weiterhin können die dargestellten Bilder qualitativ-visuell bewertet werden. Vergleicht man Bild 8 und Bild 9, ist ein deutlicher Qualitätsunterschied der beiden Debayer-Ansätze zu erkennen.

6. Zusammenfassung und Ausblick

Es wurde ein hochauflösender CMOS-Bildsensor an einen FPGA angeschlossen. Die aufgenommenen Bilder wurden zur Überprüfung der Ansteuerung auf einem TFT-Monitor ausgegeben. Es wurde mit verschiedenen Architekturansätzen eine Auflösung von 640 x 480 Pixel bei einer maximalen Bildwiederholfrequenz von etwa 4 Fps erreicht. Als Teil der Verarbeitungskette wurden verschiedene Debayer-Algorithmen implementiert.

Die Anzeige auf dem TFT-Display war durch den Xilinx XPS-TFT-Core auf eine Auflösung von 640 x 480 Pixel limitiert. Der Chrontel DVI-Controller würde bei geeigneter Ansteuerung bis zu 1600 x 1200 Pixel schaffen, allerdings sind der XPS-TFT-Core sowie dieser Chip und dessen Ansteuerung sehr schlecht dokumentiert. Dies verzögert eine Eigenentwicklung, wir planen jedoch den Einsatz eines TFT-IP-Cores, der 1024 x 768 Pixel erlauben wird. Dies wiederum erhöht die Anforderungen an den Datendurchsatz des Systems und es bleibt abzuwarten, ob die vorgestellten Architekturansätze diesen Durchsatz leisten können.

Weiterhin ist ein verbessertes Layout der Schnittstellenplatine in Vorbereitung, um die Signalqualität zu verbessern und die Taktrate zur Ansteuerung des Kameramoduls zu erhöhen.

6.1. Cam-Core mit eigener Speicheranbindung

Aktuell in Arbeit ist die Anbindung des Cam-Core mit dediziertem PLB an den DDR2-Speicher. Dies ermöglicht das direkte Schreiben der RGB-Daten in das RAM. Somit müssen die Daten nicht durch den PPC fließen, was eine weitere Erhöhung des Datendurchsatzes ermöglicht. Der PPC fungiert in diesem Aufbau nur zur Steuerung und Konfiguration der einzelnen Komponenten, vom Datenfluss ist er abgekoppelt.

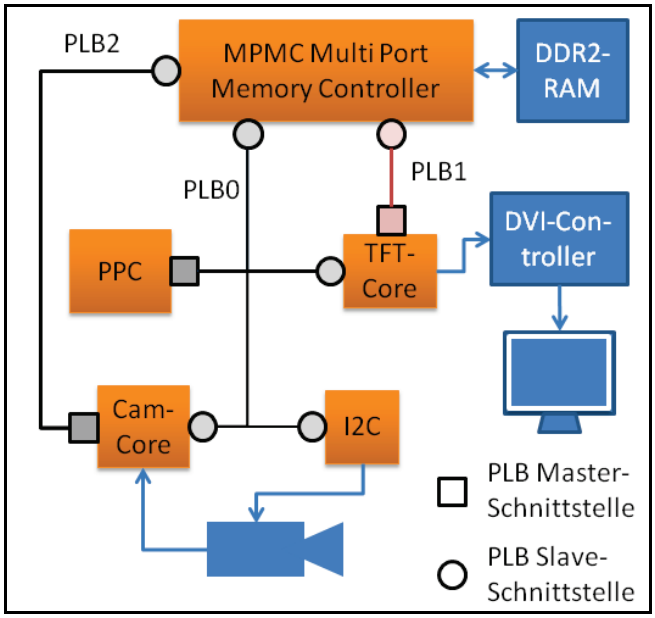


Bild 10: Cam-Core mit eigener Speicheranbindung

Damit hätte der Cam-Core direkten Speicherzugriff (Direct Memory Access DMA).

Eine weitere Durchsatzerhöhung kann durch Optimierung des MPMC und des Zugriffs auf das DDR2-RAM erzielt werden.

7. Quellen

[1] Xilinx Virtex-5 FXT ML510 Embedded Development Platform http://www.xilinx.com/products/devkits/HW-V5-ML510-G.htm

[2] Bahadir K. Gunturk, John Glotzbach, Yucel Altunbasak, Ronald W. Schafer and Russel M. Mersereau, "Demosaicking: Color Filter Array Interpolation," Exploring the imaging process and the correlations among three color planes in single-chip digital cameras in: IEEE Signal Processing Magazine Volume 22, Issue 1 2005, S. 44–54.

[3] Iván Olaf Hernández Fuentes, Miguel Enrique Bravo-Zanoguera and Yáñez Galaviz Guillermo, "FPGA Implementation of the Bilinear Interpolation Algorithm for Image Demosaicking," in: Proceedings of International Conference on Electrical, Communications, and Computers CONIELECOMP 2009, S. 25–28.

Herausforderungen bei der Integration eines Open Source CAN-Controllers in ein SOPC

Christian Seibt, Gregor Burmberger

HTWG Konstanz, Brauneggerstraße 55, 78462 Konstanz

chseibt@htwg-konstanz.de, gregor.burmberger@htwg-konstanz.de

Im Rahmen einer Bachelorarbeit an der Hochschule Konstanz, Technik, Wirtschaft und Gestaltung, soll ein Open Source CAN-Controller in ein FPGA-basiertes SOPC integriert werden. Ziel ist es, das SOPC als Teilnehmer in einem größeren CAN Bussystem-Verbund einzusetzen.

1. Einleitung

Ein System-On-a-Programmable-Chip (SOPC) ist vergleichbar mit einem anwendungsspezifischen Mikrocontrollersystem. Der Vorteil der FPGA-Implementierung ist, dass das SOPC flexibel an die Anforderungen angepasst werden kann. Es besteht meist aus einem Prozessorkern (Softcore), welcher mit beliebigen, wieder benutzbaren Komponenten (IP Cores) wie z.B. RAM, IOs, JTAG-Interface oder anderen Peripheriemodulen erweiterbar ist.

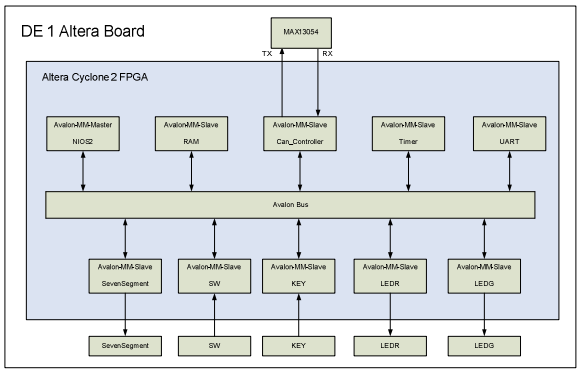


Abbildung 1: Aufbau eines SOPC Systems mit CAN Controller und IOs

Text Der FPGA-Hersteller *Altera* bietet drei lizenzfreie Software-Pakete für die Realisierung eines SOPC an. *Quartus II 9.0 Webedition* ist ein Softwareprogramm zur Erstellung eines VHDL- Designs. Das Tool *SOPC Builder* in der *Quartus II 9.0 Web Edition*, ermöglicht es, ein SOPC mit einem lizenzfreien *NIOS II* Prozessor und verschiedenen IP Cores zu erstellen. Die *NIOS II 9.0 IDE* Entwicklungsumgebung ermöglicht die Erstellung der zugehörigen Benutzersoftware eines SOPC. Mit *ModelSim-Altera 6.4a* können VHDL-Designs sowie ein ganzes SOPC inklusive Anwendersoftware simuliert werden. Es ist auch eine Simulation eines CAN-Bussystem mit zwei gleichen möglich. Eine Möglichkeit für eine FPGA Entwicklungsumgebung bietet *Altera* mit dem *DE1 Altera Board*. Das Board beinhaltet einen 90 nm *Cyclone II* FPGA mit 20000 Logic Elements, Externer Speicher, RS232, SD-Card Socket, PS/2 Port und andere IOs. Der *NIOS II* Softcore ist in drei Konfigurationen einsetzbar: *fast*, *standard* und *economy*. In einem *Cyclone II* FPGA ergibt sich folgende Performance:

Tabelle 1: Vergleich Nios Konfiguration[1]

Typ	NIOS II/f	NIOS II/s	NIOS II/e
Architektur	32 Bit	32 Bit	32 Bit
Data Cache	512 - 64 kb	-	-
Instruction Cache	512 - 64 kb	512 - 64 kb	-
Debug Module	JTAG	JTAG	JTAG
MIPS/MHz	1.105	0.518	0.107
LE	ca. 3500	ca. 2000	<1000

2. Herausforderungen

Ein Nachteil von SOPCs ist, dass komplexere IP Cores wie z.B. CAN-Controller u.ä. meist lizenzpflichtig sind. *Opencores* [2] ist eine Internet Community auf der verschiedene IP Cores auf Open Source Basis verfügbar sind. Die auf *Opencores* verfügbaren IPs sind überwiegend für das offene *Wishbone*-Bussystem konzipiert, da auch die meisten Open Source Prozessoren diesen unterstützen. *Altera* benutzt ihr proprietäres *Avalon* Bussystem, welches jedoch zu *Wishbone* ähnlich ist und sich an dieses mit wenig Zusatzlogik anpassen lässt. Auf *Opencores* existiert ein CAN Controller, der zu dem *NXP SJA1000* CAN-Controller kompatibel ist. Bei Open Source Quellen ist die kor-

rekte Funktionsweise fraglich, da verschiedene Nutzer eigenständig daran arbeiten. Bei dem hier vorliegenden Open Source CAN-Controller existieren zwei verschiedene Versionen, eine veraltete Verilog Version mit Testbench und eine aktuelle VHDL-Version ohne Testbench, die im Laufe der Entstehungsphase von der Verilog Version übersetzt wurde. In der Testbench wird ein Busnetzwerk mit zwei Teilnehmern Simuliert. Bei der Anordnung von zwei gleichen CAN-Controller-IP Cores können Übertragungsfehler, welche auf einer fehlerhaften Implementierung auf Sende- und Empfangsseite beruhen nicht erkannt werden.

3. Realisierung

Um ein kostengünstiges SOPC mit einem CAN-Controller zu erstellen, wurde bei der Integration der CAN-Controller von *Opencores* verwendet. Zur Funktionsprüfung des Open Source *SJA1000* mit einem verifizierten CAN-Controller werden kostengünstige Mikrocontroller mit CAN-Funktionalität eingesetzt. Ein SOPC wird auf ein *Altera DE1* Entwicklungsboard geladen und an ein Mikrocontrollerboard mit CAN-Funktion angeschlossen.

4. Übersetzungsfehler

Für die Simulation wurde die aktuelle VHDL-Version mit der Verilog Testbench verwendet. Eine Simulation von Mixed Signals wird in der lizenzfreien Modelsim Version von *Altera* leider nicht unterstützt. Es wird hierzu ein Mixed Signal Simulator benötigt der z.B. in der Vollversion von ModelSim integriert ist. Bei der Simulation des CAN-Controllers ergaben sich Übertragungs- und Funktionsfehler, die sich u.a. auf Übersetzungsfehler des Tools *X-HDL Translator 3.2.53* von Verilog zu VHDL zurückführen ließen.

Verilog besitzt eine Fork/Join-Struktur in der Testbench. VHDL und damit auch der VHDL Translator unterstützen diese Funktion in derselben Struktur nicht. Fork/Join ist eine Funktion, die in einer Testbench die Befehle/Prozesse parallel ausführt und erst bei einer vollständigen Abarbeitung aller Prozesse das Programm fortsetzt. In VHDL ist die gleiche Funktion mit einer andern Strukturform möglich.

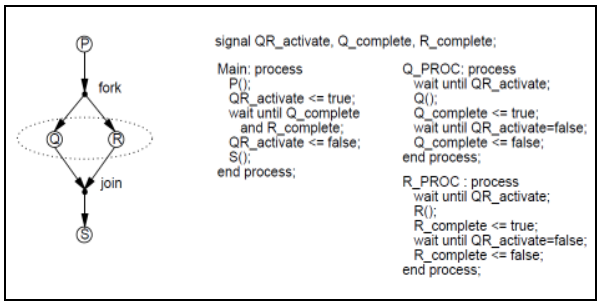


Abbildung 2: Fork/Join Lösungsstruktur [4]

Weitere Fehler:

- Übersetzung von „don't care" Bits in if-Funktionen: Bei if-Funktionen werden in VHDL für don't care Signale ein *std_match()* benutzt [3]. In der Übersetzten VHDL-Version fehlte dieser Befehl. Bei einer Übersetzung mit einer neueren Version des *X-HDL 3.2.54* wurde dieser Fehler nicht mehr festgestellt.
- Vektoraddition: Bei einer Vektoraddition in Verilog hängt die Resultierende Bitlänge einer Bitaddition von der Länge des Ergebnisses ab. Wenn ein 15 Bit Vektor mit einem 15 Bit Vektor addiert wird und die Bitlänge des Ergebnissignals 16 Bit lang ist, wird das Overflow Bit bei der Addition berücksichtigt. Bei einem 15 Bitvektor als Ergebnissignal nicht [5]. In VHDL wird die Länge eines Additionsergebnisses von der maximalen Länge der Summanten bestimmt. Bei einer Situation wie im obigen Beispiel wird das Overflow Bit in einer Addition nicht berücksichtigt, da die Länge der Summanten nur 15 Bit beträgt [3]. Das Übersetzungstool berücksichtigt diesen Fehler nicht. Bei einer Verilog-VHDL Übersetzung entstehen somit im VHDL Code immer dann Fehler, wenn Bitvektoren bei einer Addition nicht eindeutig definiert sind.

5. Busanpassung

Um den *SJA1000* CAN-IP Core für ein *Altera* SOPC System benutzen zu können, musste die Busschnittstelle angepasst werden. Das *Avalon*- und das *Wishbone*-Bussystem sind sehr ähnlich. Beide unterstützten Multi Master und Multi Slave Systeme. Die maximale Busbreite im *Wishbone*-Bus beträgt 64 Bit. Der *Alavon*-Bus unterstützt nur eine Busbreite von 32 Bit. Für die Schnittstelle müssen folgende Signalverknüpfungen erstellt werden:

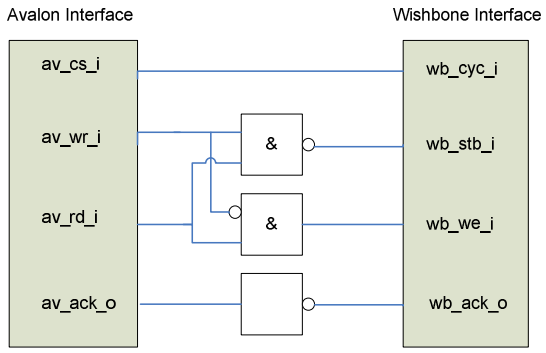


Abbildung 3: Avalon-Wishbone Verknüpfung

6. Treibersoftware

Für die Verwendung des CAN-Controllers in einer Anwendersoftware wird ein Treiber benötigt. Der Treiber ist eine Schnittstelle zwischen CAN-Controller und Anwendersoftware. Er stellt Schreib- und Lesemethoden, sowie eine Interrupt Bearbeitung für den CAN-Controller zur Verfügung. Um in einem SOPC Daten in ein IP Core Register zu schreiben, muss der Arbeitsspeicher umgangen werden, um zu verhindern, dass die Daten verloren oder verzögert werden. Mit dem Deklarieren eines C-Pointers volatile wird dies nicht erreicht. Die Hardware Abstraction Layer (HAL) beinhaltet C–Makros um dies zu umgehen, so genannte IORD und IOWR Befehle [6].

7. Simulation

Für eine korrekte Funktionsweise des Open Source CAN-Controllers wird das komplette System inklusive Anwendersoftware simuliert.

Bei der Simulation eines SOPC Systems interagieren alle Entwicklungsumgebungen (*Quartus*, *NIOS II* und *Modelsim*) miteinander. Die generierten Simulationsdateien werden von dem *SOPC Builder* automatisch generiert. Die Wichtigsten Simulationsdateien werden hier kurz zusammengefasst:

1. Testbench in der SOPC *.vhd* Datei: In der Hauptdatei des SOPC wird eine Testbench generiert. Die Testbench initialisiert alle Eingangssignale zu Null, generiert einen Voreingestellten clock und resetet das System.

2. Modelsim Projektdatei *.mpf*: Mit dieser Projektdatei kann in Modelsim das SOPC Projekt geladen werden.

3. *setup_sim.do*: Dieses do-File wird im geladenen SOPC Projekt ausgeführt. Bei der Ausführung werden alle Simulationsdateien für die Simulation des SOPC kompiliert und geladen.

4. *RAM.hex*: Die Standard Initialisierungsdatei eines Ram Blocks ist *RAM.hex*. Die erstellte *RAM.hex* Datei beinhaltet die Initialisierung des *NIOS II* Prozessors.

Um das SOPC mit einem Anwenderprogramm zu Simulieren wird die *NIOS II 9.0* Software benötigt. *NIOS II 9.0* überschreibt beim kompilieren das von dem *SOPC Builder* erstellte *RAM.hex* Datei mit einem Anwendersoftware *RAM.hex* Datei. Die Zeitspanne von 100 µs Simulationszeit reicht aus, um ein Booten des NIOS II Prozessors und eine Initialisierung des CAN-Controllers zu simulieren. Die Zeit zur Ausführung der Simulation beträgt wenige Sekunden.

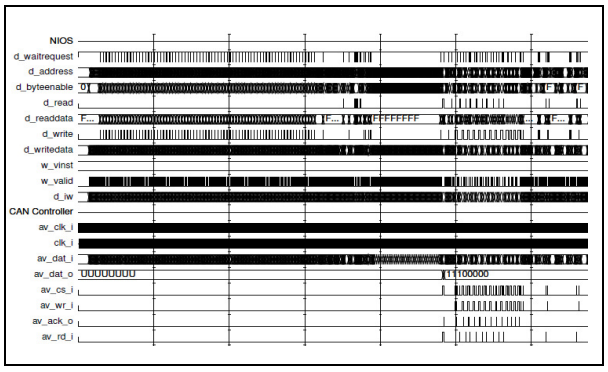


Abbildung 4: 100 µs Simulation des SOPC mit CAN-Controller und Anwendersoftware

8. Funktionstest

Für einen Hardwaretest wird eine Adapterplatine mit einem *MAX13054* an das DE1 Altera Board angeschlossen. Der *MAX13054* ist ein High-Speed CAN Tranceiver und wandelt den 3,3V IO Pegel des FPGAs in einen CAN-Pegel. Auf der Adapterplatine wurde ein Jumper hinzugefügt, mit diesem der Abschlusswiderstand in einem CAN Bussystem hinzugefügt werden kann.

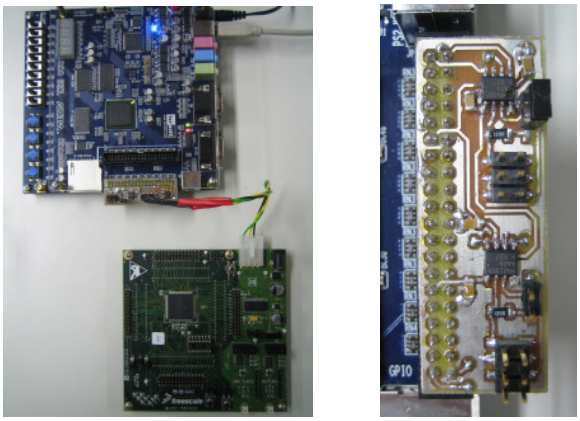


Abbildung 5: links, Altera DE1 Board mit CAN Bus Adapter und Anschluss an ein EVB9S12XF512E Board; rechts, CAN Bus Adapterplatine

9. Fehlergenerator

Zur Überprüfung einer korrekten Fehlerbehandlung des Open Source CAN-Controllers nach ISO16845, wurde ein Fehlerprotokollgenerator auf VHDL Basis erstellt. Der Fehlergenerator ist ein einem Frame Mode und einem Bit Mode einstellbar. Im Frame Mode können unterschiedliche CAN-Nachrichten auf dem Bus gesendet werden. Dies ermöglicht die Generierung von vordefinierten Error-Nachrichten in Form von Stuff Error, CRC Error oder Form Error Fehlern. Im Bit Mode kann ein beliebiges Bit in einer CAN-Nachricht mit einem dominanten Pegel verfälscht werden.

10. Fazit

Die Implementierung eines Open Source CAN-Controllers als wieder verwendbaren IP Core ist eine kosten-günstige Variante für die eigene Verwendung. Nachteile sind Programmierfehler, welche durch nicht implementierte Testfunktionen entstehen. Die Integration und Simulation eines IP Core in ein SOPC ist mit lizenzfreien Softwareprogrammen von *Altera* vereinfacht. Jedoch ist die das Zusammenspiel und Funktion der einzelnen Softwaretool sehr komplex und garantiert auf Anhieb keine gewünschte Funktionsweise. Die Überprüfung einer korrekten Funktionsfähigkeit beansprucht viel Zeit. Um den Vorgang zu beschleunigen hilft Erfahrung in den Entwicklungssoftware von Altera und eine gute Erfahrung im Bereich VHDL.

11. Referenzen

[1] Altera, NIOS 2 Preformance Benchmarks, Version 4.0, 2009

[2] Open Source Webseite: www.opencores.org

[3] IEEE Standard: VHDL Reference Manual, 1997

[4] Daniel D. Gajski, Frank Valhid, Sanjiv Narayan, “A VHDL Front-End for Embedded Systems”, Technical Report, Dept. Of Information and Computer Science, University of California, 1993

[5] IEEE Standard Hardware Description Language Based on the Verilog Hardware Description Language, 1996

[6] Altera, NIOS 2 Software Developer’s Handbook Version 9.1, 2009

Untersuchung der Einsatzmöglichkeiten der JTAG-Schnittstelle für programmierbare Logikbausteine

Denis Schneider, B. Eng.

HTW Aalen, Beethovenstraße 1, 73430 Aalen

schneiderdenis@gmx.de

Die vorliegende Arbeit befasst sich mit der Untersuchung der Einsatzmöglichkeiten der JTAG-Schnittstelle in Endprodukten aus dem Automotive Bereich. Diese sind eingebettete Systeme, welche für Multimedia Anwendungen entwickelt werden. Dafür werden immer häufiger programmierbare Logikbausteine, wie FPGA und CPLD verwendet. Diese Bausteine übernehmen verschiedene Aufgaben in Produkten, wie z.B. die Ansteuerung diverser Bildschirme.

Das Ziel der Arbeit war es, die JTAG-Schnittstelle für die Konfiguration von FPGA- und CPLD-Bausteinen einzusetzen. Dadurch ließen sich Produktionskosten sparen, denn momentan werden CPLD-Bausteine für die Serienfertigung vor dem Einbau ins Endprodukt konfiguriert. Mit der Konfiguration über die JTAG-Schnittstelle könnten diese Bausteine im System konfiguriert werden, was einen Produktionsschritt einsparen würde. Außerdem würde somit eine Möglichkeit zur Rekonfiguration der Bausteine im Feld realisiert werden.

Die Konfiguration der programmierbaren Logik ist aber nicht die einzige Stärke der JTAG-Schnittstelle. Ursprünglich wurde diese Schnittstelle zum Testen von Bausteinen oder ganzen Systemen entwickelt. Durch die Ansteuerung der Schnittstelle in einem eingebetteten System wäre es somit denkbar, nicht nur die Konfiguration von FPGA- oder CPLD-Bausteinen im System zu ermöglichen, sondern auch das Testen und Debuggen dieser Bausteine zu erleichtern.

Um diese Aufgaben zu realisieren, wurden von mir im Laufe Arbeit zwei unterschiedliche Konzepte für die JTAG-basierte Konfiguration von programmierbaren Bausteinen in einem eingebetteten System erarbeitet. Diese Konzepte mussten umgesetzt und anschließend auf ihre Effizienz bezüglich des Ressourcenverbrauchs im System untersucht werden.

1. Einleitung

Programmierbare Logikbausteine, wie FPGA[1] oder CPLD[2] werden heutzutage immer öfter eingesetzt. Ursprünglich wurden diese Bausteine für Rapid Prototyping[3] entwickelt, weil sie die Möglichkeit der Rekonfiguration für den Entwickler bieten. Dadurch wurde es erstmals möglich, komplexe anwenderspezifische Schaltungen in einem Baustein zu realisieren, ohne auf ASIC[4] zurückgreifen zu müssen.

Durch die stetig sinkenden Preise für die programmierbare Logik, werden FPGA- und CPLD-Bausteine heutzutage immer öfter in Endprodukten eingesetzt. Hersteller, wie Altera oder Marktführer Xilinx, bieten zertifizierte Lösungen für verschiedene Einsatzgebiete. Im Automotive Bereich bietet z.B. Altera mit den Cyclone-FPGA und den MAX-CPLD Low-Cost-Lösungen für diverse Anwendungen.

Neben der eigentlichen programmierbaren Logik bieten vor Allem FPGA-Bausteine weitere Leistungsmerkmale, wie z.B. Soft- und Hard-IP[5]-Cores. Dabei handelt es sich um bereits vorgefertigte Strukturen, die entweder als Quellcode für FPGA vorliegen (Soft-Core), oder als fertige Schaltung in den Baustein integriert sind (Hard-Core). Ein einfaches Beispiel eines solchen IP-Cores wäre ein Multiplizierer, der komplexe Multiplikationen in einem Takt ausführen kann. Mit der programmierbaren Logik können solche Geschwindigkeiten nicht erreicht werden. Außerdem können die IP-Cores eingebettete Prozessoren, Ethernet oder I²C-Schnittstellen oder verschiedene Filter für die digitale Signalverarbeitung sein.

---

1 Field Programmable Gate Array

2 Complex Programmable Logic Array

3 Schnelle Prototypen-Entwicklung

4 Application Specific Integrated Circuit

5 Intelectuall Property

Mit einem solchen Angebot auf nur einem Baustein ist es offensichtlich, dass FPGA-Bausteine sehr wichtig für die moderne Hardware-Entwicklung geworden sind. Der größte Vorteil der programmierbaren Logikbausteine liegt allerdings in der Rekonfigurierbarkeit. Damit können sehr schnell Design-Änderungen in der Entwicklung oder Aktualisierungen an der Firmware vorgenommen werden, ohne einen neuen Baustein verwenden zu müssen.

FPGA-Bausteine besitzen einen SRAM[6]-basierten Konfigurationsspeicher. Damit müssen diese Bausteine bei jedem Einschalten neu konfiguriert werden. Der Konfigurationsspeicher bei einem CPLD besteht dagegen aus EEPROM[7]-Zellen, somit behält er seine Konfiguration dauerhaft.

Für die Konfiguration von FPGA-Bausteinen werden seitens der Hersteller unterschiedliche Konfigurationsmethoden angeboten. Im Rahmen dieser Arbeit wurden folgende Konfigurationsverfahren betrachtet:

- Passive Konfiguration (seriell/parallel)
- Aktive Konfiguration (seriell/parallel)

Beide Verfahren sind sehr schnell. Die Nachteile dieser Verfahren liegen zum einen an den dedizierten Schnittstellen und zum anderen an der fehlenden Unterstützung für CPLD-Bausteine. Eine weitere Konfigurationsart, die diese Nachteile ausgleicht, ist die JTAG-basierte Konfiguration. Hierbei handelt es sich um eine normierte Methode, die sowohl für FPGA- als auch für CPLD-Bausteine eingesetzt werden kann.

2. Stand der Technik

In diesem Kapitel wird zunächst die Funktionsweise des normierten JTAG-Verfahrens beschrieben. Anschließend wird näher auf das Konfigurationsprinzip JTAG-konformer Bausteine eingegangen.

2.1. Die Boundary-Scan-Norm (JTAG)

JTAG steht für Joint Test Action Group und bezeichnet den IEEE 1149.1 Standard mit dem Namen „Standard Test Access Port and Boundary Scan Architecture“. Dieser Standard ist auch unter dem Namen Boundary-Scan bekannt. Entwickelt wurde das Verfahren Mitte der 80er Jahre als sich führende Halbleiter-Hersteller versammelt und die Joint Test Action Group gegründet haben. Im Jahr 1990 wurde das Verfahren standardisiert, die neueste Fassung des Standards ist aus dem Jahr 2001.

Ursprünglich wurde das Verfahren zum Testen und Debuggen elektronischer Hardware entwickelt. Die JTAG entwickelte ein Verfahren, mit dem es möglich wurde, sowohl die Verbindungen zwischen einzelnen Bausteinen auf der Platine zu testen als auch die Signale im Baustein, z.B. einem IC[8], zu messen. Der Testvorgang beim Boundary-Scan-Verfahren konnte komplett, ohne physische Eingriffe ins Testsystem vorzunehmen, durchgeführt werden.

Die Idee des Boundary-Scan-Verfahrens war es, alle Bausteine auf der Platine in einer seriellen Schiebekette miteinander zu verbinden. Dafür wurden zusätzliche Speicherkomponenten in die Bausteine integriert. Diese Komponenten befinden sich zwischen den I/O-Pins und der eigentlichen Kernlogik des Bausteins (Abbildung 1). Für die Ansteuerung dieser Komponenten wurde eine universelle Schnittstelle geschaffen, die nur aus vier Leitungen besteht: einem seriellen Eingang TDI[9], einem seriellen Ausgang TDO[10], einer Steuerleitung TMS[11] und einer Taktleitung TCK[12]. Manche Bausteine besitzen noch eine optionale fünfte Leitung TRST[13], welche die Testlogik zurücksetzen kann. TDI- und TDO-Leitungen der Bausteine auf der Platine werden zu einer Leitung verbunden. TMS und TCK liegen an allen Bausteinen parallel an.

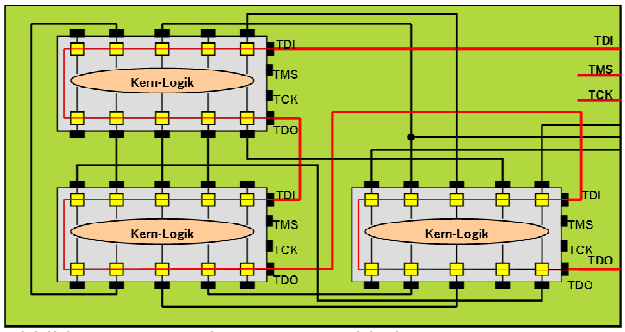


Abbildung 1: Boundary-Scan-Architektur

Mit dieser geschickten Anordnung kann somit die Struktur der Platine aufgelöst und ein Pfad durch die gesamte Platine gezogen werden. Dieser Pfad ist von der eigentlichen Funktion der Platine unabhängig. Die Speicherelemente zwischen den Pins und der Kernlogik der Bausteine sind dabei als virtuelle Testpunkte zu betrachten. Die Signale an diesen Testpunkten lassen sich messen und können

6 Static Random Access Memory

7 Electrically Erasable Programmable Read Only Memory

8 Integrated Circuit

9 Test Data Input

10 Test Data Output

11 Test Mode Select

12 Test Clock

13 Test Logic Reset

anschließend seriell am TDO-Ausgang ausgegeben werden.

Neben den Speicherkomponenten zwischen den I/O-Pins und der Kernlogik wurden im Standard weitere Merkmale einer Boundary-Scan-dedizierten Hardware definiert. Intern besteht eine solche Hardware aus mehreren Registern (Abbildung 2).

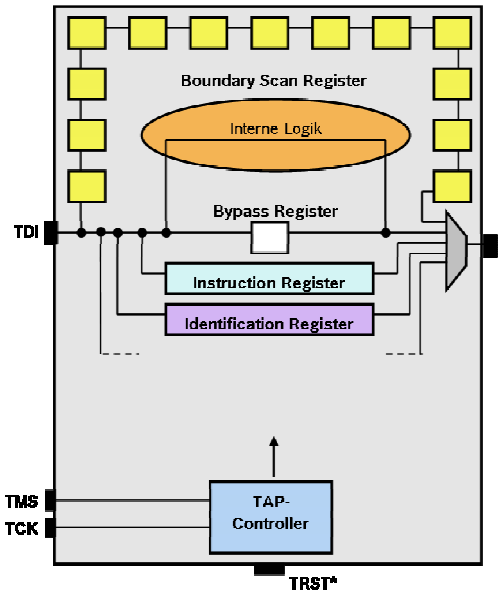


Abbildung 2: Boundary-Scan-Hardware

Im Standard sind drei Register vorgeschrieben: Instruction Register, Boundary-Scan Register und Bypass-Register. Es können aber auch andere Register definiert werden, wie z.B. das Device-Identification Register.

Die Funktion der Register ist folgende:

- Instruction Register: Dieses Register wird dazu benutzt, den Baustein in einen bestimmten Betriebsmodus zu versetzen. Dafür wird eines der Datenregister zwischen TDI und TDO geschaltet.
- Boundary-Scan Register: Das Boundary-Scan-Register besteht aus einer Aneinaderreihung von sog. Boundary-Scan-Zellen, das sind die Speicherkomponenten zwischen den I/O-Pins und der Kernlogik des Bausteins. Dieses Register wird für diverse Testmodi verwendet.
- Bypass-Register: Das Bypass-Register hat eine Länge von einem Bit. Es wird dazu benutzt, den Baustein aus einer Kette von Bausteinen zu lösen, ihn zu umfahren („bypass“).

Zugriffe auf alle Register der Hardware werden über den sog. TAP[14]-Controller ermöglicht (Abbildung 3).

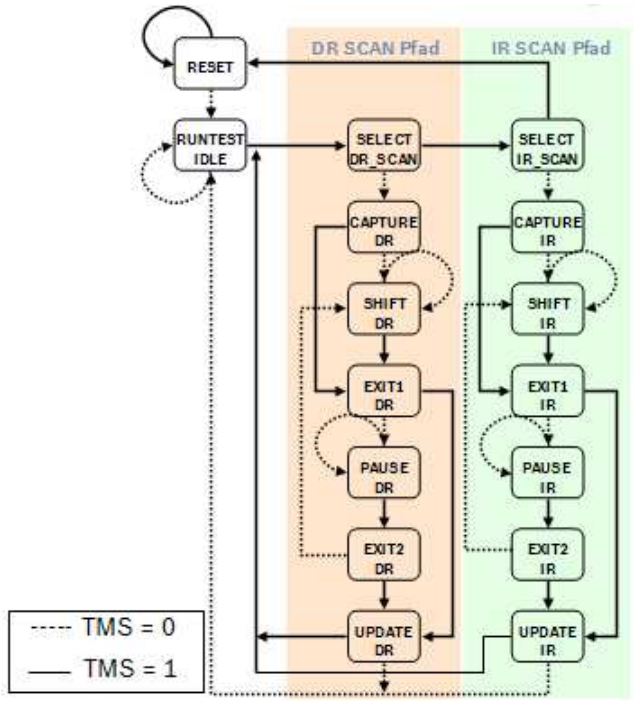


Abbildung 3: Zustandsdiagramm des TAP-Controllers

Der TAP-Controller ist ein Zustandsautomat mit 16 Zuständen. Die Zustandsübergänge werden durch das TMS-Signal gesteuert und mit der steigenden Flanke vom TCK-Signal eingeleitet. Die Abbildung 3 zeigt den Zustandsautomaten und die Bedingungen für die Zustandsübergänge.

Alle Zustände im TAP-Controller haben zwei Ausgänge. Wichtig sind hierbei die zwei Hauptpfade, IR_SCAN und DR_SCAN. Mit ihnen wird bestimmt, ob ein Instruction- oder ein Datenregister momentan aktiv ist. IR_SCAN wird durchlaufen, wenn eine Anweisung in das Instruction-Register geladen werden soll. DR_SCAN wird durchlaufen, wenn eines der Datenregister angesprochen wird. Welches Datenregister verwendet wird, hängt davon ab, welche Anweisung zuvor in das Instruction-Register geladen wurde.

Der TAP-Controller bildet das interne Protokoll der JTAG-Schnittstelle. Alle Betriebsmodi werden durch dieses Protokoll gesteuert.

2.2. In-System-Configuration (ISC)

Nach der Entwicklung des IEEE 1149.1 Standards haben die Hersteller von programmierbaren

[14] Test Access Port

Logikbausteinen schnell gemerkt, dass das sich Verfahren sehr gut zum Konfigurieren von bereits verbauten Bausteinen einsetzen lässt. Das Problem dabei war, dass jeder Hersteller seine eigene Methode für die Konfiguration benutzt hat. Deshalb war es nicht möglich, Bausteine verschiedener Hersteller in einer JTAG-Kette zu programmieren. Daraufhin entstanden Ideen für herstellerunabhängige Konfigurationsverfahren.

Die Entwicklung des neuen Standards begann im Jahr 1996 und im Juli 1998 wurde der IEEE 1532 Standard fertig gestellt. Dieser Standard ist auch unter dem Namen In-System-Configuration bekannt. Die aktuelle Version des Standards ist aus dem Jahr 2002.

IEEE 1532 basiert auf dem IEEE 1149.1 Standard und benutzt dieselbe physikalische Schnittstelle und denselben TAP-Controller. Der Unterschied zwischen den beiden Standards liegt in einem erweiterten Register-Set (Tabelle 1). Diese Register versetzen den Baustein in spezielle Betriebsmodi, die für die Konfiguration benötigt werden. Neben bestimmten Pflicht-Modi gibt es noch weitere, diese sind aber herstellerabhängig.

Tabelle 1: ISC-Betriebsmodi

ISC-Betriebsmodi	Funktion
IDCODE, USERCODE	Pflicht im IEEE 1532
ISC_ENABLE	Aktiviert den ISC-Modus
ISC_DISABLE	Deaktiviert den ISC-Modus
ISC_ERASE	Löscht den Konfigurationsspeicher
ISC_PROGRAM	Beschreibt den Konfigurationsspeicher
ISC_NOOP	Keine Operation

IEEE 1532 bietet viele Vorteile für den Nutzer. In erster Linie ist das Verfahren sehr effizient, was Programmierzeit und Kosten angeht. Außerdem ermöglicht es das sog. „concurrent Programming“, das bedeutet, dass mehrere Bausteine verschiedener Hersteller in einer JTAG-Kette konfiguriert werden können.

Der prinzipielle Ablauf des Konfigurationsvorgangs ist im IEEE 1532-Standard definiert. Die Konfiguration wird dabei in mehreren Schritten vorgenommen. Damit diese Schritte ausgeführt werden, muss der TAP-Controller des Bausteins immer eine bestimmte Zeit im Zustand RUNTEST/IDLE verbleiben. Nachfolgend werden die einzelnen Schritte eines Konfigurationsvorgangs beschrieben.

Zuerst wird der Baustein aus dem Normalbetrieb genommen und in den Konfigurationsmodus versetzt:

1. Laden der Anweisung ISC_ENABLE im Zustand *SHIFT_IR*.
2. Laden der Daten, die zu dieser Anweisung gehören, im Zustand *SHIFT_DR*.
3. Eine bestimmte Zeit im Zustand *RUN_TEST/IDLE* abwarten.

Die zweite Sequenz löscht den Konfigurationsspeicher des Bausteins. Die Schritte 5. und 6. können wiederholt werden, wenn im Baustein mehrere Sektoren gelöscht werden müssen:

4. Laden der Anweisung ISC_ERASE im Zustand *SHIFT_IR.*
5. Laden der Daten, die zu dieser Anweisung gehören, im Zustand *SHIFT_DR*.
6. Eine bestimmte Zeit im Zustand *RUN_TEST/IDLE* abwarten.

Im nächsten Schritt wird der Konfigurationsspeicher des Bausteins mit neuen Daten beschrieben. Die Schritte 8. und 9. werden so oft wiederholt, bis der Konfigurationsspeicher vollständig beschrieben wurde:

7. Laden der Anweisung ISC_PROGRAM im Zustand *SHIFT_IR*.
8. Laden der Daten, die zu dieser Anweisung gehören, im Zustand *SHIFT_DR.*
9. Eine bestimmte Zeit im Zustand *RUN_TEST/IDLE* abwarten.

Nach der Konfiguration wird dem Baustein signalisiert, dass der Konfigurationsvorgang beendet wird:

10. Laden der Anweisung ISC_DISABLE im Zustand *SHIFT_IR*.
11. Eine bestimmte Zeit im Zustand *RUN_TEST/IDLE* abwarten.
12. Laden der Anweisung BYPASS im Zustand *SHIFT_IR*.

Mit der letzten Anweisung wird der Baustein in den Normalbetrieb versetzt, und nimmt das Verhalten an, welches durch die Konfigurationsdaten beschrieben wurde. Die Konfigurationsdaten, die in den Konfigurationsspeicher des Bausteins geladen werden, werden bei der Design-Entwicklung für den Baustein in speziellen Dateien abgelegt.

Normalerweise befinden sich diese Daten in einer Bitimage-Datei. Dieses Bitimage wird z.B. für die aktive oder passive Konfiguration verwendet.

Für die JTAG-basierte Konfiguration können spezielle Dateien erzeugt werden, die sowohl die Konfigurationsdaten beinhalten als auch den Konfigurationsablauf beschreiben. Bei diesen Dateien handelt es sich um das Serial Vector Format (SVF) und Jam™ Standard Test and Programming Language (STAPL).

2.3. Dateiformate für die JTAG-basierte Konfiguration

Das Serial Vector Format wurde 1991 unter Zusammenarbeit von Texas Instruments und Teradyne entwickelt. Das Ziel der Entwicklung war es, eine herstellerunabhängige Beschreibungssprache für JTAG-Operationen zu erstellen. So entstand ein Dateiformat zum Austausch von Boundary-Scan-Testvektoren. Heutzutage wird das Serial Vector Format von der Fa. ASSET InterTech gepflegt.

SVF wird hauptsächlich für ATE[15] verwendet. Aber auch als Programmierdatei für programmierbare logische Schaltungen, wie FPGA oder CPLD, ist das Dateiformat bestens geeignet.

Das Jam STAPL-Format ist der JEDEC[16]-Standard der Jam Sprache der Fa. Altera. Ursprünglich entwickelte Altera diese Sprache für die Programmierung ihrer CPLD. Das standardisierte Format sollte ein herstellerunabhängiges Format werden. Jam STAPL hat sich aber bis heute nicht bei allen Herstellern durchgesetzt.

Eine Jam STAPL-Datei beinhaltet Algorithmen, die Signale für die IEEE 1149.1- Schnittstelle erzeugen. Verschiedene JTAG-konforme Bausteine können mithilfe einer Jam STAPL-Datei konfiguriert werden.

2.3.1 Serial Vector Format (SVF)

Eine SVF-Datei besteht aus einer Folge von ASCII-Anweisungen, die beschreiben, wie die JTAG-state-machine durchlaufen werden soll. Eine SVF-Anweisung kann maximal 256 Zeichen pro Zeile enthalten, darf aber auf mehreren Zeilen verteilt sein. Die Anweisungen werden mit einem Semikolon abgeschlossen. Jede Anweisung enthält ein Kommando und die dazugehörigen Parameter. Auskommentierte Zeilen werden mit vorangestellten „//“ oder „!“ dargestellt. Die Abbildung 4 zeigt einen Ausschnitt aus einer SVF-Datei, welche im Rahmen dieser Arbeit benutzt wurde.

Die Anweisungen in der SVF-Datei werden sequentiell abgearbeitet. Sprünge innerhalb der Datei, Schleifen oder Rekursionen sind nicht möglich.

Nachfolgend eine Übersicht über die wichtigsten Kommandos beim Serial Vector Format:

- SIR: (*S*can *I*nstruction-*R*egister) Läuft den IR_SCAN-Pfad des TAP-Controllers durch und lädt eine Anweisung in das Instruction-Register.
- SDR: (*S*can *D*ata-*R*egister) Läuft den DR_SCAN Pfad des TAP-Controllers durch und lädt Testvektoren in das Datenregister.
- RUNTEST: versetzt den TAP-Controller in einen stabilen Zustand für eine bestimmte Anzahl von Takten, eine bestimmte Zeit oder beides.
- ENDIR: gibt den Endzustand für die Operationen im IR_SCAN Pfad an.
- ENDDR: gibt den Endzustand für die Operationen im DR_SCAN Pfad an.
- STATE: Versetzt den TAP-Controller in einen der stabilen Zustände. Als Zustandsübergänge können alle Zustände des TAP-Controllers angegeben werden. Stabile Zustände sind RESET, IDLE, DRPAUSE oder IRPAUSE.
- FREQUENCY: gibt die maximale TCK Frequenz an. Diese Frequenz ist gültig, bis der nächste Aufruf von FREQUENCY erfolgt oder die SVF-Datei zu Ende ist. Als Parameter wird hierbei die Frequenz in Hz angegeben. Diese Frequenz stellt die maximale sichere Frequenz für den Baustein dar.

Weitere Kommandos, wie TRST, HIR, HDR, TIR, TDR, PIO, PIOMAP werden hier nicht behandelt. Diese Kommandos werden für die parallele Konfiguration von mehreren Bausteinen verwendet. Da der Schwerpunkt der Arbeit nicht auf der Konfiguration mehrerer Bausteine lag, und somit auch keine Testumgebung für diese Konfiguration vorhanden war, wurden diese Anweisungen nicht weiter betrachtet, Informationen zu diesen Anweisungen können der SVF-Spezifikation entnommen werden.

[15] Automatic Test Equippment. Verschiedene Messtechniken, die zum Testen von Chips und Elektronik-Bauteilen verwendet werden.

[16] US-Amerikanische Organisation zur Standardisierung von Halbleitern

Abbildung 4: Ausschnitt aus einer SVF-Datei

2.3.2 JAM STAPL Format

Im Gegensatz zum SVF-Format handelt es sich bei einer Jam STAPL-Datei um eine richtige Sprache mit Verzweigungen, Schleifen und Abfragen. Dadurch erhofften sich die Entwickler kürzere Ausführungszeiten. Außerdem kann eine Jam STAPL-Datei entweder einer ASCII-Datei oder eine kompilierte Byte-Code-Datei sein. Im weiteren Verlauf dieser Arbeit wurden immer Byte-Code-Dateien verwendet.

Eine STAPL-Datei besteht aus zwei Teilen. Im ersten Teil befindet sich der Programmieralgorithmus, der ähnlich dem SVF aus Anweisungen besteht. Dieser Programmieralgorithmus ist z.B. für alle Bausteine derselben Baureihe gleich. Den zweiten Teil einer Jam STAPL-Datei bilden die Konfigurationsdaten. Diese Daten werden bei Jam STAPL komprimiert in einem Array abgelegt. Zur Laufzeit muss die Software, die eine Jam STAPL-Datei ausführt, diese Daten dekomprimieren.

Die Anweisungen im Programmieralgorithmus beinhalten einen optionalen Bezeichner, ein Kommando und dazugehörige Parameter. Diese können Konstanten, Variablen oder Ausdrücke sein, mit dem dazugehörigen Typbezeichner (Boolean oder Integer). Normalerweise nimmt eine Anweisung eine Zeile in der STAPL-Datei ein, dies ist aber keine Vorschrift. Jede Anweisung wird mit einem Semikolon abgeschlossen. Auskommentierte Zeilen werden mit einem vorangestellten Apostroph angegeben, diese werden beim Lesen durch die Software ignoriert. Im Gegensatz zu SVF gibt es bei Jam STAPL keine Einschränkungen, was die Zeilenlänge oder die maximale Größe einer Datei angeht.

Abbildung 5 zeigt einen Ausschnitt aus einer Jam STAPL-Datei im ASCII-Format.Die grobe Struktur der STAPL Datei ist dabei wie folgt aufgebaut:

- ***NOTE*** Anweisungen enthalten Textstrings, die den Inhalt der STAPL Datei dokumentieren.
- ***ACTION*** Anweisungen beschreiben die Schritte, die gemacht werden müssen, um einen kompletten Vorgang, wie z.B. Programmieren oder Löschen eines Bausteins, auszuführen. Für jede solcher Operationen gibt es ACTION Anweisungen, die eine Liste von Prozeduren enthalten. Manche dieser Prozeduren können optional oder empfohlen sein, es kann also vom Benutzer festgelegt werden, ob diese aufgerufen werden sollen oder nicht. Solche Prozeduren werden mit den zusätzlichen Bezeichner RECOMMENDED oder OPTIONAL beschrieben. In der Abbildung 5 ruft z.B. die ACTION „PROGRAM“ die Prozeduren „L0“, „DO_BLANK_CHECK“, „DO_VERIFY_RECOMMENDED“ und „L25“ auf.
- Ein ***PROCEDURE*** Block enthält eine Liste von Jam STAPL Aufrufen. Diese können Variablen sein, die wiederum andere Prozeduren

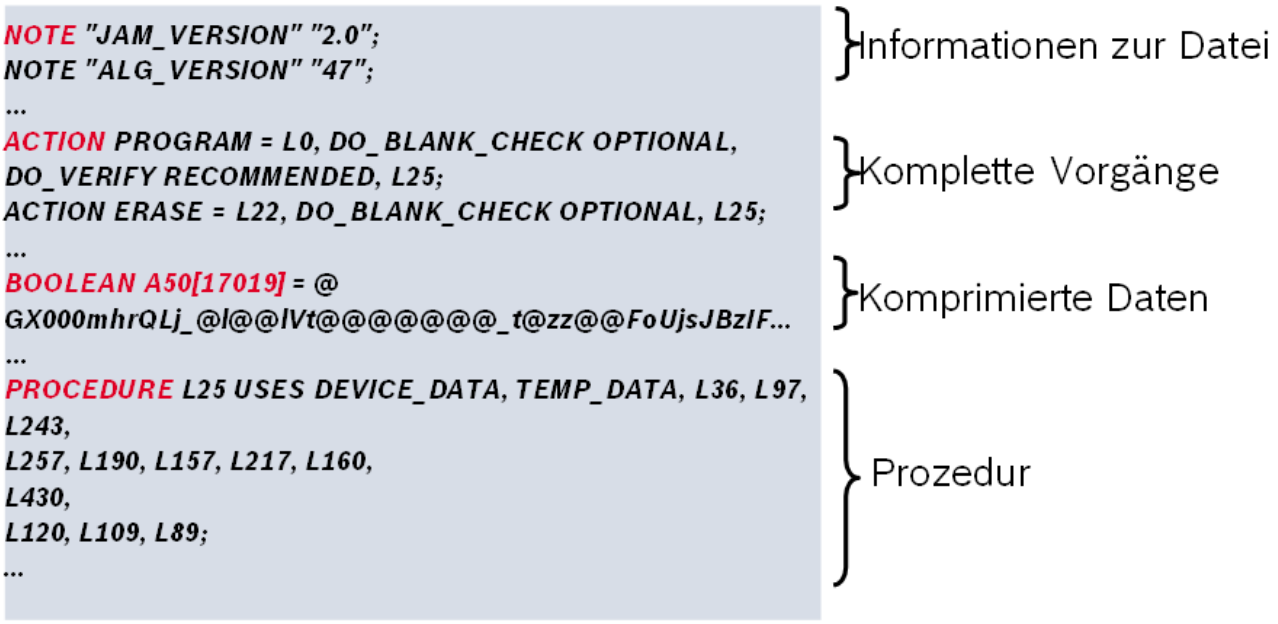


Abbildung 5: Ausschnitt aus einer ASCII Jam STAPL Datei

aufrufen. Abgeschlossen werden die Prozeduren immer mit einer Anweisung ***ENDPROC***.

- ***CRC*** Anweisung steht am Ende der Datei und stellt die Prüfsumme als eine Hexadezimalzahl dar. In der Prüfsumme sind alle Zeichen, ausgenommen *Carriage Return (CR)*, aufsummiert. So kann die Software beim Ausführen der Jam STAPL-Datei, prüfen, ob die Datei korrekt geladen wurde; indem die Zeichen noch mal gezählt und mit dem angegebenen CRC Wert verglichen werden.

Diese Anweisungen bilden das Grundgerüst einer Jam STAPL-Datei. Im Ausschnitt aus der Abbildung 5 ist noch das Array mit komprimierten Konfigurationsdaten zu sehen. In eckigen Klammern wird dabei die Größe der Konfigurationsdaten in Bit angegeben. In diesem Fall beträgt die Größe der Konfigurationsdaten 17019 Bit oder ca. 2 KByte.

Bei Jam STAPL müssen die Variablen innerhalb des PROCEDURE- oder DATA-Blocks deklariert werden. Die Anweisungen NOTE, ACTION und CRC müssen außerhalb aller Blöcke erfolgen. Alle anderen Anweisungen, bis auf weitere PROCEDURE-Anweisungen, dürfen nur innerhalb eines PROCEDURE-Blocks erfolgen.

Jam STAPL unterstützt 32-Bit lange, vorzeichenbehaftete Integer- und Booleanwerte als Datentypen. Eindimensionale Arrays von diesen Typen dürfen ebenfalls verwendet werden.

Ein Jam STAPL-Programm startet, indem der Benutzer eine ACTION Anweisung aufruft. Wenn diese ACTION beendet wird, stoppt das Programm. Wenn eine andere ACTION ausgeführt werden soll, muss die Software neu gestartet werden. Prozeduren, die zu der aufgerufenen ACTION gehören, werden der Reihe nach aufgerufen und durch die Anweisung ENDPROC beendet. CALL Anweisungen werden zum Aufrufen von Prozeduren benutzt. GOTO Anweisungen erlauben das „Springen“ innerhalb einer Prozedur.

Das Jam STAPL-Format erlaubt kein Verlinken von mehreren Dateien. Ebenso darf keine andere Datei innerhalb einer Jam STAPL-Datei vorkommen.

Dies ist nur ein grober Überblick über den Aufbau einer Jam STAPL-Datei. In Wirklichkeit ist das Format sehr viel komplexer. Neben den bereits angesprochenen Funktionen bietet es unter Anderem Schleifen, Verzweigungen und Rekursion an. Im Rahmen dieser Arbeit musste keine Software erstellt werden, die dieses Format verstehen sollte, es wurden frei zugängliche Jam STAPL-Applikationen verwendet. Ausführlichere Informationen zum Jam STAPL-Format können der Spezifikation entnommen werden.

3. Realisierung

Für die Realisierung der JTAG-basierten Konfiguration wurde eine Testplatine zur Ansteuerung eines Bildschirms im Fahrzeug verwendet. Das Herzstück dieser Platine ist ein 32-Bit Mikrocontroller der Fa. Fujitsu. Dieser generiert Bildsignale für den Bildschirm, steuert diverse LED’s und regelt die gesamte Funktion der Platine. Um die Bildsignale an

den Bildschirm zu senden, wird ein CPLD der Fa. Altera verwendet (Abbildung 6).

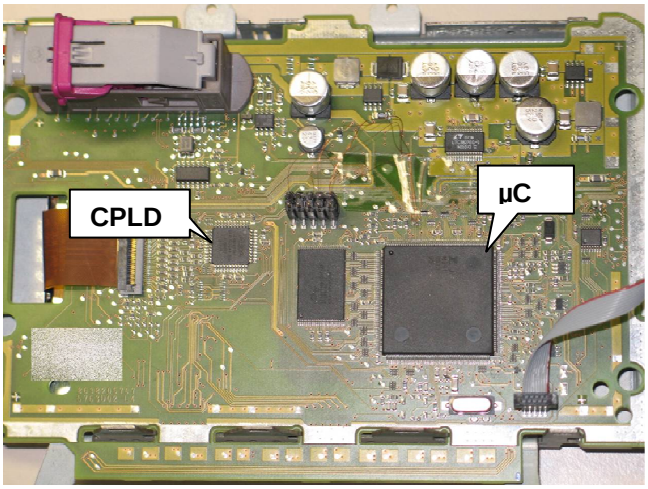


Abbildung 6: Hardware-Entwicklungsplattform

Bei dem verbauten Mikrocontroller handelt es sich um einen 32-Bit RISC-Controller aus der MB91467D-Familie, mit einer CPU-Taktfrequenz von 96 MHz und einem Flashspeicher von 1088 KB. Der Mikrocontroller besitzt einen SRAM-Speicher mit einer Größe von 64 KByte. Der Speicher teilt sich in zwei Bereiche auf: einen DRAM[17]-Bereich und einen IDRAM[18]-Bereich. Die Abbildung 7 zeigt einen Ausschnitt aus dem verfügbaren Speicherbereich des Mikrocontrollers.

Für den Mikrocontroller bietet Fujitsu eine Entwicklungsumgebung, FR Softune Workbench, an. Diese beinhaltet einen C-Compiler, Assembler, Linker und einen Debugger. Die Konfigurationssoftware, die im Rahmen der Arbeit entstand, wurde in der C-Sprache mithilfe dieser Entwicklungsumgebung geschrieben.

Beim Erarbeiten des Konzepts für die Konfiguration mithilfe der JTAG-Schnittstelle in einem eingebetteten System, wurden zwei unterschiedliche Methoden in Betracht gezogen. Die erste Methode basiert auf der Portierung einer Jam STAPL-Applikation der Fa. Altera. Die zweite Methode besteht aus dem Entwickeln einer eigenen Applikation zum Ansteuern der JTAG-Schnittstelle, basierend auf dem Serial Vector Format.

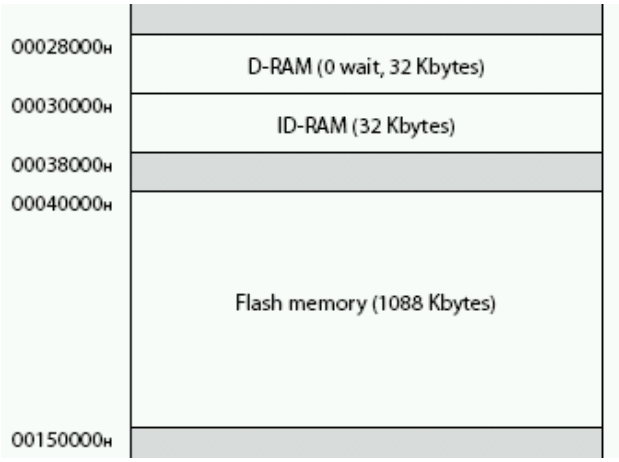


Abbildung 7: Speicherbereiche im µ-Controller

Die grundlegenden Unterschiede beider Konzepte liegen in der Verarbeitung der Konfigurationsdaten. Altera verspricht durch das Nutzen von Jam STAPL-Dateien, geringere Dateigrößen, womit der Speicheplatz gespart werden kann, und schnelle Ausführungszeiten, was sich positiv auf die Konfigurationsdauer auswirkt. Das wird mit der Kompression der Konfigurationsdaten in einer Jam STAPL-Datei argumentiert. Dafür arbeitet eine Jam STAPL-basierte Konfigurationssoftware vollkommen transparent für den Benutzer. Es gibt keine Möglichkeit, weder in den Programmieralgorithmus, noch in die Konfigurationsdaten einzugreifen. Dadurch lässt sich eine Jam STAPL-Applikation nur schwer zum Entwerfen eigener Testverfahren einsetzen.

Das Serial Vector Format bietet dem Benutzer dagegen den Vorteil, dass der Konfigurationsalgorithmus für ihn leicht lesbar ist. Hier bieten sich somit Möglichkeiten, den Ablauf dieses Algorithmus den Wünschen des Benutzers entsprechend anzupassen. Dadurch kann eine SVF-Konfigurationsdatei als Vorlage für eigene Testalgorithmen dienen.

Da beide Konfigurationsverfahren in einem eingebetteten System zum Einsatz kommen sollen und einen Mikrocontroller als Hardwareplattform nutzen, gibt es weitere wichtige Kriterien. Eine Jam STAPL-basierte Applikation ist sehr komplex, wodurch sie sehr viel Ressourcen im System beansprucht. Eine eigene Applikation kann dagegen so gestaltet werden, dass sie nur eine geringe Ressourcen-Auslastung verursacht.

3.1. Jam STAPL-basierte Konfiguration

Für die Konfiguration mithilfe einer Jam STAPL-basierten Anwendung bietet Altera den Jam STAPL

[17] (hier) Data RAM, Datenspeicher

[18] (hier) Instruction/Data RAM, Anweisungen- und Datenspeicher

Byte Code Player Version 2.2[19] (nachfolgend Jam Player genannt), an. Diese Applikation wurde in der C-Sprache geschrieben und auf den Einsatz in einem eingebetteten System vorbereitet. Sie ermöglicht Zugriff zu JTAG-kompatiblen Bausteinen und kann somit zum Testen und Konfigurieren verwendet werden. Dazu liest und verarbeitet sie die Daten aus einer Jam STAPL Byte-Code-Datei.

Die Abbildung 8 stellt die interne Struktur des Jam Players dar. Diese Struktur wurde so gewählt, um die I/O Funktionen separat von den anderen Funktionen zu halten, da diese plattformabhängig sind. Alle anderen Funktionen sind plattformunabhängig und können mit nahezu jeder beliebigen Hardware genutzt werden. Um die Software mit einer eigenen Hardwareplattform nutzen zu können, müssen diese Funktionen an die Hardware-Umgebung angepasst werden.

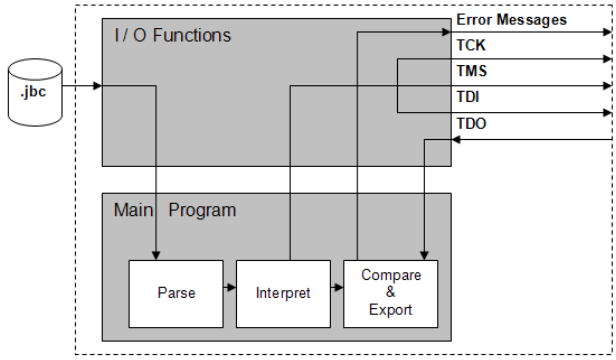


Abbildung 8: Interne Struktur des Jam Players

Über diese I/O-Funktion wird zum einen der Zugriff auf die Byte-Code-Datei ermöglicht und zum anderen werden hier die Signale für die physikalische Schnittstelle des JTAG-Interface erzeugt. Außerdem werden hier alle für den Benutzer relevanten Statusinformationen angezeigt.

Bei der Ausführung liest der Jam Player zunächst die gesamte Byte-Code-Datei ein und führt die gewünschte ACTION-Anweisung aus. Dafür erzeugt er Signale für das JTAG-Interface und liest die Antwort des Bausteins auf diese Signale. Anschließend wird dem Benutzer mitgeteilt, ob die Aktion erfolgreich ausgeführt wurde.

Zum Benutzen der Jam STAPL-basierten Applikation in einem eingebetteten System, sollte eine Portierung des Jam Players auf den Mikrocontroller vorgenommen werden. Dafür müssen folgende Schritte durchgeführt werden:

- Der Inhalt der Byte-Code-Datei muss im Programmcode des Jam Players abgelegt werden, da der Mikrocontroller kein Dateisystem besitzt, und somit keine Datei-Zugriffe unterstützt.
- Der Jam Player erzeugt standardmäßig Signale für die serielle Schnittstelle einer Windows-Umgebung. Für den Mikrocontroller müssen diese Signale an die definierten JTAG-Pins des Mikrocontrollers umgeleitet werden.
- Die Statusmeldungen müssen ebenfalls umgeleitet werden.

Der wichtigste Schritt bei der Portierung ist die Übergabe des Inhalts der Byte-Code-Datei. Unter einer Betriebssystem-Umgebung wird diese Byte-Code-Datei zusammen mit dem Programm aufgerufen. Die Software geht dabei folgenderweise vor:

1. Bestimmen der Dateigröße
2. Öffnen der Datei im binären Modus (.jbc ist eine binäre Datei)
3. Speicherplatz für den Puffer, der die Daten bekommt, anfordern
4. Datei in den Datenpuffer einlesen
5. Datei schließen

Da eine eingebettete Lösung kein Dateisystem besitzt, muss hier anderweitig vorgegangen werden. Die Daten aus der Byte-Code-Datei werden zuerst mithilfe eines kleinen Tools in einem abgelegt. Da die Daten binär sind, müssen sie gleichzeitig in das ASCII Format umgewandelt werden. Wichtig ist hierbei, dass das Daten-Array als konstant deklariert wird. Ansonsten werden die Daten im DRAM-Bereich des Mikrocontrollers landen. Aber auch so entsteht an dieser Stelle ein Problem mit der Portierung des Jam Players. Der DRAM-Bereich des Mikrocontrollers wird beim Portieren so stark belastet, dass er dafür nicht ausreicht.

Der Grund für die hohe Belastung des RAM-Speichers liegt an der Struktur der Jam STAPL Byte-Code-Datei und an der Verarbeitung dieser Datei zur Laufzeit. Wie bereits gezeigt wurde, besteht die Jam-Datei aus einem Programmieralgorithmus und den Konfigurationsdaten, die in der Datei in komprimierter Form vorliegen.

Zur Laufzeit passiert im Speicher des Mikrocontrollers folgendes:

1. Der Programmcode des Jam Players wird zunächst zusammen mit dem Inhalt der Byte-Code-Datei im ROM-Speicher abgelegt.
2. Zur Verarbeitung des Byte-Codes kopiert der Jam Player den gesamten Inhalt der Datei in den RAM-Speicher. Dazu muss im RAM-Speicher ein Puffer angelegt werden, der diesen Inhalt aufnimmt.

[19] Quelle: www.altera.com

3. Im RAM-Speicher werden anschließend die komprimierten Daten dekomprimiert. Das bedeutet, dass sowohl die gesamte Jam STAPL-Datei, als auch die dekomprimierten Daten sich im RAM-Speicher befinden.

Laut Altera ist es so gewünscht, dass der RAM-Speicher stärker als der ROM-Speicher belastet wird. Im Fall eines eingebetteten Systems, wo es keinen dynamischen, sondern nur einen statischen RAM-Speicher gibt, ist das aber ein großer Nachteil.

Altera beschreibt den Speicherverbrauch des Jam Players in der „Application Note 122“. Daraus kann entnommen werden, dass der Jam Player mindestens den Speicher benötigt, der der Größe der Byte-Code-Datei entspricht. Zusätzlich wird für jeden weiteren Baustein, der sich in der JTAG-Kette befindet, Speicherplatz benötigt. Außerdem wird weiterer Speicherplatz für die Dekompression beansprucht.

Die Byte-Code-Datei beim vorliegenden CPLD besitzt eine Gesamtgröße von 36 KByte. Die Konfigurationsdaten beanspruchen 2 KByte Speicher nach der Dekompression. Daraus ergib sich, dass der RAM-Bereich im Mikrocontroller mindestens 38 KByte Platz haben muss, damit die Software ausgeführt werden kann. Diesem stehen jedoch nur 32 KByte zur Verfügung. Dieses Problem kann nur durch eine Erweiterung des DRAM-Bereichs gelöst werden. In den Linker-Einstellungen der Fujitsu-Entwicklungsumgebung kann die Adressierung der Speicherbereiche verändert werden. Somit kann der DRAM-Bereich ganz einfach vergrößert werden. Das kann auf zwei verschiedene Arten vorgenommen werden. Entweder durch das Nutzen eines externen RAM-Speichers, der als DRAM adressiert wird. Oder durch das Vergrößern des internen DRAM-Bereichs, indem ein Teil des IDRAM-Bereichs dem DRAM-Bereich hinzugefügt wird. Da der DRAM-Bereich nur um ca. 10 KByte vergrößert werden musste, wurde hier die zweite Möglichkeit angewendet. Dadurch ließ sich die Software kompilieren und auch ausführen.

Mit der Portierung des Jam STAPL Byte Code Players wurde eine Möglichkeit der JTAG-basierten Konfiguration realisiert. Schwachstellen der Software bilden der hohe Speicherverbrauch und die fehlende Möglichkeit auf die Algorithmenebene zugreifen zu können, um einfache Testverfahren auszuführen.

Der nächste Abschnitt dieses Kapitels befasst sich mit der Entwicklung einer SVF-basierten Konfigurationssoftware, die diese Schwachstellen ausgleichen könnte.

3.2. SVF-basierte Konfiguration

Das Konzept der zweiten Methode bestand darin, eine Applikation zu entwickeln, die auf dem Serial Vector Format aufbaut. Wie bereits erwähnt wurde, wird eine SVF-Datei sequentiell abgearbeitet. Somit können die, in der SVF-Datei enthaltenen, Anweisungen, direkt aufgerufen und die Testvektoren der Reihe nach an die Funktionen übergeben werden. Diese Daten müssen nicht, wie beim Jam Player, komplett eingelesen werden, da sie während eines Programmier- oder Testvorganges nur einmal benutzt werden. Somit liegt die Vermutung nahe, dass die SVF-basierte Konfiguration weniger Ressourcen im Mikrocontroller verbraucht.

Die Idee war es, dem Serial Vector Format ähnliche Funktionen zu implementieren. Diese Funktionen sollen die gleichen Aufgaben, wie die SVF-Kommandos erfüllen können und die gleichen Parameter besitzen. Durch diese Vorgehensweise kann eine einfachere Übergabe der Konfigurationsdaten aus der SVF-Datei erzielt werden.

Für die SVF-basierte Konfigurationssoftware wurde ein hierarchischer Aufbau konzipiert, der wie folgt ausschaut:

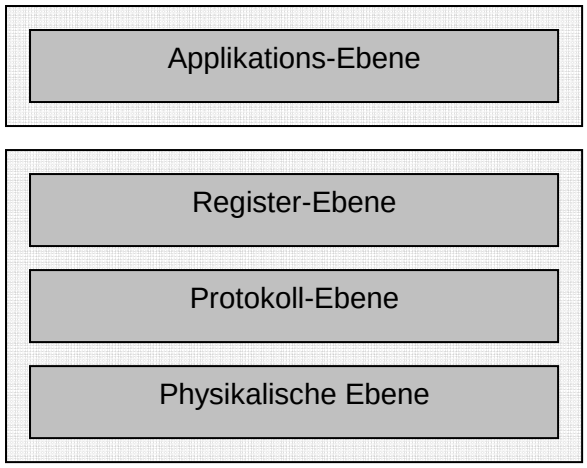


Abbildung 8: Aufbau der SVF-basieten Konfigurations-SW

In der untersten Ebene der SVF-Applikationen wurden Funktionen zum Ansteuern des JTAG-Interface definiert. Hier werden Signale für TDI, TMS und TCK erzeugt und die TDO-Signale empfangen. Die zweite Ebene ermöglicht den Zugriff auf den TAP-Controller. Hier wurden Funktionen zum Initialisieren des JTAG-Protokolls und zum Wechsel in jeden beliebigen Zustand erstellt. Die dritte Ebene ermöglicht den Zugriff auf beliebige Register der JTAG-Hardware. Hier wurden Schiebefunktionen definiert, die sowohl das Instruction-Register als auch die Datenregister

ansprechen können. Die oberste Ebene beinhaltet den eigentlichen Konfigurationsalgorithmus und die Konfigurationsdaten. Dieser Algorithmus kommt aus der SVF-Datei und muss an diese Schicht übergeben werden.

Die unteren drei Ebenen wurden so gestaltet, dass sie universell, sowohl für die Konfiguration, als auch zum Testen der Hardware, eingesetzt werden können. Die oberste Ebene wurde so gestaltet, dass diese für den Benutzer leicht zugänglich ist und jederzeit um eigene Algorithmen, z.B. Testverfahren erweitert werden kann.

Bei der Umsetzung wurde besonders Wert darauf gelegt, die Applikation so einfach und ressourcenschonend wie möglich zu gestalten. Nachfolgend gibt es eine Übersicht über die einzelnen Funktionen.

Physikalische Ebene:

- void **jtag_tdi** (unsigned int tdi)
  Erzeugt ein HIGH oder LOW Signal am TDI-Pin
- void **jtag_tms** (unsigned int tms)
  Erzeugt ein HIGH oder LOW Signal am TMS-Pin
- void **jtag_tck** (void)
  Erzeugt ein HIGH-LOW Wechsel am TCK-Pin
- int **check_output** ()
  Liest bei Bedarf den Wert am TDO-Pin und liefert diesen zurück.

Protokoll-Ebene:

- void **goto_reset_idle**()

  Wird beim Start der Software aufgerufen. Durch das Setzen von TMS auf HIGH und fünfmal Takten von TCK wird der TAP-Controller in Zustand RESET gebracht, wodurch die Schnittstelle initialisiert wird. Anschließend wird der TAP-Controller in den Zustand IDLE gebracht, wo er auf weitere Anweisungen warten kann.

- void **goto_jtag_state**(state jtag_state)

  Diese Funktion ermöglicht den Wechsel in jeden beliebigen Zustand des TAP-Controllers. Der jeweils aktive Zustand wird immer zwischengespeichert. Der Zustandswechsel erfolgt durch das Setzen von TMS auf HIGH und einmal Takten von TCK.

Register-Ebene:

- void **do_irshift** (int length, int instruction)

  Diese Funktion zwingt den TAP-Controller den IR_SCAN Pfad durchzulaufen und eine Anweisung in das Instruction-Register zu laden. Die Anweisung wird bitweise über die TDI-Leitung mit der steigenden Flanke von TCK geschoben. Diese Funktion hat dieselbe Funktionalität wie das SIR-Kommando des Serial Vector Formats.

- void **do_drshift** (int length, char *tdi_data, char *tdo_data, char *mask_data)

  Diese Funktion zwingt den TAP-Controller den DR_SCAN Pfad durchzulaufen und einen Testvektor in das jeweils aktive Daten-Register zu laden. Die Testvektoren kommen aus der SVF-Datei und werden als Zeiger an die Funktion übergeben. Der Testvektor wird bitweise mit der steigenden Flanke von TCK über die TDI-Leitung geschoben. Mit der Fallenden Flanke von TCK wird die Antwort am TDO-Pin empfangen. Diese Antwort muss anschließend maskiert und mit der erwarteten Antwort verglichen werden. Diese Funktion hat dieselbe Funktionalität wie das SDR-Kommando des Serial Vector Formats.

- void **run_test** (unsigned int cycles)

  Diese Funktion bewegt den TAP-Controller in den Zustand RUN_TEST/IDLE für eine bestimmte Anzahl von Taktzyklen. Während dieser Wartezeit wird der interne Konfigurationsspeicher des Bausteins mit den Daten, die zuvor mit der **do_drshift**-Funktion geladen wurden, beschrieben. Die benötigten Taktzyklen werden aus der SVF-Datei ermittelt und mit der Frequenz am TCK-Pin verrechnet. Diese Funktion hat dieselbe Funktionalität wie das RUNTEST-Kommando des Serial Vector Formats.

Applikations-Ebene:

In dieser Ebene befindet sich der gesamte Konfigurationsalgorithmus mit den benötigten Daten. Der Algorithmus kommt aus der SVF-Datei. Die Übergabe der Daten wurde mithilfe eines Hilfstools „SVF2C“ realisiert. Dieses Tool konvertiert alle relevanten Anweisungen aus der SVF-Datei in einen fertigen C-Code. Diese Vorgehensweise funktioniert nur, weil die Funktionen in der SVF-basierten Konfigurationssoftware nach dem Vorbild der SVF-Kommandos geschrieben wurden. Somit lassen sich die SVF-Kommandos relativ einfach durch C-Funktionen ersetzen.

Die Ausgabe von „SVF2C“ ist eine C-Datei. In dieser Datei befinden sich alle Funktionsaufrufe mit den dazugehörigen Testvektoren, die bei der Konfiguration verwendet werden. Die Datei wird in das Source-

Verzeichnis der SVF-basierten Applikation kopiert und in die „main.c“-Datei integriert. Nun kann mit Softune Workbench FR der gesamte Quellcode der SVF-basierten Applikation kompiliert und anschließend in den Speicher des Mikrocontrollers geladen werden. Der Mikrocontroller braucht anschließend lediglich alle Funktionsaufrufe sequentiell abzuarbeiten um den Konfigurationsvorgang vorzunehmen. Der Vorteil dieser Vorgehensweise, ist dass der Benutzer jederzeit eigene Testvektoren erstellen und integrieren kann. Dafür müssen lediglich die **do_irshift**()- und **do_drshift**()-Funktionen mit neuen Parametern versehen werden.

Somit wurde die zweite Möglichkeit der Konfiguration mithilfe des JTAG-Interface realisiert. Im Gegensatz zum Jam STAPL Byte Code Player bietet die SVF-basierte Konfigurationssoftware für den Benutzer die Möglichkeit, auf eine relativ einfache Art und Weise eigene Test-Verfahren zu erstellen. Im nächsten Abschnitt werden beide Applikationen auf ihre Effizienz untereinander verglichen.

3.3. Effizienzvergleich

Laufzeiten: Die Laufzeiten beider Applikationen wurden mit einem Reload-Timer gemessen und können der Tabelle 2 entnommen werden.

Tabelle 2: Ausführungszeiten Jam STAPL und SVF

Aktion	Jam STAPL	SVF
Löschen	0,204 s	0,125 s
Konfigurieren	1,951 s	3,240 s
Verifizieren	0,635 s	0,380 s
Blank Check	0,637 s	0, 380 s

Beim Betrachten der Laufzeiten fällt auf, dass die SVF-basierte Applikation immer schneller als der Jam Player ist, außer beim eigentlichen Konfigurationsvorgang. Der Grund hierfür sind die Programmierzeiten der einzelnen EEPROM-Zellen. Bei der SVF-basierten Applikation werden die maximal sicheren Programmierzeiten verwendet, wogegen der Jam Player die genauen Zeiten zu kennen scheint.

In der Testphase wurden verschiedene Wartezeiten beim SVF-basierten Konfigurationsvorgang getestet. Es wurde festgestellt, dass es sogar möglich ist, die Konfigurationszeit auf unter 1 Sekunde zu bringen, indem die RUNTEST-Zyklen in der SVF-Datei um Faktor 10 verkleinert werden. Da seitens Altera keine Bestätigung dieser Zeiten vorlag, ist es eine unsichere Methode und somit nicht für den Einsatz geeignet.

In diesem Punkt könnte die Arbeit also noch verbessert werden, wenn genaue Programmierzeiten der EEPROM-Zellen im Baustein bekannt werden.

Speicherverbrauch: Der Speicherverbrauch beider Applikationen ist in der nachfolgenden Tabelle 3 dargestellt.

Tabelle 3: Speicherbrauch Jam STAPL und SVF

Speicherbereich	Jam STAPL	SVF
RAM	ca. 38 KByte	ca. 3 KByte
ROM	ca.124 KByte	ca. 63 KByte

Die Gründe für den unterschiedlichen Speicherbedarf liegen in der Implementierung der SVF-basierten Applikation. Die Konfigurationsvektoren werden hierbei aus der SVF-Datei extrahiert und befinden sich im ROM-Speicher. Bei der Ausführung der Applikation werden diese in direkten Funktionsaufrufen nacheinander an die Applikation übergeben. Dadurch wird das komplette Einladen der Datei in den RAM-Speicher erspart.

Ein weiterer Grund ist die unterschiedliche interne Verarbeitung der Konfigurationsdaten. Da die SVF-basierte Applikation alle Daten direkt bekommt, kann sie diese sofort mit den beiden Schiebefunktionen an die JTAG-Schnittstelle senden. Somit wird der Gebrauch von weiteren Funktionen, welche beim Jam STAPL Byte Code Player zum Umwandeln von einzelnen Bytes in Funktionsaufrufe verwendet werden, gespart. Die direkten Funktionsaufrufe sind nur möglich, weil das Serial Vector Format sequentiell und ähnlich dem C-Code aufgebaut ist.

3.4. Datenkompression bei Jam STAPL

Bei den durchgeführten Vergleichen beider JTAG-basierten Konfigurationsverfahren wurde festgestellt, dass eine SVF-basierte Konfiguration für eine eingebettete Lösung die bessere Alternative darstellt. Bis auf die Programmierzeit, arbeitet die SVF-basierte Konfiguration effizienter und verbraucht weniger Ressourcen, als der Jam Player. Die Problematik der

Programmierzeit lässt sich aber auch beseitigen, wenn die genauen Programmierzeiten des CPLD-Bausteins bekannt werden.

Dennoch stellt sich die Frage der Daseinberechtigung des Jam STAPL Byte Codes. Vor allem die Kompression der Konfigurationsdaten, die in der SVF-basierten Konfigurationssoftware fehlt, sollte dem Jam STAPL Byte Code eigentlich Vorteile verschaffen. Um dies zu analysieren wurden weitere theoretische Überlegungen vorgenommen, wie sich beide Konfigurationsarten im Falle eines großen Bausteins verhalten.

Der CPLD-Baustein, der im Laufe dieser Arbeit verwendet wurde, ist der kleinste seiner Familie mit nur 32 Makrozellen. Somit ist auch der Konfigurationsspeicher relativ gering. Die SVF- und Jam STAPL-Konfigurationsdateien in diesem Fall haben fast dieselbe Größe: die SVF-Datei ist 33 KByte groß und die Jam STAPL-Datei 36 KByte. Die SVF-Datei besteht zum großen Teil aus unkomprimierten Daten, die sich auch oft wiederholen können. Jam STAPL dagegen besteht zu einem großen Teil aus dem Konfigurationsalgorithmus, der ziemlich komplex ist. Die eigentlichen Konfigurationsdaten liegen in der Datei in komprimierter Form vor. In der Abbildung 5 ist ein Ausschnitt aus dieser Jam STAPL-Datei dargestellt, die Konfigurationsdaten besitzen gerade mal eine Größe von ca. 2 KByte.

Für die Analyse des Ressourcenverbrauchs im Fall eines großen CPLD-Bausteins mit 512 Makrozellen, wurde ein Testdesign für diesen Baustein erstellt und daraus die SVF- und Jam STAPL-Dateien gebildet. Die Größe der Jam STAPL-Datei stieg dabei leicht um 4 KByte auf rund 40 KByte. Die SVF-Datei dagegen vergrößerte sich auf ca. 360 KByte, also mehr als das 10-fache. Der Grund für diesen Unterschied liegt in der Datenkompression und dem Aufbau des Jam STAPL-Formats.

Da beide CPLD-Bausteine aus der gleichen MAX3000A-Familie stammen, hat sich am Konfigurationsalgorithmus in der Jam STAPL-Datei nichts verändert! Dieser bleibt für Bausteine gleicher Familie immer gleich. Was sich geändert hat, ist die Größe der Konfigurationsdaten. Im Fall des 512-Makrozellen-CPLD stieg die Größe der Konfigurationsdaten auf rund 30 KByte. Komprimiert besitzen diese Daten eine Größe von nur 5 KByte. Die Frage, die sich stellt, was genau bewirkt diese Kompression im Fall eines eingebetteten Systems?

Wie bereits bei der Portierung des Jam Players gezeigt wurde, werden beim Ausführen des Jam Players sowohl der Programmcode als auch die Byte-Code-Datei zunächst im ROM-Speicher abgelegt. Zur Verarbeitung der Byte-Code-Datei und zur Dekompression der Konfigurationsdaten, kopiert der Jam Player anschließend die gesamte Byte-Code-Datei in den RAM-Speicher des Mikrocontrollers. Somit muss der RAM-Speicher den Platz für die Byte-Code-Datei und für die dekomprimierten Daten zur Verfügung stellen. Beim 32-Makrozellen-CPLD benötigte die Byte-Code-Datei ca. 38 KByte RAM-Speicher. Beim großen 512-Makrozellen-Baustein werden hier schon rund 70 KByte Speicherplatz benötigt. Daraus lässt sich ableiten, dass das Jam STAPL-Format immer mehr RAM-Speicher beanspruchen wird, je größer der zu konfigurierende Baustein wird. Der Hintergrundgedanke dieser Vergehensweise ist die Einsparung des ROM-Speichers. Hier liegen nur der Programmcode, an dem sich nichts ändert und die komprimierte Byte-Code-Datei, die nur geringfügig an der Größe zunimmt, da der Großteil der Datei aus dem Konfigurationsalgorithmus besteht.

Die SVF-basierte Applikation arbeitet dagegen genau umgekehrt. Hier werden genauso der Programmcode und der Inhalt der SVF-Datei im ROM-Speicher abgelegt. Der Unterschied ist, dass die SVF-Daten nicht in den RAM-Speicher kopiert werden muss und dieser somit entlastet wird. Der Nachteil allerdings, dass die Software immer mehr ROM-Speicher benötigen wird, je größer die SVF-Datei wird, da die Daten unkomprimiert vorliegen.

Zusammenfassend lässt sich sagen, dass beide Verfahren ihre Vor- und Nachteile haben. Mit der Jam STAPL-basierten Konfiguration lässt sich der ROM-Speicher einsparen, dafür wird der RAM-Speicher ziemlich stark beansprucht. Die SVF-basierte Konfiguration entlastet den RAM-Speicher, benötigt im Gegenzug aber immer mehr ROM, je größer der zu konfigurierende Baustein wird. Für den Fall des 32-Makrozellen-CPLD, der im Laufe dieser Arbeit konfiguriert wurde, und des Fujitsu-Mikrocontrollers mit dem begrenzten RAM-Speicher, ist die SVF-basierte Konfiguration die bessere Alternative.

4. Zusammenfassung/Ausblick

Im Rahmen dieser Arbeit wurden Verfahren für die JTAG-basierte Konfiguration realisiert und erfolgreich getestet. Die Ergebnisse der Arbeit können dazu verwendet werden, die Konfigurationsvorgänge in Produkten bequemer zu gestalten. Dadurch, dass zwei unterschiedliche Konzepte realisiert wurden, die Jam STAPL-basierte Konfiguration und die SVF-basierte Konfiguration, kann, je nach Anwendungsfall, die am besten geeignete Methode verwendet werden.

Die Erkenntnisse dieser Arbeit lassen sich in folgenden Punkten zusammenfassen:

- Es wurden zwei verschiedene Möglichkeiten realisiert, die JTAG-Schnittstelle in einem eingebetteten System anzusteuern.
- Dadurch wurde ermöglicht, programmierbare Logikbausteine im Feld zu konfigurieren. Die Konfigurationssoftware kann dabei sowohl als eigenständige Software agieren als auch ein Teil der Gesamt-Firmware im Mikrocontroller sein.
- Die SVF-basierte Konfiguration verbraucht wenig RAM-Speicher, dafür aber sehr viel ROM, wenn ein großer Baustein konfiguriert werden soll. Die Jam STAPL-basierte Konfiguration benötigt sehr viel RAM-Speicher, geht dafür aber sparend mit dem ROM-Speicher um.
- Die Konfiguration kann auf beliebige JTAG-konforme Bausteine angewendet werden. Dazu werden nur die Konfigurationsdaten (SVF oder Jam STAPL) der jeweiligen Bausteine benötigt.
- Neben der Konfiguration wurde mit der SVF-basierten Software eine Plattform geschaffen, die das Testen im System ermöglicht. Diese Plattform kann um Testverfahren erweitert werden, die für den Benutzer von Interesse sind.

Die SVF-basierte Konfigurationssoftware, die im Rahmen dieser Arbeit entstanden ist, lässt sich problemlos für die Konfiguration von einem Logikbaustein einsetzen. Für die Konfiguration von mehreren Bausteinen in einer JTAG-Kette, müssen die im Rahmen dieser Arbeit nicht betrachteten SVF-Funktionen implementiert werden.

Ebenso können die Programmierzeiten der Software verbessert werden. Dafür wäre es vorteilhaft, die genauen Programmierzeiten der Bausteine von Altera bestätigt zu bekommen.

Für die Realisierung von JTAG-Testverfahren muss die Software ebenfalls noch erweitert werden, vor Allem in der Auswertung der Ergebnisse eines Testablaufs. Aber auch in der Generierung der benötigten Testvektoren und deren Übergabe an die Software, können Verbesserungen erzielt werden. Kommerzielle JTAG-Tools bieten in diesem Zusammenhang viele komplexe Lösungen, wie z.B. automatische Testvektorgenerierung, die vom Schaltkreisdesign abhängig ist. Solche Verfahren sind nur mit einem großen Aufwand realisierbar.

Für das Testen der Funktionalität einzelner Bausteine, wie CPLD oder FPGA kann die Applikation bereits jetzt verwendet werden. Neue Test- und Debugging-Möglichkeiten, wie benutzerspezifische Register, die in neueren FPGA's angelegt werden können, können durch die Software ebenso angesprochen werden. Das Verwenden von diesen Registern wird bei der Entwicklung zukünftiger Hardware-Komponenten eine große Rolle beim Testen und Debuggen spielen.

5. Literatur

Altera Application-Notes

[1] Application Note 88 „Using the Jam Language for ISP & ICR via an Embedded Processor"
[2] Application Note 122 "Using Jam STAPL for ISP & ICR via an Embedded Processor"
[3] Application Note 95 "In-System Programmability in MAX Devices"
[4] Application Note 39 "IEEE 1149.1 Boundary Scan Testing in Altera Devices"
[5] Application Note 100 "In-System Programmability Guidelines"

Datenblätter

[6] Altera MAX3000A Data Sheet
[7] Fujitsu FR60 MB91460D Series Data Sheet

Spezifikationen

[8] STAPL Specification: www.jedec.org/download/search/jesd71.pdf
[9] SVF Specification: http://www.asset-intertech.com/support/svf.pdf

JTAG-Normen

[10] IEEE 1149.1 - "Standard Test Access Port and Boundary Scan Architecture"

[11] IEEE 1532 – In System Configuration

Andere

[12] Boundary Scan Tutorial:

http://www.asset-intertech.com/pdfs/Boundary-Scan_Tutorial_2007.pdf

[13] Altera Configuration Handbook

[14] Fujitsu MB91460 Series User's Manual

[15] Softune™ Workbench User's Manual

[16] Softune™ Linkage Kit Manual for V6

Konzeption und Entwicklung eines Delta-Sigma-Wandlers in VHDL

Alexander Riske

Dirk Jansen, Tobias Volk

Hochschule Offenburg, Badstraße 24

0781/ 205 179, Alexander.Riske@fh-offenburg.de

Zusammenfassung:
Delta-Sigma-Wandler, auch Bitstream-Wandler genannt, werden in der Audiowelt heute nahezu ausschließlich verwendet. Sie sind bei überzeugenden klanglichen Qualitäten auch preiswert und relativ einfach herzustellen.

1. Einleitung

Das Funktionsprinzip des Delta-Sigma-Modulators besteht auf der kontinuierlichen Differenzbildung des Eingangssignal mit dem Ausgangssignal. Der Unterschied beider Signale wird dann aufintegriert und somit der Ausgangsfehler reduziert.

Durch die Modulatorschaltung wird die Quantisierungsrauschenergie in den höheren Frequenzbereich verschoben. Dieser Frequenzbereich wird für die Verarbeitung der Signale aber nicht verwendet, sondern durch den nachfolgenden Tiefpass gefiltert und unterdrückt somit das gesamte Rauschen. Das Verfahren ist unter dem Namen „Noise Shaping“ bekannt.

Die Ordnung des Wandlers wird durch die Anzahl der Integratoren und somit der Gegenkopplungen bestimmt. Je höher die Ordnung des Modulators, desto weiter wird die Rauschleistung im Spektrum verschoben und somit wächst der Nutzungsfrequenzbereich.

In meiner Studienarbeit sollte eine akustische Ausgabe einer WAVE- Datei mit dem SIRIUS- Prozessor realisiert werden. Hierzu sollte ein digitales Modul entwickelt und mit dem Prozessor auf einem FPGA realisiert werden.

Abb. 1 *Stud- PDA, entwickelt in der HS- Offenburg*

Der Prozessor soll dazu eine Melodiedatei von einer SD-Karte auslesen und in ein pulsverhältnismoduliertes Signal (PPM) umwandeln. Der Mittelwert der Überabtastung entspricht immer dem aktuellen Eingangswert.

2. Hardware

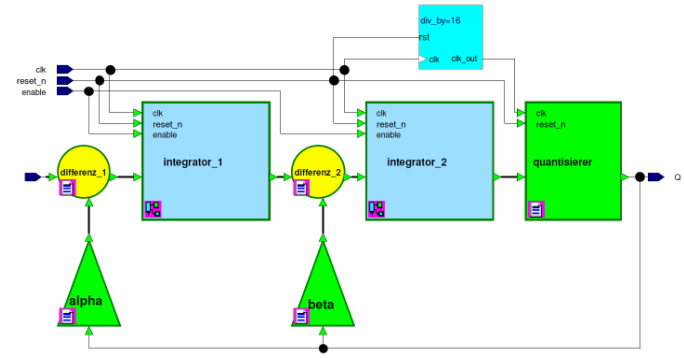


Abb. 2 *Hardwareumsetzung der Sigma-Delta-Modulators*

Im Modulator wird die Differenz aus dem Eingangs- und dem Ausgangssignal über eine oder mehrere Rückkoppelschleifen gebildet. Die Differenz der beiden Signale bildet den momentanen Fehler zwischen dem Eingang- und dem Ausgangsignal. Dieser Fehler wird dann integriert und vom Quantisierer bewertet. Eine digitale „1“ steht im einfachsten Fall für eine positive und eine „0“ für eine negative Abweichung.

Bei genügend hoher Überabtastung tritt zwischen zwei Abtastzeitpunkten nur eine geringe Signaländerung auf, so dass die Verwendung eines einfachen 1-Bit Wandlers möglich wird. Die entstehende serielle Bitfolge stellt ein pulsdichtemoduliertes Signal mit einer höheren Abtastfrequenz dar. Die Bitfolge dieses Datenstromes kann das Originalsignal komplett und eindeutig beschreiben.

Dieser Bitstream wird als Übergabesignal an den Tiefpassfilter weitergeleitet. Die Unterdrückung der hochfrequenten Rauschanteile sowie die Umwandlung des pulsdichtemodulierten Signals in ein analoges Signal werden vom Tiefpassfilter übernommen.

Die Überabtastrate des Gesamtsystems beträgt:

$$\mathrm{OSR} = \frac{48\ \mathrm{MHz}}{16 \cdot 44{,}2\ \mathrm{kHz}} \approx 68$$

3. Software

3.1. Wave-Format

Das WAVE-Dateiformat (auch WAVE RIFF genannt) ist ein Format zur digitalen Speicherung von Audiodaten, das von Microsoft definiert wurde. Die Amplituden der einzelnen Abtastpunkte werden nacheinander in einer Datei gespeichert.

Das WAVE-Format garantiert die Speicherung und Wiedergabe von akustischen Signalen in hoher Qualität, eine Datenkompression wird dabei nicht unterstützt. Das Format arbeitet mit einer Sampletiefen von 8 und 16 Bit und einer Abtastrate von bis zu 44,2 kHz. Dies führt zu einer Datenmenge von 88,2 kB (705 kBit) pro Sekunde. Die Größe der WAVE-Datei bestimmt sich aus den Aufnahmeeigenschaften.

Um das WAVE-Dateiformat vollständig zu beschreiben, ist der Header von Microsoft sehr umfangreich gehalten worden. Die wichtigsten Headerpositionen und deren Bedeutung sind in der Tabelle 1 aufgeführt.

Headerposition	Bytes	Bedeutung
0	4	‚RIFF'- Signatur
4	4	Dateigröße in Byte
8	4	‚WAVE'- Signatur
22	2	Anzahl der Kanäle (1=Mono, 2= Stereo)
24	4	Samplingrate
34	2	Samplingtiefe (8, 16Bits)
44		(akustische) Nutzdaten

Tab.1 *Headerposition und Bedeutung*

Der Headerinhalt definiert demmentsprechen die Konfiguration des Delta- Sigma- Wandlers.

3.2. Softwareentwicklung

Der Dateiinhalt kann viel schneller aus dem RAM als von der SD-Karte gelesen werden, deshalb werden beim Programmstart zwei Buffer mit der Größe von 512 Bytes reserviert. Der Blöcke wird dann mit dem Inhalt der abzuspielenden Datei von der SD-Karte gefüllt. Parallel dazu arbeitet der SIRIUS Prozessor alle 22,6 ms (44,2 kHz) die Interrupt Service Routine *isr_sound()* ab und stellt die Daten des Buffers der Hardware zur Verfügung.

Im Wechselbetrieb wird das Auffüllen der neuen Daten sowie das Wandeln des Bufferinhaltes bis zum Dateiende wiederhollt.

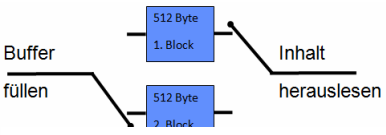


Abb 3. *Wechselbetrieb der beiden RAM-Buffer*

4. Software

Die Funktion des Delta-Sigma-Wandlers ist durch das Abspielen mehrer WAVE-Dateien sowie erstellten Testsignale, wie in der Abbildung 4, erfolgreich nachgewiesen.

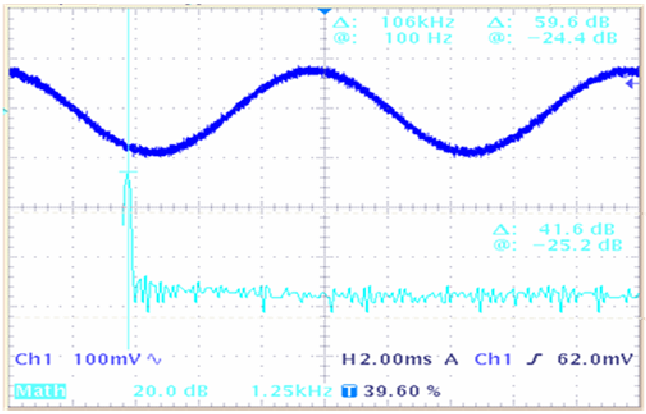


Abb 4. *Ausgangssignal des Tiefpassfilters*

Der Signal-Rauschabstand eines Testsignals mit 100 Hz beträgt mindestens 40 dB.

Zurzeit wird das Gesamtsystem im „Stud-PDA" eingesetzt und den erfolgreichen Abschluss des Projektes.

5. Referenzen

[1] D. Jansen, N. Fawaz, M. Durrenberger, D. Bau: "A Small High Performance Microprocessor Core SIRIUS For Embedded Low Power Designs, Demonstrated in a Medical Mass Application Of an Electronic Pill (ePille©)", Embedded System Design Topic, Techniques and Trends, ISBN 978-0-387-72257-3, p. 363-372, June 2007, California, USA.

[2] A. Riske „Konzeption und Entwicklung eines Delta-Sigma-Wandlers in VHDL" Februar 2010

Implementierung eines Single Pass Connected Component Labeling Algorithmus zur Detektion von leuchtenden Objekten in Nachtszenen im Automotive Umfeld

Dipl.-Inform. (FH) Steffen Jaeckel, Prof. Dr.-Ing. Axel Sikora und Prof. Dr. Wolfgang Rülling

SIZ Embedded Design und Networking, Poststr. 35, 79423 Heitersheim

Tel. +49-7634-6949341, Mail jaeckel@stzedn.de

Der Connected Component Labeling (CCL) Algorithmus wird in vielen bildverarbeitenden Systemen eingesetzt, die zur Objekterkennung genutzt werden. Er wird als Basisalgorithmus verwendet, um bestimmte Teile eines Bildes als ein zusammenhängendes Objekt zu identifizieren und somit die weitere Verarbeitung zu vereinfachen.
Die ursprüngliche Definition des Connected Component Labeling Algorithmus benötigt zwei Durchläufe durch das Bild, wobei im ersten Durchlauf das Labeling durchgeführt wird. In einem zweiten Durchlauf werden sich berührende Labels zusammengeführt.
In diesem Beitrag wird ein Single-Pass-CCL-Algorithmus zur Detektion von leuchtenden Objekten im Automotive Umfeld vorgestellt, der in vielen Fällen zufriedenstellende Ergebnisse liefert. Der Algorithmus wurde auf einem Altera FPGA implementiert und in die Systemumgebung einer Bilderkennung aus dem Automotive Umfeld integriert. Ebenfalls wurde eine web-basierte Testumgebung erstellt. Diese Implementierung dient auch als Framework für die Entwicklung und Umsetzung verbesserter Algorithmen.

1. Motivation

Immer mehr Automobile werden mit Kameras ausgestattet. Diese können mit einem oder mehreren Bildaufnehmern zahlreiche Funktionen übernehmen. Hierzu zählen Sicherheits-, bzw. Warnfunktionen, wie z.B. Lane Keep Assist (LKA), Lane Change Assist (LCA), Verkehrzeichenerkennung, Erkennung von Fußgängern oder anderen verletzlichen Straßenbenutzern (Vulnerable Road Users, VRUs) und Komfortfunktionen, wie z.B. ein automatisches Fern- und Abblendlicht oder eine Nachtsichtunterstützung.

Hierbei ist davon auszugehen, dass kamerabasierte Systeme in wenigen Jahren in praktisch allen Neuwagen zu finden sein werden, auch in kostengünstigeren Kleinwagen. Deswegen ist es wichtig, dass die Basisfunktionen in kostengünstiger Hardware umgesetzt werden können und nicht schnelle und teure Mikroprozessoren hierfür verwendet werden.

Eine Basisaufgabe ist neben der Vorverarbeitung und Vorfilterung die Erkennung zusammenhängender Objekte, die dann als Region of Interest (ROI) näher untersucht werden kann.

2. Architektur objekterkennender Systeme

2.1. Überblick

Objekterkennende Systeme gliedern sich in die bildverarbeitenden Systeme ein. Es handelt sich dabei um Systeme die ein aufgenommenes Bild vorverarbeiten und unter dem Einsatz bildverarbeitender Algorithmen bestimmte Merkmale extrahieren.

Die Aufgaben in einem objekterkennenden System werden in mehreren Schritten durchgeführt (vgl. Bild 1).

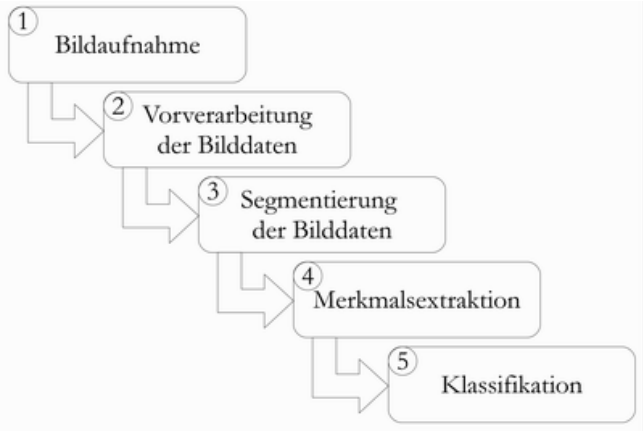


Bild 1 Die einzelnen Ablaufschritte bei der Objekterkennung

Dieser Beitrag beschäftigt sich hauptsächlich mit dem Schritt 4. Es wird jedoch auch kurz auf die

Bearbeitungsschritte 2 und 3 eingegangen da diese Schritte die Grundlage für eine erfolgreiche Merkmalsextraktion legen.

2.2. Vorverarbeitung

Der Schritt der Vorverarbeitung dient dazu, Fehler in den Bilddaten zu korrigieren und die Bildqualität zu verbessern.

$$U(x,y) = \begin{pmatrix} x-1,y-1 & x,y-1 & x+1,y-1 \\ x-1,y & x,y & x+1,y \\ x-1,y+1 & x,y+1 & x+1,y+1 \end{pmatrix} \quad (1)$$

In der Bildverarbeitung werden vielfach lokale Operationen ausgeführt. Hierbei werden neben dem Bildpunkt an Position *(x, y)* auch die direkt benachbarten Bildpunkte, die lokale Umgebung *U*, mit einbezogen, vgl. Gl. (1).

$$m = \frac{1}{16} \cdot \begin{pmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{pmatrix} * U(x,y) \quad (2)$$

Zur Verbesserung der Eingangsbilddaten wird ein Gauß Tiefpassfilter angewendet, vgl. Gl. (2). Es existieren einige Tiefpassfilter, denen ein besseres Glättungsverhalten nachgesagt wird, wie der Median oder der Olympic Filter. Die Implementierung dieser Filter auf einem FPGA ist allerdings wesentlich komplexer, weshalb deren Verwendung außen vor bleibt.

2.3. Segmentierung

Bei der Segmentierung wird das gesamte Bild in einzelne Bereiche, bzw. Objekte mit gleichen Eigenschaften unterteilt. Dabei muss darauf geachtet werden, dass das richtige Segmentierungsverfahren in der benötigten Auflösung verwendet wird. Einerseits würde ein zu detailreiches Bild die nächsten Schritte der objekterkennenden Operationen unnötig aufblähen. Auf der anderen Seite dürfen aber die relevanten Bildinformationen durch eine zu geringe Auflösung nicht verloren gehen.

Laut [5] gibt es mehrere verschiedene Klassen von Segmentierungsverfahren. Dazu zählen

- pixelorientierte Verfahren
- kantenbasierte Verfahren
- regionenbasierte Verfahren
- modellbasierte Verfahren

In Bild 2 wird das pixelorientierte Segmentierungsverfahren der adaptiven Schwellwertoperation demonstriert. Der Ausdruck adaptive Schwellwertoperation bedeutet dabei, dass auf Grund der entstandenen Ergebnisse einer ersten Schwellwertoperation die Grenze des Schwellwertes angepasst (adaptiert) werden kann, um das Ergebnis zu beeinflussen.

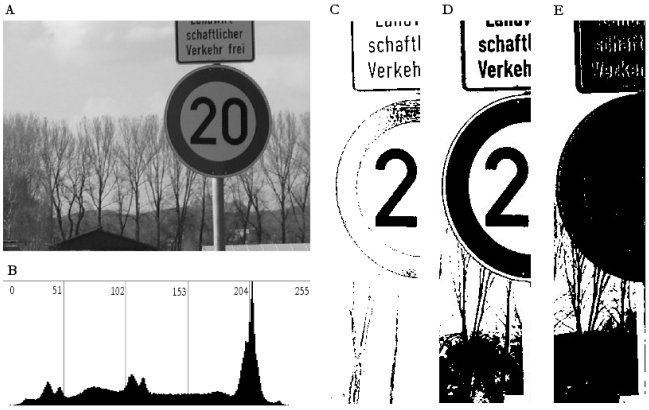


Bild 2 Segmentierung eines Bildes bei ungleichmäßigem Hintergrund: **A** Originalbild von Fb78; **B** Histogramm; **C** bis **E** Ausschnitt des Bildes nach Schwellwertoperation mit einem globalen Schwellwert von 40, 95 bzw. 130

Bei der Segmentierung mit einem Schwellwert wird aus einem Farb-, bzw. Graustufenbild ein Binärbild extrahiert. An Stellen, an denen der Wert des Bildpunktes unterhalb der Grenze des Schwellwerts liegt, wird der Bildpunkt zu Hintergrund, an Stellen oberhalb der Grenze wird der Punkt zu Vordergrund. Wie in Bild 2 klar erkenntlich ist, entstehen merkliche Unterschiede in den Ergebnissen in Abhängigkeit von der Wahl des Schwellwertes.

2.4. Merkmalsextraktion

Nach [4] werden in diesem Schritt die für die Klassifikation bedeutsamen Merkmale extrahiert und in einer Liste abgelegt. Dies ist der schwierigste Schritt in der Objekterkennung beim computerbasierten Sehen, da für die Merkmale "scharfe" Kriterien angewendet werden. Im Gegensatz dazu werden bei der menschlichen Merkmalsextraktion "weiche" Kriterien angewendet, was zu einer höheren Robustheit in Bezug auf die Falscherkennung von Objekten führt. Für die Merkmalsextraktion stehen mehrere Ansätze zur Verfügung. So lassen sich Ansätze basierend auf Kanten und Ecken realisieren, wie in [3] beschrieben. Laut [6] wäre auch ein Ansatz der auf skalierungsinvarianten Merkmalen basierend zielführend. Man kann auch den Ansatz einer Hauptkomponentenanalyse nach [8] verflogen. Der vorgestellte Connected Component Labeling (CCL) Algorithmus unterscheidet sich hauptsächlich durch seine Einfachheit und sequenzielle Ausführbarkeit von den meisten anderen Ansätzen.

Der CCL-Algorithmus wird in vielen bildverarbeitenden Systemen eingesetzt, die zur Objekterkennung genutzt werden. Er wird als Basisalgorithmus verwendet, um bestimmte Teile eines Bildes als ein

zusammenhängendes Objekt, eine so genannte Zusammenhangskomponente (ZHK), zu identifizieren und somit die weitere Verarbeitung zu vereinfachen.

Die ursprüngliche Definition des Connected Component Labeling Algorithmus ist in [10] beschrieben und benötigt zwei Durchläufe durch das Bild, wobei im ersten Durchlauf das Labeling durchgeführt wird, bei dem allen Bildpunkten eines Bildes ein Label zugewiesen wird. Sich berührende Labels werden mit einer modifizierten Variante des in [9] vorgestellten *UNION* Operator verkettet. Im zweiten Durchlauf wird mit Hilfe des *FIND* Operators aus [9] für jedes Label die Wurzel gesucht und ihm zugewiesen. Nach dem zweiten Durchlauf besitzen sich berührende Bildpunkte ein eindeutiges Label und können für weitere Schritte getrennt betrachtet werden.

In der Implementierung des CCL Algorithmus gibt es grundsätzlich die Möglichkeit, unterschiedliche Arten der Connectivity/Verbindung/Nachbarschaft einzubeziehen. Eine Möglichkeit ist die Benutzung der 4-connectivity, bzw. Vierernachbarschaft, wobei die vier direkt anliegenden Bildpunkte zur Betrachtung hinzugezogen werden. Außerdem gibt es noch die 8-connectivity, bzw. Achternachbarschaft, bei der alle acht umliegenden Bildpunkte betrachtet werden. Die Auswahl der verwendeten Art der Nachbarschaft hängt von der Komplexität der Eingangsdaten und dem angestrebten Ergebnis ab und variiert je nach Einsatzgebiet und der verfügbaren Leistung des bildverarbeitenden Systems.

Grundsätzlich lässt sich der Connected Component Labeling Algorithmus laut [1] und [11] auf jegliche Datenstrukturen anwenden. So wäre es denkbar, eine Version des CCL-Algorithmus zur Vorbereitung der Merkmalsextraktion aus den rohen Bilddaten einzusetzen und eine abgewandelte Form des CCL-Algorithmus auch in späteren Arbeitsschritten, wie der Klassifikation, zu verwenden.

2.5. Klassifikation

Die Klassifikation in der Bildverarbeitung reiht sich ein in die Methoden der Mustererkennung (vgl. Schritt 4 in Bild 1). Dabei ist das Ziel der Klassifikation, den durch die Merkmalsextraktion gefundenen Objekten Bedeutungen zuzuweisen.

Die Klassifikation umfasst laut [5] zwei grundlegende Aufgaben.

Es müssen Beziehungen zwischen den Bildeigenschaften und den Objektklassen so detailliert wie möglich herausgearbeitet werden.

Es muss der optimale Satz an Bildeigenschaften herausgefunden werden, so dass mit einer möglichst einfachen Klassifizierungsmethode die Objekte in die verschiedenen Klassen eingeteilt werden können.

Dem Klassifikationsverfahren muss bereits vor der Ausführung bekannt sein, welche Bedeutungen denn erkannt werden sollen. Dies kann entweder statisch geschehen, durch eine feste Vorgabe der Merkmale einer Bedeutung oder man die Merkmale auch durch Verfahren des Maschinellen Lernens ermitteln. Tiefere Einblicke in Klassifikationsverfahren können bezogen werden aus den Werken [12], [5] und [2].

3. Connected Component Labeling

3.1. Klassischer Algorithmus

Der Connected Component Labeling Algorithmus wird auf einem binarisierten Bild ausgeführt, das üblicherweise aus einer Schwellwertbildung eines Graustufenbildes gewonnen wurde. Dabei wird das binarisierte Bild pixelweise beginnend von links oben, Zeile für Zeile durchlaufen und seine Vierernachbarschaft betrachtet. Aufgrund der zeilenweisen Betrachtung des Gesamtbildes müssen bloß das nördliche und das westliche Label betrachtet werden, so dass am Ende des Bildes jedes Pixel, das nicht Hintergrund ist, ein Label besitzt.

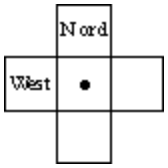


Bild 3 Vierernachbarschaft

Für den Fall, dass sich mehrere Labels berühren, wird mittels des Union-Find-Algorithmus ein passendes Label ausgewählt und zugewiesen. Dazu wurde die in [9] vorgestellte Implementierung dahingehend modifiziert, dass nicht die Grösse der Zusammenhangskomponente (ZHK) als Entscheidung genommen wurde, welche ZHK denn zugewiesen wird, sondern immer die kleinere ZHK ausgewählt wird.

Mit dem Abschluss der Bearbeitung des Bildes wurde jedem Pixel, das nicht als Hintergrund gilt, ein Label vergeben. Dabei können auch komplexe Beziehungen der ZHKs entstehen, wenn verschiedene Regionen zum Beispiel V-förmig zusammenlaufen oder eine Art Schneckenform besitzen. Solche Fälle müssen daher speziell behandelt werden, weshalb der Algorithmus zwei Durchläufe benötigt.

Im ersten Durchlauf werden alle Pixel betrachtet. Wenn ein Pixel kein Hintergrund ist, wird ihm ein bestimmtes Label zugewiesen, das in eine Tabelle eingetragen wird, die den Dimensionen des zu

bearbeitenden Bildes entspricht. In einer zweiten Tabelle werden nebenher die Verkettungen, die aus dem Union-Find Algorithmus entstehen, gespeichert.

```
function find(a) returns int
      result = a
      do {
            -- search parent of a
            result = result.parent
      } while (result != result.parent)
      -- Path compression
      while(a != result)
      {
            help = a.parent
            a.parent = result
            a = help
      }
      return result

function union(a, b) returns int
      aRoot = find(a)
      bRoot = find(b)
      if(aRoot < bRoot)
            bRoot.parent = aRoot
            return aRoot
      else
            aRoot.parent = bRoot
            return bRoot
```

Listing 1 Modifizierter Union-Find Algorithmus

3.2. Single Pass Union-Find Ansatz

Angelehnt an den klassischen Ansatz wurde mit Hilfe von Prof. Dr. Wolfgang Rülling eine modifizierte Variante des CCL Algorithmus entworfen. Diese Variante nutzt gewisse Eigenschaften aus, die sich dadurch ergeben, dass das Bild bloß einmal durchlaufen wird.

- Bei der Abarbeitung des Bildes von oben nach unten entsteht eine Verkettungsstruktur, in der sämtliche Kanten von rechts nach links verlaufen.
- Wegen der Abarbeitung der Zeilen von links nach rechts müssen maximal Ketten der Länge 2 durchlaufen werden.

Der Algorithmus setzt die gleiche Union-Find Implementierung ein, die in Listing 1 vorgestellt wurde. Bei der Ausführung stehen die Bildpunkte bzw. ihre Repräsentanten der "alten Zeile" sowie die der "neuen Zeile" zur Verfügung. Die Zeilen werden von links nach rechts durchlaufen und an jeder Position müssen folgende drei Operationen ausgeführt werden.

1. Der Bildpunkt in der neuen Zeile verweist zunächst auf sich selbst. Ist der linke Nachbar P_{links} gesetzt, soll der aktuelle Bildpunkt P_{neu} auf den Repräsentanten des linken Nachbarn verweisen: *UNION*(P_{links}, P_{neu})
2. In der alten Zeile wird für jeden Bildpunkt P_{alt} die *FIND()* -Operation ausgeführt. Dabei wird der Repräsentant des derzeit bearbeiteten Bildpunkts P_{alt} ermittelt und die Kette in der alten Zeile komprimiert. Hier werden maximal Ketten der Länge 2 durchlaufen und es entstehen Ketten der Länge 1.
3. Wenn der Bildpunkt P_{neu} einen Nachbarn P_{alt} in der darüber liegenden alten Zeile besitzt, muss außerdem eine *UNION* Verknüpfung von P_{alt} und P_{neu} erfolgen, *UNION*(P_{alt}, P_{neu}). Dabei werden nicht die wirklichen Bildpunkte verwendet, sondern ihre Repräsentanten die in den Schritten 1 und 2 ermittelt wurden. Dabei muss beachtet werden:

   Wenn der Repräsentant von P_{alt} noch in der alten Zeile liegt, muss man den Repräsentanten von P_{alt} auf den Repräsentanten von P_{neu} (in die neue Zeile) zeigen lassen.

   Wenn der Repräsentant von P_{alt} bereits in der neuen Zeile liegt, muss man den rechten Repräsentanten auf den linken zeigen lassen.

Außerdem muss beachtet werden

- dass man in der Implementierung erkennen muss, ob der Repräsentant eines Bildpunkts in der alten oder in der neuen Zeile liegt.
- dass man, sobald die neue Zeile vollständig durchlaufen wurde, in der alten Zeile nach existenten Repräsentanten suchen muss, die in die alte Zeile zeigen. Diese Komponenten können ausgegeben werden, da sie ihren Repräsentanten in der alten Zeile haben und somit als abgeschlossen gelten.

3.3. Schneller Ansatz

Der derzeit implementierte Algorithmus ist eine Modifikation des klassischen CCL-Algorithmus. Das Hauptaugenmerk beim Entwurf dieses Algorithmus lag darauf in konstanter Laufzeit eine Extraktion der gewünschten Merkmale zu erreichen. Dieser Algorithmus wurde exakt beschrieben und in *VHDL* implementiert. Er arbeitet wie der Algorithmus aus Kap. 3.1 in einer Vierernachbarschaft. Dieser Ansatz setzt im Gegensatz zum Single Pass Union-Find Ansatz aus Kap. 3.2 keine Verkettungen der

Bildpunkte mittels Union-Find ein, sondern erlaubt maximal eine Verkettung einer Komponente pro Zeile. Durch diese Anpassung kann die Implementierung ein konstante Durchlaufzeit und eine hohe Bearbeitungsrate von 1 Bildpunkt pro Takt erreichen.

4. Derzeitige Implementierung

4.1. Überblick

Im Laufe der Arbeiten wurde ein bildverarbeitendes System implementiert, das als Vorverarbeitungsstufe in einem objekterkennenden System dient. Dabei wurden die Punkte 2 bis 4 aus Bild 1 in ein System auf Basis eines Altera FPGA implementiert, s. Kap. 4.2.

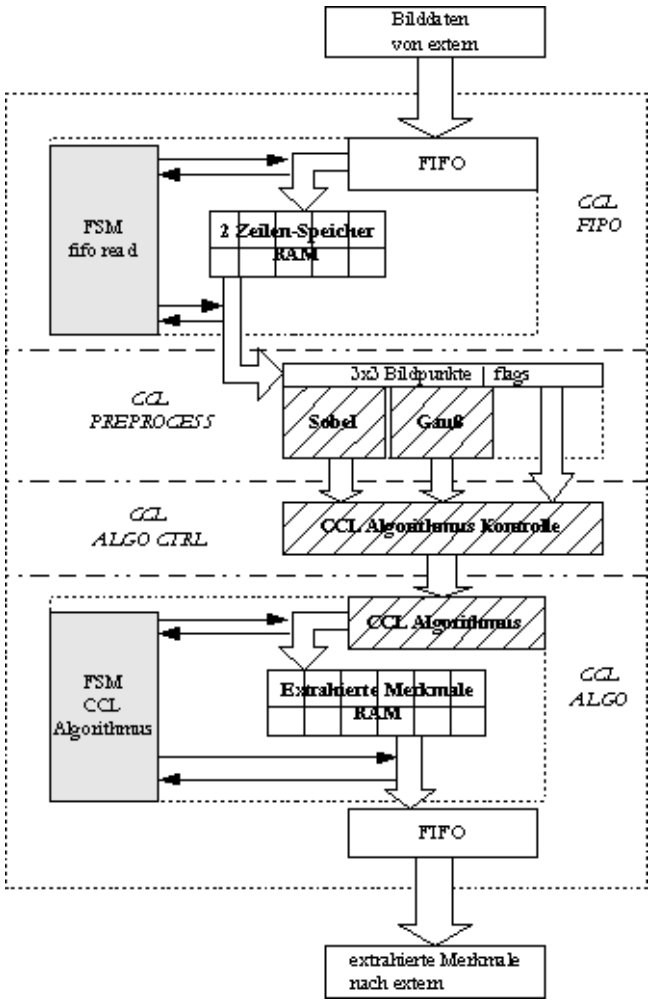


Bild 4 Überblick der implementierten Funktionalität

Die Vorverarbeitung der Bilddaten, vgl. Punkt 2 in Bild 1, wurde in den Modulen *CCL FIPO* und *CCL PREPROCESS*, vgl. Bild 4, implementiert. Das *CCL FIPO* Modul erzeugt eine 3x3 Pixel große Matrix. Diese Matrix wird im *CCL PREPROCESS* Modul zur Vorverarbeitung der Bilddaten, wie in Kap. 2.2 gezeigt, benötigt.

Die Segmentierung der Bilddaten, vgl. Punkt 3 in Bild 1, ist im *CCL ALGO CTRL* Modul implementiert. Dabei wird auf den Tiefpass-gefilterten Originalpixel, wie in Kap. 2.3 gezeigt, eine adaptive Schwellwertoperation angewendet, was zu einer Binarisierung des Eingangsbilds führt.

Dieses binarisierte Bild wird als Eingang zur Merkmalsextraktion im *CCL ALGO* genutzt.

4.2. Implementierungsdetails

Das Gesamtsystem wurde dabei auf folgende Rahmendaten ausgelegt:

- Bildformat: 640x480 Pixel
- Pixeltiefe: 8 Bit
- Max. Labels: 64
- Merkmale:
  - Anzahl Pixel
  - Max.- & Durchschnitts-Grauwert
  - Anzahl Kantenpixel
  - Durchschnitts-Gradientenwert
  - Schwerpunkt
  - Bounding Box

Dabei sind die Parameter „Bildformat“, „Pixeltiefe“ und „maximale Anzahl an Labels“ zur Kompilierzeit einstellbar.

Das System kann selbst für einen Altera Cyclone3 Low-Cost FPGA kompiliert und synthetisiert werden. Eine Begrenzung der verwendbaren Taktfrequenz entsteht nur durch die Grenzen der eingesetzten Hardware. In Tab. 1 wird der Ressourcenverbrauch des implementierten bildverarbeitenden Systems gezeigt.

Tab. 1 Ressourcenverbrauch absolut

Modul	Logic Cells	Dedicated Logic Registers	Block RAMs
FIPO	521	326	2
PREPROC	122	74	0
ALGOCTRL	31	24	0
ALGO	647	239	3

Die Daten aus Tab. 1 beziehen sich auf die verwendete Logik für den CCL Algorithmus und die

Extraktion des Merkmals der Fläche (Anzahl der Bildpunkte) in einer ZHK. Die zwei FIFOs am Modul-Eingang, bzw. Ausgang (vgl. Bild 4) gehen nicht in den gezeigten Ressourcenverbrauch ein. Sie sind jedoch nötig, um eine Integration in das Gesamtsystem zu ermöglichen.

5. Ausblick

Die Implementierung dieses ersten CCL-Blocks diente zum Aufbau der gesamten Infrastruktur. In diesen Rahmen konnte mittlerweile ein erweiterter CCL-Algorithmus [7] integriert werden, der eine größere Anzahl von benachbarten Zellen in die Zusammenhangsbetrachtung einbezieht und auch noch mehr Sonderfälle berücksichtigt.

Über diesen wird gesondert berichtet.

6. Literaturverzeichnis

[1] M. B. Dillencourt, H. Samet, and M. Tamminen. A general approach to connected-component labeling for arbitrary image representations. *J. ACM*, 39 (2): 253–280, 1992. ISSN 0004-5411.

[2] R. C. Gonzalez and R. E. Woods. *Digital Image Processing (3rd Edition)*. Prentice Hall, August 2007. ISBN 013168728X.

[3] C. Harris and M. Stephens. A combined corner and edge detector. In *Proc. 4th Alvey Vision Conference*, pages 147–152, 1988.

[4] I. Jahr. *Lexikon der industriellen Bildverarbeitung*. Spurbuchverlag, 2003. ISBN 3887782879.

[5] B. Jähne. *Digitale Bildverarbeitung (German Edition)*. Springer, 2005. ISBN 3540249990.

[6] D. Lowe. Object recognition from local scale-invariant features. In *ICCV99*, pages 1150–1157, 1999.

[7] N. Ma, D. G. Bailey, and C. T. Johnston. Optimised single pass connected components analysis. In *International Conference on Field-Programmable Technology*, pages 185–192, 2008.

[8] K. Pearson. On lines and planes of closest fit to a system of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, series 6, 2: 559–572, 1901.

[9] W. Rülling. *Effiziente Algorithmen und Datenstrukturen*. Vorlesungsskript an der Hochschule Furtwangen University, 2008.

[10] A. Rosenfeld and J. L. Pfaltz. Sequential operations in digital image processing. *Journal of the ACM*, 1966.

[11] H. Samet and M. Tamminen. Efficient component labeling of images of arbitrary dimension represented by linear bintrees. *IEEE Trans. Pattern Anal. Mach. Intell.*, 10 (4): 579–586, 1988.

[12] R. Schmid. *Industrielle Bildverarbeitung*. Vieweg Verlagsgesellschaft, 2002. ISBN 3528049456.

Hardware-Modell für AES

Stefan Lober, Andreas Siggelkow

Fakultät Elektrotechnik und Informatik

Hochschule Ravensburg-Weingarten, Doggenriedstraße, 88250 Weingarten

stefan.lober@hs-weingarten.de siggelkow@hs-weingarten.de

Der erste Schritt im Systementwurf eines SoC ist immer öfter die Modellierung. Aus diesem Modell heraus kann aufgrund des Timings und der Flächenanordnung entschieden werden, welche Teile des Systems in Hardware und welche in Software realisiert werden können. Das erstellte Modell sollte möglichst schnell sein, am besten Echtzeit-Anforderungen unterstützen.
Der im Modell erstellte Code sollte, ohne re-compiliert werden zu müssen, auf dem Zielsystem lauffähig sein. Nur so können Kunden des SoC schon vor der Verfügbarkeit einer realen Hardware ihre Applikations-Software erstellen und auf der virtuellen Hardware testen.
Vorgestellt wird der erste Schritt, die Modellierung der AES-Verschlüsselung als Software auf einem Standard Virtual Prototype. Dafür wird die CoMET® Entwicklungs-Umgebung der Firma VaST Systems eingesetzt.

1. Motivation

Ein immer größer werdendes Problem beim Entwurf digitaler Chips oder gar digitaler Systeme auf einem Chip (SoC[1]) ist, dass die Entwicklungszeit immer mehr verkürzt werden muss um überhaupt mit dem Produkt Geld zu verdienen (Time to Market). Allerdings werden die Produkte immer komplexer, auch im Consumer-Bereich, und damit einhergehend nimmt die Verzahnung von Software und Hardware zu. Diese Software- / Hardware-Interaktion verlängert allerdings noch einmal die Entwicklungszeit (siehe Abbildung 1). Zunächst muss die Spezifikation für das SoC erstellt werden, dann wird klassischerweise die Hardware entwickelt, die Hardware gefertigt und auf dem fertigen Silizium wird dann erst die Software entwickelt. Daran schließt sich der Systemtest an. Stellt man jetzt fest, dass ein Fehler gemacht wurde, ist die Re-Design-Phase entsprechend lang.

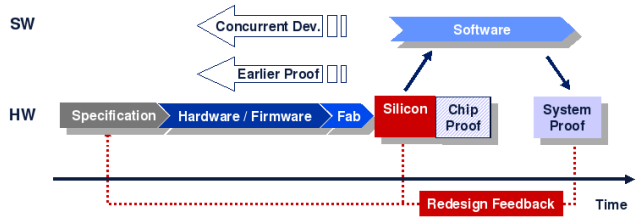


Abbildung 1: Klassische HW- / SW-Entwicklung

Somit liegt nahe, dass man die Software und die Hardware parallel entwickeln möchte um Zeit zu sparen und um die Hardware schon mit den Anforderungen der Ziel-Software testen zu können. Dies ist mit Hilfe eines Virtual Prototype möglich (siehe Abbildung 2). Dieser Virtual Prototype (VP) modelliert die Ziel-Hardware inklusive CPU, und auf diesem Virtual-Prototype kann die Ziel-Software binärkompatibel geschrieben werden. Ist der reale Chip gefertigt, muss nur noch der Binärcode vom VP in den Speicher der Hardware transferiert werden.

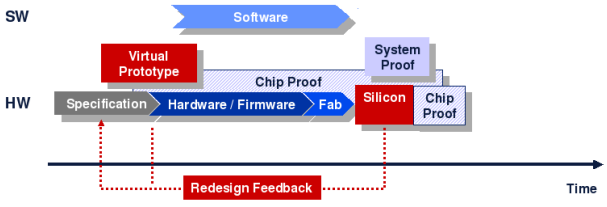


Abbildung 2: HW- / SW-Entwicklung mit Virtual Prototype

Mit diesem Vorgehen hat man mehrere Vorteile: eine verbesserte Time-to-Market, der Kunde des SoC kann frühzeitig (vor Erhalt des Chips) seine Applikations-Software entwickeln und man kann während des Entwurfsprozesses verschiedene Designvarianten ausprobieren (Cachegrößen, Hardware- / Software-Realisierungen alternativ austauschen, CPU Varianten, …).

[1] System on a Chip

2. Gesamtprojekt

Ziel des Projekts ist es, den AES-Algorithmus[2] auf einem SoC zu implementieren. Der erste Teil dieses Projektes ist eine reine Software-Lösung, die auf dem ARM9 des VaST Systems Entwicklungsboards läuft. Das zweite Teil-Projekt wäre, den Algorithmus als Virtual Prototype zu modellieren und in Hardware zu beschreiben.

Anhand der Hardware-Beschreibung und der Software-Implementierung soll die Performance beider Systeme untersucht werden. Mittels diesen Ergebnissen soll entschieden werden, welche Teile des Algorithmus für Hardware und welche für Software geeignet sind. Mit mehreren Performance-Tests soll eine optimale Co-Entwicklung zwischen Hardware und Software für den AES-Algorithmus gefunden werden (siehe Abbildung 3).

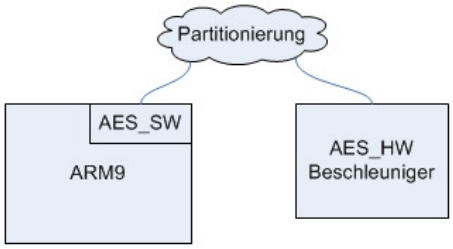


Abbildung 3: HW- / SW-Partitionierung für AES

3. Aufgabenstellung

In dieser Projektarbeit soll der AES-Algorithmus auf der VaST-Plattform als Softwarelösung implementiert werden. Dieses Teilprojekt wird als die Software-Implementierung des Gesamtprojekts, die im Kapitel 2 genannt ist, genutzt.

Es ist völlig ausreichend die erstellte Software auf einem Standard-VP[3] zu erstellen und die Funktionsweise der Hardware und Software in der VaST-Entwicklungsumgebung zu simulieren, da das VaST-Entwicklungsboard noch nicht zur Verfügung steht. Die dabei benötigten Eingaben (Plaintext, Key) und die Ausgabe (Ciphertext) sollen in definierten Speicherbereichen abgelegt werden. Es soll gezeigt werden, dass die Verschlüsselung / Entschlüsselung eines Blockes / States (128 Bit) mit den 3 AES-Varianten AES128, AES192 und AES256[4] korrekt durchgeführt wird, um einen Wert zum Vergleich mit der Hardwarelösung und der Hardware- / Software Co-Design Lösung zu haben.

[2] Advanced Encryption Standard

[3] Virtual Prototype: SW-Simulation eines SoC [1]

[4] Zahl hinter „AES“ gibt die Schlüssellänge an

4. Grundlagen

4.1. VaST-Plattform

Hauptwerkzeug der VaST-Plattform zur Entwicklung dieses Projekts ist die „CoMET® System Engineering Environment“, im Folgenden nur noch „CoMET“ genannt. CoMET wird für die Erstellung von VPs verwendet. Es dient als Umgebung für die parallele Entwicklung der Hardware und Software. Bei Projekten, in denen Hardware und Software entwickelt wird, werden in dieser Umgebung alle Hardware- und Software-Teil-Projekte verwaltet. Die Hardware-Entwicklung erfolgt in einer SystemC-ähnlichen Syntax und die Entwicklung der Software erfolgt in C [1].

4.2. AES-Algorithmus

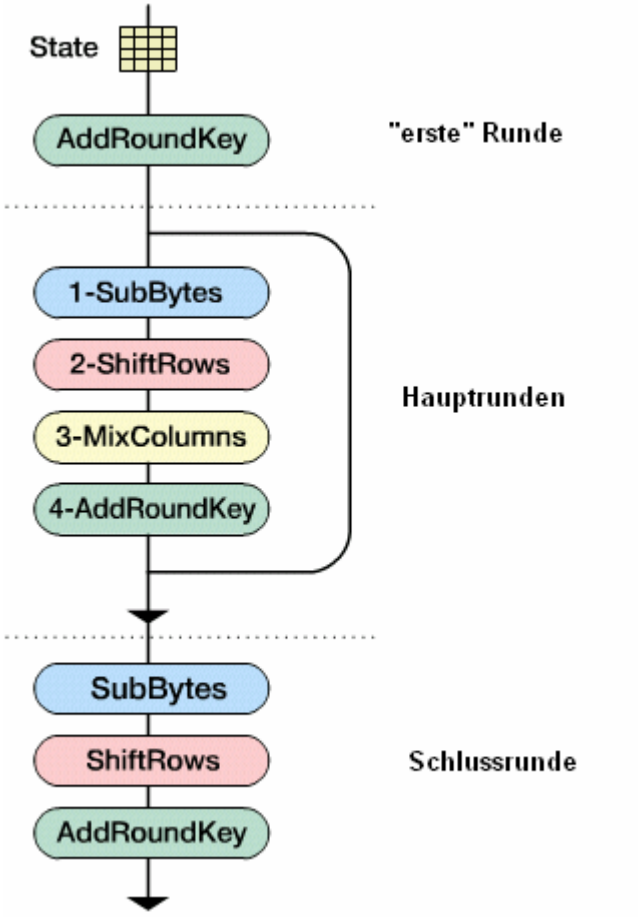


Abbildung 4: Ablauf der AES-Verschlüsselung [2]

Hinter dem Advanced Encryption Standard verbirgt sich der Rijndael-Algorithmus von Joan Daemen und Vincent Rijmen. Rijndael gewann eine öffentliche Ausschreibung des NIST[5], in der ein Nachfolger für

[5] National Institute of Standards and Technology

den nicht mehr sicheren DES-Algorithmus[6], gesucht wurde.

Der AES-Algorithmus ist eine Block-Cipher mit einer Blockgröße von 128 Bit. Es werden Schlüssel-Längen von 128, 192 und 256 Bit unterstützt. Je nach Schlüssellänge durchläuft jeder Block 10, 12 oder 14 Runden. Zuerst wird der Schlüssel von 128, 192 oder 256 Bit, auf die Schlüsselbit-Anzahl aus Formel 1, erweitert. Aus diesem erweiterten Schlüssel werden die so genannten Rundenschlüssel extrahiert.

Pro Runde werden die Funktionen „SubBytes“, „ShiftRows“, „MixColumns“ und „AddRoundKey“ auf 128 Bit große Blöcke (sogenannten „states“) angewandt. Ausnahmen sind die erste Runde, in der nur ein Rundenschlüssel addiert wird, und die letzte Runde, in der die „MixColumns“-Funktion nicht benötigt wird (siehe Abbildung 4).

Die grobe Funktionsweise des AES-Algorithmus kann z.B. unter [3] oder [4] nachgelesen werden. Der genaue Ablauf des Algorithmus, mit mathematischen Hintergrund-Informationen (z.B. Galois-Felder), befindet sich im AES-Standard [5].

5. Hardware

5.1. Vorhandenes

Mit der CoMET Entwicklungs-Umgebung wird ein Tutorial mitgeliefert, welches die Erstellung eines Standard Virtual Prototype erläutert. Dabei wird ein System, bestehend aus einem Virtual System Prototype[7] und einer Virtual Platform[8] erstellt. Dieser zweischichtige Hardware-Entwurf besteht hauptsächlich aus einem ARM9-Prozessor, einem Speicher und dem dazugehörigen Bussystem (siehe Abbildung 5) und wird als Virtual Prototype für dieses Projekt genutzt.

5.2. Speicheraufbau

Es ist ein großer Speicher vorhanden, der aus 3 Teil-Speichern besteht (siehe Tabelle 1, Speicher-Nr. 1 - 3). Die erstellte AES-Hardware-Komponente (siehe Kapitel 5.3) befindet sich ebenfalls in diesem Speicherbereich (Speicher-Nr. 4).

Tabelle 1: Speicheraufbau

Nr.	Inhalt	Größe	Startadr.
1	reset vector [6]	8 MB	0x0
2	code [6]	64 MB	0xA0000000
3	data [6]	48 MB	0xE0000000
4	AES-HW	---	0xF0000000

Die relevanten Ein- / Ausgabe-Daten für Plaintext, Key und Ciphertext werden im Datenspeicher (Speicher-Nr: 3) hinterlegt. Dabei steht jedem Ein- bzw. Ausgabe-Wert 16 MByte (48 MByte / 3) zur Verfügung, was 16.777.216 Ein- bzw. Ausgabe-Zeichen im ASCII-Format[9] entspricht. Die genaue Darstellung des Daten-Speicher-Segments für die Ein- / Ausgabe findet sich in Tabelle 2.

Tabelle 2: Datenspeicher für Plaintext, Key und Ciphertext

Nr.	Inhalt	Größe	Startadr.
1	Plaintext	16 MB	0xE0000000
2	Key	16 MB	0xE1000000
3	Ciphertext	16 MB	0xE2000000

Aus Einheits-Gründen wurden die 3 Bereiche für Plaintext, Key und Ciphertext gleich groß gewählt, obwohl für den Key nur 128, 192 bzw. 256 Bit Speicherplatz benötigt werden. In diesem Projekt wurde mehr Wert auf die Funktionalität des Algorithmus als auf die Sicherheit gelegt. Deshalb wird der Key auch im schreib- / lesbaren Speicher abgelegt. In einem realen Einsatzgebiet darf dieser Schlüssel nicht jedermann zugänglich sein, da er das Geheimnis des AES-Algorithmus darstellt (Kerckhoffs' sches Prinzip).

5.3. AES-Hardware-Komponente

Für den AES-Algorithmus wird eine eigenständige Komponente erstellt, die alle Funktionalitäten abarbeitet, die der Algorithmus bietet. Diese Komponente wird als Peripherie-Baustein auf der Virtual Platform integriert und läuft damit auf dem ARM9-Prozessor (siehe Abbildung 5). Die standardmäßigen Ein- und Ausgänge des Bussystems werden für Plaintext, Key und Ciphertext nicht benutzt. Diese 3 Ein- / Ausgaben werden softwareseitig simuliert. Das heißt, die Eingaben werden nicht an Eingängen eingelesen und an Ausgängen ausgegeben, sondern im Algorithmus hinterlegt. Von

[6] Data Encryption Standard

[7] VSP ist der Top-Level der simulierten Hardware [6]

[8] 2nd Level, enthält z.B. Peripherie-Bausteine [6]

[9] American Standard Code for Information Interchange

dort werden sie in den Speicher geschrieben, der Algorithmus wird ausgeführt und das Ergebnis (Ciphertext) wird wieder in den Speicher geschrieben bzw. auf der Entwicklungskonsole ausgegeben (siehe Kapitel 6.2).

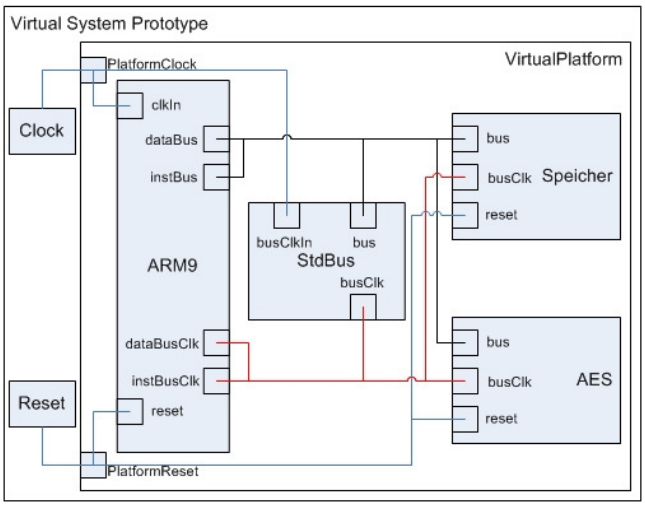


Abbildung 5: Blockdiagramm der Hardware

6. Software

In den folgenden Abschnitten wird der Programmablauf der erstellten Module erklärt.

6.1. Konfiguration

Config

Allgemeine Parameter, die die Konfiguration des Programms betreffen, befinden sich im „*Config*"-Modul. Dort ist die Rundenanzahl des Algorithmus, abhängig von der AES-Version, das irreduzible Polynom, das für die Berechnungen im $GF(2^8)$[10] erforderlich ist und die Anzahl von Bytes, die ein state enthält, hinterlegt. Dies sind nur einige Beispiele der Konfigurationsparameter. Dieses Modul muss bei normaler Anwendung des Algorithmus nicht angepasst werden.

UserInput

Bedingt durch die Simulation der Hardware werden im „*UserInput*"-Modul Änderungen durch den Benutzer erforderlich. So kann hier z.B. die AES-Version (AES128, AES192 oder AES256) gewählt werden und es kann gewählt werden, ob die Daten verschlüsselt oder entschlüsselt werden sollen. In diesem Konfigurationsmodul wird der Plaintext und der Key hinterlegt.

6.2. Programmsteuerung

Hauptkomponente

Zuerst wird der Key, der im UserInput-Modul hinterlegt ist, in den Speicher kopiert, da die zugrunde liegende Hardware simuliert wird. Danach wird überprüft, ob der Schlüssel genau 128, 192 oder 256 Bit (je nach AES-Version) lang ist. Falls diese Voraussetzung nicht erfüllt ist, wird der Algorithmus sofort abgebrochen. Andernfalls wird der Key in die Variable, die für den AES-Schlüssel vorgesehen ist (aesKey), gespeichert. Um den Initialschlüssel von 128, 192 bzw. 256 Bit zu erweitern, um genügend Rundenschlüssel zur Verfügung zu haben, wird die Funktion „*keyExpansion(aesKey)*", mit einer Referenz auf den Schlüsselbytefolge (Key-Schedule) als Übergabeparameter, aufgerufen. Die Funktionalität von „keyExpansion" ist im Kapitel Schlüssel-Erweiterung nachzulesen. Mit der Funktion „*initAddRoundKey(aesKey)*" wird die Key-Schedule dem Modul AddRoundKey zur Verfügung gestellt.

Nachdem der Schlüssel „eingelesen" und vor verarbeitet wurde, wird nun der Plaintext „eingelesen". Der Plaintext wird ebenfalls zuerst vom Modul UserInput in den Speicher kopiert, so lange die Hardware simuliert wird. Der Speicher wird dann immer auf ein Vielfaches von 128 Bit mit einem Platzhalter-Zeichen aufgefüllt, da der AES-Algorithmus blockbasiert verschlüsselt und keine nicht vollständig gefüllten Blöcke verschlüsselt werden können. Dieser evtl. „verlängerte" Plaintext wird dann, in 128-Bit-Blöcken, in die dafür vorgesehene Variable (state) kopiert. Je nachdem welcher Modus gewählt wurde, wird nun entweder die Funktion „*encryption(state)*" oder „*decryption(state)*" aufgerufen, also der 128 Bit state entweder verschlüsselt oder entschlüsselt. Danach wird der nächste 128 Bit Block aus dem Speicher in die state Variable kopiert und dieser dann ebenfalls ver- bzw. entschlüsselt. Dieser Vorgang wiederholt sich so lange, bis die komplette Eingabe-Zeichenfolge Block für Block vom Speicher in die state Variable kopiert und verschlüsselt wurde.

Nach jeder Verschlüsselungs- / Entschlüsselungs-Operation wird der ver- / entschlüsselte state auf der Entwicklungskonsole ausgegeben. Abschließend erfolgt die Ausgabe der gesamten ver- / entschlüsselten Eingabe-Zeichenfolge.

[10] Galois-Feld

Verschlüsselung

Das „*Encryption*“-Modul führt die AES-Verschlüsselung durch. Dabei werden die Verschlüsselungs-Runden (siehe Abbildung 4) abgearbeitet und die Verschlüsselungs-Funktionen aus dem Kapitel 6.3 aufgerufen. Der Ablauf in diesem Modul ist wie im AES-Standard [5] definiert.

In der ersten Verschlüsselungs-Runde wird die Funktion „*addRoundKey(state, „erste“ Runde)*“ aufgerufen. In den Runden 2 bis 9 (AES128), 2 bis 11 (AES192), 2 bis 13 (AES256) werden immer folgende Funktionen, in der gleichen Reihenfolge, aufgerufen (siehe Kapitel 6.3):

- subBytes(state[i][j])
- shiftRows(state)
- mixColumns(state)
- addRoundKey(state, round)

In der finalen Verschlüsselungsrunde werden folgende Funktionen aufgerufen:

- subBytes(state[i][j])
- shiftRows(state)
- addRoundKey(state, “letzte” Runde)

Die Aufrufe der Funktion „*outputEncState(state)*“ im Quellcode dienen der Ausgabe des aktuellen states für Debug-Zwecke. Diese Funktion existiert im Verschlüsselungs- und Entschlüsselungs-Modul.

Entschlüsselung

Im „*Decryption*“-Modul werden die Entschlüsselungs-runden abgearbeitet, wie im AES-Standard [5] definiert. Dabei wird der Verschlüsselungs-Algorithmus (siehe Abbildung 4) in umgekehrter Reihenfolge verarbeitet. In der ersten Runde wird die Funktion „*addRoundKey(state, „letzte“ Runde)*“ aufgerufen. In den Runden 2 bis 9 (AES128), 2 bis 11 (AES192), 2 bis 13 (AES256) werden folgende Funktionen aufgerufen (siehe Kapitel 6.4):

- invShiftRows(state)
- invSubBytes(state[i][j])
- addRoundKey(state, maxRounds - round)
- invMixColumns(state)

In der letzten Entschlüsselungsrunde werden diese Funktionen aufgerufen:

- invShiftRows(state)
- invSubBytes(state[i][j])
- addRoundKey(state, “erste” Runde)

Schlüssel-Erweiterung

Im Modul „*KeyExpansion*“ befinden sich zwei Funktionen. Die Funktion „*initRCon(rCon)*“ wird benötigt, um die rCon[11]-Tabelle, die in der ersten Zeile aus Zweier-Potenzen und sonst aus Nullen besteht, zu erzeugen. Die Anzahl an Spalten der rCon-Tabelle ist gleich der Anzahl an Rundenschlüsseln.

Die Funktion „*keyExpansion(aesKey)*“ dient der Erweiterung des initalen AES-Schlüssels. Dabei wird der initiale Schlüssel auf folgende Länge erweitert (siehe Formel 1):

- 128 Bit (16 Byte) → 1408 Bit (176 Byte)
- 192 Bit (24 Byte) → 2496 Bit (312 Byte)
- 256 Bit (32 Byte) → 3840 Bit (380 Byte)

$$\#keyScheduleBits = \#keyBits * (rounds + 1)$$

Formel 1: Anzahl der Key-Schedule Bits

Die Key-Schedule bezeichnet den gesamten Schlüssel. Abbildung 6 zeigt die Key-Schedule für einen initalen Schlüssel von 128 Bit. Spalten in dieser Key-Schedule werden mit w(i) bezeichnet.

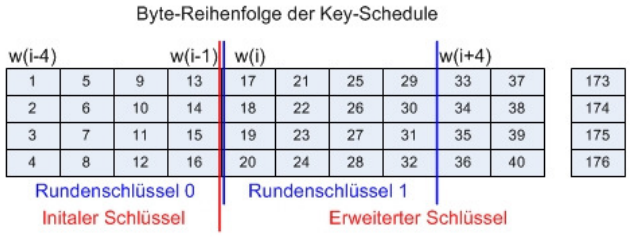


Abbildung 6: Key-Schedule für AES128

Als erstes werden alle bisher nicht benötigten Schlüsselbits auf Null gesetzt, bevor die rCon-Tabelle initialisiert wird. Die Grundoperation ist immer eine XOR-Verknüpfung der vorherigen Spalte (w(i-1)) mit der Spalte an aktueller Position, des vorherigen Rundenschlüssels:

- w(i-4) für AES128
- w(i-6) für AES192
- w(i-8) für AES256

Diese Operation wird auf jedes Byte des erweiterten Schlüssels angewandt. Im Beispiel für AES128:

$$w(i) = w(i-1) \oplus w(i-4)$$

Formel 2: Grundoperation der Key-Expansion

$\oplus$ entspricht dabei der XOR-Verknüpfung.

[11] rCon: Name aus dem AES-Standard [5] entnommen

Es wird nun aufgrund der aktuellen Spalte, beginnend mit der ersten noch nicht erweiterten Spalte (im Beispiel die Bytes 17 - 20 (Spalte w(i))), unterschieden.

Falls der Schlüssel für eine erste Spalte eines Rundenschlüssels (im Beispiel: w(i), w(i+4),...) expandiert wird, muss jedes Byte der Spalte w(i-1) mit dessen entsprechendem Byte der S-Box ersetzt werden (SubByte-Funktion in Kapitel 6.3). Des Weiteren wird jedes Byte der Spalte w(i-1) in der Key-Schedule Tabelle um eine Position nach oben rotiert. Das Byte an oberster Position wird an die niederste Position verschoben. Die Byte-Reihenfolge der Spalte w(i-1) sieht dann, von oben nach unten, wie folgt aus: 14, 15, 16, 13. In diesem Fall wird die Grundoperation aus Formel 2 auf die rotierte und substituierte Spalte w(i-1) angewandt. Zusätzlich zur Grundoperation erfolgt eine XOR-Verknüpfung mit der entsprechenden Spalte der rCon-Tabelle (im Beispiel: der ersten Spalte, da der erste Rundenschlüssel (Rundenschlüssel 1) expandiert wird). Im Beispiel für AES128:

$$w(i) = w(i-1)[rot\ \&\ sub] \oplus w(i-4) \oplus rCon(i/4-1)$$

Formel 3: Operation für erste Spalten der Rundenschlüssel

Falls die Version AES256 gewählt wurde und die aktuelle Spalte w(i) die Fünfte der 8 Spalten des Rundenschlüssels ist, wird jedes Byte der Spalte w(i-1) mit dessen entsprechendem Byte der S-Box ersetzt, bevor die Grundoperation, aus Formel 2 zur Schlüssel-Expansion angewandt wird.

$$w(i) = w(i-1)[sub] \oplus w(i-8)$$

Formel 4: Schlüssel-Erweiterung für AES256

Nach der kompletten Schlüssel-Erweiterung wird, aus Debug-Gründen, die gesamte Key-Schedule auf der Entwicklungskonsole ausgegeben.

6.3. Verschlüsselungs-Funktionen

SubBytes

Mit Hilfe der S-Box, die im Modul „*SubBytes*“ hinterlegt ist, erfolgt eine monoalphabetische Verschlüsselung[12]. Die S-Box ist als 256 Byte großes Array aufgebaut, das zu jedem Element, außer der Null, sein multiplikatives Inverses in GF(2^8) enthält [3].

Der Funktion „*subBytes(byte)*“ wird ein Byte übergeben. Durch Shift-Operationen werden die höherwertigen 4 Bit des übergebenen Bytes als Zeile der S-Box und die niederwertigen 4 Bit als Spalte der S-Box aufgefasst. Das Byte, das sich am Schnittpunkt dieser Zeile und Spalte in der S-Box befindet, wird zurückgegeben (siehe Abbildung 7). Die S-Box wurde aus dem AES-Standard [5], Seite 16 übernommen.

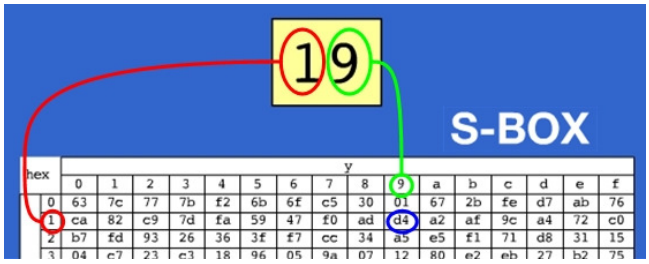


Abbildung 7: Byte-Substitution mittels S-Box

ShiftRows

Ein state ist immer ein 4 * 4 Byte großer Block, auf den Operationen angewendet werden. Die Funktion „*shiftRows(state)*“, im gleichnamigen Modul, rotiert zyklisch und zeilenabhängig die Bytes nach links [3]. Es wird wie folgt zyklisch und byteweise nach links rotiert:

- Zeile 0: wird nicht rotiert
- Zeile 1: um eine Byte-Position
- Zeile 2: um zwei Byte-Positionen
- Zeile 3: um drei Byte-Positionen

Diese Rotation ist in Abbildung 8 grafisch dargestellt.

Abbildung 8: Prinzip von shiftRows [7]

Da ein Verschiebevorgang (Vertauschen) auf ein Array angewandt nicht ohne Zwischenspeichern von Bytes auskommt, werden Bytes in temporären Variablen zwischen gespeichert. Nachdem die verbleibenden Bytes verschoben wurden, wird das zwischengespeicherte Byte an die „freigewordene“ Position kopiert.

MixColumns

In der Funktion „*mixColumns(state)*“ werden nicht, wie der Namen andeutet, Spalten vertauscht, sondern die Spalten eines state werden mit einer 4 x 4 Matrix multipliziert. Diese Matrix ist aus [5], Seite 18.

Die Multiplikation der Elemente erfolgt durch Schiebeoperationen. So ist z.B. die Multiplikation mit 2 eine Verschiebung des Spaltenelements um eine Bitposition nach links. Die Multiplikation mit 3 ist eine Links-Schiebeoperation um eine Bitposition gefolgt

[12] gleiches Plaintext- und Ciphertext-Alphabet

von einer XOR-Verknüpfung mit dem ursprünglichen Spaltenelement (siehe Formel 5).

$$a' = (a << 1) \oplus a$$

Formel 5: Multiplikation mit 3 im GF(2^8)

Die XOR-Verknüpfung kann im GF(2^8) als Addition eingesetzt werden, da dort die beiden Operationen äquivalent sind ([4], Seite 171-172). Diese Funktion ist im Modul „*MixColumns*“ enthalten.

AddRoundKey

Die „*AddRoundKey(state, runde)*“-Funktion ist eine bitweise XOR-Verknüpfung des aktuellen state mit dem aktuellen Rundenschlüssel (siehe [5]). Diese Funktion erhält als Übergabeparameter u.a. die aktuelle Rundennummer. Mittels dieser Nummer können die entsprechenden Bytes des Rundenschlüssels aus der Key-Schedule extrahiert werden. Danach erfolgt die XOR-Verknüpfung des, ebenfalls als Parameter übergebenen, states und des Rundenschlüssels. Diese Funktion wird für die Ver- und Entschlüsselung eingesetzt.

6.4. Entschlüsselungs-Funktionen

Da der AES-Algorithmus komplett umkehrbar ist (siehe [4], Seite 70), befinden sich die Entschlüsselungs-Funktionen, die sich teilweise nur gering von den Verschlüsselungs-Funktionen unterscheiden, in denselben Modulen wie die Verschlüsselungs-Funktionen.

invSubBytes

Die Funktion „*invSubBytes(byte)*“ ist fast genau die gleiche, wie „subBytes“ in Kapitel 6.3, mit der Ausnahme, dass das als Übergabeparameter vorhandene Byte mit dem entsprechenden Byte der inversen S-Box ([5], Seite 22) substituiert wird.

invShiftRows

Die Funktionen „shiftRows“ (Kapitel 6.3) und „*invShiftRows(state)*“ unterscheiden sich nur in der Schieberichtung. Bei „shiftRows“ wird nach links rotiert, die „invShiftRows“-Funktion rotiert die Bytes nach rechts.

invMixColumns

Der Unterschied zwischen „mixColumns“ und „*invMixColumns(state)*“ besteht darin, dass bei „invMixColumns“ die Matrixmultiplikation mit der inversen Matrix der in „mixColumns“ eingesetzten Matrix durchgeführt wird. Diese inverse Matrix befindet sich in [5], Seite 23.

7. Auswertung

7.1. Key-Expansion

Im AES-Standard [5], Seite 27 - 32, befinden sich 3 Beispiele für die Schlüssel-Erweiterung für AES128, AES192 und AES256. Diese 3 Beispiele wurden mit der erstellten Software getestet und mit den Werten aus dem AES-Standard verglichen. Dabei wurden keine Unterschiede festgestellt.

7.2. Verschlüsselung eines state

Auf den Seiten 35 - 46 des AES-Standard [5] befinden sich 3 Verschlüsselungen eines state. Dieser state wird mit AES128, AES192 und AES256 verschlüsselt. Es ist jeder „Zwischenzustand“ des Verschlüsselungs-Vorgangs angegeben.

Der gleiche state, mit gleichem Schlüssel, wurde in der erstellten Software verschlüsselt und mit dem verschlüsselten state des AES-Standard verglichen, wobei keine Unterschiede festgestellt wurden.

7.3. Entschlüsselung eines state

Ebenfalls im AES-Standard [5] auf den Seiten 35 - 46 werden die in Kapitel 7.2 verschlüsselten states wieder Schritt für Schritt entschlüsselt. Diese Entschlüsselung wurde ebenfalls auf der entwickelten Software durchgeführt. Die Entschlüsselungs-Funktion liefert die gleichen Ergebnisse, die im AES-Standard angegeben sind.

7.4. AES-Key kleiner 128, 192 bzw. 256 Bit

Es wurde die Verwendung eines zu langen Schlüssels sowie die Verwendung eines zu kurzen Schlüssels überprüft. Zu lange Schlüssel werden abgeschnitten, da maximal nur 128, 192 oder 256 Bit, je nach verwendeter AES-Version, in den Speicher übernommen werden. Zu kurze Schlüssel wurden anfangs mit Nullen aufgefüllt. Da dies aber die Schlüsselsicherheit beeinträchtigt, muss zwingend ein Schlüssel mit minimaler Länge von 128, 192 oder 256 Bit vorliegen.

7.5. Zeichenfolgen ver- / entschlüsseln

Zur Verschlüsselung von Zeichenfolgen wurde die Eingabezeichenfolge „the first test of the aes_algorithm“ verwendet.

Als Schlüssel für die AES128-Verschlüsselung wurde die Zeichenfolge „mySecret aes_key“, für AES192 „Nobody knows the AES_key“ und für AES 256 die Zeichenfolge „The secret AESkey is 256bit long“ verwendet. Als Platzhalterzeichen der Eingabe wurde

zuerst das ASCII-Zeichen 0x0 gewählt, was aber zu Problemen führte. Mit dem ASCII-Zeichen 0x20 (Leerzeichen), als Platzhalter, konnte die verschlüsselte Zeichenfolge wieder entschlüsselt werden.

Die Ausgabezeichenfolge der AES128-Verschlüsselung wurde mit der Ausgabe der Java-Bibliothek „Java Cryptography Extension [8], die in Java-JRE seit der Version 1.4.2 integriert ist [3], verglichen. Die beiden Ausgaben waren unterschiedlich. Es stellte sich heraus, dass die Java-Bibliothek als Platzhalterzeichen für die Eingabezeichenfolge das ASCII-Zeichen 0x0d (Carriage Return) einsetzt. Nach der Anpassung des Platzhalter-Zeichens in der erstellten Software zu 0x0d stimmten die Ausgaben überein.

Der Vergleich zwischen dem erstellten Programm und der Java-Bibliothek konnte nur für AES128 durchgeführt werden, da AES192 und AES256 von der Java-Bibliothek standardmäßig nicht unterstützt werden [8].

Beim Entschlüsseln der verschlüsselten Zeichenfolge werden mehr Zeichen eingelesen, als die Verschlüsselungs-Funktion ausgibt. Die ursprüngliche Eingabe und die entschlüsselte Zeichenfolge stimmen überein, bis auf 16 Byte (ASCII-Zeichen), die an die entschlüsselte Zeichenfolge angehängt werden. Es konnte bisher noch nicht nachvollzogen werden, warum die Entschlüsselungs-Funktion mehr Zeichen einliest als verschlüsselte Zeichen vorliegen.

8. Fazit / Ausblick

Die Implementierung des AES-Algorithmus in Software auf der VaST-Plattform führte zu keinen größeren Problemen. Die blockweise Verschlüsselung und Entschlüsselung ist problemlos möglich. Bei der Entschlüsselung von Zeichenfolgen werden mehr Zeichen entschlüsselt als verschlüsselt wurden.

Der nächste Schritt zur Entwicklung eines SoC für den AES-Algorithmus ist die Beschreibung des Algorithmus in Hardware.

Durch Performance-Tests des Hardware- / Software-Projekts soll eine optimale Implementierung des AES-Algorithmus auf dem VaST-Entwicklungsboard herausgefunden werden.

Die unter Performance-Aspekten günstigste Lösung (HW, SW oder eine Mischform) soll auf dem SoC integriert werden.

9. Literatur

[1] VaST Systems

http://www.vastsystems.com

[2] Funktionsweise AES

http://www.kubieziel.de/blog/archives/937-Wie-AES-funktioniert.html

[3] Wikipedia AES

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

[4] W. Ertel: *Angewandte Kryptographie*, Hanser München / Wien (2003)

[5] AES Standard

http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf

[6] CoMET Tutorial → CIF Module - Simple VSP → Overview

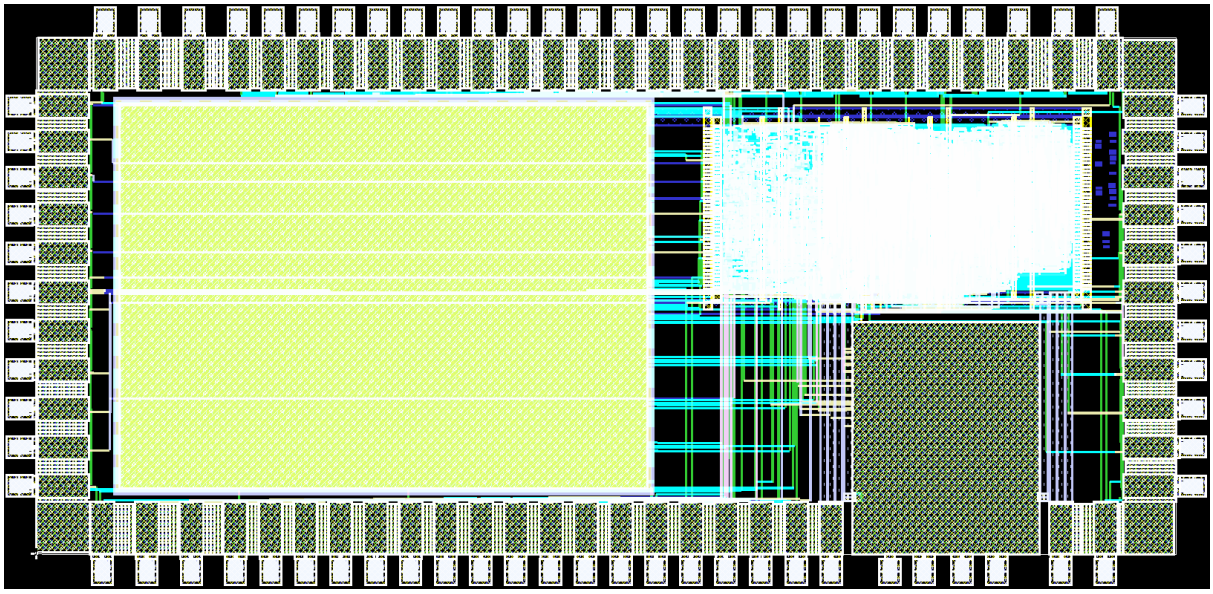
[7] AES on the GPU

http://http.developer.nvidia.com/GPUGems3/gpugems3_ch36.html

[8] Java Cryptography Extension

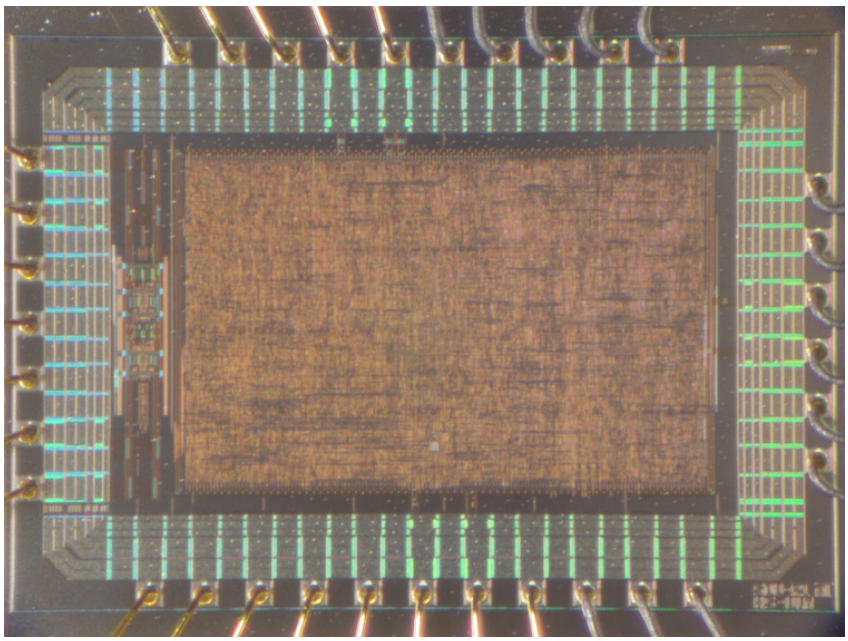
http://java.sun.com/developer/technicalArticles/Security/AES/AES_v1.html

PDA Prozessor IC



Entwurf:	Hochschule Offenburg Bearbeiter: M.Sc. Daniel Bau Betreuer: Prof. Dr.-Ing. Dirk Jansen
Layouterstellung:	Hochschule Offenburg (Standardzellenentwurf & IP-Cores)
Technologie:	UMC 0,18 µm Logic GII-1P6M
Chipfertigung:	Europractice, MPW Run 2261
Herstelldatum:	Juni 2009
Kostenträger:	MPC-Mittel FH-Verbund Baden-Württemberg
Chipdaten:	Chipgröße: 3,16 mm x 1,52 mm Gehäuse: CQFP80
Funktion:	Das PDA Prozessor IC beinhaltet ein Mikrocontrollersystem, wobei der an der Hochschule entwickelte SIRIUS Softcore Prozessor zum Einsatz kommt. Zur Kommunikation mit der Außenwelt sind Peripherien wie SPI-Controller, Timer, Audio-Unit und Interrupt-Controller integriert. Desweiteren befindet sich im ASIC die Ansteuerlogik für einen externen Speicher und ein Display mit Touchcontroller. Ein von dem UMC-Design-Kit bereitgestellter PLL IP-Core generiert den Systemtakt.
Testergebnisse:	Das PDA-Design wurde als VHDL-Entwurf konzipiert und in vorrangegangenen Arbeiten auf einem FPGA emuliert. Zum Einstieg in die 0,18 µm Technologie wurde ein zuerst rein digitaler ASIC verifiziert. Der dazu bereitgestellte Design-Kit wurde entsprechend angepasst. Das Design wurde mit Hilfe der Cadence- und Mentor-Software geroutet und über Europractice gefertigt.

Testchip für einen Sigma-Delta-A/D-Umsetzer



Entwurf:	Hochschule Ulm, Institut für Kommunikationstechnik Bearbeiter: Ante Trutin, Tobias Hartmann, Rudolf Ritter Betreuer: Prof. Dipl.-Phys. Gerhard Forster		
Layouterstellung:	Hochschule Ulm, Labor Mikroelektronik (Mixed Signal-Entwurf) Analogteil: Full Custom Design Digitalteil: Standardzellen-Entwurf		
Technologie:	C35B4C3 0,35 µm CMOS 4 Metal / 2 Poly / HR Poly		
Chipfertigung:	Fa. AMS, Österreich, über Europractice		
Herstelldatum:	III. Quartal 2009		
Kostenträger:	MPC-Gruppe Baden-Württemberg		
Chipdaten:	Chipfläche:	3,00 x 2,10 mm^2	
	Gehäuse:	QFN 48	
	Funktionsblöcke:	Analogteil:	$\Sigma\Delta$-Modulator zweiter Ordnung
		Digitalteil:	Si-Filter dritter Ordnung Kompensationsfilter Tiefpassfilter 20 kHz

Funktion:

Der Testchip enthält einen Sigma-Delta-A/D-Umsetzer für eine Nutzsignalbandbreite von 20 KHz bei 12 bit Auflösung. Die Abtastrate beträgt 20 MHz. Der zweistufige Modulator ist vollsymmetrisch in SC-Technik als Full-Custom-Design ausgeführt. Das dreiteilige digitale Filter wurde auf der Basis einer High-Level-Synthese mit Matlab-Simulink entworfen und von dort automatisch in VHDL-Code umgesetzt. Nach der Schaltungssynthese mit Design Compiler wurde das komplette Mixed-Signal-Back-End mit den Cadence-Tools Encounter und Viruoso XL umgesetzt (siehe Workshopband 41, Seite 11-22). Erste Funktionstests sind erfolgreich verlaufen. Zur Erhöhung des Dynamikbereichs ist ein Redesign des Analogteils in Planung.

MULTI PROJEKT CHIP GRUPPE

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

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

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

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

Hochschule Heilbronn
Prof. Dr. Gessler, (07940) 1306-184
gessler@hs-heilbronn.de

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

Hochschule Konstanz
Prof. Dr. Burmberger, (07531) 206-255
gregor.burmberger@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