MPC

MULTI PROJEKT CHIP GRUPPE
BADEN-WÜRTTEMBERG

Herausgeber: Hochschule Ulm **Ausgabe:** 47 **ISSN** 1868-9221 **Workshop:** Offenburg Februar 2012

Cooperating Organisation
Solid-State Circuit Society Chapter
IEEE German Section

Inhaltsverzeichnis

Tagungsband zum Workshop der Multiprojekt-Chip-Gruppe Baden-Württemberg

Die Deutsche Bibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie.

Herausgeber: Gerhard Forster, Hochschule Ulm, Prittwitzstraße 10, D-89075 Ulm

Diesen Workshopband und alle bisherigen Bände finden Sie im Internet unter:

http://www.mpc.belwue.de

Ein Energy Harvesting IP für den Einsatz in einem 0,18 µm CMOS ASIC

Jens Rechtsteiner, Gerhard Forster

***Zusammenfassung*—Dieser Beitrag befasst sich mit der Entwicklung eines IP-Blocks, der bei monolithischer Integration in ein ASIC dessen Betrieb mit einer ungeregelten Versorgungsspannung ab circa 30 mV ermöglichen soll. Spannungen dieser Größenordnung sind mit Peltier-Elementen zu erzielen, die mit der Temperaturdifferenz zwischen der menschlichen Haut und der Umgebung betrieben werden. Damit sollen verlustleistungsarme ASICs ermöglicht werden, die am menschlichen Körper ohne Batterie funktionsfähig sind. Der IP-Block beinhaltet Komponenten eines Sperrschwingers, eine Spannungsreferenz und diverse Regler. Die zur Verfügung gestellte Spannung beträgt wahlweise 1,8 V oder 3,3 V. Über einen Speicherkondensator kann überschüssige Energie zwischengespeichert werden. Der IP wurde in einer 0,18 µm-CMOS-Technologie (United Microelectronics Corporation, UMC, Taiwan) realisiert. Zur Untersuchung von Prototypen wurde ein Testchip designed.**

***Schlüsselwörter*—Energy Harvesting, CMOS, Full custom design, Schaltregler, Aktiver Gleichrichter, Sperrschwinger, Peltierelement.**

I. Einleitung

Der anhaltende Trend zu integrierten Schaltungen, die immer weniger Energie benötigen (Micropower ICs), ermöglicht zunehmend deren autarken Betrieb ohne die klassische Stromversorgung, z. B. durch Batterien. Dies ist möglich, wenn die zum Betrieb erforderliche Energie der Umwelt entnommen werden kann (Energy Harvesting). Besonders geeignet hierzu ist die Nutzung der folgenden Energieformen [1]-[6]:

- Licht (über Photovoltaik)
- Elektromagnetische Wellen (über Antennen)
- Vibration (über Piezokristalle)
- Temperaturdifferenzen (über Peltierelemente)

Jens Rechtsteiner, rechtsteinerj@gmx.de und Gerhard Forster, forster@hs-ulm.de sind Mitglieder der Hochschule Ulm, Eberhard-Finckh-Straße 11, 89075 Ulm.

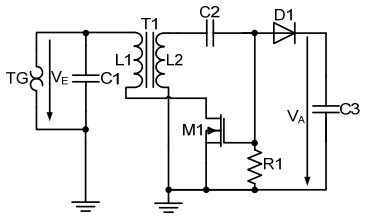


Abbildung 1: Sperrschwinger

Ziel der vorliegenden Arbeit war es, eine Versorgungseinheit für zukünftige Chips zu entwickeln, die die Temperaturdifferenz zwischen der menschlichen Hautoberfläche und der Umgebung nutzbar macht. Dazu sollen Peltierelemente, verfügbar in zahlreichen Ausführungen, eingesetzt werden. Aus Akzeptanzgründen soll das Gesamtsystem möglichst kompakt sein, die Versorgungseinheit also möglichst direkt in den Nutzchip integriert werden. Entsprechend klein sind allerdings die von den Peltier-Elementen zu erwartende Spannung und Leistung. Angesetzt wird ein zu liefernder Laststrom von mindestens 10 µA bei einer für die aktuellen Halbleiterprozesse standardisierten Versorgungsspannung von wahlweise 1,8 V oder 3,3 V.

Im nun folgenden Kapitel 2 wird die vom menschlichen Körper mittels Peltierelement extrahierbare Leistung abgeschätzt. In Kapitel 3 wird, ausgehend von den zu erwartenden Leistungsdaten, der Systemansatz dargestellt und im Kapitel 4 das Schaltungskonzept detailliert ausgearbeitet. Kapitel 5 zeigt Simulationsergebnisse des gesamten Energy Harvesting IPs und Kapitel 6 den Layoutentwurf. Kapitel 7 gibt eine kurze Zusammenfassung und endet mit dem Ausblick.

II. Energiewandlung mit Peltierelementen

Ein Peltierelement setzt sich aus p- und n-dotierten Thermopaaren zusammen [7], welche bei einer Temperaturdifferenz $T_1 - T_2$ entsprechend ihrer Seebeckkoeffizienten α_p und α_n eine Spannung V erzeugen:

$$V = \int_{T_1}^{T_2} \left(\alpha_p(T) - \alpha_n(T)\right) dT \qquad (1)$$

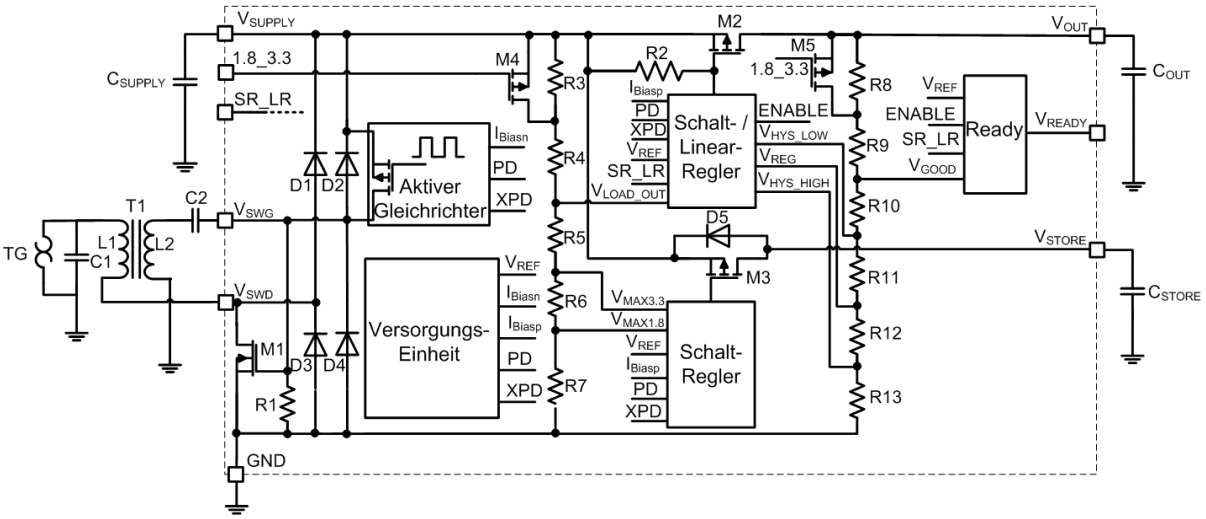


Abbildung 2: Blockschaltbild des IPs

Bei Annahme einer Temperaturdifferenz von 10 K zwischen Hautoberfläche und Umgebung und einer Differenz der Seebeckkoeffizienten von 287 µV/K des Materials Bi2Te3 (Bismuttellurid) wäre damit eine Spannung von 2,87 mV pro Thermopaar zu erwarten. Die gelieferte Leistung des Peltierelements hängt von dessen Wirkungsgrad $\eta_{Peltier}$ und dem Wärmefluss Q ab. Der Wirkungsgrad eines Peltierelements setzt sich aus einem reduzierten Wirkungsgrad η_r und dem Wirkungsgrad eines Carnot-Kreises η_{Carnot} zusammen [1].

$$P = Q \cdot \eta_{Peltier} = Q \cdot \eta_r \cdot \eta_{Carnot} \tag{2}$$

Die Carnot-Effizienz ist ebenfalls abhängig von dem Temperaturunterschied zwischen Umgebungstemperatur T_U und Körpertemperatur T_K.

$$\eta_{Carnot} = \frac{T_K - T_U}{T_K} \tag{3}$$

Der reduzierte Wirkungsgrad eines Peltierelements wird hauptsächlich von seiner figure of merit ZT gebildet, die wiederum abhängig von den Materialeigenschaften des Peltierelements ist [1], [8], [9].

$$\eta_r = \frac{\sqrt{1+ZT}-1}{\sqrt{1+ZT}+T_U/T_K} \tag{4}$$

$$ZT = \frac{\rho \cdot \alpha^2}{\kappa} \cdot \frac{T_K + T_U}{2} \tag{5}$$

Bi2Te3 besitzt einen Seebeck-Koeffizienten α = -287 µV/K, einen spezifischen Widerstand ρ = 1,1×10^5 S·m/m^2 und eine thermische Leitfähigkeit κ = 1,20 W/(m·K) [10]. Daraus resultiert nach (5) eine figure of merit ZT = 2,3 bei Raumtemperatur. Für den Wirkungsgrad des Peltierelements erhält man also $\eta_{Peltier} \approx 0{,}9\,\%$.

Der Mensch ist ein großer Energiespeicher und gibt bei den unterschiedlichsten Aktivitäten Energie ab, die erfasst und verwendet werden kann. Über den Weg der Nahrungsaufnahme gibt der menschliche Körper eine Leistung von 130 W [2] ab. Diese Leistung kann nur zu ca. 25 % genutzt werden und wird zusätzlich von der Carnot-Effizienz begrenzt [1], [2]. Dadurch könnte bei Raumtemperatur mit dem errechneten Wirkungsgrad des Peltierelements eine Leistung von ca. 300 mW von der Hautoberfläche gewonnen werden. In unserer Anwendung begrenzt sich die nutzbare Fläche auf die Größe einer Armbanduhr. Zusätzlich fällt nicht die gesamte Temperaturdifferenz zwischen Körper und Umgebung über dem Peltierelement ab. Das liegt zum einen am Stromfluss durch das Peltierelement, der dieses erwärmt und zum anderen an den thermischen Widerständen, die jeder Thermogenerator aufweist [1]. Das Leistungspeltierelement MPG-D651[1] ist in der Lage, unter günstigen Bedingungen eine Spannung von ca. 37,5 mV bei einem Innenwiderstand von 175 Ω und einer Fläche von 9 mm² zu liefern. Auf der Gehäusefläche einer Armbanduhr, welche ca. 5 cm² aufweist, könnten ca. 50 Elemente verbaut werden, wodurch eine Leistung von ca. 0,4 mW erreichbar ist.

III. Systemansatz

Die vom Peltierelement gelieferte Spannung ist mit ca. 40 mV zu klein, um eine integrierte Schaltung zu betreiben. Damit ein Transistor geschaltet werden kann, muss mindestens seine Schwellenspannung erreicht werden. Zur Erhöhung von Eingangsspannungen sind unterschiedliche Schaltungen bekannt, z.B. Aufwärtswandler oder Sperrwandler [11], [12]. In beiden Konzepten wird ein pulsweitenmoduliertes

[1] http://www.micropelt.com/down/datasheet_mpg_d651_d751.pdf

Ansteuersignal benötigt. Dieses kann aber mit der geringen vom Peltierelement gelieferten Spannung nicht erzeugt werden. Deshalb wird eine Schaltung benötigt, die sich selbstständig mit ihrer Eigenfrequenz aufschwingt und ausreichende Spannungen erreicht, um einen Transistor zu schalten. Dazu eignet sich der Sperrschwinger nach Abbildung 1, beschrieben in [13] und [14]. Die Oszillatorfrequenz f dieses Sperrschwingers wird im Wesentlichen von L_2 und C_2 bestimmt:

$$f = \frac{1}{2\pi\sqrt{L_2 \cdot C_2}} \qquad (6)$$

Die Schwingamplitude hängt ab von der Eingangsspannung V_E, dem Windungsverhältnis des Transformators und von der Ausgangslast. Das Wechselsignal am Ausgang des Sperrschwingers muss in eine Gleichspannung entsprechend der gewünschten Versorgungsspannung für die zu betreibende Nutzschaltung gewandelt werden. Hinzu kommen weitere Features, die ein solcher IP erfüllen sollte. Die Systemanforderungen an den IP lassen sich folgendermaßen zusammenfassen:

- Aus dem Signal des Sperrschwingers soll die Versorgungsspannung V_{SUPPLY} zum Betrieb des IPs erzeugt werden.
- Der an Gleichrichterdioden auftretende Spannungsabfall soll durch den Einsatz aktiver Gleichrichterschaltungen vermieden werden.
- Aus der Versorgungsspannung V_{SUPPLY} soll wahlweise die Ausgangsspannung V_{OUT} = 1,8 V oder 3,3 V erzeugt werden.
- Die Regelung der Ausgangsspannung V_{OUT} soll wahlweise mit oder ohne Hysterese möglich sein.
- Der IP soll beim Hochfahren das Erreichen der gewählten Ausgangsspannung signalisieren.
- Überschüssige Energie soll gespeichert werden können.
- Die Eigenstromaufnahme des IPs soll so klein wie möglich sein.
- Die Zahl der externen Bauelemente soll möglichst klein sein.

IV. Schaltungskonzept

Zur Umsetzung der oben genannten Anforderungen wurde das in Abbildung 2 dargestellte Schaltungskonzept entwickelt. Es beinhaltet einen Sperrschwinger, eine Versorgungseinheit, einen aktiven Gleichrichter, zwei unterschiedliche Regler und eine Einheit zur Erzeugung des Bereitschaftssignals.

Da der IP keinen herkömmlichen V_{DD}-Pin besitzt, muss er seine Versorgungsspannung V_{SUPPLY} selbst erzeugen. Dazu wird das Ausgangssignal des Sperrschwingers V_{SWG} von der Diode D_2 gleichgerichtet und über den Kondensator C_{SUPPLY} geglättet. Die Diode D_4 begrenzt die negative Spannung von V_{SWG} auf -0,7 V.

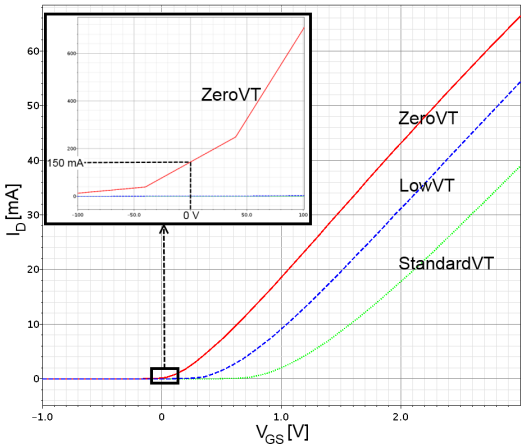


Abbildung 3: NMOS-Transferkennlinien

Die Versorgungseinheit liefert ab einer ausreichenden Spannung V_{SUPPLY} die Biasströme $I_{Biasn,p}$, versetzt den IP über die Signale *PD* und *XPD* in einen definierten Zustand und erzeugt eine Referenzspannung V_{REF}. Durch den aktiven Gleichrichter wird anschließend der Spannungsabfall über D_2 eliminiert. Um V_{SUPPLY} mittels V_{REF} auf unterschiedliche Spannungswerte zu verglichen, beinhaltet der IP zwei Spannungsteiler. Mit dem externen Signal *1.8_3.3* an den Transistoren M_4 und M_5 wird die Ausgangsspannung V_{OUT} auf die standardisierten Versorgungsspannungen der UMC-Technologie von 1,8 V bzw. 3,3 V festgelegt. Der Schalt-/Linearregler erzeugt V_{OUT} über den PMOS-Schalter M_2. Der Widerstand R_2 gewährleistet, dass zu Beginn des Spannungsaufbaus M_2 zunächst sperrt, damit der Aufbau von V_{SUPPLY} nicht verzögert wird. Über das externe Signal *SR_LR* wird bestimmt, ob V_{OUT} analog oder digital geregelt werden soll. Der Schaltregler erzeugt über M_3 eine Spannung V_{STORE} an dem Speicherkondensator C_{STORE}, die über Diode D_5 zurück in den IP gespeist werden kann. Der Ready-Block erzeugt ein digitales Ausgangssignal, das der zu versorgenden Schaltung Betriebsbereitschaft signalisiert.

Nicht integrierbar sind wegen ihrer Größe der Speicherkondensator C_{STORE} (falls erforderlich), die Abblockkondensatoren C_{OUT} und C_{SUPPLY}, sowie der Koppelkondensator C_2 und der Transformator T_1 des Sperrschwingers. Die Funktionalität der Dioden D_1 … D_4 kann mit Hilfe der Schutzdioden der Pins V_{SWG} und V_{SWD} umgesetzt werden. Nachfolgend werden die wichtigsten Blöcke aus Abbildung 2 genauer beschrieben.

A. Sperrschwinger

Zum Anschwingen des Sperrschwingers muss durch die Primärspule von T_1 ein Strom fließen. Dazu muss Transistor T_1 leitend sein, noch bevor sich an seinem Gate eine Spannung aufbauen konnte. Dies ist nur mit einem Depletion Type-Transistor möglich. Depletion

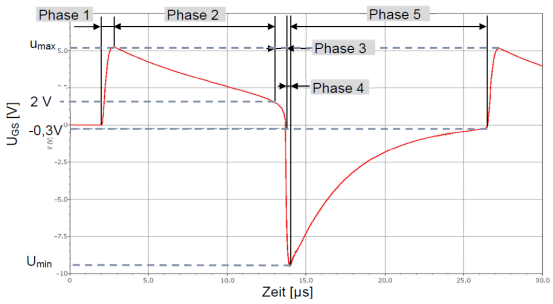


Abbildung 4: Ausgangsspannung des Sperrschwingers

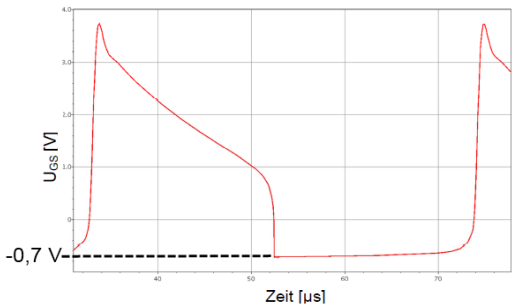


Abbildung 5: Ausgangsspannung V_{SWG} des Sperrschwingers mit Schutzdioden

Type-Transistroen stehen in modernen Halbleiterprozessen aber nicht zur Verfügung. Immer häufiger werden hingegen Enhancement Type-Transistoren mit unterschiedlichen Schwellenspannungen angeboten, um einen günstigeren Trade-off zwischen maximaler Taktfrequenz und Verlustleistung zu ermöglichen. Der Halbleiterprozess von UMC verfügt über einen sogenannten ZeroVT-Transistor. Anhand anfänglicher Simulationen (siehe Abbildung 3) konnte gezeigt werden, dass ein solcher Transistor mit entsprechend großem W/L-Verhältnis bereits bei U_{GS} = 0 V einen ausreichenden Leitwert aufweist, um das Anschwingen des Sperrschwingers zu ermöglichen. Eingebaut in den Sperrschwinger nach Abbildung 1 erhält man den in Abbildung 4 dargestellten zeitlichen Verlauf der Gate-Source-Spannung U_{GS1}.

Die Signalperiode der Oszillation lässt sich in 5 Phasen einteilen: In Phase 1 liefert das Peltierelement eine Spannung, die über M_1 gegen *GND* geschaltet ist. Dadurch wird in L_1 eine Spannung induziert, die mit dem Übersetzungsverhältnis von T_1 auf L_2 transformiert wird und mit C_2 auf das Gate von M_1 gekoppelt wird. Somit steigt die Spannung am Gate von M_1 in Phase 1 an, bis der Durchschaltewiderstand von M_1 sein Minimum erreicht hat und in L_1 keine Spannung mehr induziert wird. In Phase 2 wird die Spannung über dem Widerstand R_1 abgebaut, bis sich der Stromfluss durch M_1 negativ ändert und in Phase 3 zu einem schnelleren Spannungsabfall führt. Wird U_{GS} < -0,3 V, beginnt M_1 zu sperren, was eine abrupte Stromänderung durch M_1 zur Folge hat, woraus in Phase 4 eine große negative Spannungsinduktion am Gate von M_1 resultiert. Durch das Sperren von M_1 wird keine Spannung an L_1 induziert und die Spannung am Gate von M_1 wird in Phase 5 über dem Widerstand R_1 abgebaut. Der Spannungsabbau erfolgt schneller als in Phase 2, da durch das Sperren von M_1 keine Gegeninduktion auftritt. Während L_1 stromlos ist, weil M_1 sperrt, speichert der Kondensator C_1 Energie vom Peltierelement. Dadurch kann im leitenden Zustand der Primärspule zusätzlich Energie von C_1 geliefert werden. Ist die Kapazität C_1 allerdings zu groß, nimmt C_1 auch im leitenden Zustand der Primärspule Energie auf. C_1 muss deshalb in Abhängigkeit von L_1 dimensioniert werden. Bei einer Induktivität L_1 = 7,5 µH erweist sich die Kapazität C_1 = 333 nF als geeignet. Der ZeroVT-Transistor M_1 hat ein W/L-Verhältnis von 3 mm/500 nm. Ein größeres W/L-Verhältnis führt zu keinem besseren Ergebnis, da der Durchschaltewiderstand bereits hinreichend klein ist. Der Kondensator C_2 muss abhängig von der Sekundärspule von T_1 dimensioniert werden. Nach (6) wird die Schwingfrequenz f von C_2 und der Induktivität L_2 der Sekundärspule bestimmt. Sie soll außerhalb des hörbaren Bereichs liegen, also f > 20 kHz betragen. Mit Simulationen hat sich eine Kapazität C_2 = 330 pF bei einer Induktivität L_2 = 75 mH der Sekundärspule als geeignet erwiesen. Die Ausgangsspannung des eigentlichen Sperrschwingers sinkt bis auf V_{SWG} = -10 V, was für den Prozess von UMC unzulässig ist. Zur Begrenzung dienen die Dioden D_3 und D_4, welche den Schutzdioden der Padzellen entsprechen. Abbildung 5 zeigt den Verlauf der Spannung V_{SWG} unter Berücksichtigung der Dioden. Bei zu hohem Energieüberschuss, der nicht eingespeichert werden kann, müsste die positive Amplitude von V_{SWG} extern begrenzt werden, um die maximale Verlustleistung des IPs zu begrenzen. Mit den Schutzdioden D_1 und D_2 wird die vom Sperrschwinger gelieferte Spannung zur Versorgungsspannung V_{SUPPLY} gleichgerichtet und mit C_{SUPPLY} geglättet.

B. Versorgungseinheit

Die Versorgungseinheit setzt sich aus einer Bandgap-Referenz, einer Biasschaltung und einem Power-On-Reset zusammen. Die Bandgap-Referenz erzeugt eine temperaturstabile Spannung, die als Vergleichsspannung für die Komparatoren dient. Die Biasschaltung liefert konstante Biasströme, um die Komparatoren in einem festen Arbeitspunkt zu betreiben. Der Power-On-Reset hat die Aufgabe, den IP in einen definierten Zustand zu versetzen.

1) Bandgap-Referenz

Die Bandgap-Referenz soll eine von der Versorgungsspannung V_{SUPPLY} und Temperatur T unabhängige Spannung V_{REF} = 1 V erzeugen, die bereits bei $V_{SUPPLY} \geq$ 1,5 V geliefert wird. Die Low-Voltage

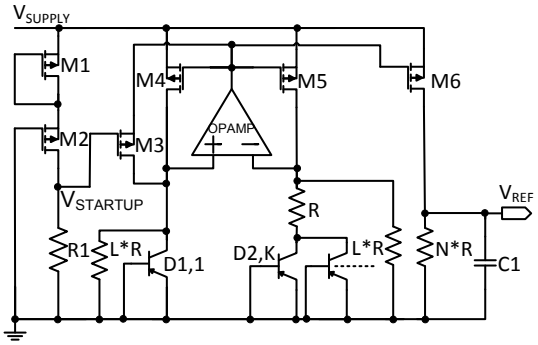


Abbildung 6: Bandgap-Referenz

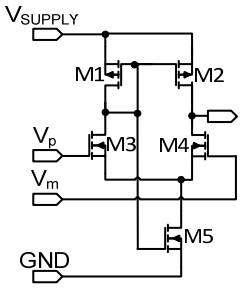


Abbildung 7: Operationsverstärker

Bandgap-Referenz aus Abbildung 6, beschrieben in [15], ist dafür geeignet. Die Referenzspannung wird in dieser Schaltung von der Dimensionierung der Widerstandsverhältnisse N und L, der Temperaturspannung V_T, der Anzahl der Bipolartransistoren K, der Anzahl der Stromspiegelpfade n und dem Spannungsabfall V_{D1} über der Diode D_1 bestimmt:

$$V_{REF} = n \cdot V_T \cdot N \cdot \ln K + \frac{N}{L} \cdot V_{D1} \qquad (7)$$

Die Temperaturabhängigkeit kann durch Nullsetzen der Ableitung eliminiert werden:

$$\frac{\partial V_{REF}}{\partial T} = n \cdot N \cdot \ln K \cdot \frac{\partial V_T}{\partial T} + \frac{N}{L} \cdot \frac{\partial V_{D1}}{\partial T} \overset{!}{=} 0 \qquad (8)$$

Der Operationsverstärker der Spannungsreferenz muss von Anbeginn arbeiten, noch bevor die Biasströme, die zur Versorgung der Komparatoren und übrigen Operationsverstärker dienen, vorhanden sind. Der selbstversorgende OPV aus Abbildung 7 arbeitet ohne externe Biasströme. Dazu wird das Gate von M_5 nicht mit einer Referenzspannung angesteuert, sondern mit den Gates der Transistoren M_1 und M_2 verbunden, welche die Funktion der aktiven Last übernehmen. Daraus resultiert auch eine Stromstabilisierung durch Gleichstromgegenkopplung. Der Operationsverstärker regelt seinen Ausgang so, dass sich gleiche Spannungen über D_1 einerseits und D_2 und R anderseits einstellen. Um den stromlosen Arbeitspunkt der Bandgap-Referenz zu überwinden, wird eine Startup-Schaltung benötigt. Diese setzt sich aus den Transistoren M_1, M_2, M_3 und dem Widerstand R_1 zusammen. Beim Hochlaufen steigt die Ausgangsspannung des Operationsverstärkers mit der Versorgungsspannung V_{SUPPLY} an, wodurch M_3 zunächst leitet und über der Diode D_1 eine Spannung erzeugt. Der Operationsverstärker regelt daraufhin die Spannung über D_2 und R auf den gleichen Wert. Der Stromfluss durch M_4, M_5 und M_6 ist identisch und ergibt über dem Widerstand $N{\cdot}R$ die Referenzspannung V_{REF}. Durch die Startup-Schaltung wird diese zunächst über den Arbeitspunkt V_{REF} = 1 V gehoben. Erreicht V_{SUPPLY} einen Wert von circa 1,6 V, so wird M_2 leitend und erzeugt über dem Widerstand R_1 eine Spannung $V_{STARTUP}$, wodurch M_3 sperrt und die Schaltung in den gewünschten Arbeitspunkt V_{REF} = 1 V zurückfällt. Die mit der Dimensionierung erreichten Ergebnisse der Bandgap-Referenz sind in Tabelle 1 zusammengefasst.

Tabelle 1: Kenndaten Bandgap-Referenz

Min. Versorgungsspannung	$V_{SUPPLY,\min} = 1{,}6$ V
Nominalspannung	$V_{REF} = 1$ V
Absoluttoleranz (Fertigungstoleranz)	$\Delta V_{REF} = 20$ mV
Versorgungsspannungsabhängigkeit	$\Delta V_{REF}/\Delta V_{SUPPLY} = 3$ mV/V
Temperaturabhängigkeit	$\Delta V_{REF}/\Delta T = 0{,}28$ mV/K

2) Bias-Schaltung

Die Bias-Schaltung hat die Aufgabe, die Komparatoren und Operationsverstärker mit konstantem Strom zu versorgen. Dazu werden geeignete Referenzspannungen für die Bias-Transistoren der Komparatoren und Operationsverstärker benötigt. Hierzu wird das Schaltungskonzept nach Abbildung 8a verwendet [15], [16]. Die stromsparende Dimensionierung der Schaltung erfolgte durch Anwendung der folgenden Beziehung:

$$I_{REF} = \frac{2}{R_2^2 \cdot KP_n \cdot \frac{W_7}{L_7}} \cdot \left(1 - \frac{1}{\sqrt{K}}\right)^2 \qquad (9)$$

Mit R_2 = 169 kΩ, W_7/L_7 = 7,5 und K = 4 ergibt sich der Referenzstrom $I_{Biasn,p} = 450$ nA. Um auch in dieser Bias-Schaltung den unerwünschten Arbeitspunkt zu überwinden, wird die gleiche Startup-Schaltung verwendet wie in der Bandgap-Referenz. Der Operationsverstärker hat die Aufgabe, die Abhängigkeit der Biasströme $I_{Biasn,p}$ von der Versorgungsspannung V_{SUPPLY} zu vermindern. Er ist identisch mit dem Operationsverstärker aus Abbildung 7. Die durch Simula-

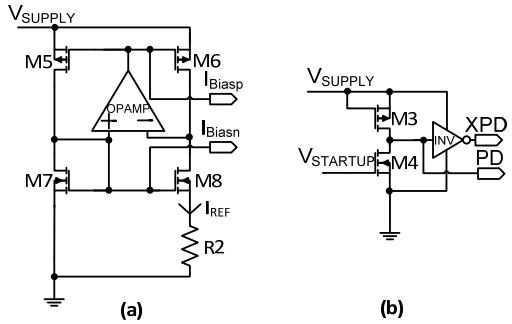


Abbildung 8: (a) Bias-Schaltung, (b) Power-On-Reset

Tabelle 2: Kenndaten der Bias-Schaltung

Min. Versorgungsspannung	$V_{SUPPLY,\min} = 1{,}6$ V
Nominalstrom	$I_{REF} = 450$ nA
Versorgungsspannungsabhängigkeit	$\Delta I_{REF}/\Delta V_{SUPPLY} = 1$ nA/V
Temperaturabhängigkeit	$\Delta I_{REF}/\Delta T = 1{,}1$ nA/K

tion ermittelten Ergebnisse der Bias-Schaltung sind in Tabelle 2 aufgelistet.

3) Power-On-Reset

Die Referenzspannung V_{REF} = 1 V ist erst ab einer Versorgungsspannung $V_{SUPPLY} \geq 1{,}6$ V vorhanden. Das Gleiche gilt im schlechtesten Fall für die Bias-Ströme, wodurch nicht genau bestimmt werden kann, wie sich die Schaltung bei $V_{SUPPLY} < 1{,}6$ V verhält. Es besteht die Gefahr, dass sich in der Schaltung verwendete Flipflops in einem metastabilen Zustand oder Inverter im Umschaltpunkt befinden, was in einem großen Stromfluss resultiert. Durch die nur sehr geringe, vom Peltierelement gelieferte Leistung, kann im schlimmsten Fall der Strom durch eine einzige Inverterstufe den Aufbau der Versorgungsspannung V_{SUPPLY} verhindern, wodurch die gesamte Schaltung funktionsunfähig wäre. Diese Situation soll der Power-On-Reset (Abbildung 8b) mit Hilfe der Signale *PD* und *XPD* vermeiden. M_3 und M_4 bilden einen kapazitiven Teiler, wobei M_3 deutlich größer ist als M_4 und somit auch seine Kapazität. Steigt V_{SUPPLY} an, so steigt auch *PD* an. Das Gate von M_4 ist mit der Startup-Spannung $V_{STARTUP}$ verbunden. Erreicht die Referenzspannung ihren Wert V_{REF} = 1 V, so leitet M4 und schaltet *PD* auf *GND*. *XPD* entspricht dem invertierten Signal von *PD*. Der Power-On-Reset wechselt die Signale *PD* und *XPD* bei einer Versorgungsspannung $V_{SUPPLY} = 1{,}7$ V. In der Simulation aus Abbildung 9 sind alle beschriebenen Ausgangssignale der Versorgungseinheit dargestellt.

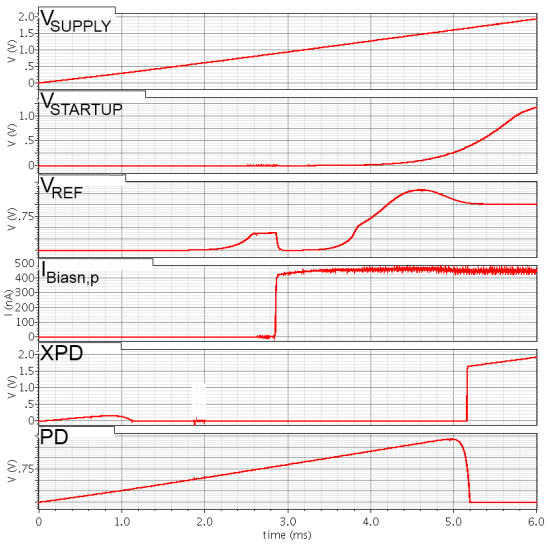


Abbildung 9: Ausgangssignale der Versorgungseinheit

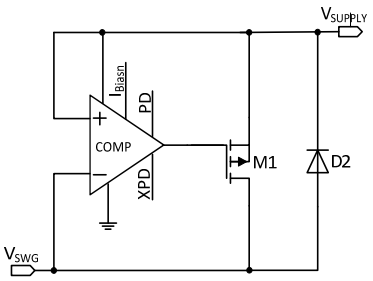


Abbildung 10: Aktiver Gleichrichter

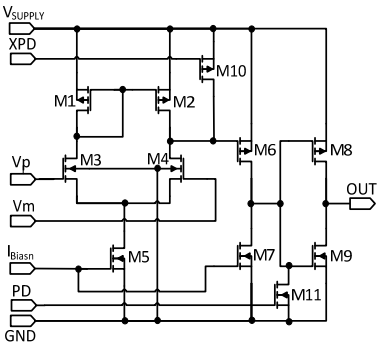


Abbildung 11: Komparator COMP

C. Aktiver Gleichrichter

Der aktive Gleichrichter setzt sich aus einem PMOS-Transistor M_1 und einem Komparator COMP (Abbildung 10) zusammen. Der PMOS-Transistor M_1 überbrückt die Diode D_2, die die Ausgangsspannung des Sperrschwingers V_{SWG} gleichrichtet. Sobald der Komparator mit Strom versorgt wird, vergleicht er die Versorgungsspannung V_{SUPPLY} mit der Ausgangsspannung des Sperrschwingers V_{SWG}. Dazu muss sein Eingangs-Gleichtaktbereich bis an seine eigene Versorgungsspannung heranreichen.

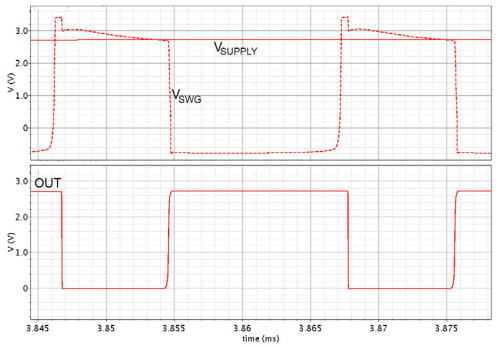


Abbildung 12: Signalverlauf des aktiven Gleichrichters

Tabelle 3: Kenndaten des aktiven Gleichrichters

Stromaufnahme (bei V_{SUPPLY} = 2 V)	I_{SUPPLY} = 2,3 µA
Spannungsverstärkung	V = 48 dB
Einschaltverzögerung	t_{ON} = 0,8 µs
Ausschaltverzögerung	t_{OFF} = 0,5 µs
Durchschaltewiderstand von M_1	R_{ON} = 160 Ω

Diese Aufgabe wird durch den Komparator in Abbildung 11 mit seiner N-Typ-Eingangsstufe erfüllt. Zur Erhöhung des Verstärkungsfaktors enthält er eine Source-Schaltung als zweite Verstärkerstufe, gefolgt von der Inverter-Ausgangsstufe. Damit sich der Komparator bei unzureichender Versorgungsspannung in einem definierten Zustand befindet, besitzt er die Eingänge *PD* und *XPD* für den Power-On-Reset. Somit entspricht die Ausgangsspannung *OUT* des Komparators während des undefinierten Bereichs der Versorgungsspannung V_{SUPPLY}. Nachdem der undefinierte Bereich überwunden ist, wird die Ausgangsspannung *OUT* von den Eingangsspannungen V_p und V_m bestimmt. Wird V_{SUPPLY} an V_p von V_{SWG} an V_m überschritten, liegt die Ausgangsspannung *OUT* des Komparators auf *GND* und der PMOS-Schalter M_1 wird leitend und schließt die Diode D_2 kurz. Ein typischer Signalverlauf ist in Abbildung 12 dargestellt. Beim Zuschalten kommt es zum kurzzeitigen Einbruch der Ausgangspannung des Sperrschwingers V_{SWG}, da zu diesem Zeitpunkt nur der Durchschaltewiderstand von M_1 zwischen den beiden Spannungen liegt. Verzögerungen, insbesondere beim Ausschalten, mindern die Effizienz und müssen minimiert werden. In Tabelle 3 sind die Kenndaten des aktiven Gleichrichters zusammengefasst.

D. Regler für die Ausgangsspannung V_{OUT}

Die Ausgangsspannung V_{OUT} soll wahlweise, je nach Anwendung, digital oder analog geregelt werden können. Für die digitale Regelung wird ein Schaltregler verwendet und für die analoge Regelung ein Linearregler.

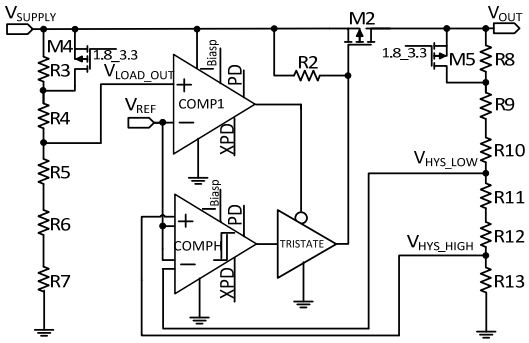


Abbildung 13: Schaltregler für die Ausgangsspannung

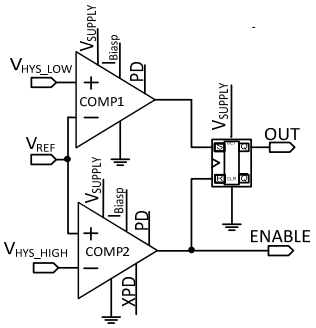


Abbildung 14: Komparator mit Hysterese

Tabelle 4: Kenndaten des Schaltreglers

Nominalschwelle	$V_{OUT,min}$ = 1,81/3,31 V $V_{OUT,max}$ = 1,83/3,33 V
Zu erwartende Toleranz	ΔV_{OUT} < 30 mV

1) Schaltregler

Der Schaltregler setzt sich aus einem Komparator $COMP_1$ und einem Komparator mit Hysterese $COMP_H$ zusammen (Abbildung 13). $COMP_1$ unterscheidet sich von *COMP* (Abbildung 11) lediglich darin, dass er eine PMOS-Differenzstufe enthält. Er dient dazu, den Schaltregler über den Tri-State-Buffer erst dann zu aktivieren, wenn V_{SUPPLY} eine Spannung von 2 bzw. 3,5 V erreicht hat, um V_{OUT} auf 1,8 bzw. 3,3 V laden zu können. Damit soll das schnelle Hochfahren von V_{SUPPLY} sicher gestellt werden. Hat V_{SUPPLY} den erforderlichen Wert erreicht, regelt der Komparator mit Hysterese $COMP_H$ die Ausgangsspannung V_{OUT} auf den eingestellten Wert. Mit der Hysterese wird die Zahl der Schaltvorgänge und somit der Stromverbrauch reduziert.

Der klassische Ansatz der Hysteresebildung über Mitkoppelwiderstände ist hier nicht möglich, weil diese sehr hochohmig sein müssten und mit ihrer Rückwirkung den Spannungsteiler R_8 … R_{13} belasten würden. Deshalb wird die Hysterese, wie in Abbildung 14 gezeigt mit zwei Komparatoren ($COMP_1$, $COMP_2$) und einem Latch erzeugt. Dabei

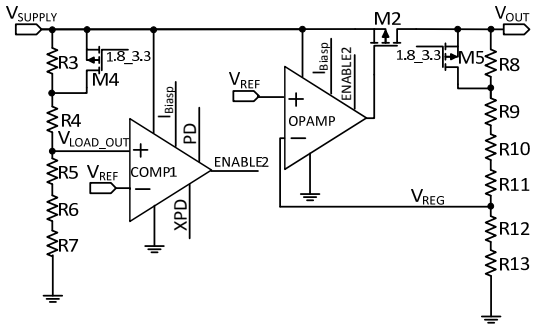


Abbildung 15: Linearregler für die Ausgangsspannung

Tabelle 5: Kenndaten Linearregler

Nominalspannung	V_{OUT} = 1,82 V
Absoluttoleranz	ΔV_{OUT} = 2 mV
Zeitkonstante	τ = 1 ms

wird V_{OUT} von $COMP_1$ auf die minimale und von $COMP_2$ (identisch mit $COMP_1$) auf die maximale Ausgangsspannung V_{OUT} (Tabelle 4) überprüft und der Schaltzustand des Schaltreglers wird durch das Latch bestimmt. Tabelle 4 zeigt die wichtigsten Daten des Schaltreglers.

2) Linearregler

Der Linearregler ist in Abbildung 15 dargestellt. Er enthält einen Operationsverstärker, der die über den Spannungsteiler R_8 … R_{13} heruntergeteilte Ausgangsspannung V_{OUT} mit V_{REF} vergleicht und entsprechend den Transistor M_2 im Triodenbereich betreibt. Auch dieser Regler soll erst arbeiten, wenn V_{SUPPLY} den gewünschten Wert von 2 bzw. 3,5 V erreicht hat. Dazu liefert $COMP_1$ ein $ENABLE_2$-Signal, das den Operationsverstärker sperrt, bis V_{SUPPLY} aufgebaut ist. Die Kenndaten des Linearreglers sind in Tabelle 5 dargestellt.

E. Schaltregler für den Energiespeicher

Der Schaltregler für den Energiespeicher besteht aus zwei Komparatoren $COMP_1$ und $COMP_2$ und einem Schalter *SWITCH* (Abbildung 16). Es werden zwei Komparatoren benötigt, da die zulässige Maximalspannung von 4 V unabhängig von den unterschiedlichen Ausgangsspannungen V_{OUT} (1,8/3,3 V) ist. Jedoch werden abhängig von V_{OUT} die Spannungspunkte des Spannungsteilers verschoben. Dadurch ergeben sich zwei unterschiedliche Spannungsabgriffe am Spannungsteiler für die maximale Versorgungsspannung V_{SUPPLY}. Da hochohmige Spannungsteiler große Flächen benötigen, wird ein zweiter Komparator einem weiteren Spannungsteiler vorgezogen. Die Ausgänge werden je nach Signal *1.8_3.3* auf das Gate von M_3 geschaltet. Ist die maximale Versorgungsspannung

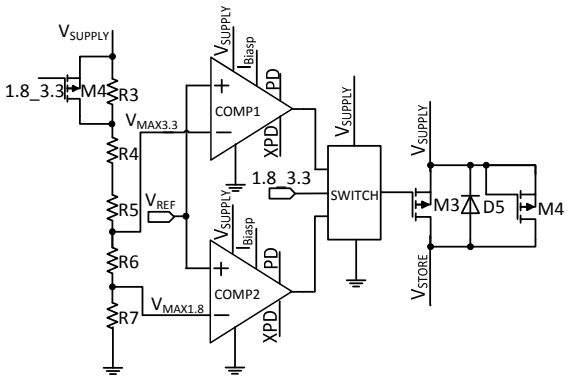


Abbildung 16: Schaltregler zum Laden des Speicherkondensators

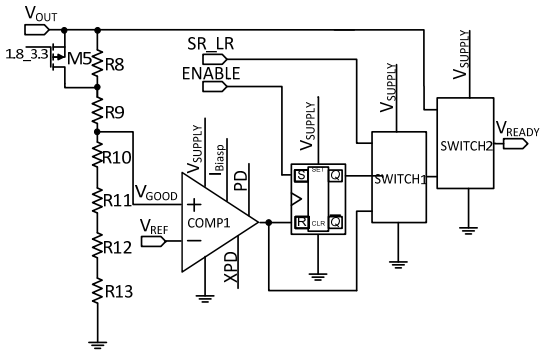


Abbildung 17: Ready-Signal-Erzeugung

V_{SUPPLY} erreicht, kippt der Ausgang des Komparators auf *GND*, wodurch M_3 leitend wird und am Speicherkondensator die Spannung V_{STORE} aufbaut. Sollte V_{SUPPLY} unter V_{STORE} abfallen, fließt über die Diode D_5 Energie von V_{STORE} zurück nach V_{SUPPLY}, aber nur wenn der Spannungsunterschied eine Diodenspannung (ca. 0,7 V) überschreitet. Zur Verringerung dieser Spannung bei kleinen Strömen ist der Diode ein LowVT-Transistor M_4 in Diodenschaltung parallel geschaltet. Er wird bereits ab einem Spannungsunterschied V_{STORE} - V_{SUPPLY} > 0,3 V leitend.

F. Ready-Signal-Erzeugung

Mit dem Signal V_{Ready} (Abbildung 17) wird der zu versorgenden Schaltung signalisiert, dass ihre Versorgungsspannung V_{OUT} den gewünschten Wert erreicht hat. Dafür wird bei Erreichen der Ausgangsspannung V_{OUT} ein Spannungssprung von *GND* auf V_{OUT} erzeugt. Mit diesem Signal kann in der zu versorgenden Schaltung ein Power-On-Reset realisiert werden. Die Schaltung setzt sich aus einem Komparator $COMP_1$, einem Latch und zwei Schaltern $SWITCH_1$ und $SWITCH_2$ zusammen. Die Ausgangsspannung von $COMP_1$ kippt auf V_{SUPPLY}, kurz bevor V_{OUT} erreicht ist. Zusammen mit dem *ENABLE* Signal von $COMP_2$ (Abbildung 14) wird ein Latch angesteuert. Über *SR_LR* wird bestimmt, welcher Eingang von $SWITCH_1$ auf den Eingang von $SWITCH_2$ geschaltet wird. Ist V_{OUT} analog geregelt, wird das Ausgangssig-

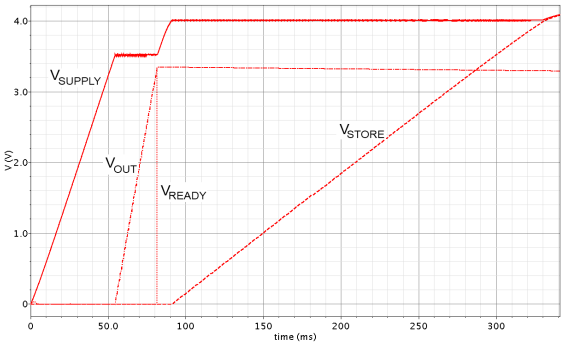


Abbildung 18: Transient-Analyse des Spannungsaufbaus

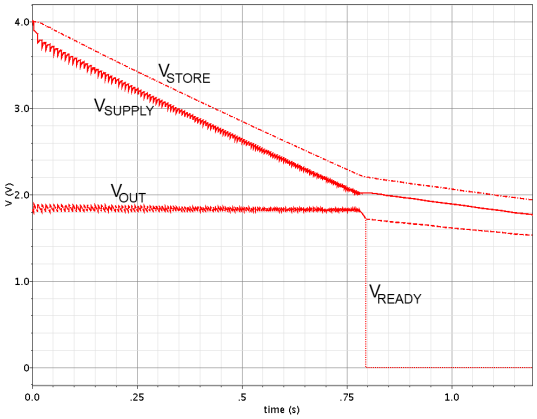


Abbildung 19: Transient-Analyse des Speicherbetriebs

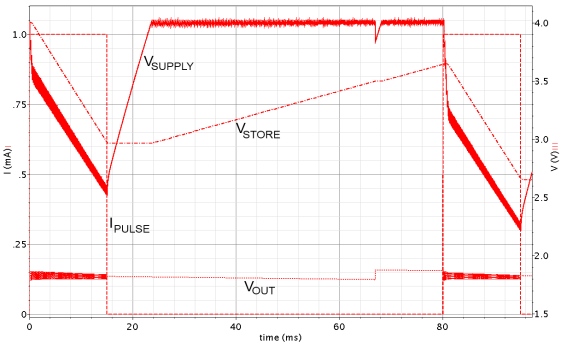


Abbildung 20: Transient-Analyse für den Pulsbetrieb

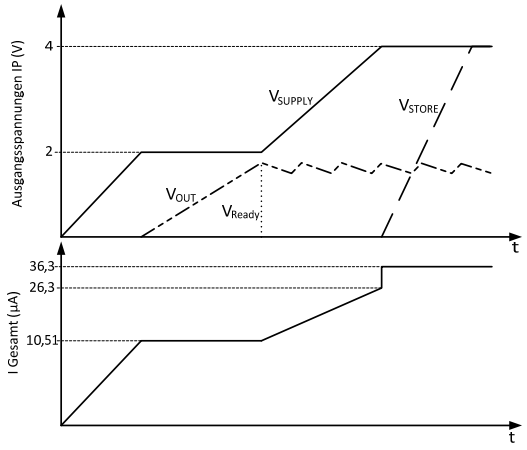


Abbildung 21: Stromaufnahme, abhängig vom Betriebszustand

nal von $COMP_1$ auf den Eingang von $SWITCH_2$ geschaltet, ist V_{OUT} digital geregelt, das Ausgangssignal des Latch. Der Grund für die Erzeugung unterschiedlicher Ready-Signale liegt am Komparator mit Hysterese. Dieser wird verwendet, um die Anzahl der Schaltvorgänge zu reduzieren. Wird der zu versorgenden Schaltung bei der Betriebsart mit dem Schaltregler vor dem Erreichen der gewünschten Ausgangsspannung V_{OUT} Bereitschaft signalisiert, muss von V_{SUPPLY} mehr Energie an V_{OUT} geliefert werden. Das hat einen schnelleren Spannungsabfall von V_{SUPPLY} zur Folge, immer wenn der PMOS-Schalter M_2 (Abbildung 13) leitend wird. Dadurch erhöhen sich auch die Schaltvorgänge von $COMP_1$, was eine höhere Stromaufnahme zur Folge hat. Dies wird durch das Latch vermieden, das erst bei Erreichen von $V_{HYS,HIGH}$ das Bereitschaftssignal erteilt.

V. Gesamtsimulation

Zur Reduktion der Simulationsdauer und geeigneten Signaldarstellung ist bei den Gesamtsimulationen die Beschaltung geeignet gewählt worden. Die Simulation aus Abbildung 18 zeigt den Aufbau der unterschiedlichen Spannungen des IPs mit C_{SUPPLY} = 3 µF, C_{OUT} = 2 µF und C_{STORE} = 6 µF. Das Peltierelement liefert in diesem Beispiel eine Spannung von 300 mV und am Ausgang V_{OUT} fließt ein Laststrom von 30 µA. Zunächst steigt die Versorgungsspannung V_{SUPPLY} an. Hat diese 3,5 V erreicht, wird C_{OUT} bis V_{OUT} = 3,3 V geladen, wodurch V_{READY} einen Spannungssprung von *GND* auf V_{OUT} macht. Anschließend steigt V_{SUPPLY} bis auf eine Spannung von 4 V an. Danach wird C_{STORE} bis auf V_{STORE} = 4 V geladen. Wegen des Energieüberschusses steigen in diesem Fall V_{SUPPLY} und V_{STORE} gemeinsam weiter an, was durch eine externe Beschaltung vermieden werden muss.

Abbildung 19 zeigt den IP im Speicherbetrieb. Die Quellspannung des Peltierelements wird hier entfernt, wodurch der Sperrschwinger kein Signal mehr erzeugt. Die verwendeten Daten sind: C_{SUPPLY} = 2 µF, C_{OUT} = 3 µF, C_{STORE} = 10 µF und Laststrom = 30 µA. V_{SUPPLY} sinkt in diesem Fall bis auf circa V_{STORE} - 0,3 V. Bei dieser Differenz wird der LowVT-Transistor M_4 (Abbildung 16) leitend und speist Energie von C_{STORE} zurück zum IP, der dadurch die Ausgangsspannung V_{OUT}, abhängig von der Dimensionierung von C_{STORE}, für eine bestimmte Zeitspanne aufrechterhält.

Abbildung 20 zeigt das Verhalten des IPs für den Fall, dass am Ausgang V_{OUT} ein kurzzeitig großer Pulsstrom I_{PULSE} = 1 mA entnommen wird bei weiterhin gleichbleibender Energiezufuhr und Beschaltung wie oben. Die Versorgungsspannung V_{SUPPLY} bricht

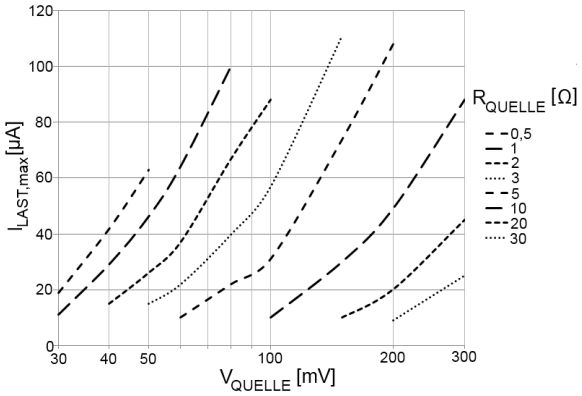


Abbildung 22: Maximal erreichbarer Laststrom

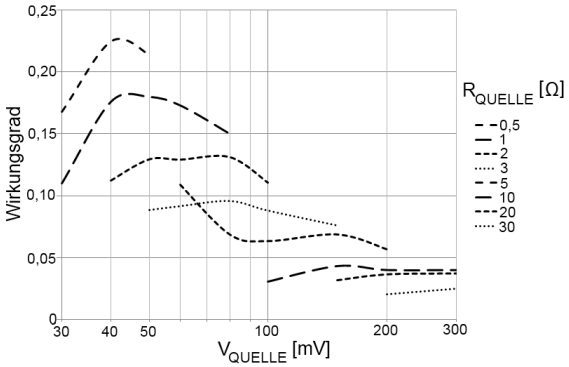


Abbildung 23: Maximal erreichbarer Wirkungsgrad

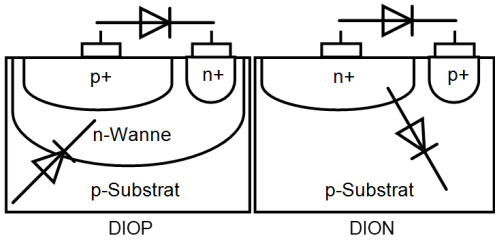


Abbildung 24: Schichtenaufbau zweier pn-Dioden

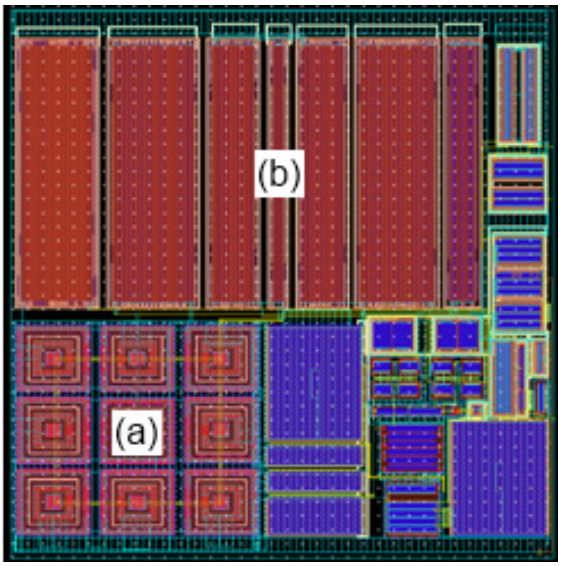


Abbildung 25: Layout der Versorgungseinheit (a) Bipolartransistoren, (b) Widerstände

hierbei ein, so dass ebenfalls Energie aus dem Speicherkondensator bezogen wird. Damit sind Stromimpulse bis zu 1 mA möglich. Die maximale Dauer des Stromimpulses hängt von der Größe des Speicherkondensators ab.

Abbildung 21 zeigt die Stromaufnahme I_{GESAMT} des IPs in den unterschiedlichen Betriebszuständen. Beim Aufbau der Ausgangsspannung V_{OUT} werden 10,5 µA verbraucht und beim Laden des Speicherkondensators 36,3 µA, da der Stromverbrauch der Bauteile von der Versorgungsspannung V_{SUPPLY} abhängig ist.

Abbildung 22 zeigt den maximal lieferbaren Laststrom in Abhängigkeit vom Innenwiderstand R_{QUELLE} und von der Spannung V_{QUELLE} des Peltierelements. Danach kann der IP bei einem Innenwiderstand von 1 Ω und einer Peltierspannung von 30 mV einen maximalen Dauer-Laststrom I_{LAST} von 10 µA liefern. Der Wirkungsgrad beträgt in diesem Fall 11 % (Abbildung 23).

VI. Layout-Entwurf

Das Layout basiert auf einem 0,18 µm Halbleiterprozess von UMC. Der Prozess 1P6M enthält eine Polysiliziumlage und sechs Metalllagen. Zur Erstellung des Layouts stehen die Bauteile, wie beispielsweise die Widerstände, in unterschiedlichen Ausführungen zur Verfügung. Abbildung 24 zeigt den Schichtenaufbau der im UMC Prozess vorhandenen Dioden. Bei der Auswahl der Dioden müssen die möglichen Spannungen an den Anschlüssen beachtet werden, da sich durch die pn-Übergänge zusätzliche Dioden bilden, die ein Fehlverhalten der Schaltung hervorrufen können. Bei der Diode *DION* ist die Anode mit dem p-Substrat verbunden, das geerdet ist. Sie ist deshalb nicht als Gleichrichterdiode D_1 oder D_2 geeignet, bei der sich an der Kathode die Versorgungsspannung aufbauen soll. Für die Dioden D_3 und D_4, deren Anode auf *GND* liegt, kann die Diode dagegen verwendet werden. Bei der Diode *DIOP* sind aufgrund der N-Wanne beide Anschlüsse bei Anwendung positiver Spannungen gegen *GND* isoliert. Dieser Diodentyp ist daher für die Gleichrichterdioden D_1 und D_2 eingesetzt. Möglicherweise wären die Dioden aufgrund der vorhandenen Schutzdioden an den Pads entbehrlich, allerdings standen keine Simulationsmodelle für die Schutzdioden zur Verfügung.

In Abbildung 25 ist das Layout der Versorgungseinheit dargestellt. Da hierbei besonderes Gewicht auf das Matching gelegt werden muss, sind die wichtigen Bauteile verschachtelt platziert. Dazu sind die acht Bipolar-Transistoren $D_{2,8}$ der Bandgap-Referenz (siehe Abbildung 6) um den Transistor D_1 angeordnet (Abbildung 25a). Auch die Widerstände sind ver-

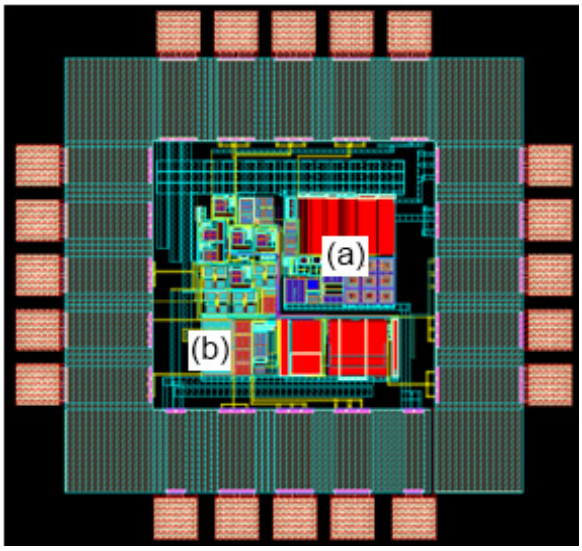


Abbildung 26: Chip-Layout (a) Transistor des Sperrwandlers, (b) Versorgungseinheit

schachtelt angeordnet und weisen alle eine senkrechte Anordnung auf (Abbildung 25b). Durch diese verschachtelte Platzierung sollen die Toleranzen gemittelt und die Spannungsreferenz noch unabhängiger von Temperatur und Versorgungsspannung sein. Die Differenzstufen der Operationsverstärker sind als Kleeblatt-Strukturen ausgeführt. Alle Transistoren sind von Guard-Ringen umgeben, um Übersprechen der Bauteile zu vermeiden.

Abbildung 26 zeigt das Gesamtlayout. Der Transistor des Sperrschwingers (Abbildung 26b) ist mit möglichst großem Abstand von der Versorgungseinheit (Abbildung 26a) platziert, um diese vor Störsignaleinkopplung zu schützen. Zusätzlich besitzt die Versorgungseinheit einen separaten *GND*-Anschluss, der keine Störsignale aufweist.

Der Corebereich des IPs belegt eine Fläche von 330 µm x 330 µm. Zu Testzwecken wurde mit dem Core ein eigener Chip durch Hinzufügen geeigneter E/A-Zellen erstellt. Der Gesamtchip ist Pad-limited mit einer Gesamtfläche von 890 µm x 890 µm. Alle Layoutarbeiten wurden mit Virtuoso XL durchgeführt. Das Design ist mit LVS und DRC geprüft.

VII. Zusammenfassung

Für die autarke Versorgung integrierter Schaltkreise wurde eine Versorgungseinheit entwickelt, die aus der Temperaturdifferenz zwischen Mensch und Umgebung eine für den Betrieb mikroelektronischer Schaltungen geeignete Versorgungsspannung erzeugt. Die Schaltung wurde mit Simulationen verifiziert und in ein Layout überführt. Der damit fertiggestellte ASIC wurde zur Prototypenfertigung freigegeben. Die wichtigsten Ziele wie das Generieren einer ausreichenden Ausgangsspannung aus der sehr kleinen Peltierspannung wurden erreicht und sind in der Tabelle 6 zusammengefasst. Verbesserungspotenzial besteht möglicherweise noch bei der Dimensionierung, speziell des Sperrschwingers, um ein Strom- und Spannungsoptimum zu erreichen.

Tabelle 6: Kenndaten des IPs

Peltierspannung	> 30 mV
Ausgangsspannung	1,8 V / 3,3 V
Regelung	analog oder digital
Max. zulässiger Laststrom	1 mA
Stromaufnahme	min. 10,5 µA max. 36,5 µA
Wirkungsgrad	bis zu 25 %
Energiespeicher	Kondensator
Externe Bauteile	Max. 5 Kondensatoren, Transformator
Chipfläche IP-Core	350 µm x 350 µm
Chipfläche Testchip	890 µm x 890 µm

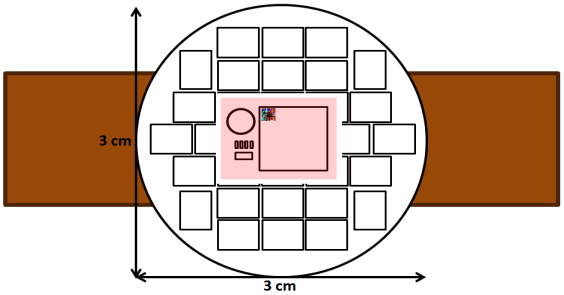


Abbildung 27: Applikation

Nach der Fertigung soll der Chip in eine aufgebaute Messumgebung eingebettet und damit verifiziert werden. Anschließend kann er in zukünftigen Projekten als IP zur Spannungsversorgung verwendet werden. In Abbildung 27 ist eine mögliche Applikation abgebildet, die verdeutlicht, dass das Gesamtsystem trotz externer Bauteile im Bauraum einer Armbanduhr verbaut werden könnte. Für Anwendungen mit höherer Temperaturdifferenz können die externen Bauelemente angepasst werden.

Danksagung

Die Autoren bedanken sich bei Eugen Ringwald und Josef Schäfer für die technische Unterstützung während des gesamten Projekts, bei Michael Päzold, Daniel Moosmann und Manuel Wohlhüter für ihre Arbeit am Simulator sowie bei Avramelos Vasileios für seine Voruntersuchungen an Hardware-Komponenten.

LITERATURVERZEICHNIS

[1] S. Priya, D. J. Inman, „Energy Harvesting Technologies“, *Springer Science+Business Media*, LLC 2009.

[2] T. Starner, J. A. Paradiso, „Energy Scavenging for Mobile and Wireless Electronics”, *IEEE CS and IEEE ComSoc,* 2005.

[3] T. Starner, J. A. Paradiso, „Human Generated Power for Mobile Electronics”, *CRC Press*, 2004.

[4] J. Rabaey, F. Burghardt, D. Steingart, M. Seeman, P. Wright, „Energy Harvesting - A Systems Perspective”, *IEEE International Electron Devices Meeting*, 2007.

[5] P. D. Mitcheson, E. M. Yeatman, G. Kondala Rao, A. S. Holmes, T. C. Green, „Energy Harvesting From Human and Machine Motion for Wireless Electronic Devices”, *Proceedings of the IEE Vol.* 96 *No.* 9, September 2008.

[6] C. Grimm, T. Herndl, T. Kazmierski, C. Links, P. Mitcheson, J. Parker, „Energy Harvesting Systems: Principles, Modelling and Performance Optimisation”, 2009.

[7] S. Lineykin, S. Ben-Yaakov, „Spice compatible equivalent circuit of the energy conversion process in thermoelectric modules”, 23*rd IEEE Israel Convention*, 2004.

[8] I. M. Belov, M. P. Volkov, S. M. Manyakin, „Optimization of Peltier Thermocouple Using Distributed Peltier Effect”, *IEE Eighteenth International Conference on Thermoelectrics,* 1999.

[9] S. Kandasamy, K. Kalantar-zadeh, G. Rosengarten, W. Wlodarski, "Modelling of a Thin Film Thermoelectric Micro-Peltier Module”, *IEEE Region* 10 *Conference*, 2004.

[10] http://en.wikipedia.org/wiki/Bismuth_telluride, April 2012.

[11] M. Wens, M. Steyaert, „Design and Implementation of Fully-Integrated Inductive DC-DC Converters in Standard CMOS”, *Springer Science+Business Media*, B.V. 2011.

[12] U. Tietze, C. Schenk, „Halbleiter-Schaltungstechnik“, *Springer-Verlag Berlin Heidelberg*, 2002.

[13] J. G. Linvill, R. H. Mattson, „Junction Transistor Blocking Oscillators”, *PROC. IRE, vol.* 43*, pp.* 1632-1639; November 1955.

[14] S. M. Korzekwa, „Transistor Blocking Oscillator Analysis”, *IRE Transactions on Circuit Theory,* December 1961.

[15] R. J. Baker, „CMOS Circuit Design, Layout, and Simulation“ *IEEE Press*, Piscataway 2005.

[16] L. Schumm, G. Forster, „Ein CMOS-Leistungsverstärker für niedrige Versorgungsspannungen“, *Workshop der Multiprojekt-Chip-Gruppe Baden-Württemberg*, Juli 2011.

Jens Rechtsteiner ist seit dem Abschluss seines Bachelorstudiums 2010 Masterstudent an der Hochschule Ulm. Er absolviert ein Masterstudium in Systems Engineering and Management, welches er voraussichtlich im Wintersemester 2011 abschließen wird.

Gerhard Forster ist seit 1992 Professor für Elektronik und Mikroelektronische Schaltungen an der Hochschule Ulm. Sein Schwerpunkt liegt auf dem Gebiet des Entwurfs von Mixed-Signal-ASICs. Nach seinem Studium der Physik an der Universität Heidelberg befasste er sich zunächst in einer Forschungsabteilung (heute Daimler Forschungszentrum) mit der Entwicklung und Anwendung neuer Halbleiterprozesse. Er ist Herausgeber des vorliegenden Tagungsbandes.

Hochschule Offenburg
University of Applied Sciences

Integration des Altera Avalon Bus-Systems in einen 32 Bit Softcore mit Harvard-Architektur und Dual Cache

Michael Schmidt, Sebastian Stickel, Florian Zowislok, Dirk Jansen

***Zusammenfassung*—Der 32 Bit Softcore SIRIUS Hulk, entwickelt an der Hochschule Offenburg, verfügt über einen Befehls- und Datencache sowie über eine Harvard-Architektur und ist mit diversen Peripherieeinheiten wie Timer, UART, Interrupt-Controller, ROM, AHI (Acoustic Human Interface), PIO und SPI ausgestattet. Dieses Design des SIRIUS Hulk wurde im Rahmen einer Masterthesis um das prozessorinterne Altera Avalon Bus-System ergänzt, wodurch sich das Gesamtsystem auf elegante Art und Weise mit weiterer Peripherie, sogenannten IP-Cores (Intellectual Property), ergänzen und erweitern lässt. Die generelle Funktion des Altera Avalon Bus-Systems sowie die Implementierung und Integration des Avalon Interfaces in den Softcore werden in dieser Veröffentlichung ebenso behandelt wie die Anbindung von IP-Cores an denselben mit Hilfe von Altera Qsys und die Eigenschaften des Softcores SIRIUS Hulk.**

***Schlüsselwörter*—SIRIUS, Prozessorkern, Prozessor, Softcore, Harvard Architektur, FPGA, Altera, Avalon Bus, Memory Mapped, Cache, Interface.**

I. Einleitung

Als Teil des Instituts für angewandte Forschung (IAF) entwickelt das ASIC Design Center der Hochschule Offenburg seit 1989 anwenderspezifische Schaltungen (ASICs). Ferner wurde vor einigen Jahren mit der Entwicklung eines hauseigenen Prozessorkerns namens SIRIUS für Forschung und Lehre der Studenten begonnen. Dieser verfügte seinerzeit über eine Von-Neumann-Architektur, einen 16 Bit breiten Datenbus sowie über einen beschränkten RISC-Befehlssatz und konnte sämtliche Register-Register Befehle in einem Taktzyklus abarbeiten. Darüber hinaus verfügte er über zwei Betriebsmodi, den 16- und den 32 Bit Modus, die es ihm ermöglichen, optimal auf den Zieleinsatz abgestimmt zu werden. Diese

Michael Schmidt, mschmid7@stud.hs-offenburg.de, Sebastian Stickel, sebastian.stickel@hs-offenburg.de, Florian Zwislok und Dirk Jansen, d.jansen@hs-offenburg.de sind Mitglieder der Hochschule für Technik, Wirtschaft und Medien – Offenburg, University of Applied Sciences, Badstraße 24, 77652 Offenburg.

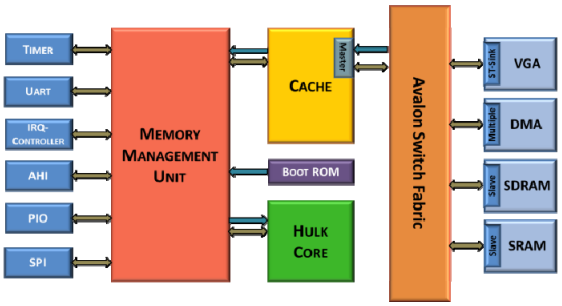


Abbildung 1: Prozessorsystem SIRIUS Hulk mit Altera Avalon Bus System und Peripherie.

Prozessorkernphilosophie wurde als Grundlage für eine ganze Prozessorfamilie verwendet, die nun unter dem Familiennamen SIRIUS weiterentwickelt wird.
Der bisherige Kern SIRIUS wurde in Sirius Janus umbenannt und kommt beispielsweise im hauseigenen PDA in UMC 0.18 µm Technologie zum Einsatz. Als kompaktesten Vertreter gibt es den Sirius Tiny, der über eine v. Neumann-Architektur und einen 16 Bit breiten Datenbus verfügt. Ferner wurde der Befehlssatz weiter eingeschränkt, sodass dieser Prozessorkern auch zur Lehre der Studenten in einem Kurs eingesetzt werden kann, in dem diese den Core selbst in VHDL programmieren und somit erste tiefere Einblicke in moderne Prozessorarchitektur gewinnen können.

II. Sirius Hulk

Als performantester Vertreter tritt der Softcore *Sirius Hulk* auf, da bei ihm auf den bekannten Flaschenhals der v. Neumann-Architektur verzichtet und ihm stattdessen eine Harvard-Architektur zu Grunde gelegt wurde, wodurch er über einen getrennten Daten- und Instructionbus verfügt. Der große Vorteil einer Harvard-Architektur kommt zum Tragen, wenn Daten und Instruktionen gleichzeitig im Speicher angefragt werden müssen. Der Datenbus ist im Gegensatz zu *Tiny* oder *Janus* stets 32 Bit breit, der Befehlsdatenbus dagegen 16 Bit. Dies lässt sich auf den zum Einsatz kommenden Aufbau der Instruktionen zurückführen, die, wie in Abbildung 3 aufgeführt, aus 6 Bit Opcode, 4 Bit Target und 4 Bit Source bestehen. Die verbleibenden zwei Bits sind ungenutzt. Somit verfügt der *Sirius Hulk* über 16 Register wie Instructionpointer-Register (IP), Framepointer-Register (SA), Stackpoin-

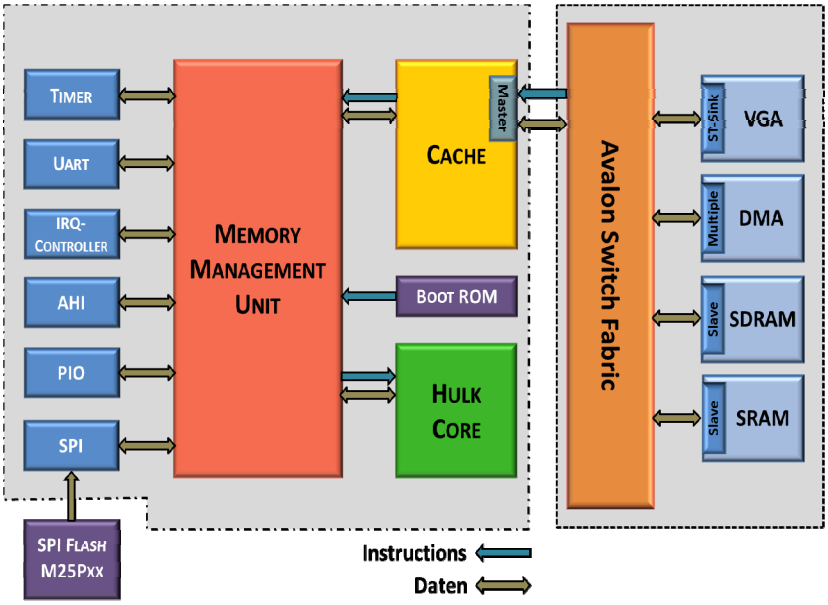


Abbildung 2: Gesamtsystem *Sirius Hulk Avalon*

Abbildung 3: Aufbau der Instructions

ter-Register (SR), Immediate-Register (IM) und zwölf General-Purpose-Register (R0-RB).

Des Weiteren ist er gemäß den 6 Bit Opcode in der Lage, maximal 64 verschiedene Befehle auszuführen. Sowohl der Datenadressbus als auch der Instruction-Adressbus verfügen jeweils über eine Bitbreite von 32, womit sich insgesamt 4 GB Adressraum ansprechen lässt.

Die Codeausführung wurde zur optimalen Prozessorauslastung als dreistufige Fetch-Decode-Execute-Pipeline ausgeführt, in der im Idealfall (1-Cycle Befehle) stets parallel ein Befehl geholt (Fetch), ein Befehl decodiert (Decode) und ein Befehl ausgeführt (Execute) wird. Befehle, die mehr als einen Taktzyklus benötigen, unterbrechen die Pipeline entsprechend. Ausgehend von dieser Strategie wird bei einem Fetch immer der übernächste Befehl angefragt, während der nächste bereits decodiert wird.

III. Cache und Bootvorgang

In diesem Kapitel werden der implementierte Cache und der Ablauf des Bootens näher beleuchtet.

A. Cache

Um die Codeausführung zu beschleunigen, wurde dem *Sirius Hulk* neben den zuvor genannten Eigenschaften auch ein Cache hinzugefügt, sodass auf häufig benutzten Code und Daten schnell und effizient zugegriffen werden kann [1]. Ein Cache beschreibt hierbei einen für den Core transparenten Zwischenspeicher, der die am häufigsten verwendeten Befehle und Daten automatisch vorrätig hält, sodass diese nicht im Arbeitsspeicher angefragt werden müssen. Befehlscache und Datencache wurden beim *Hulk* getrennt ausgeführt und verfügen jeweils über vier satzassoziative Cacheblöcke mit jeweils 4 kB Speicher, sodass dem Softcore insgesamt sowohl 16 kB Daten- als auch Befehlscache zur Verfügung stehen. Die Größe einer Page wurde zu 64 Byte gewählt, die auch stets komplett aus dem Arbeitsspeicher gelesen oder in diesen geschrieben werden. Eine schematische Darstellung der beiden Caches sowie deren Anbindung an Prozessorkern und Arbeitsspeicher ist in Abbildung 4 zu sehen. Sollte der Core Daten oder Befehle am Cache anfragen, die dieser nicht vorrätig hat (Cache-Miss), so wird die Codeausführung durch den Prozessorkern durch ein *Ready*-Signal angehalten, während der Cache die Daten aus dem Arbeitsspeicher

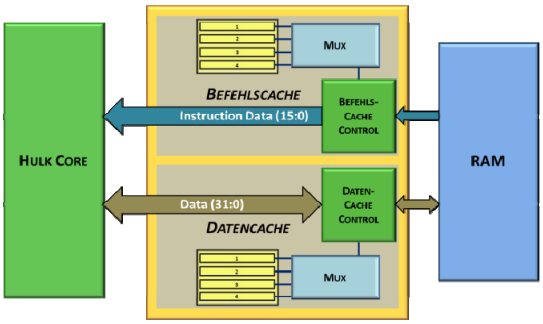


Abbildung 4: Struktur des Cache mit schematischer Anbindung an Core und RAM (ohne MMU)

nachlädt. Analog hierzu wird der Core ebenfalls angehalten, falls ein Schreibvorgang auf den Cache stattfinden soll, dieser jedoch keine freie Page zur Verfügung stellen kann und somit die bis dahin älteste respektive diejenige Page, auf die am längsten kein Zugriff mehr erfolgte, in den Arbeitsspeicher zurück schreiben muss. Damit der Core bei vorliegenden Daten (Cache-Hit) nicht angehalten werden muss, wurde die Taktfrequenz des Cachesystems gerade doppelt so hoch gewählt wie die des Prozessorkerns. Somit hat der Cache gemäß Kapitel II zwei Core-Takte Zeit, die Daten dem Prozessor zur Verfügung zu stellen.

Der Befehlscache hat im Gegensatz zum Datencache nicht die Möglichkeit, schreibend auf den Arbeitsspeicher zuzugreifen, da die Befehlsdaten durch das ausgeführte Programm definiert sind und nicht geändert werden dürfen. Der prinzipielle Signalfluss sowie die beteiligten Komponenten sind in Abbildung 4 dargestellt. Sowohl die Organisation des Caches als auch Zugriffsstrategien werden in [1] und [2] näher beleuchtet.

B. Bootvorgang

Nachdem das System eingeschaltet oder ein Reset durchgeführt wurde, sind alle flüchtigen Speicher des Systems noch unbeschrieben. Der Prozessorkern fragt nun die Adresse 0x7FFFFF00 am Instructionbus an. Auf diese Adresse ist das interne Boot-ROM gemappt (siehe Abbildung 2). In diesem ROM befindet sich ein kompakter Bootloader, der unter anderem das interne SPI-Modul initialisiert, sodass der Core auf den angeschlossenen SPI-Flash zugreifen kann, in dem das eigentliche Programm, das ausgeführt werden soll, abgelegt ist. Der Bootloader kopiert nun bis zu einer bestimmten vorgegebenen Adresse des SPI-Flashs dessen Daten in den Datencache. Ist diese Endadresse erreicht, wird ein sogenannter Cache-Flush ausgeführt, der die nun nicht mehr vorhandene Konsistenz zwischen Arbeitsspeicher und Datencache wieder herstellt. Der Bootvorgang ist ab dieser Stelle abgeschlossen und der Core kann an die Startadresse des Hauptprogramms springen, worauf die Befehle über den Befehlscache pageweise angefragt und in diesem nach und nach abgelegt werden. Der gesamte Bootvorgang sowie die zur Verwendung kommenden Prozessorbefehle werden in [1] und [2] ausführlich beschrieben.

IV. Altera Avalon Bus

Um die Einsatzmöglichkeiten und die Flexibilität des *Hulk* in Bezug auf die Peripherie zu erweitern, wurde beschlossen, das Gesamtsystem an den Altera Avalon Bus anzubinden. Ursprünglich unter anderem für den Altera-eigenen Softcore *Nios* entwickelt, stellt das Altera Avalon Bus-System Verbindungen zwischen den verschiedensten Prozessorkomponenten wie DMA Controller, UART- oder SPI-Peripherie her und verwaltet diese anhand entsprechender Vorgaben. Neben dem Avalon Bus gibt es noch diverse andere SoC-Bussysteme (System-On-Chip) wie beispielsweise AMBA oder Wishbone. Die Wahl fiel schlussendlich auf den Avalon Bus, da bereits viele der bisherigen Komponenten des Instituts für Angewandte Forschung der Hochschule Offenburg auf Altera FPGAs getestet wurden und auch schon diverse *Megafunctions*, dies sind von Altera zur Verfügung gestellte IP-Cores[1], genutzt werden. Der Einsatz eines solchen Bussystems birgt keinerlei Nachteile, weil sich mit ihm völlig neue Möglichkeiten im Hinblick auf die Vielfalt der Prozessor-Interfaces ergeben und die Entwicklungszeit signifikant verkürzt wird, da viele der Schnittstellen als fertige IP-Cores mit Busanbindung zu Verfügung stehen. Darüber hinaus können auch eigene IP-Cores mit Avalon Bus Interface entwickelt und eingesetzt oder bei Bedarf auch Dritten angeboten werden.

Tabelle 1: Altera Avalon Bus Interfaces

Interface	**Verwendung**
Clock	Taktdomänen
Reset	Resetdomänen
Interrupt	Interrupts
Streaming	Dauerhafter Bitstrom
Memory-Mapped	Adressraumbezogene Einteilung der Buskomponenten

In den nachfolgenden Unterkapiteln wird nun näher auf den Altera Avalon Bus, dessen Funktionsweise und Eigenschaften sowie das entwickelte Avalon-Interface für den *Sirius Hulk* eingegangen.

A. Interfaces

Das Altera Avalon Bus-System unterscheidet grundsätzlich zwischen fünf verschiedenen Interfaces, wie sie in Tabelle 1 aufgelistet sind. Das Clock- und das Reset-Interface sind in der Regel mit einem der anderen verknüpft, sodass die jeweiligen Taktdomänen und Flankensensitivitäten der Signale definiert sind. Solch eine Zugehörigkeit ist etwa beim Memory-Mapped-Interface zwingend notwendig. Das Streaming-Interface stellt eine Verbindung zwischen zwei Busteilnehmern mit einem dauerhaften Bitstrom dar (Streaming), wie es zum Beispiel bei manchen VGA

[1] http://www.altera.com/products/ip/altera/mega.html

Tabelle 3: Master-Slave Mapping, Quelle: [2]

Master Address	**32-Bit Master Data**	
	When Accessing a 16-Bit Slave Port	When Accessing a 64-Bit Slave Port
0x00	$OFFSET[1]_{15..0}$:$OFFSET[0]_{15..0}$	$OFFSET[0]_{31..0}$
0x04	$OFFSET[3]_{15..0}$:$OFFSET[2]_{15..0}$	$OFFSET[0]_{63..32}$
0x08	$OFFSET[5]_{15..0}$:$OFFSET[4]_{15..0}$	$OFFSET[1]_{31..0}$
0x0C	$OFFSET[7]_{15..0}$:$OFFSET[6]_{15..0}$	$OFFSET[1]_{63..32}$
…	…	…

IP-Cores verwendet wird. Das Memory-Mapped-Interface beschreibt eine adressbezogene Einteilung der verschiedenen Buskomponenten zueinander. So besitzt jeder Teilnehmer eine Adresse oder einen Adressraum, unter dem er angesprochen werden kann oder für den er zuständig ist.

Im weiteren Verlauf wird auf das Memory Mapped Interface näher eingegangen, da dies jenes Interface darstellt, welches neben dem Clock- und Reset- Interface am häufigsten beim *Sirius Hulk* zum Einsatz kommt. Eine detaillierte Beschreibung aller Interfaces sowie deren Konfigurationsmöglichkeiten findet sich in [3] und [4].

B. Avalon Memory Mapped Interfaces und System Interconnect Fabric

Jeder Busteilnehmer verfügt beim Memory Mapped Interface (im folgenden MMI) über einen bestimmten Adressraum. Unter der theoretischen Voraussetzung, dass dieser 32 Bit groß ist und jeder Teilnehmer nur eine Adresse benötigt, wären somit 4.294.967.296 Buskomponenten möglich.

Allgemein wird bei einem MMI zwischen Master-Interfaces und Slave-Interfaces unterschieden. Der Unterschied besteht darin, dass ein Master stets eine Verbindung initiiert und ein Slave eine solche annimmt. Ein Slave-Device kann also in der Regel keine Verbindung von sich aus zu einem anderen Gerät aufbauen. Eine Ausnahme hiervon bilden bestimmte Slave-Interfaces, die einen Master dazu veranlassen können, eine Übertragung zu beginnen. Diese sind in der Praxis jedoch selten relevant. Die Verbindungsherstellung und die Datenübertragung zwischen den verschiedenen Komponenten ist Aufgabe der *System Interconnect Fabric*, die alle Busteilnehmer zu einem großen Bussystem vereint. Dieser Bus Management Logik, auch *Avalon Switch Fabric* genannt, werden unter anderem folgende Aufgaben zuteil:

Tabelle 2: Typische Signale eines Memory Mapped Master Interfaces

Signaltyp	**Signalname**
Adresse	Address
Daten	Write_Data Read_Data
Kontrollsignal	Byte_Enable Burstcount Waitrequest Write_Request Read_Request Read_Data_Valid IRQ

- Address Decoding
- Datapath Multiplexing
- Wait State Insertion
- Dynamic Bus Sizing
- Native Address Alignment
- Arbitration for Multimaster Systems

Durch sie ist es dem Nutzer beispielsweise möglich, einen 32 Bit Master und einen 16 Bit Slave miteinander zu verbinden. Die Adressübersetzung sowie die Verwaltung der Datenpakete übernimmt die System Interconnect Fabric ohne Zutun des Programmierers gemäß Tabelle 3. Des Weiteren regelt sie die Arbitration in Multimaster Systemen, wie sie bei Systemen mit DMA-Controllern vorkommt. Sollten zwei Master gleichzeitig auf einen Slave zugreifen wollen, so wird dem ersten der beiden anfragenden Master die Verbindungsherstellung erlaubt, während der zweite auf die Beendigung dieser warten muss. Diese Arbitration wird über das Waitrequest-Signal von der System Interconnect Fabric gesteuert. Neben Kontrollsignalen verfügt ein MMI über mindestens ein Adresssignal und, je nach Lese- und/oder Schreibmöglichkeit des Master- respektive des Slave-Devices, über diverse Datensignale. Eine typische Signalkombination eines Master-Interfaces ist in Tabelle 2 aufgeführt.

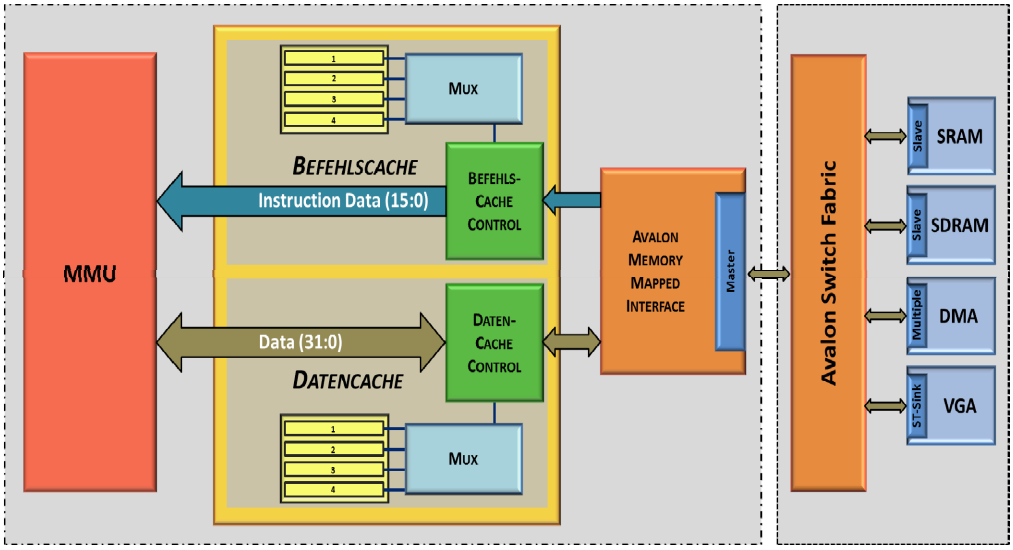


Abbildung 5: Anbindung des Avalon Memory Mapped Master Interface an Cache und Avalon Switch Fabric

Eine wichtige Eigenschaft des Altera Avalon Memory Mapped Interfaces ist die Möglichkeit, eine Lese- oder Schreibanfrage auf ein Slave im sogenannten Burst-Modus auszuführen. Hierbei wird dem beteiligten Slave mit Hilfe des Burstcount Signals mitgeteilt, wie viele Datenworte ab der auf dem Adresssignal anliegenden Adresse gelesen oder geschrieben werden sollen. Initiiert ein Master beispielsweise einen Lesevorgang auf ein Slave ab Adresse 0x410 mit einem Burstcount Wert von 0x10, so wird das Slave-Device 16 Datenworte ab Adresse 0x410 liefern, also die Daten von Adresse 0x410 bis 0x41F. Bursting kann somit dazu beitragen, die Kommunikationszeit zwischen Buskomponenten massiv zu verkürzen, da nicht für jede Adresse eine eigene Anfrage gesendet werden muss. Allerdings muss beachtet werden, dass während eines Burstvorgangs kein anderer Master auf das beteiligte Slave-Device zugreifen kann. Der Wert des Burstcount-Signals hat also direkten Einfluss auf die zuvor erwähnte Waitrequest-Zeitdauer bei Multimaster-Systemen. Ebenso ist es möglich, die Buskommunikation zu pipelinen, was den Busdurchsatz insgesamt vergrößern kann. Weitere spezielle Aufgaben und Eigenschaften der System Interconnect Fabric sowie Signale des MMI werden in [4] aufgeführt und verdeutlicht.

C. Sirius Avalon Memory Mapped Master Interface

Um den Softcore *Sirius Hulk* an das Altera Avalon Bussystem anbinden zu können, musste ein Master Interface entwickelt werden, durch welches der Core Zugriff auf die Busperipherie hat und Lese- und Schreibvorgänge initiieren kann. Die Anbindung des Master Interfaces an den *Hulk* erfolgt über die Daten- und Befehls-Kontrolleinheit des Caches. Somit bildet das Master-Interface die Schnittstelle zwischen *Sirius Hulk* und der *System Interconnect Fabric* mit den Avalon IP-Cores gemäß Abbildung 1 bzw. Abbildung 5. Die zur Verwendung kommenden Bussignale sind:

- ADDRESS (31:0)
- BYTE_ENABLE (3:0)
- BURSTCOUNT (4:0)
- WAITREQUEST_N
- WRITE_REQ
- READ_REQ
- READDATA_VALID
- WRITE_DATA
- READ_DATA
- IRQ (2:0)

Wie an Abbildung 1 und Abbildung 2 zu sehen ist, erfolgt die Unterscheidung zwischen Daten und Befehlen im Master-Interface. Dies geschieht mit Hilfe eines Selektorprozesses, der registriert, von welcher Cache-Kontrolleinheit das Master-Interface angesprochen wird. Ausgehend von dieser Selektion erfolgt ein Mapping von Daten-, Adress-, und Kontrollsignalen auf den Altera Avalon Bus. Zuständig hierfür sind zwei unabhängige State-Machines. Die erste entscheidet in einem Clock-Process, ob es sich um eine Lese- oder Schreibanfrage handelt und wechselt in Abhängigkeit hiervon in einen READ- oder WRITE-State. In einem Output-Process wird ausgehend von diesem State die entsprechende von Datencache- oder Befehlscache-Kontrolleinheit übermittelte Adresse an den Avalon Bus weitergegeben und entweder eine Lese- oder Schreibanfrage eingeleitet.

Die zweite State-Machine ist für die Kommunikation mit dem Cache zuständig und regelt die Adressierung der entsprechenden Cache Zeilen und deren Dateninhalt. Hierfür verfügt sie neben einem IDLE-

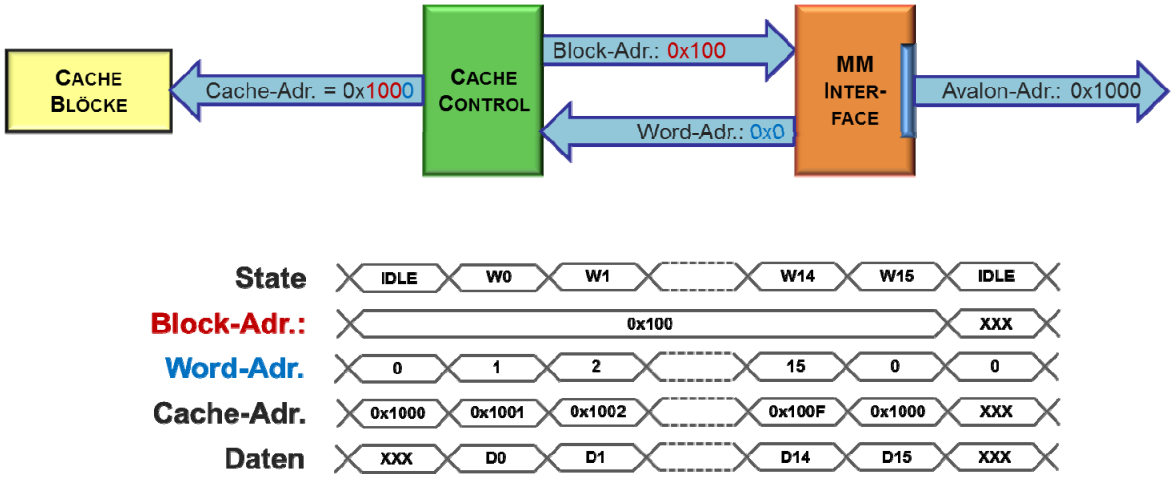


Abbildung 6: Ablauf der Adresskommunikation zwischen Master-Interface und einer Cache-Kontrolleinheit bei einem Schreibvorgang

State über 16 READ- und 16 WRITE-States, sechzehn States deshalb, da wie in Kapitel III.A aufgeführt, stets eine komplette Cachezeile gelesen oder geschrieben wird und eine Cachezeile aus 64 Byte Daten und somit 16 x 32 Bit besteht. Es wird also in jedem dieser States genau ein Datenwort an den Avalon Bus übermittelt oder von diesem angenommen. An dieser Stelle kommt nun auch das Burstcount Signal zum Tragen, das angibt, wie viele dieser States nacheinander durchlaufen werden. Bisher ist dies auf einen oder alle 16 R/W-States beschränkt.

Die Adressierung der Cachedaten geschieht in einer Interaktion zwischen besagter State-Machine und Cache-Kontrolleinheit. Letztere stellt dem Master-Interface eine Blockadresse zur Verfügung, an die das Master-Interface vier Bits mit dem Wert 0 anfügt. Die so erhaltene Adresse bildet diejenige Adresse, die auf den Avalon Bus gegeben wird. Beispielsweise würde eine Blockadresse von 0x3FF eine Avalon-Adresse von 0x3FF0 zur Folge haben. Gleichzeitig sendet das Master-Interface eine 4 Bit breite Word-Adresse an die Cache-Kontrolleinheit zurück und inkrementiert diese in jedem State um Eins, ausgehend vom Wert Null im IDLE-State. Somit würden bei obigem Beispiel nacheinander die Word-Adressen 0x0, 0x1, 0x2 bis 0xF gebildet werden. Diese sich erhöhende Word-Adresse fügt nun die Cache-Kontrolleinheit ihrerseits wieder an die Blockadresse an und bildet somit die Cacheadresse, die die jeweiligen Datenworte innerhalb einer Cachezeile adressiert. Wiederum im vorhergehenden Beispiel würden im Cache die Adressen 0x3FF0, 0x3FF1, 0x3FF2 bis 0x3FFF angefragt werden.

Abbildung 6 verdeutlicht diesen Ablauf nochmals Anhand eines Schreibvorgangs mit der Blockadresse 0x100. Sollte das Master-Interface einen Lese- oder Schreibvorgang auf ein Slave beginnen, dessen Interface gerade belegt ist, dem Master also ein Low-aktives Waitrequest_N Signal gemäß Kapitel IV.B übermittelt wird, behält der momentane State so lange seine Gültigkeit, bis der Avalon Bus wieder frei ist und der aktuelle Vorgang fortgesetzt werden kann.

V. Cache Bypass

Um dem Core ein direktes Zugreifen auf die Avalon Bus-Peripherie zu ermöglichen, war es notwendig, einen sogenannten „Cache Bypass“ zu implementieren. Mit der bisherigen Konfiguration ohne Bypass wären Daten, die der Core auf einer Busperipherie wie beispielsweise einer UART-Schnittstelle ausgeben möchte, stets im Cache abgelegt worden und hätten die Peripherie somit nie oder nur bei einem Write-Back des Caches erreicht. Der implementierte Cache-Bypass genügt dabei einer *Non-Write-Allocate* Strategie, bei der die Daten immer am Cache vorbei auf der nächsthöheren Zielebene gelesen oder geschrieben werden, der Cache also nie angesprochen wird. Dies spart zum einen Rechenzeit, erhöht aber auch gleichzeitig die Cache Hit Rate, womit dieser effizienter wird. Damit steht mehr Speicherplatz für andere Daten zu Verfügung, die somit im Cache vorgehalten werden können.

Die Adressräume der Bus-Peripherie, die nicht dem Cache zugeordnet werden sollen, liegen nahezu beliebig im gesamten 32 Bit-Adressraum des *Sirius Hulk*. Somit war es naheliegend, die Unterscheidung, ob eine angesprochene Adresse vom Cache behandelt werden soll oder nicht, mit Hilfe sogenannter Segmentierung in der Memory Management Unit (MMU) zu verbinden.

Die MMU ist unter anderem für die Übersetzung der virtuellen in physikalische Adressen und die Generierung der Chipselect-Signale zuständig. Aktuell können in der MMU vier Segmente angelegt werden, in deren Konfigurationsregister angegeben werden kann, ob der mit den Segmenten verknüpfte physikalische Adressraum am Cache vorbeigeführt oder von diesem

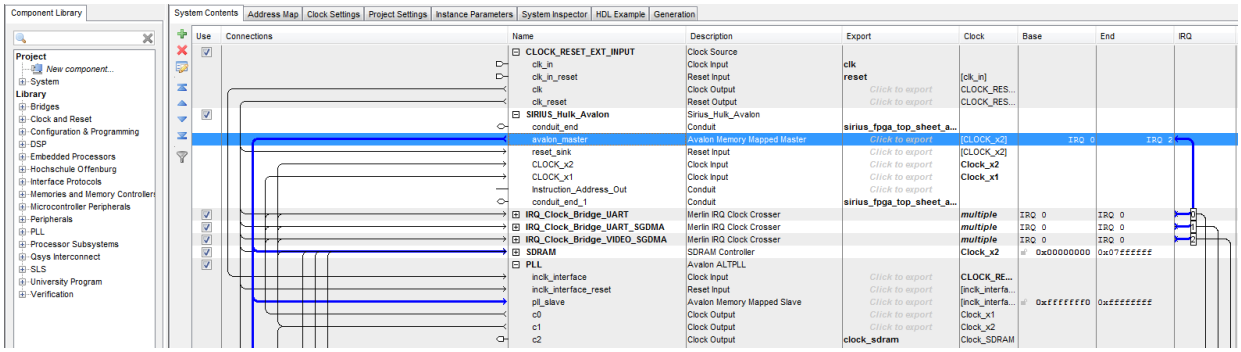


Abbildung 7: Ausschnitt aus Altera Qsys

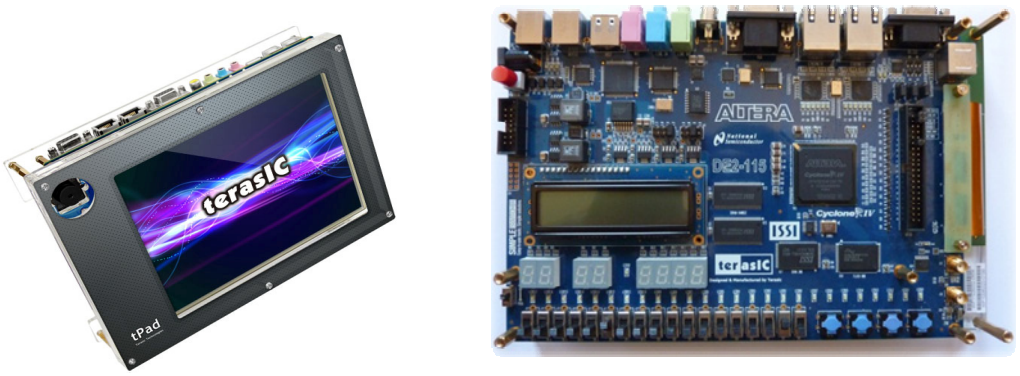


Abbildung 8: Entwicklungsboard Terasic tPad

behandelt werden soll. Anhand dieser Konfiguration wird den Cache-Kontrolleinheiten parallel zu ihren Chipselect-Signalen ein *Activate-Cache*-Signal zur Verfügung gestellt, mit dem die in ihnen implementierte State-Machine entsprechend gesteuert wird. Diese State-Machines mussten somit um die entsprechenden BYPASS-READ- und BYPASS-WRITE-States erweitert werden. Wichtig war hierbei unter anderem eine klare Trennung der Daten und Adressen des langsamen Taktes des Cores und des schnellen Taktes des Caches bzw. des Avalon Busses mit Hilfe geeigneter Register-Prozesse. Weiterhin war es nötig, Steuersignale des Cores und Datensignale vom Memory Mapped Master Interface für eine korrekte Bypass-Funktion zu registern. Zur Vorverifizierung wurden an dieser Stelle mehrere Simulationen mit ModelSim durchgeführt[2]. Weiterhin wurde ein Testprogramm im FPGA ausgeführt, bei dem der gesamte Cache durch den Bypass ausgeschaltet wurde. Das System stellte seine Funktionalität bei dauerhaft aktiviertem Bypass beider Caches unter Beweis.

VI. Altera Qsys

Mit dem Programm Altera Qsys stellt Altera ein Tool zur Verfügung, mit dem die *System Interconnect Fabric* erstellt werden kann. Hierzu bietet es eine grafische Oberfläche, unter der direkt die verschiedenen IP-Cores ausgewählt und in das zu erstellende System übernommen werden können. Weiterhin ist es möglich, je nach IP-Core diesen zu konfigurieren und auf seine Bedürfnisse hin anzupassen. Beispielsweise kann die Größe eines FIFO-Speichers mit Avalon Streaming-Interface individuell eingestellt werden. Die einzelnen Clock-, Reset-, Interrupt- und Avalon Bus-Verbindungen können, nachdem die gewünschten Komponenten dem System hinzugefügt worden sind, per Mausklick erstellt werden. Die gesamte *System Interconnect Fabric* mit allen Komponenten wie Arbitern, Adress-Alignment- oder Timing-Modulen wird darauf hin von Qsys erstellt und konfiguriert. Eventuelle sich überlappende Speicherbereiche oder sonstige Fehler werden erkannt und dem Anwender mitgeteilt. Dieser kann darüber hinaus entscheiden, ob das zu generierende System zur Simulation mit ModelSim, zur Synthese oder beidem verwendbar sein soll. Entsprechend dieser Auswahl erstellt Qsys verschiedene Gesamtsysteme, die anschließend in ModelSim simuliert oder beispielsweise mit Altera Quartus synthetisiert werden können.

Das *Sirius Hulk*-System mit Avalon Memory Mapped Master Interface wurde wie in Abbildung 7 zu sehen als neuer IP-Core der Qsys Bibliothek hinzugefügt und kann somit jederzeit als Grundlage neuer Projekte und Systeme dienen.

[2] http://www.model.com

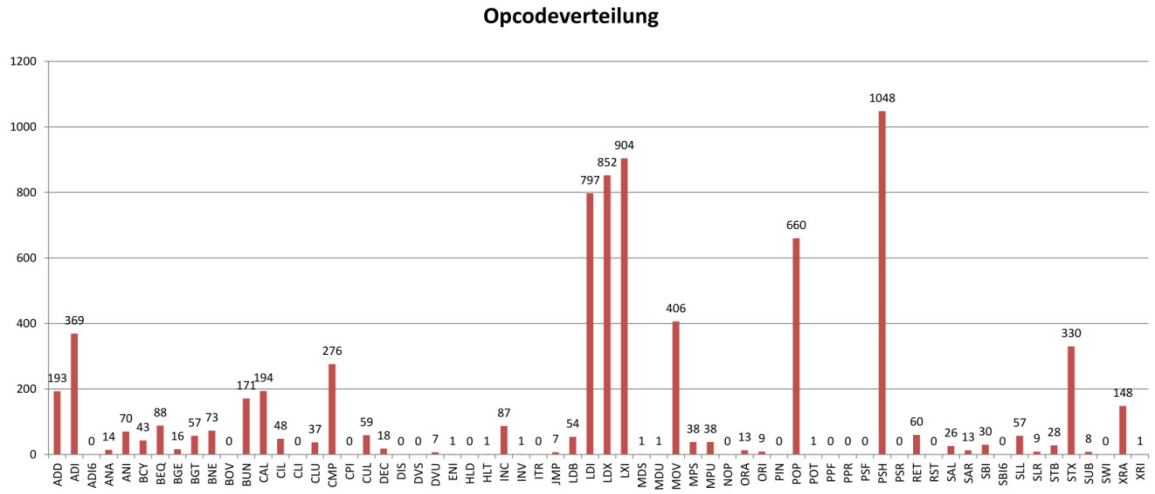


Abbildung 9: Opcodeverteilung (Benchmark *Coremark*)

VII. Aktueller Entwicklungsstand

Die Entwicklung und Verifizierung des Gesamtsystems fand auf dem in Abbildung 8 gezeigten Entwicklungsboard namens *tPad* von Terasic statt. Neben einer Vielzahl an Schnittstellen verfügt es über einen 8 Zoll-Touchscreen, eine 5 Megapixel-Kamera und den Altera FPGA Cyclone IV EP4CE115F29C7N mit 114.480 Logikzellen[3].

Im Zuge der Entwicklung wurde das in Kapitel 0 beschriebene Avalon Memory Mapped Master Interface erstellt, an den Dual Cache angebunden und auf Funktion überprüft. Weiterhin wurde das so erhaltene *Sirius Hulk Avalon*-System in Altera Qsys eingebunden und somit ein Gesamtsystem mit Avalon *System Interconnect Fabric* und Avalon Peripherie IP-Cores erstellt. Zu diesen gehören bisher:

- UART
- PLL
- SDRAM Controller
- SRAM Controller
- DMA Controller
- SD Card Controller
- VGA Display Controller mit Streaming-Interface

Das gesamte Bussystem und damit einhergehend auch alle Busteilnehmer, wurden mit entsprechendem C-Programmcode auf dem *tPad* getestet und stellten ihre Funktionalität unter Beweis. Die Taktfrequenz des Cores betrug dabei 40 MHz und die des Caches sowie der Avalon-Peripherie 80 MHz.

VIII. Codeanalyse

Um eine bessere Einschätzung der verwendeten Prozessorbefehle im Hinblick auf deren Häufigkeit und Effekte möglicher Optimierung derselben treffen zu können, wurde eine Codeanalyse vorgenommen. Als Codegrundlage diente hierzu das Benchmark Programm *EEMBC Coremark*, das direkt von der Coremark-Homepage bezogen werden kann.[4] Diese Benchmark wurde mit dem Ziel entwickelt, einen praxisnahen Workload abzubilden, um einen Prozessorkern möglichst aussagekräftig in Bezug auf einen beliebigen Praxiseinsatz bewerten zu können. Nachdem die entsprechende Codeanpassung an den *Sirius Hulk* erfolgt war, wurde eine Excel-Auswertung vorgenommen, die die erzeugte List-Datei des Assemblers auswertet, in der alle Codeelemente in Assembler aufgelistet sind und somit quantitativ erfasst werden können. In Abbildung 9 sind die Ergebnisse dargestellt. Wie zu sehen ist, kommt der PSH-Befehl am häufigsten vor, da mit diesem Parameterübergaben über den Stack an aufgerufene Funktionen abgehandelt werden. Darüber hinaus kommen die Befehle LXI, LDX (Load Double Word) und LDI (Load Integer Immediate) besonders häufig vor. Der Befehl LXI addiert eine Konstante zum SA-Registerwert und legt das Ergebnis in einem Zielregister ab.

Der vom *Sirius Hulk* erreichte Coremark-Wert belief sich in einem ersten Test auf 0,78 Coremark/MHz. Man muss allerdings beachten, dass der verwendete Little-C-Compiler (LCC) keine Möglichkeiten der Codeoptimierungen bietet. Es muss beachtet werden, dass es sich hier um eine rein statische Analyse handelt, die nur die reine Häufigkeit der Opcodes im vorliegenden Programm ermittelt und eventuelle mehrfache Unterfunktionsaufrufe und somit mehrfa-

[3] http://www.tpad.terasic.com

[4] http://www.coremark.org

che Ausführung bestimmter Codeabschnitte nicht berücksichtigt.

IX. Ausblick

In naher Zukunft können weitere IP-Cores dem *Sirius Hulk Avalon*-System hinzugefügt werden, um es auf Zielhardware wie dem *tPad* von Terasic anzupassen und dessen Funktionen und Interfaces im vollen Umfang nutzen zu können.

Ferner kann über Timing-Constraints und optimierter Synthese versucht werden, die zeitkritischen Pfade des Systems zu minimieren und so die maximale Taktfrequenz des Systems zu erhöhen, sodass zum Beispiel der auf dem *tPad* verbaute SDRAM mit den für diese Speicherklasse üblichen 133 MHz betrieben werden kann. Dies würde einem Core-Takt von 66,5 MHz entsprechen.

X. Fazit

Die Erstellung eines Avalon Memory Mapped Master Interfaces und die Anbindung des RISC-Softcores *Sirius Hulk* des Instituts für angewandte Forschung der Hochschule Offenburg an das Altera Avalon-Bussystem ermöglicht es, den Softcore auf vielfältige Weise mit IP-Cores zu erweitern und so auf den jeweiligen Zieleinsatz hin zu optimieren. Darüber hinaus verfügt er über ein modernes Prozessordesign durch Komponenten wie Dual-Cache, Harvard Architektur sowie den reduzierten Befehlssatz.

Danksagung

Besonderer Dank gilt an dieser Stelle allen Mitarbeitern des ASIC Design Centers und des Instituts für Angewandte Forschung (IAF) der Hochschule Offenburg, die uns stets mit Rat und Tat zur Seite standen und es so ermöglichten, diese Veröffentlichung im Rahmen einer Master-Thesis zu erstellen.

Literaturverzeichnis

[1] S. Stickel, “Entwicklung eines Cache Speichers mit DDR-RAM-Anbindung für eine 32 Bit Softcore mit Harvard Architektur und Verifikation in einem FPGA”, Master-Thesis, Hochschule Offenburg, April 2011.

[2] S. Stickel, D. Jansen, F. Zowislok, M. Durrenberger, „32 Bit Softcore mit Harvard Architektur und Double Cache“, 45. *Workshop der Multiprojekt-Chip-Gruppe Baden-Württemberg*, Albstadt-Sigmaringen, Feb. 2011.

[3] Altera, “Avalon Interface Specifications”. http://www.altera.com/literature/manual/mnl_avalon_spec.pdf, Datum des Zugriffs: Februar 2012.

[4] Altera, “SOPC Builder User Guide”. http://www.altera.com/literature/ug/ug_sopc_builder.pdf, Datum des Zugriffs: Februar 2012.

Michael Schmidt erhielt den akademischen Grad des B. Eng in Elektro- und Informationstechnik im Jahr 2010 von der Hochschule Offenburg und studiert derzeit im Masterprogramm. Die vorliegende Veröffentlichung wurde im Rahmen seiner Master-Thesis publiziert.

Digitally Assisted Analog: Adaptive IQ Correction Algorithms for Homodyne Receivers

Simran Singh, Michael Epp, Georg Vallant, Heinz-Peter Bürkle

***Abstract*—The homodyne receiver is suitable for on-chip integration, however its I and Q channel component mismatch results in insufficient image rejection, degrading its performance. By characterizing the IQ mismatch and providing correction algorithms in the digital domain, the performance degradation can be compensated. Therefore the dynamic range of the homodyne receiver can be increased. This paper focuses on algorithms, which correct the IQ mismatch adaptively.**

***Index Terms*—Adaptive IQ correction algorithms, IQ mismatch homodyne receiver, digitally assisted analog, image rejection ratio, LMS & Circularity.**

I. Introduction

The miniaturization of integrated circuits continues and consequently enables electronic devices to become more complex and compact. However, stronger miniaturization also means, that the non-idealities of analog circuitry become more dominant. One current trend is to mitigate the effects of these non-idealities by using the interdisciplinary concept of "Digitally Assisted Analog".

The aim of digitally assisted analog is to reduce the complexity of the analog components, enabling simpler analog circuit design. Simpler analog circuits offer increased energy efficiency and allow for better integration. The total cost of the IC production is reduced. The performance degradation due to the non-idealities of the simpler analog circuitry is mitigated in the digital domain by providing corresponding correction algorithms.

One example of digitally assisted analog can be seen in the communication fields. Usage of the simpler homodyne receiver architecture allows better integration due to its simple analog components. This is used as an alternative to the heterodyne receiver, which has highly demanding analog components with integration difficulties [1] [2].

Simran Singh and Heinz-Peter Bürkle, heinz-peter.buerkle@htw-aalen.de, are with Hochschule Aalen, Beethovenstr.1, D-73430 Aalen, Germany.

Michael Epp and Georg Vallant, georg.vallant@cassidian.com, are with Cassidian Electronics, Woerthstraße 85, D-89077 Ulm, Germany.

However, the homodyne receiver suffers from analog component mismatches, which lead to insufficient image rejection. Insufficient image rejection causes spurs, which limit the spurious free dynamic range (SFDR).

By characterizing the effects of the analog component mismatch and its relationship to the image rejection degradation, digital algorithms can be deployed, which improve the image rejection ratio (IRR).

This contribution focuses on adaptive digital correction algorithms for the IQ mismatch in the homodyne receiver. The algorithm analysis, its complexity and performance will be presented. First the IQ mismatched is analyzed and characterized, in the next step it is compensated via digital algorithms.

II. Receiver Architecture for an IC solution

There are two basic types of receiver architecture available:

1) Heterodyne architecture.
2) Homodyne architecture.

A. Heterodyne Receiver

The heterodyne receiver (shown in figure 1) has superior down conversion performance. However it requires high performance analog components. Due to IC technology limitations, not all of the components can be realized on chip.

For example, after selecting the band of interest, there are two stages of filtering in the heterodyne receiver, one for image rejection and one for anti-aliasing. The analog filters that operate at higher frequencies with high filtering requirement and is therefore realized off-chip, for e.g. as SAW (surface acoustic wave) filter.

The ADC digitizes the signal at a certain IF and so the performance requirement for the ADC must be met at a higher frequency, making it more difficult to achieve. Even with the possibility of IF sampling, the SNR and SFDR of the ADC has to be sufficient at its operating frequency. This means more development effort and increased costs.

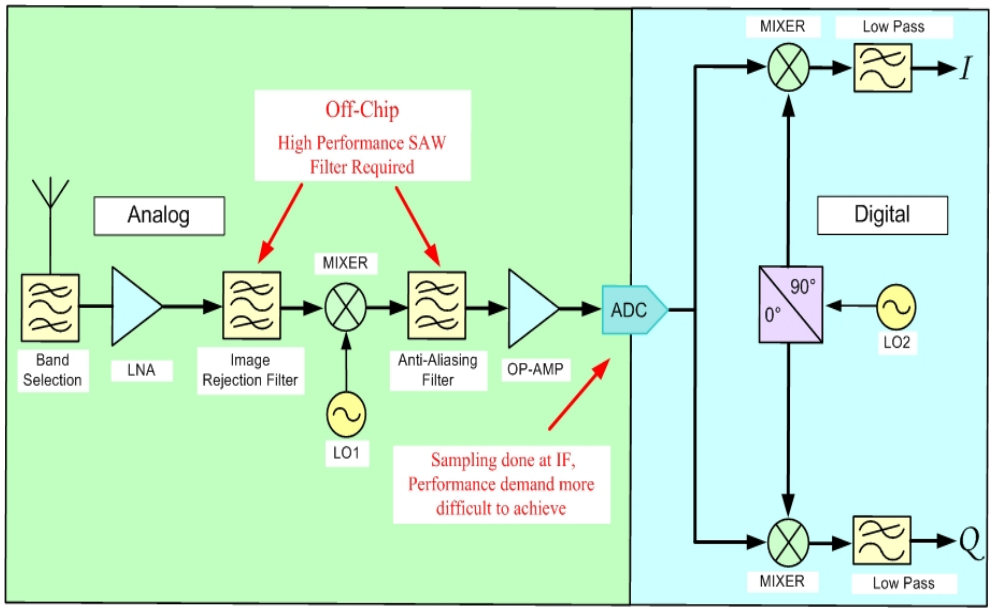


Figure 1: Heterodyne receiver architecture

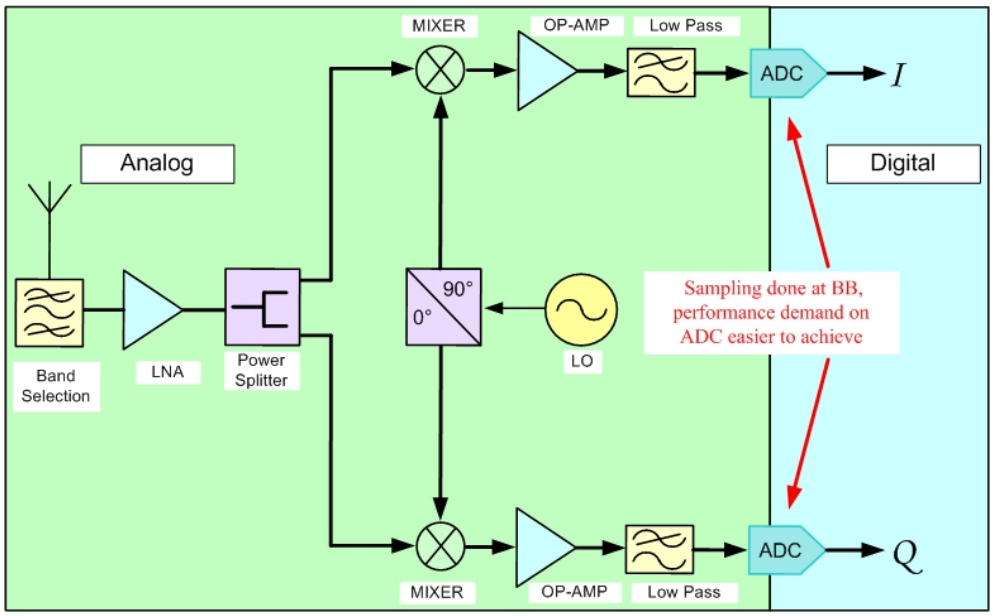


Figure 2: Homodyne receiver architecture

B. Homodyne Receiver

The homodyne architecture (shown in fig. 2) consists of analog components with less performance demand in comparison to the heterodyne receiver. For example, the ADC samples at baseband and therefore its performance demand is easier to be met. These simple analog components are suited for on-chip integration but there are now certain problems that were not present in the heterodyne architecture.

The main drawback is the finite matching of the analog components in the I and Q path. Having two analog RF paths, the mismatches between discrete analog components result in incomplete image rejection. DC offset is also a problem in the Zero-IF down conversion due to the parasitic coupling between the mixer inputs resulting in self-mixing of the signals, which contributes to the DC component.

Despite having obvious drawbacks, the homodyne architecture is simple and easy to be integrated with less production cost and therefore it is chosen for integrated solutions with increasing preference over the past years.

The types of mismatches that occur due component mismatch can be analyzed and their relationship to the performance degradation can be mathematically modelled followed by algorithms that mitigate these effects.

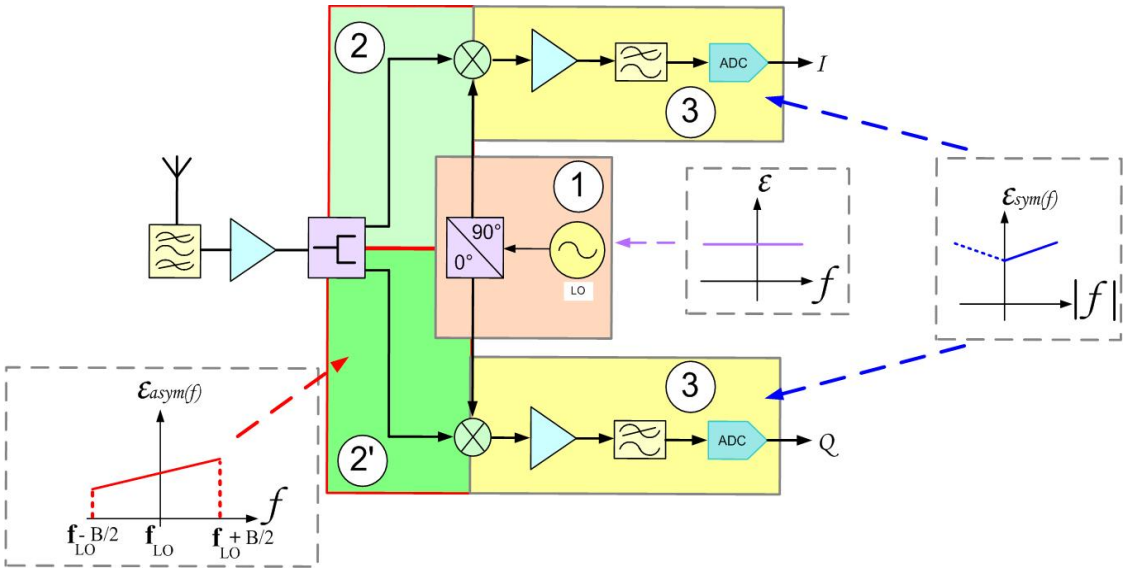


Figure 3: Types of IQ mismatches across the frequency bandwidth in a Homodyne receiver

III. Classifying Types of IQ Mismatches in a Homodyne Receiver

The IQ mismatch varies across the frequency bandwidth and for a different LO frequency the IQ mismatch variations across the bandwidth will be different. The IQ mismatch variation across the bandwidth can be classified into three categories:

1) Static IQ mismatch.
2) Asymmetrical IQ mismatch (frequency-dependent).
3) Symmetrical IQ mismatch (frequency-dependent).

Firstly, the static IQ error gives a constant mismatch between the I and Q path. The LO frequency is fixed so the IQ mismatch generated across the bandwidth is also constant. As seen in fig. 3 in block labelled 1, the local oscillator (LO) generated frequency does not have a phase difference of 0° and 90° but with a certain offset for e.g. 0 ° and 90°+$\Delta\varphi$. The next type of IQ mismatch is the frequency dependent asymmetrical mismatch. The IQ mismatch depends on the frequency at RF (shown in Fig. 3 with blocks labelled 2 and 2').

After the complex down conversion, the RF signals, whose frequencies were above the LO frequency, will be positive frequencies. For RF signals, whose frequencies were below the LO frequency, will be negative frequencies after the complex down conversion.

Therefore the IQ mismatch for positive frequencies is not the same as for negative frequencies. This gives birth to the asymmetrical, frequency-selective nature of the IQ mismatch across the bandwidth. The third type of IQ mismatch is the symmetrical error.

After the down conversion, on each respective I and Q channels, the signals are real and no distinction can be made between positive and negative frequencies (if the RF frequency was above or below the LO frequency).

The IQ mismatch which occurs for positive frequencies is the same as for negative frequencies, resulting in a symmetrical IQ mismatch. The symmetry behaviour of the IQ mismatch across the bandwidth is shown in the fig. 3 above in block labelled 3.

The symmetrical IQ mismatch can be corrected in each respective I and Q path alone. However, in order to correct the asymmetrical errors both I and Q signal must be taken. Only by taking the complex signal, positive and negative frequencies can be differentiated (if the RF frequency was above or below the LO frequency).

Only then, the asymmetrical IQ mismatch that occurred at RF can be corrected. The IQ mismatch is divided as follows:

1) Amplitude Mismatch, $A\varepsilon$. Amplitude of both I & Q path signals are not the same.
2) Phase Mismatch, $\varphi\varepsilon$. The phase difference of the I & Q path signals is not 90°.

The image rejection ratio (IRR) depends on the IQ mismatch. The IQ mismatch varies across the bandwidth, so at different frequencies the IRR is different. It is also essential to examine the relationship between the IQ mismatch and the image rejection. DC offset is also an issue in a homodyne receiver.

Though this does not contribute to the IQ mismatch, it disrupts adaptive IQ correction algorithms which extract statistical information from the signal. Therefore, DC offset also has to be removed before applying correction algorithms.

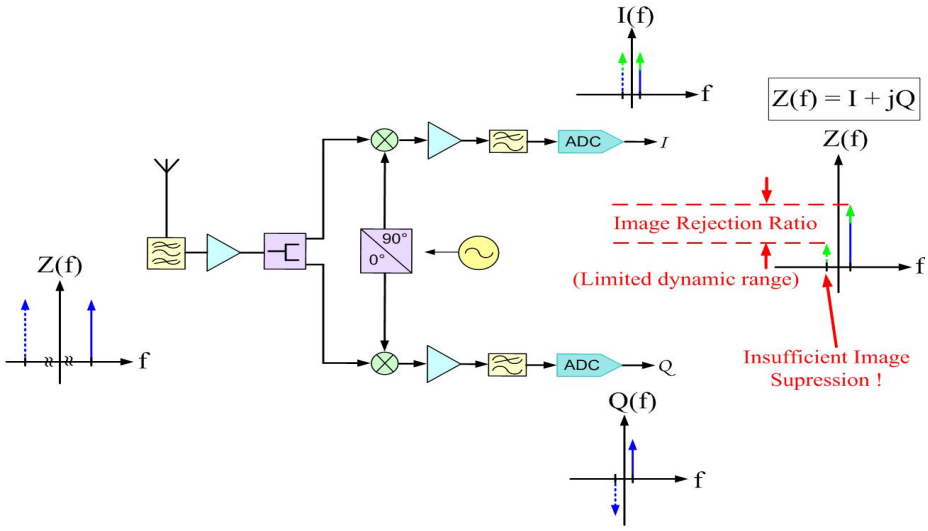


Figure 4: Insufficient image rejection with IQ mismatch

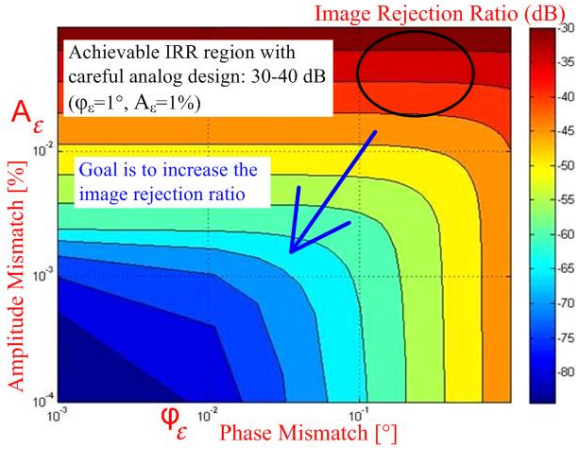


Figure 5: IQ mismatch vs IRR

IV. IQ MISMATCH AND IMAGE REJECTION RATIO

As shown in Figure 4, because the analog components in the I and Q channels are not perfectly matched, the frequency components on both the channels are imbalanced. Due to this, the images (mirror frequencies) do not cancel each other out completely. The remaining image component manifests itself as a spur.

This spur limits the dynamic range and causes performance limitation in the homodyne receiver. The IQ mismatch consists of an amplitude mismatch and phase mismatch. In the next step the relationship between the IRR to the amplitude and phase mismatch will be derived. For a received signal $s(t)$, after complex down conversion without IQ mismatch, no mirror frequency components are present.

$$s(t) = \cos(\omega t) + j\sin(\omega t)$$

$$s(t) = e^{j\omega t}$$

Now let $z(t)$ be the received signal with IQ mismatch, with the I-channel being ideal and only the Q-channel containing the amplitude and phase mismatch. Due to this IQ mismatch, the mirror frequency components remain as seen below.

$$z(t) = \cos(\omega t) + j(1 + A_\varepsilon)\sin(\omega t + \varphi_\varepsilon)$$

$$z(t) = (k_1)e^{j\omega t} + (k_2)e^{-j\omega t} = k_1 s(t) + k_2 s^*(t)$$

The image rejection ratio (IRR) is simply the logarithmic ratio between the desired frequency components and its mirror frequency components.

$$z(t) = (k_1)e^{j\omega t} + (k_2)e^{-j\omega t}$$

$$\mathrm{IRR} = 20\log\left(\frac{|k_1|}{|k_2|}\right) = 10\log\left(\frac{|k_1|^2}{|k_2|^2}\right)$$

Now the relationship for the IQ mismatch and the image rejection ratio can be established as follows.

$$z(t) = \cos(\omega t) + j(1 + A_\varepsilon)\sin(\omega t + \varphi_\varepsilon)$$

$$z(t) = e^{j\omega t}\left(1 + \frac{A_\varepsilon}{2} + j\frac{\varphi_\varepsilon}{2}\right) + e^{-j\omega t}\left(-\frac{A_\varepsilon}{2} + j\frac{\varphi_\varepsilon}{2}\right)$$

$$z(t) \approx e^{j\omega t}(1) + e^{-j\omega t}\left(-\frac{A_\varepsilon}{2} + j\frac{\varphi_\varepsilon}{2}\right)$$

$$\text{with}\left(1 + \frac{A_\varepsilon}{2} + j\frac{\varphi_\varepsilon}{2}\right) \approx 1$$

$$\mathrm{IRR} = 10\log\left(\frac{1}{\left(\frac{A_\varepsilon}{2}\right)^2 + \left(\frac{\varphi_\varepsilon}{2}\right)^2}\right)$$

Using the approximate formula, the resulting image rejection for the amplitude mismatch of 1% and the phase mismatch of 1° results in -39.95 dB. This is cross checked against the measured image in a FFT spectrum, which was at -40 dB. Figure 5shows the resulting IRR for various IQ mismatch combinations.

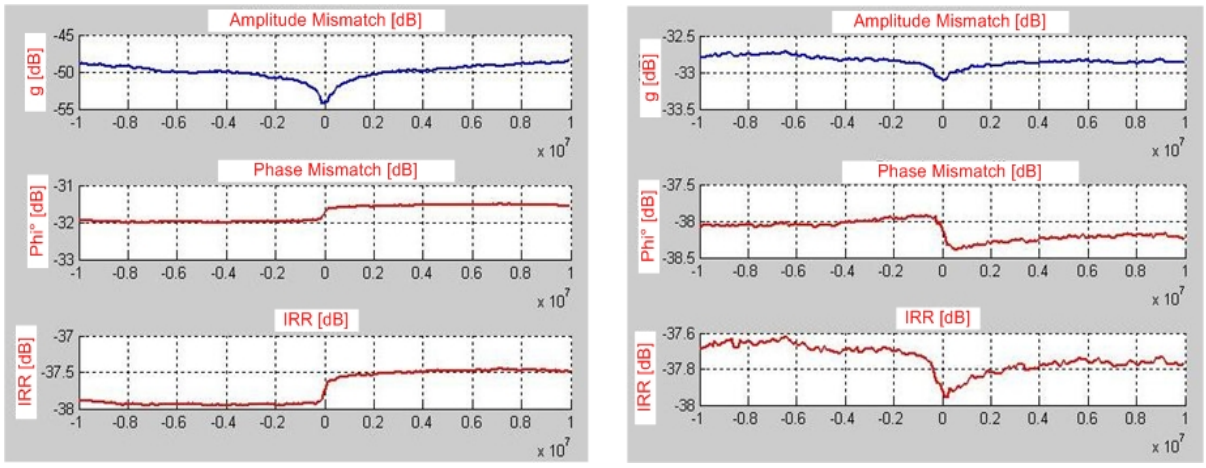


Figure 6: Measured IQ mismatch on various Homodyne receivers

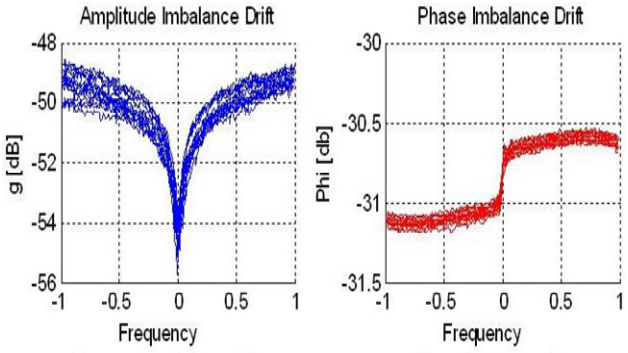


Figure 7: IQ mismatch fluctuations

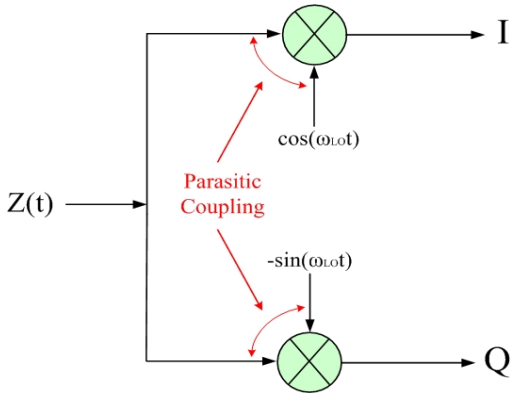


Figure 8: DC offset in Homodyne receivers

V. Variations in the IQ mismatch

As shown in figure 6 above, it can be seen that the amplitude mismatch $A\varepsilon$ and the phase mismatch $\varphi\varepsilon$ vary over the frequency. This is known as the frequency dependent IQ mismatch or the form of the IQ mismatch. The IQ mismatch form surely varies from receiver to receiver due to manufacturing tolerances. Also, as seen in figure 7, the IQ mismatch fluctuates over very small temperature variations, even if the operating conditions are kept nearly constant. From figure 6, it can be deduced that if correction of the IQ mismatch was to be attempted via a one time calibrated FIR filter, it has to be done specifically for each Homodyne receiver (requiring laboratory calibration). However the main disadvantage (seen in figure 7) of a fixed correction is the fact that the IQ mismatch varies over time and would lead to the performance degradation of the fixed correction. For this reason, adaptive IQ correction algorithms are to be attempted that can deal with these IQ mismatch variations and are valid independently of receiver specific IQ mismatch.

VI. DC offset in Homodyne receivers

The DC offset causes the I and Q path signal to have different levels. This causes problems when extracting IQ mismatch information via statistical methods as in the case of adaptive IQ correction algorithms.

The main reason for this DC offset is the parasitic coupling between the two mixer inputs (shown in figure 8 above). The LO to RF leakage causes a portion of LO frequency to be mixed into the received signal. Upon down conversion, this leaked LO frequency falls at exactly 0 Hz contributing to the DC portion of the signal.

This gives a static DC offset as the LO frequency is fixed, leading to constant LO to RF leakage. At the same time, there is also a RF to LO leakage causing a portion of the RF signal to leak into the LO input. The capacitive coupling between the mixers inputs is frequency dependent, so the RF to LO leakage varies according to the RF frequency. This causes a dynamic DC offset. It is a prerequisite that the DC offset is removed before using adaptive algorithms [3].

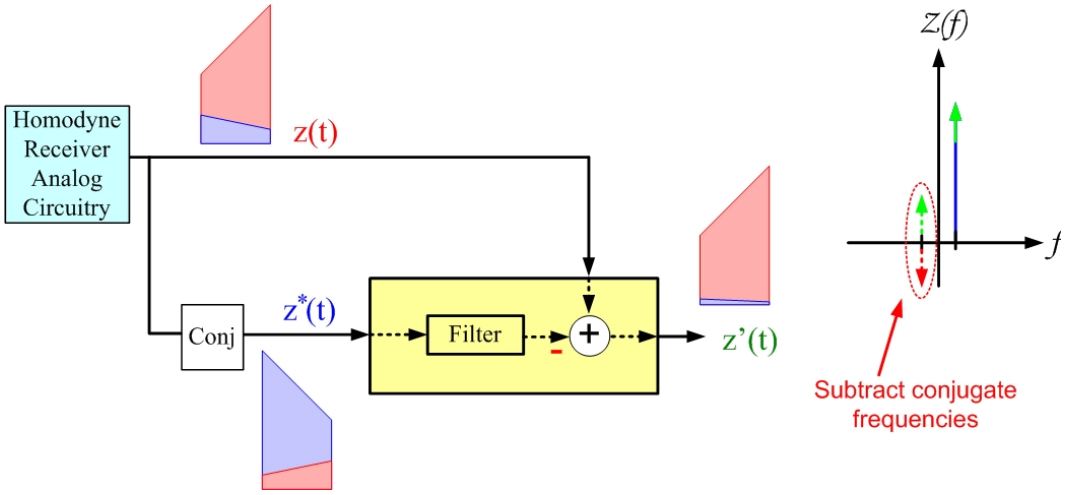


Figure 9: IQ mismatch correction principle

VII. Adaptive IQ Correction Algorithms

A. Introduction

Due to the fact that there are variations in the IQ mismatch during operation, algorithms that correct the IQ mismatch in an adaptive manner are to be examined. Digital filters are to be considered whose coefficients are constantly adapting to the input data with an algorithm that governs the coefficient change.

The term adaptive is used because the filter is constantly adjusting its coefficients in correspondence to an observed quantity. This quantity offers the information needed for the adaptation process to respond to the change. For reacting towards the change, an algorithm is needed based on which the tap update decision is made.

Adaptive algorithms investigated rely on statistical properties of signals. It can either have a reference signal, using which the coefficient change is made by considering the statistical properties of the two signals. When there is no reference signal, an alternative method can be utilized, where certain statistical information of the signal itself is used for the adaptation process (no additional signal is needed).

When a reference signal is used it is called supervised adaptation. When no reference signal is used, it is known as blind adaptation. There are many different applications for the supervised adaptation. One type of application is known as interference cancellation (IC).

This can be extended to the IQ mismatch case, considering the image as the interference. The goal is to improve the image rejection by taking the image as source of interference and cancelling its effect on the signal.

When no reference signal is available, blind adaptation or better known as Blind Source Separation (BSS) can be used. In the BSS version, there is no reference or desired signal. An attempt is made to restore certain statistical properties of a signal that were lost during the complex down conversion due to incomplete image rejection. In this case the signal is referenced to itself to detect distorted statistical properties which are then corrected leading to an improved IQ matching [4] [5].

B. Underlying Principal of IQ Mismatch Correction

Let the complex IQ signal at the output of the Homodyne receiver be known as complex signal $z(t)$. As seen in figure 9, the signal $z(t)$ contains it image as a spur in the spectrum. The goal of improving the image rejection ratio is achieved either by reducing the amplitude mismatch and phase mismatch (in the I and Q path respectively). Alternatively the same effect can be achieved by subtracting a scaled version of the image (containing the mirror frequencies) from the signal (seen in figure 9).

By simply taking the conjugate of the complex signal $z(t)$, its mirror frequencies $z^*(t)$ are obtained. The discrete samples of $z(t)$ can be considered as a vector, with each element being a complex element. Vector $z^*(t)$ is obtained simply by taking the conjugate of each of the elements in $z(t)$.

$$\vec{z} = [z_1 \quad z_2 \quad z_3 \quad \cdots \quad z_n]^T$$

$$\vec{z}^* = [z_1^* \quad z_2^* \quad z_3^* \quad \cdots \quad z_n^*]^T$$

The conjugate signal $z^*(t)$ is scaled and subtracted from the desired signal $z(t)$ resulting in a signal $z'(t)$ with improved image rejection. By doing this, the image spur that that was limiting the dynamic range is further reduced (see figure 9) and the system performance increases [6]-[11].

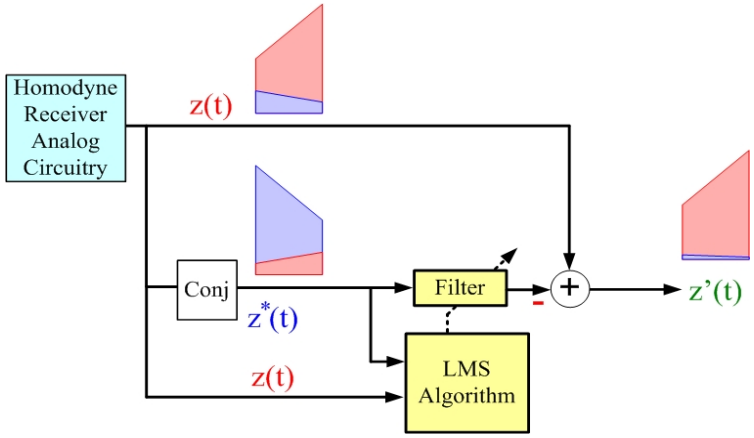


Figure 10: Adaptive IQ correction via LMS algorithm

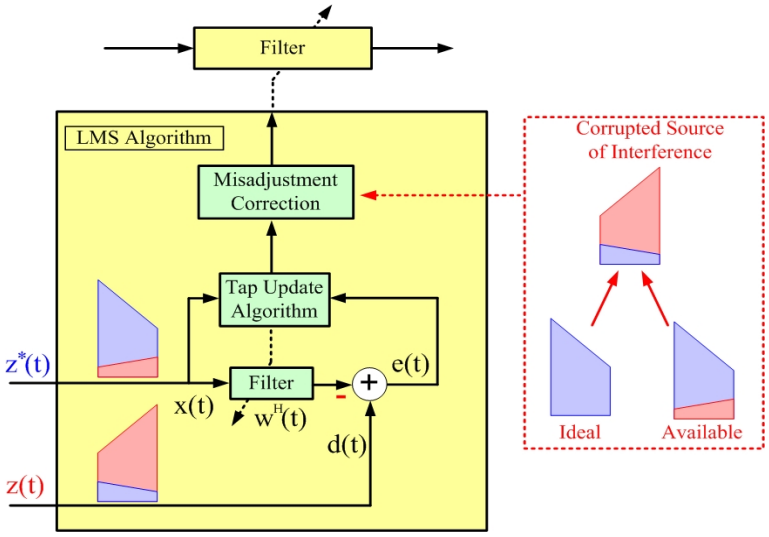


Figure 11: Inner structure of the LMS algorithm

C. Interference Cancellation - Least Mean Square Algorithm

Interference cancellation (IC), as the name suggests, is an algorithm that takes a given source of interference and removes its influence from a desired signal. This is done by de-correlating the desired signal from the source of interference, achieved via subtracting spectral components of the interference from the desired signal. This requires a signal that serves as a reference source for the interference. The IC method can be extended to the IQ mismatch correction problem. The source of interference needed is provided by taking the conjugate of the signal. By de-correlating the conjugate from the desired signal via spectral subtraction, the IRR can be increased.

The least mean square (LMS) algorithm, a stochastic gradient algorithm, will be used for the purpose of IC. The structure for IQ correction with IC LMS algorithm is shown in figure 10. Both $z(t)$ (the desired signal) and its conjugate $z^*(t)$ (source of interference) are fed to the LMS algorithm block. The LMS algorithm (one tap version) is as follows:

$$e(t) = d(t) - w^H(t)x(t) \qquad \text{Filtering}$$
$$w(t+1) = w(t) + \mu x(t)e^*(t) \qquad \text{Tap update}$$

The scaling factor, known as the weight tap $w(t)$, is initialized with the value 0. Update of the weight is done by measuring the cross-correlation between the $x(t)$ and the output signal $e(t)$ with a small adaptation constant μ called the step-size.

After convergence, $x(t)$ is optimally scaled by $w^H(t)$[1] before being subtracted from $d(t)$, leading to e(t) being, in mean, de-correlated from $x(t)$. The convergence interval is dependent on the step-size μ. Smaller μ gives a smaller convergence interval, achieving better IRR.

However, before an improvement in the IRR can be achieved, the convergence misadjustment has to be corrected. The conjugate of the signal $z^*(t)$ contains also the desired signal $z(t)$ which causes the misadjustment at convergence. Proposed in figure 11 is a LMS algorithm structure with misadjustment correction [6]-[12].

[1] H-Hermitian transpose

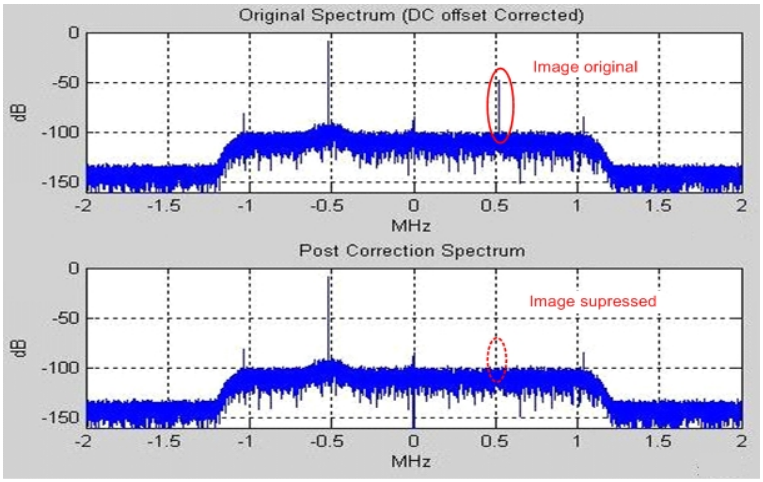


Figure 12: LMS algorithm tested with a single tone CW signal

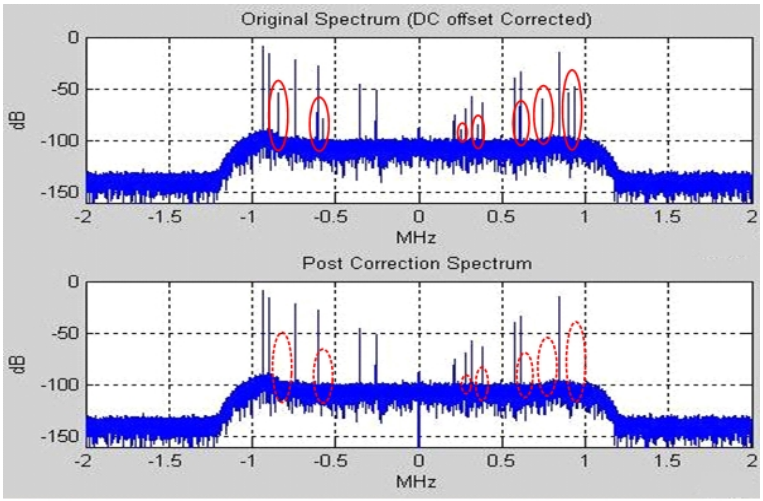


Figure 13: LMS algorithm tested with a multi tone CW signals

In accordance with the Wiener filter theory, optimal convergence of the weight tap $w(t)$ is given as:

$$w_{opt} = \frac{P}{R}$$

where P is the cross-correlation $E[x(t)d^*(t)]$ and R is the autocorrelation $E[x(t)x^H(t)]$. To remove the conjugate frequencies $s^*(t)$ from the signal $z(t)$, the desired weight tap $w^H(t)$ value must equal k_2 when scaling $x(t)$. Therefore $w(t)$ must equal k_2^*.

$$d(t) = z(t) = k_1 s(t) + k_2 s^*(t)$$

$$x(t) = z^*(t) = k_1^* s^*(t) + k_2^* s(t)$$

P and R of the IQ signal can be expressed as follows:

$$R = E[x(t)x^H(t)] = E[z^*(t)z(t)]$$
$$= (k_1^* k_1 + k_2^* k_2) E[s^*(t)s(t)]$$
$$P = E[x(t)d^*(t)] = E[z^*(t)z^*(t)]$$
$$= 2(k_1^* k_2^*) E[s^*(t)s(t)]$$

Therefore the effect of the non-ideal source of interference on the convergence of the optimal weight of $w(t)$ can be calculated [6]-[12].

$$w_{opt} = \frac{2(k_1^* k_2^*) E[s(t)s^*(t)]}{(k_1 k_1^* + k_2 k_2^*) E[s(t)s^*(t)]}$$

with $k_1 k_1^* + k_2 k_2^* \approx 1$ and $k_1^* \approx 1$

$w_{opt} \approx 2k_2^*$ (wanted $w_{opt} = k_2^*$)

This is misadjustment is corrected by multiplying it with a correction factor. The LMS algorithm is tested for a single tone CW signal (as shown in figure 12). Here it can be seen that the image has been suppressed down to the noise level. The LMS algorithm is also capable of correcting the IQ mismatch for a multi tone CW signal (shown in figure 13). The LMS algorithm is simple and is suited for hardware implementation. Its convergence speed and convergence interval is dependent on the step-size (adaptation constant). Being a stochastic gradient algorithm, it exhibits good tracking capabilities. However it requires a source of interference for the adaptation process. Convergence becomes a problem if both positive and negative frequency components are simultaneously present.

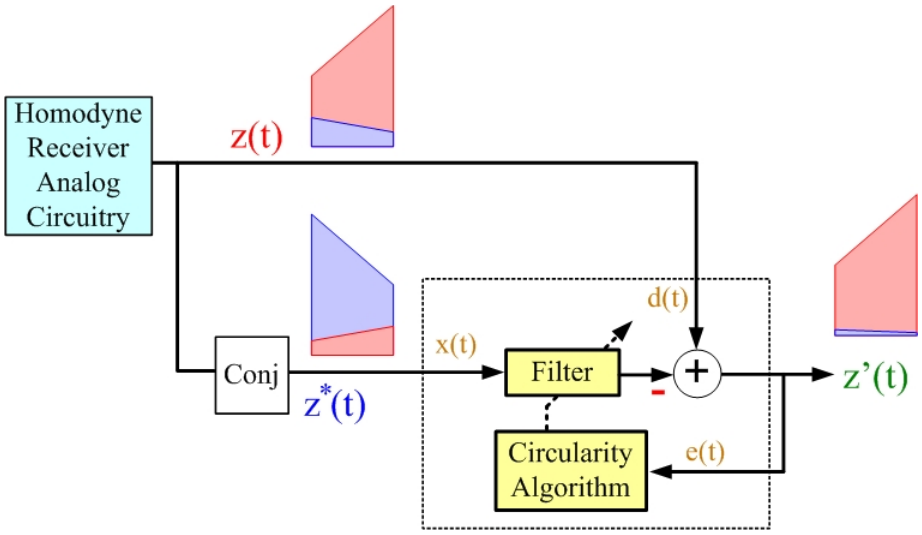


Figure 14: Adaptive IQ correction via circularity based algorithm

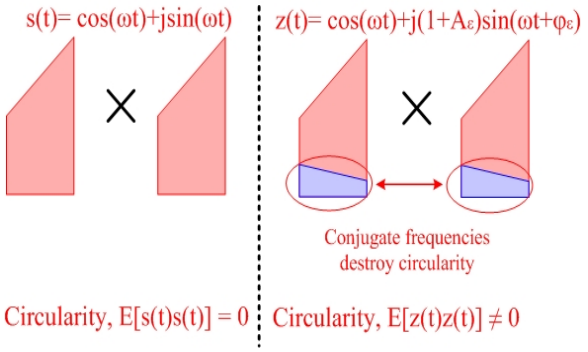


Figure 15: Circularity with and without IQ mismatch

D. Blind Source Separation-Circularity based algorithm

The interference cancellation adaptive algorithm requires a signal as the interference source (the conjugate signal in the IQ mismatch case) as a reference for the adaptation process. An alternative solution to requiring a reference signal is offered by the Blind Source Separation (BSS) algorithms. No extra signal is needed as reference. The internal statistical properties of the signal are used as reference, mainly the second order statistical properties of a signal.

The autocorrelation function (ACF) measures the signal's $z(t)$ cross-correlation with itself in the function $E[z(t)z(t-\tau)^*]$. The pseudo autocorrelation function (PACF) measures the signal's cross- correlation to its conjugate in the function $E[z(t)z(t-\tau)]$. If the PACF remains unaffected by the time delay τ, the signal is said to be second order stationary, meaning $E[z(t)z(t-\tau)] = E[z(t)z(t)]$.

For the IQ mismatch correction, a special case property of the second order statistics will be considered, which is lost due to the presence of the IQ mismatch. Consider the signal is not only second order stationary, but its PACF $E[z(t)z(t-\tau)] = 0$. In this case the signal is said to be circular. This is only valid for complex signals.

For real signals $E[z(t)z(t)]$ will not equal 0. As the IQ signals are complex signals, the circularity property of second order statistics is applicable. A proper IQ signal (e.g. without IQ mismatch) is circular. However a signal being circular does not always mean the signal is also proper.

As shown in figure 9, a weighted image is subtracted from the signal to correct the IQ mismatch. This structure will also be extended to the BSS (Blind Source Separation) algorithm using the circularity measurement for its adaptation process.

The weight update is done by examining the circularity of the output signal (shown in figure 14). The circularity based algorithm (one tap version) is as follows:

$$e(t) = d(t) - w(t)x(t) \qquad \text{Filtering}$$

$$w(t+1) = w(t) + \mu e(t)e(t) \qquad \text{Tap update}$$

Re-establishing circularity corrects the IQ mismatch. When the PACF is measured, the $\mathrm{Re}\{E[z(t)^2]\}$ measures how well the magnitude of the real and imaginary component of $z(t)$ are matched (the amplitude mismatch in the IQ signal).

The $\mathrm{Im}\{E[z(t)^2]\}$ provides information about the orthogonality of the real and imaginary component of $z(t)$ (the phase mismatch in the IQ signal). Restoring circularity means correcting the amplitude and phase mismatch.

If $E[z(t)^2]$ (PACF) has a positive value, the weighted quantity of the image is subtracted from the signal. If $E[z(t)^2]$ has a negative value, the weighted quantity of the image is added to the signal. This is done to bring $E[z(t)^2]$ towards zero by adding or subtracting a weighted value of $z^*(t)$ using an instantaneous estimate of $E[z(t)^2]$ [13]-[19].

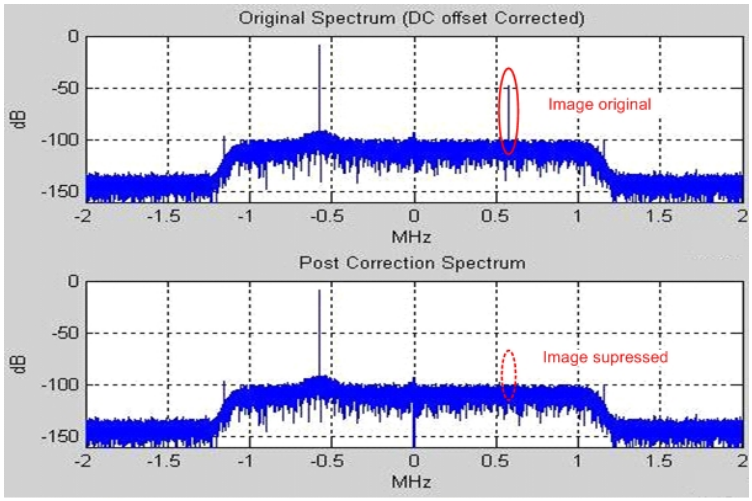


Figure 16: Circularity based algorithm tested with a single tone CW signal

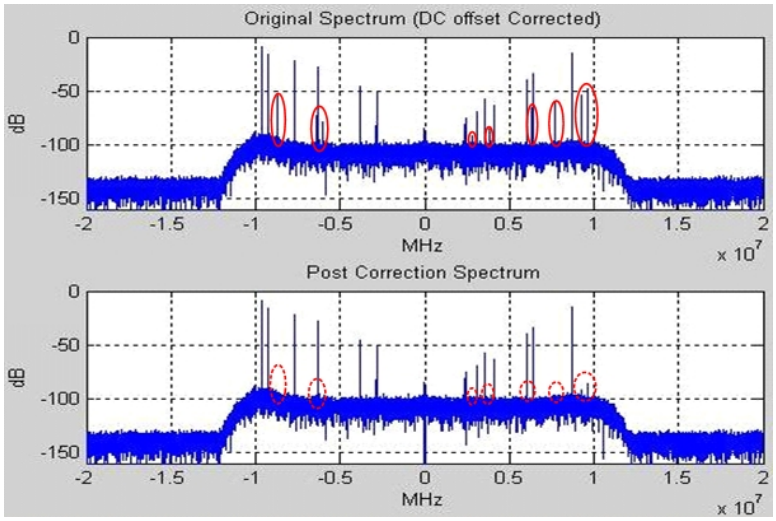


Figure 17: Circularity based algorithm tested with a multi tone CW signal

Restoring the circularity would mean removing the component of $s^*(t)$ from $z(t)$.

$$z(t) = k_1 s(t) + k_2 s^*(t)$$

For this, an optimal weight tap is needed, which subtracts the exact amount of $s^*(t)$ from $z(t)$. This results in $z(t)$ consisting only of s(t). However for this to happen, the subtracted image weighting must correspond to the factor k_2. So the convergence of the tap must be at k_2 when scaling $z^*(t)$ to optimally reject the image.

$$k_2 z^*(t) = k_2[k_1 * s^*(t) + k_2 * s(t)]$$

$$k_2 z^*(t) \approx k_2 s^*(t) \quad (k_2 k_2 * \approx 0 \;\&\; k_1 * \approx 0)$$

To verify if w_{opt} converges at k_2, the derivation is done as seen below. Firstly the $E[z(t)^2]$ is expanded.

$$E[z(t)^2] = E[e(t)^2] = E[(d(t) - w(t)x(t))^2]$$

for small values of $w(t)$,

$$E[z(t)^2] \approx E[d(t)^2] - 2w(t)E[d(t)x(t)]$$

The approximation made above is valid for the range of the weight tap that lies at about -30dB and below. Desired is a tap weight that gives $E[z(t)^2]=0$ [13]-[19].

Therefore the optimal tap weight is at:

$$w_{opt}(t) = \frac{E[d(t)^2]}{2E[d(t)x(t)]}$$

$E[d(t)^2]$ and $E[d(t)x(t)]$ can be expressed as follows

$$E[d(t)^2] = 2k_2 E[s(t)s^*(t)]$$

$$E[d(t)x(t)] \approx E[s(t)s^*(t)]$$

The point of convergence is at:

$$w_{opt}(t) = \frac{2k_2 E[s(t)s^*(t)]}{2E[s(t)s^*(t)]} = k_2$$

It can be seen above, that in the circularity algorithm the tap converges at k_2, unlike in the LMS algorithm. The performance of the circularity based BSS algorithm for a single tone CW signal is shown in figure 16 and the performance with a multi tone signal is shown in figure 17.

The circularity algorithm shares many similar traits with the LMS algorithm. The key differences are that the circularity algorithm does not need an additional signal as reference for the adaptation process, the signal itself is the reference. It is more sensitive towards frequency components close to DC.

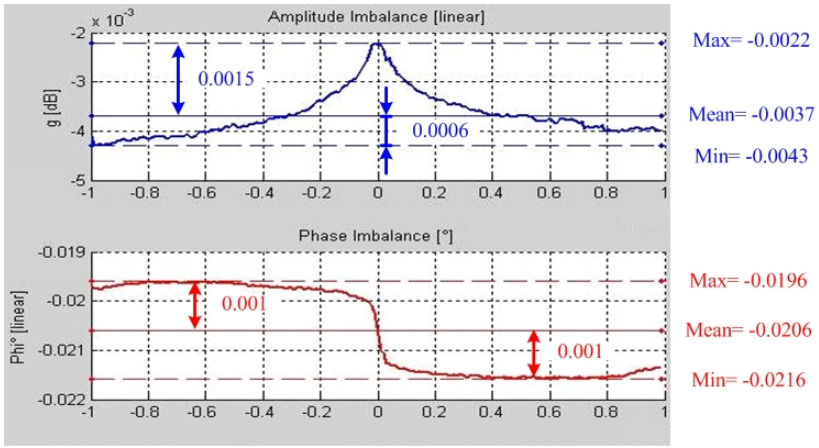


Figure 18: Worst case IRR with IQ mismatch variance across the bandwidth

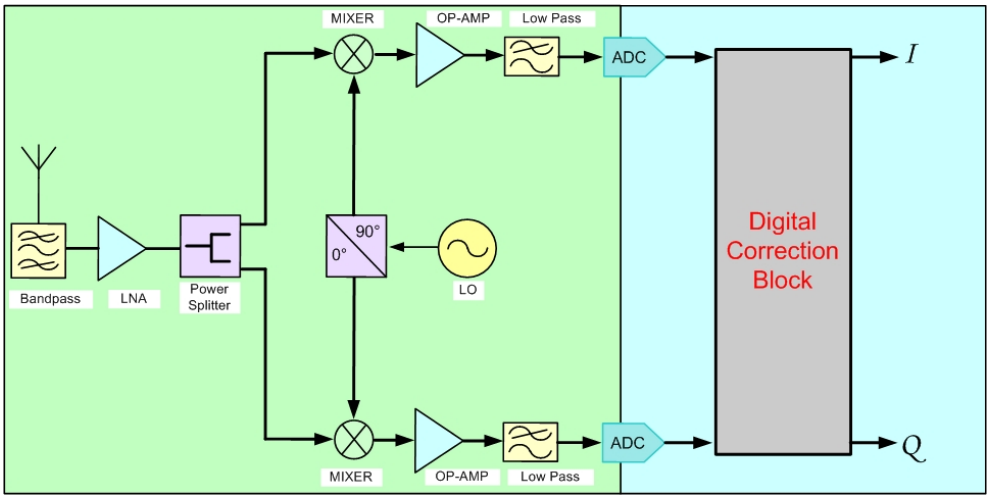


Figure 19: Homodyne receiver foreseen with digital correction block

VIII. Limitations & Constraints of Adaptive IQ Correction Algorithms

Adaptive algorithms have a convergence issue. Convergence is effected by the number of taps of the adaptive algorithm and it becomes more difficult with increasing number of taps (max. 4 taps). The one-tap variant offers best convergence, however with the drawback that it cannot correct frequency dependent IQ mismatch. The worst case IRR of a homodyne receiver using only one-tap depends on the IQ mismatch variance across the band width. For a set of measured homodyne receivers, due to their IQ mismatch variance across the band width, the worst case IRR lies at about 60 dB. This corresponds to a remaining amplitude mismatch of 0.1% and a phase mismatch of 0.1°.

Attempting to achieve this improvement via analog means would have lead to high circuit complexity, increased costs and lead to integration challenges. Via digital correction, the same result is achieved while keeping the components simple, without creating IC integration challenges and mainting a low production cost.

IX. Conclusion

Heterodyne receivers offer better dynamic range performance but their demanding analog components are difficult to integrate as IC solutions. Homodyne receivers have simple analog components suitable for IC solutions but their performance degrades due to component mismatches. The concept of digitally assisted analog is to provide correction algorithms to mitigate the effects of analog non-idealities (amplitude & phase mismatch).

Adaptive algorithms for correcting IQ mismatches are analyzed and are of interest because the same algorithm can be mapped as hardware on each chip. By providing digital correction algorithms, the homodyne architecture can be realized as a cost effective IC receiver solution without much compromise on the system performance.

ACKNOWLEDGEMENT

S. Singh on the behalf of Aalen University of Applied Sciences would like to thank Cassidian for supporting this thesis research work and enabling this contribution.

REFERENCES

[1] V. Giannini, J. Craninckx, A. Baschirotto, "Baseband Analog Circuits for Software Defined Radio", *Springer* 2008.

[2] K. Okada, S. Kousai, "Digitally-Assisted Analog and RF CMOS Circuit Design for Software-Defined Radio", *Springer* 2011.

[3] M. Mailand, R. Richter and H.-J. Jentschel, "IQ-imbalance and its compensation for non-ideal analog receivers comprising frequency-selective components", Dresden University of Technology; *Advances in Radio Science,* 2006, www.adv-radio-sci.net/4/189/2006/ .

[4] S. S. Haykin, "Adaptive Filter Theory" Fourth Edition, *Prentice Hall* 2002.

[5] D. G. Manolakis, V. K. Ingle and S.M. Kogon, "Statistical and Adaptive Signal Processing: Spectral Estimation, Signal Modeling, Adaptive Filtering and Array Processing", *Artech House* 2005.

[6] M. Valkama, M. Renfors, V. Koivunen, "Blind I/Q signal separation-based solutions for receiver signal processing", *EURASIP Journal on applied signal processing* 2005.

[7] M. Valkama, M. Renfors, V. Koivunen, "Compensation of frequency selective I/Q imbalances in wideband receivers: Models and algorithms", Mikko Valkama, Markus Renfors, Visa Koivunen, *Proc. IEEE Transaction on Signal Processing,* March 2001.

[8] M. Valkama, M. Renfors, V. Koivunen, "Blind source separation based I/Q imbalance compensation", *Proc. IEEE Transaction on Signal Processing*, October 2001.

[9] M. Valkama, M. Renfors, V. Koivunen, "On the performance of the interference canceller based I/Q imbalance compensation", *Proc. IEEE International Conference on Acoustics, Speech and Signal Processing*, June 2000.

[10] M. Valkama, M. Renfors, "Advanced DSP for I/Q imbalance compensation in a Low-IF receiver", Proc. IEEE International Conference on Communications, June 2000.

[11] Y. Li, "A Novel Adaptive Mismatch Cancellation System for Quadrature IF Radio Receivers", Ottawa-Carleton Institute for Electrical Engineering, Department of Electronics, Carleton University, Canada 1998.

[12] M. Chakraborty, "Adaptive Signal Processing Lecture Notes", Electronics & Electrical Communication Engineering Indian Institute of Technology Kharagpur, 2007.

[13] L. Antilla, M. Valkama, M. Renfors, "On Circularity *of Receivers Front-End Signals under RF Impairments", Proc. IEEE European Wireless Conference Vienna Austria*, April 2011.

[14] M. Petit, W. Haselmayr, A. Springer, "Blind Compensation of I/Q Filter Imbalance in the LTE Downlink" *Proc. IEEE European Wireless Conference Vienna Austria,*April 2011.

[15] L. Antilla, M. Valkama, M. Renfors, "Circularity-Based I/Q Imbalance Compensation in Wideband Direct-Conversion Receivers" *Proc. IEEE Transaction on Vehicular Technology,* July 2008.

[16] L. Antilla, M. Valkama, M. Renfors, "Frequency-Selective I/Q Mismatch Calibration of Wideband Direct-Conversion Transmitter", *Proc. IEEE Transaction on Circuits and Systems II*, April, 2008.

[17] L. Antilla, M. Valkama, M. Renfors, "Gradient-Based Blind Iterative Techniques for I/Q Imbalance Compensation in Digital Radio Receivers", *Proc. IEEE on Signal Processing Advances in Wireless Communications,* June 2007.

[18] L. Antilla, M. Valkama, M. Renfors, "Blind Compensation of Frequency-Selective I/Q Imbalances in Quadrature Receivers: Circularity-Based Approach", *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing,* April, 2007.

[19] L. Antilla, M. Valkama, M. Renfors, "Blind Moment Estimation Techniques For I/Q Imbalance Compensation In Quadrature Receivers" *Proc. IEEE International Symposium on Personal, Indoor and Mobile Radio Communications,* September 2006.

Simran Singh pursued the academic degree Dipl.-Ing. (FH) in electronics at the Aalen University of Applied Sciences, which he completed in December 2009. He then continued with his master studies in computer controlled systems, having received the M.Sc. degree in February 2012.

Heinz-Peter Bürkle received the academic degree Dipl.-Ing. in electrical engineering in 1991 from the University of Stuttgart and the degree Dr.-Ing. in 1997 also from the University of Stuttgart. After several years in R&D at Alcatel he is a Professor of semiconductor technology, circuit design and computer-aided circuit design at the Aalen University of Applied Sciences since 2003.

Georg Vallant received the B.Eng. in Electrical Engineering and M.Eng in Systems Engineering from the Hochschule Ulm in 2008 and 2010, respectively. Since 2010 he is PhD student at Karlsruhe Institute of Technology (KIT), working as a researcher at Cassidian, Ulm. In 2011 he was a visiting researcher at Tampere University of Technology, Finland.

Michael Epp received his Dipl.-Ing. (FH) from the Kempten University of Applied Sciences in 1997. Since 1997 he is working at CASSIDIAN in the department "Programmable Hardware" as specialist for analog-to-digital converters.

hochschule aschaffenburg
university of applied sciences

Echtzeit-BLOB-Analyse mit Lauflängenkodierung und -dekodierung auf einem FPGA

Felix Fellhauer, Marc Schmitt, Konrad Doll

***Zusammenfassung*—Die Analyse zusammenhängender Objekte (Binary Large Objects, BLOBs) ist ein weit verbreitetes Verfahren in der Bildverarbeitung. In vielen Fällen ist es aufgrund von Echtzeitanforderungen notwendig, spezielle Hardware wie Field Programmable Gate Arrays (FPGAs) zu verwenden. Beim Einsatz von FPGAs besteht die Herausforderung darin, Bilder in nur einem Durchlauf zu verarbeiten, um das Speichern kompletter Bilder auf dem FPGA zu vermeiden. Bisher vorgeschlagene Verfahren verwenden die Zeit während des Zeilenrücklaufs, um W-förmige Strukturen mit iterativen Algorithmen zu verarbeiten. Die Implementierung dieser iterativen Algorithmen benötigt auf FPGAs eine große Anzahl von Logikgattern. Eine Verwendung von Lauflängenkodierung kann diese Strukturen vermeiden. Bisher vorgeschlagene, auf Lauflängenkodierung basierende Ansätze können jedoch eine Echtzeit-BLOB-Analyse nur unter der Einschränkung realisieren, dass mit ihr eine Mindestkompression erreicht wird. Dieser Beitrag stellt einen Algorithmus vor, der Lauflängenkodierung und -dekodierung simultan anwendet, um beliebige Bildinhalte ohne Einschränkung in Echtzeit verarbeiten zu können. Das neue Verfahren wurde auf einem FPGA-System realisiert und kann Objekteigenschaften wie Extremkoordinaten, Fläche oder Anzahl der enthaltenen Löcher extrahieren. Tests haben gezeigt, dass der verwendete Algorithmus komplexe Bilder in Echtzeit verarbeiten kann, ohne bei der Lauflängenkodierung auf eine Mindestkompression angewiesen zu sein. Die vorgestellte FPGA-Implementierung kann hochauflösende Bilder mit einer Bildwiederholrate von 50 Hz verarbeiten.**

***Schlüsselwörter*—BLOB-Detection, Connected Component Labeling, Union-Find, FPGA.**

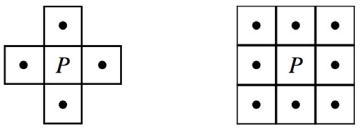
Felix Fellhauer, f.fellhauer@gmail.com, Marc Schmitt, schmitt.marc@gmail.com und Konrad Doll, konrad.doll@h-ab.de, sind Mitglieder der Hochschule Aschaffenburg, Fakultät Ingenieurwissenschaften, Labor für Rechnergestützten Schaltungsentwurf, Würzburger Straße 45, 63743 Aschaffenburg.

I. EINLEITUNG

In vielen Fällen kann für Aufgaben wie das Klassifizieren, Sortieren, Zählen oder Erkennen von Objekten in der Bildverarbeitung die Analyse zusammenhängender Objekte eingesetzt werden. Daher ist die BLOB-Analyse ein bewährtes Verfahren, um mit relativ geringem Aufwand Objekteigenschaften zu extrahieren.

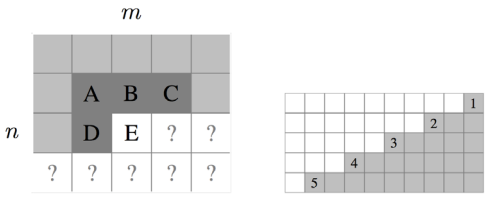
Um zusammenhängende Objekte zu ermitteln und deren Eigenschaften zu extrahieren, wurden bereits verschiedene Verfahren vorgestellt. Beim einfachsten Ansatz wird das gesamte Bild zweimal durchlaufen [1, 2]. Der erste Durchlauf erfolgt von links oben nach rechts unten. Hierbei werden alle Vordergrundpixel mit einem Label markiert. Zeitgleich wird eine Äquivalenztabelle erstellt, die angibt, welche Label als äquivalent und damit zu einem zusammenhängenden Objekt gehörig betrachtet werden müssen. Beim zweiten Durchlauf, der in entgegengesetzter Richtung erfolgt, werden alle Label entsprechend den Einträgen in der Äquivalenztabelle ersetzt. Danach steht ein Bild zur Verfügung, in dem alle Vordergrundpixel eindeutig einem Objekt zugeordnet sind. Bei dieser Art der BLOB-Analyse muss das gesamte Eingangsbild gespeichert werden. Je nach Auflösung des Eingangsbildes wird hierzu erheblicher Speicherplatz benötigt. Die Realisierung dieses Speichers für Hardwareimplementierungen kann teuer sein, wodurch dieser Ansatz nur für die Implementierung in Software interessant ist. Der notwendige Speicherbedarf kann verringert werden, wenn das Eingangsbild Pixel für Pixel innerhalb nur eines Durchlaufs verarbeitet wird. Aus diesem Grund sind Algorithmen mit nur einem Durchlauf für die Implementierung in Hardware von Vorteil. Derartige Ansätze ermöglichen die einfache, performante und flexible Implementierung zum Beispiel in modernen FPGAs.

Zur Erkennung zusammenhängender Objekte wurden bereits Ansätze mit nur einem Durchlauf vorgestellt, z.B. [3, 4]. In [3] wird das Bild ähnlich einer Faltung mit einer Nachbarschaftsmatrix durchlaufen. Auch hier müssen Bilddaten gespeichert werden, jedoch nur in Form einer einzigen Bildzeile.



(a) (b)

Abbildung 1: Darstellung zweier möglicher Modelle für Nachbarschaftsbeziehungen. Die zu *P* benachbarten Pixel sind mit einem Punkt gekennzeichnet. (a) 4-Pixel-Nachbarschaft, (b) 8-Pixel-Nachbarschaft.



(a) (b)

Abbildung 2: (a) Aufgrund der seriellen Datenübertragung von Digitalbildern stehen für die Pixel-für-Pixel Bearbeitung nur die bereits übertragenen Bildpunkte zur Verfügung. (b) Objekt, für das im Verlauf insgesamt fünf verschiedene Label benötigt werden.

Der Verzicht auf eine umfangreiche Datenpufferung macht es notwendig, ein Modul zu verwenden, das ähnlich einer Äquivalenztabelle gleichwertige Label noch während des ersten Durchlaufs auflöst. Es wurde gezeigt, dass die Verwendung von Lauflängenkodierung die Auflösung der Tabelleneinträge vereinfacht und den benötigten Speicherbedarf deutlich reduzieren kann [5]; dies jedoch nur, wenn durch die Lauflängenkodierung eine Mindestkompression erreicht wird. Der vorliegende Beitrag beschreibt einen Ansatz, der die Vorteile der Lauflängenkodierung ohne Einschränkungen nutzbar macht.

Die folgenden Abschnitte beschreiben den Algorithmus und die erzielten Ergebnisse: Abschnitt II führt in die Algorithmus-Grundlagen ein. In Abschnitt III wird auf die Ermittlung von Objekteigenschaften eingegangen. Die einzelnen Schritte des vorgestellten Ansatzes werden in Abschnitt IV zusammenfassend erläutert. Abschnitt V zeigt schließlich die Ergebnisse und die benötigten Ressourcen einer Hardwareimplementierung des vorgestellten Verfahrens auf einem FPGA.

II. ALGORITHMUS

Ausgangspunkt einer BLOB-Analyse ist ein Bild, in dem Vordergrund und Hintergrund bereits getrennt sind. Je nach Anwendung kann diese Trennung mit Hilfe eines einfachen Schwellwertfilters oder einer Vordergrund-Hintergrund Segmentierung erfolgen. In jedem Fall liegt das Bild nach dem Vorverarbeitungsschritt als binäre Matrix mit den Elementen (*m, n*) und einer Farbtiefe von einem Bit vor. In dieser Arbeit werden Vordergrundpixel schwarz und Hintergrundpixel weiß dargestellt. Die Aufgabe der BLOB-Analyse ist nun, in diesem binären Bild zusammenhängende Objekte, wie im Folgenden definiert, zu finden und gegebenenfalls deren Eigenschaften zu berechnen.

Ein Pixel ist dann benachbart zu einem Pixel P, wenn zwischen den beiden Pixeln eine Nachbarschaftsbeziehung existiert (siehe Abbildung 1). Man unterscheidet zwischen 4-Pixel- und 8-Pixel-Nachbarschaften. Der in diesem Beitrag vorgestellte Algorithmus bezieht sich auf eine 8-Pixel-Nachbarschaft, kann jedoch auf eine 4-Pixel-Nachbarschaft übertragen werden. Zwei Pixel sind miteinander verbunden, wenn es zwischen den beiden Pixeln einen Pfad bestehend aus Vordergrundpixeln gibt, wobei für jedes Pixel in dem Pfad gilt, dass es mit dem Nachfolgepixel benachbart ist. Eine Menge von Pixeln bildet ein zusammenhängendes Objekt, wenn alle Pixelpaare in dem zusammenhängenden Objekt verbunden sind.

Jedes der Vordergrundpixel wird mit einem Label hinsichtlich seiner Objektzugehörigkeit beschrieben. Pixel mit gleichem Label gehören dem gleichen Objekt an. Bei dem vorgestellten Ansatz beschreiben alle Label größer 0 die Zugehörigkeit zu einem zusammenhängenden Objekt. Das Label 0 wird für alle Hintergrundpixel vergeben.

Bei der Erkennung zusammenhängender Objekte ohne Bildpufferung soll jeder Bildpunkt schon zum Zeitpunkt seiner Übertragung eindeutig einem zusammenhängenden Objekt zugeordnet werden. Dies hat für die in Abbildung 1 gezeigten Nachbarschaftsbeziehungen zur Folge, dass zur Zuordnung nur die Bildpunkte analysiert werden können, die bereits übertragen wurden [6]. Abbildung 2a zeigt das aktuelle Pixel E mit den Koordinaten (*m, n*) und den bekannten Nachbarn A, B, C, D. Die Bildpunkte rechts und unterhalb des aktuellen Pixels (in Abb. 2a mit ”?“ markiert) können nicht betrachtet werden, da sie zu diesem Zeitpunkt noch nicht übertragen sind.

Die Zuweisung eines Labels für E geschieht in Abhängigkeit der Labelwerte der Bildpunkte A bis D. Die Label, die jeweils zwischen den Label C und D liegen ($m+1, n-1$)...($m-1, n$), werden in einem Puffer zwischengespeichert. Prinzipiell sind Ansätze denkbar, bei denen die Nachbarschaftsbeziehung vollständig überprüft wird. Hierzu wäre es jedoch notwendig, zwei statt nur einer Bildzeile zu puffern.

Bei U-förmigen Strukturen müssen äquivalente Objekte ”verschmolzen“ werden. Um die Objektzugehörigkeit bereits im Puffer befindlicher Bildpunkte ändern zu können, wird dem Labelpuffer ein weiteres Modul nachgeschaltet: die Merger-Tabelle. Für die Berechnung der Eigenschaften erkannter Objekte wird von den entsprechenden Eigenschaftsmodulen Label D beobachtet und somit die Objektzugehörigkeit des zuletzt klassifizierten Bildpunktes. Die einzelnen Schritte werden in den folgenden Abschnitten detailliert erläutert.

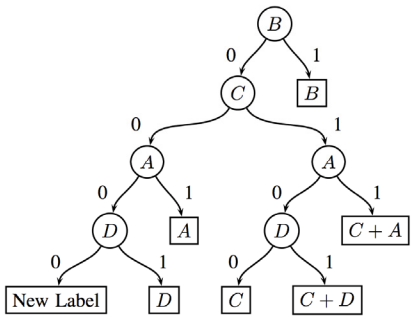


Abbildung 3: Die Zuweisung der Label geschieht unter Betrachtung der bereits zugewiesenen Label. Für die Entscheidung ist ausschlaggebend, ob der jeweilige Bildpunkt Vordergrund (’1’) oder Hintergrund (’0’) ist.

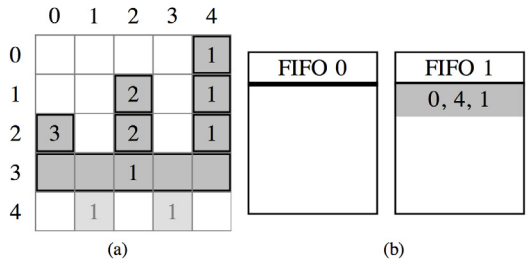


Abbildung 4: (a) Zuweisung der Objektzugehörigkeit pro Run (b) Inhalt des lauflängen-basierten Label-Puffers am Ende von Bildzeile 3 in (a): Der Inhalt des FIFO 0 wurde bereits dekodiert und ausgegeben. FIFO 1 beinhaltet Bildzeile 3.

A. Zuweisung der Label

Die Zuweisung eines Labels für Pixel E kann nach einem allgemeinen Schema erfolgen. Sind alle umliegenden Pixel Hintergrundpixel, wird ein bisher nicht verwendetes Label zugewiesen. Für den Fall, dass in der Nachbarschaft nur ein von 0 verschiedenes Label existiert, wird dieses Label vergeben. Sind unter den benachbarten Bildpunkten verschiedene Label vergeben, so wird immer das Label mit der kleinsten Zahl für Pixel E verwendet. In diesem speziellen Fall müssen zwei BLOB-Regionen zu einer verschmolzen werden. Dieser Prozess bedarf darüber hinaus gesonderter Betrachtung und wird in Abschnitt II-C ausführlich behandelt.

Die Labelzuweisung kann mit Hilfe eines Entscheidungsbaumes veranschaulicht werden [7] (siehe Abbildung 3). Der gezeigte Entscheidungsbaum verwendet als Entscheidungsparameter nicht die Label der einzelnen Bildpunkte, sondern trifft die Entscheidung allein aufgrund der Vorder- bzw. Hintergrundinformation der Nachbarpixel. Unter Verwendung dieses Entscheidungsbaums kann in der Hardwareimplementierung auf die Verwendung großer und damit langsamer Komparatoren nahezu vollständig verzichtet werden.

Die maximale Anzahl zusammenhängender Objekte muss im Vorfeld festgelegt werden. Hiervon ist die Länge des Binärwortes abhängig, welches das Label jedes Bildpunktes beschreibt. Je nach Eingangsbild können deutlich mehr Label benötigt werden, als Objekte im Bild vorhanden sind. Abbildung 2b zeigt ein relativ einfaches zusammenhängendes Objekt, für das insgesamt fünf Label benötigt werden.

B. Label-Pufferung

Um Zugriff auf die bereits zugewiesenen Label zu erhalten, müssen alle Label zwischen C und D gespeichert werden. Bestehende Ansätze verwenden hierzu einen Pufferspeicher, in dem für jedes zugewiesene Label der gesamten letzten Bildzeile ein Speicherplatz bereit gestellt wird [8]. Alle weiter zurück liegenden Pixel brauchen nicht mehr beachtet zu werden. Dieser Beitrag verfolgt für die Pufferung dieser Label einen Ansatz unter der Verwendung von Lauflängenkodierung, denn damit kann der Speicherbedarf in praktischen Anwendungen wesentlich reduziert werden.

Lauflängenkodierung ist eine in der Signal- und Bildverarbeitung häufig eingesetzte Methode zur Kompression von immer wiederkehrenden Signalwerten. Bei der BLOB-Analyse findet eine Wiederholung von Information immer dann statt, wenn mehrere nacheinander übertragene Pixel zum Vordergrund gehören. Damit sind sie benachbart und gehören demselben zusammenhängenden Objekt an.

Für die Anwendung der Lauflängenkodierung zur Pufferung von Label werden alle innerhalb einer Zeile zusammenhängenden Vordergrund-Pixel zu einem Run komprimiert. Die Angabe des Labels geschieht dann nicht für jeden Bildpunkt separat, sondern immer für einen ganzen Run. Der benötigte Speicher für einen solchen Run ist unabhängig von der Anzahl der in einem Run enthaltenen Bildpunkte. Es werden immer die erste und letzte x-Koordinate eines Runs, sowie das Label des ganzen Runs gespeichert.

Die Zuweisung der Label erfolgt jedoch, wie in Abschnitt II-A beschrieben, pixelweise. Daher müssen die lauflängenkodierten Informationen nach der Kodierung wieder pixelweise dekodiert und ausgegeben werden. Nur so kann die Betrachtung der benachbarten Bildpunkte mit Hilfe des vorgestellten Verfahrens realisiert werden.

Die Realisierung der Label-Pufferung geschieht am einfachsten mit Hilfe zweier unabhängiger Puffer zur Speicherung der lauflängenkodierten Informationen, da in diesem Fall für die Dekodierung wesentlich weniger Aufwand notwendig ist. Ein Beispiel für diese Vorgehensweise ist in Abbildung 4a und 4b aufgeführt. Einer der beiden Puffer wird immer für gerade Zeilen beschrieben, der andere für ungerade. Die Dekodierung der gespeicherten Daten beim Lesen erfolgt simultan zur Kodierung aus dem jeweils anderen Pufferspeicher. Der Puffer mit Lauflängen-

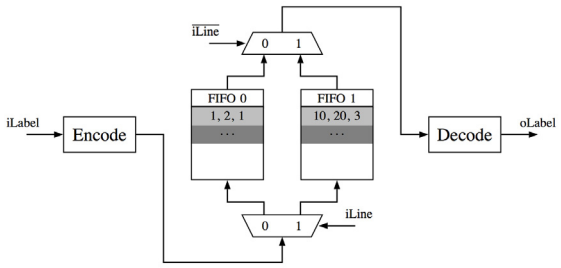


Abbildung 5: Der lauflängen-basierte Zeilenpuffer kodiert die Label der aktuellen Zeile und legt diese in einem FIFO-Speicher ab. Zeitgleich werden die kodierten Label der vorhergehenden Zeile aus dem zweiten FIFO-Speicher dekodiert und ausgegeben.

kodierung weist genau dieselben Schnittstellen auf wie eine Pufferimplementierung mit Zeilenspeicher. Die Abbildung der erwähnten Methodik in Hardware geschieht mittels zweier FIFO-Pufferspeicher, die über einen Multiplexer abwechselnd beschrieben und gelesen werden (Abbildung 5).

Für die ersten drei Label in jeder Zeile müssen gesonderte Register zur Verfügung gestellt werden, da diese bereits ausgelesen werden müssen, bevor das aktuelle Pixel E die nächste Zeile erreicht. Der für den Puffer benötigte Speicher $D(n, i, j)$ hängt von der Breite des Eingangsbildes n, der maximalen Anzahl erkennbarer Objekte i und der maximalen Anzahl der Runs j in einer Zeile ab. Gleichung 1 zeigt diesen Zusammenhang.

$$D(n, i, j) = \left(2 \cdot \log 2\ (n) + \log 2\ (i)\right) \cdot 2j \qquad (1)$$

Hieraus ergibt sich ein wesentlicher Vorteil dieses Verfahrens. Der Speicherbedarf ist nicht in erster Linie von der Auflösung des Eingangsbildes abhängig, wie es bei der Verwendung von einfachen Zeilenpuffern der Fall ist (Gleichung 2), sondern von der Anzahl der voraussichtlich enthaltenen Objekte.

$$D(n, i) = \log 2\ (i) \cdot n \qquad (2)$$

Die maximale Anzahl der Runs pro Bildzeile muss für diesen Ansatz im Vorfeld dimensioniert werden, was für die meisten Anwendungen keine Einschränkung darstellt. Zusätzlich zum geringeren Speicherbedarf ergibt sich bei der Verwendung von Lauflängenkodierung ein weiterer, wesentlicher Vorteil. Das Verschmelzen von BLOB-Regionen kann deutlich einfacher durchgeführt werden, da nun innerhalb eines Runs keine Inkonsistenzen bezüglich der Objektzugehörigkeit auftreten können. Die folgenden beiden Abschnitte behandeln diesen Aspekt näher.

C. Verschmelzen zusammenhängender Objekte

Wie in Abschnitt II-A beschrieben, müssen zusammenhängende Objekte unter bestimmten Umständen verschmolzen werden. Dies bedeutet, dass Objekte,

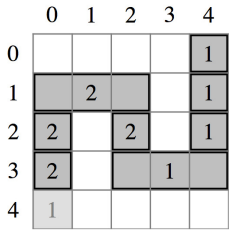


Abbildung 6: Darstellung eines Falles, bei dem die Zuweisung der Objektzugehörigkeit in der letzten Bildzeile nur unter der Verwendung einer Merger-Tabelle richtig durchgeführt werden kann.

die zu Beginn der Datenübertragung noch als zwei verschiedene Objekte mit unterschiedlichem Label behandelt wurden, im weiteren Fortschritt der Übertragung zu einem Objekt vereinigt werden. Das ist beispielsweise bei U-förmigen Strukturen der Fall. Die beiden vertikalen Zweige eines Us sind zunächst alleinstehende Objekte. Erst bei der Übertragung der horizontalen Verbindung wird klar, dass die bis dahin unabhängigen Objekte zum selben zusammenhängenden Objekt gehören.

Das Auftreten solcher Situationen erzeugt jedoch Probleme, da hierbei die Objektzugehörigkeit von Vordergrundpixeln verändert werden muss, auch wenn diese bereits dem Label-Puffer übergeben wurden. Aus diesem Grund folgt dem Label-Puffer ein weiteres Modul, das es ermöglicht, bereits gespeicherte Label zu verändern. Ein eindimensionaler Vektor, die Merger-Tabelle, enthält für jedes verwendete Label einen Eintrag, der bei erstmaliger Zuweisung eines Labels auf sich selbst verweist. Werden zwei BLOB-Regionen zusammengeführt, so kann dieser Eintrag abgeändert werden und alle bisher unter diesem Label gespeicherten Einträge werden auf den neuen verwiesen. Die beiden Label werden dann als äquivalent betrachtet. Durch die Merger-Tabelle findet also eine Abbildung der im Label-Puffer gespeicherten Label statt. Abbildung 6 zeigt einen Fall, bei dem der Eintrag in der Merger-Tabelle für Label 2 zum Zeitpunkt der Übertragung von Pixel (3, 3) geändert wird und dann auf Label 1 zeigt. Dies hat zur Folge, dass in Zeile 4 statt erneut Label 2 nun Label 1 zugewiesen wird.

Im Falle eines Verschmelzens zweier Objekte können mit der Merger-Tabelle alle im Puffer befindlichen Label abgeändert werden. Dies hat zunächst keine Auswirkung auf die Label, die bereits dekodiert wurden und in den Registern A, B und C gespeichert sind. Für diese Register muss deswegen mit Hilfe eines Eingangsmultiplexers dafür gesorgt werden, dass alle Einträge des alten Labels durch das neue ersetzt werden. Aufgrund der Zuweisung der Label pro Run kann mit der Lauflängenkodierung in vielen Fällen die Objektzugehörigkeit auch ohne die Verwendung einer Merger-Tabelle korrekt bestimmt werden. Je nach Komplexität der zu erkennenden Objekte kann daher auf die Merger-Tabelle verzichtet werden.

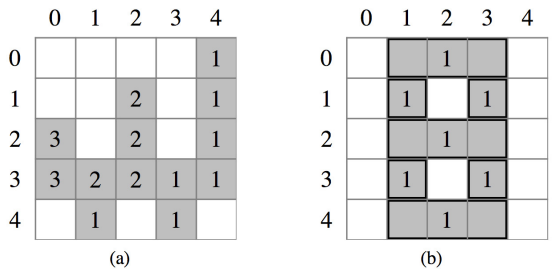


Abbildung 7: (a) Label mit einfachem Zeilenpuffer ohne Lauflängenkodierung zwischengespeichert (b) Objekt mit zwei Löchern

D. Chaining in der Merger-Tabelle

Unter Verwendung eines einfachen Zeilenpuffers zum Zwischenspeichern der Label kann es selbst bei einfachen Strukturen zu verketteten Einträgen in der Merger-Tabelle kommen. Abbildung 7a zeigt einen solchen Fall: Zu Beginn von Bildzeile 3 zeigen die einzelnen Elemente der Merger-Tabelle *MT* jeweils auf sich selbst, da jeder der drei Zweige bis dahin ein unabhängiges Objekt darstellt *MT* = [0, 1, 2, 3]. Beim weiteren Fortschreiten der Übertragung ergibt sich zunächst, dass bei Pixel (1, 3) von Label 3 auf Label 2 verwiesen (*MT* = [0, 1, 2, 2]) und schließlich bei Pixel (3, 3) von Label 2 auf Label 1. Am Ende von Zeile 3 lautet der Inhalt der Merger-Tabelle: *MT* = [0, 1, 1, 2]. Durch den nicht unmittelbar terminierten Eintrag am vierten Element wird ohne Gegenmaßnahmen in Zeile vier an der Stelle (1, 4) fälschlicher Weise Label 2 statt Label 1 vergeben. Um solche Einträge in der Merger-Tabelle zu vermeiden, wird in bestehenden Verfahren Dechaining (Aufräumen der Merger-Tabelle) mit Hilfe von iterativen Algorithmen durchgeführt [8]. Dies kann jedoch nur am Ende einer Zeile geschehen, weswegen man auf eine Pause zum Aufräumen der Merger-Tabelle angewiesen ist. Zusätzlich benötigen die notwendigen iterativen Algorithmen in Hardware mehrere Taktzyklen, was unter Umständen die Echtzeitfähigkeit des Gesamtsystems beeinträchtigt.

Bei Anwendung eines lauflängen-basierten Label-Puffers kann dieselbe Struktur problemlos verarbeitet werden (siehe Zeile 3 in Abb. 4a im Gegensatz zu Abb. 7a). Die nicht terminierten Einträge in der Merger-Tabelle bleiben zwar weiterhin bestehen, jedoch werden diese Einträge in den nachfolgenden Zeilen nicht mehr verwendet.

III. Extraktion von Eigenschaften

Bisher vorgestellte Ansätze unter der Verwendung von Lauflängenkodierung benötigen nach Ende eines jeden Runs eine bestimmte Anzahl Taktzyklen, um Objektzugehörigkeit und entsprechende Objekteigenschaften zu berechnen [5]. Dies hat den Nachteil, dass bei Unterschreitung einer Mindestkompressionsrate die Echtzeitfähigkeit nicht gewährleistet ist. So kommt es je nach Bildinhalt zu Fehlern bei der BLOB-Analyse. Der in diesem Beitrag vorgestellte Ansatz ermöglicht es, die Berechnung der Objekteigenschaften simultan zur Labelzuweisung durchzuführen, wodurch Vordergrundpixel zu jedem Zeitpunkt verarbeitet werden können.

Die mit Hilfe des vorgestellten Verfahrens errechnete Objektzugehörigkeit dient als Grundlage, um verschiedene Eigenschaften der erkannten Objekte zu bestimmen. Hierzu wird das neu zugewiesene Label und somit die Objektzugehörigkeit des gerade übertragenen Bildpunktes betrachtet. Im Falle von Verschmelzungen werden die jeweiligen Eigenschaften entsprechend akkumuliert. Um diesen Vorgang in Hardware abbilden zu können, muss zur Speicherung der Objekteigenschaften ein vollwertiges Dual-Port-RAM verwendet werden, da ein Region-Merging nur so innerhalb eines einzigen Pixel-Taktes durchgeführt werden kann. Die im Folgenden vorgestellten Objekteigenschaften haben beispielhaften Charakter und können nach Belieben erweitert werden. Ermittelt werden die Anzahl der Objekte, die Extremkoordinaten der Objekte in x- und y-Richtung, die Objektfläche, sowie die Anzahl der enthaltenen Hohlräume für jedes Objekt.

A. Anzahl der Objekte

Das Zählen der im Bild enthaltenen zusammenhängenden Objekte ist eine der grundlegenden Aufgaben der BLOB-Analyse. Dieser Parameter kann mit Hilfe eines Binärzählers extrahiert werden, der nach folgendem Schema angesteuert wird:

- Wenn ein neues Label zugewiesen wird, erhöhe den Objektzähler um 1
- Wenn ein Region-Merge zweier unterschiedlicher Label stattfindet, erniedrige den Objektzähler um 1
- Findet ein Region-Merge mit zwei identischen Label statt, lasse den Zählerstand unverändert

B. Objektfläche

Eine weitere wichtige Objekteigenschaft ist die Fläche eines BLOBs. Hierzu kann mit einem Zähler die Anzahl der zu jedem BLOB zugeordneten Pixel ermittelt werden.

C. Extremkoordinaten

Die Extremkoordinaten jedes BLOBs lassen sich durch ständiges Vergleichen der aktuellen Übertragungsposition mit einem gespeicherten Minimum und Maximum in x- bzw. y- Richtung bestimmen. Die so gewonnene „Bounding-Box“-Eigenschaft kann zur Berechnung des Mittelpunktes der umschließenden Rechtecke für die erkannten zusammenhängenden Objekte herangezogen werden:

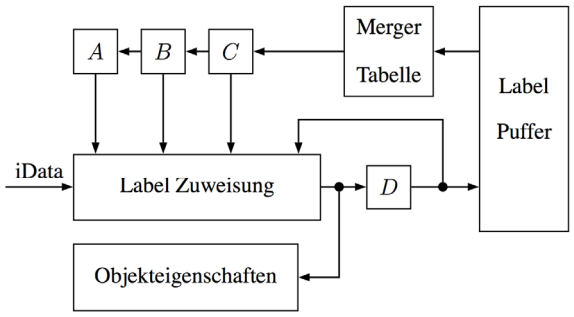


Abbildung 8: *Hardware Architektur* - Gegenüber dem Ansatz mit einfachem Zeilenpuffer (pixelweise) wird dieser hier durch einen Puffer mit Lauflängenkodierung und -dekodierung ersetzt. Dieser stellt die selben Schnittstellen, wie ein einfacher Zeilenpuffer zur Verfügung.

$$x_c = x_{\min} + \frac{x_{\max} - x_{\min}}{2} \quad y_c = y_{\min} + \frac{y_{\max} - y_{\min}}{2} \quad (3)$$

Weiter kann über die Extremkoordinaten eines zusammenhängenden Objektes ein Seitenverhältnis berechnet werden, dessen Betrachtung in vielen Fällen einen Beitrag zur Lösung einfacher Klassifikationsaufgaben leisten kann.

D. Anzahl der Hohlräume

Eine weitere Objekteigenschaft ist die Anzahl der enthaltenen Löcher in einem zusammenhängenden Objekt. Dies ist gleichzeitig ein Maß für die Komplexität eines Binärbildes. Um die Anzahl der Hohlräume zu extrahieren, wird ein einfacher Binärzähler implementiert. Dieser Zähler wird immer dann inkrementiert, wenn zwei zusammenhängende Objekte mit demselben Label verschmolzen werden.

Abbildung 7b zeigt ein zusammenhängendes Objekt, bei dem dieser Fall an den Stellen (2, 2) und (2, 4) auftritt. Die so extrahierten BLOB-Eigenschaften können als Klassifikationsmerkmale für verschiedene Aufgaben verwendet werden.

IV. Architektur des Gesamtsystems

Das in diesem Beitrag vorgestellte System besteht im Wesentlichen aus vier Modulen: Label-Zuweisung, Label-Puffer (auf Lauflängenkodierung basierend), Merger-Tabelle und Objekteigenschaften. Abbildung 8 zeigt ein schematisches Blockschaltbild der genannten Module. Der folgende Abschnitt fasst das Zusammenwirken der einzelnen Module zusammen.

Die binären Eingangsdaten werden zunächst im Block Label-Zuweisung verarbeitet. Dort wird festgestellt, ob es sich um ein Vordergrund- oder Hintergrundpixel handelt. Weiter wird auf der Grundlage der bereits bearbeiteten Bildpunkte entschieden, welchem zusammenhängenden Objekt der übertragene Bildpunkt (m, n) zugeordnet werden kann. Die Zuordnung ist an dieser Stelle noch nicht endgültig und kann sich, wie bereits erwähnt, im weiteren Verlauf der Bildübertragung ändern.

Die Register A, B, C, und D speichern Label der bereits zugewiesenen Pixel und stellen so die in Unterabschnitt II-A beschriebene Nachbarschaftsbeziehung her. Die Label, die jeweils zwischen Label C und D liegen (m+1, n−1)...(m−1, n), werden in einem Label-Puffer zwischengespeichert, der die Eingangsdaten zunächst unter der Verwendung von Lauflängenkodierung kodiert und zur Ausgabe wieder entsprechend dekodiert.

Um Änderungen der bereits zugewiesenen Label zu ermöglichen, werden die vom Label-Puffer ausgegebenen Label durch die Merger-Tabelle dekodiert, bevor sie den Registern A, B und C übergeben werden. Die Einträge in der Merger-Tabelle werden durch Steuersignale des Label-Zuweisungs-Moduls angelegt und bei Bedarf geändert. Um Eigenschaften der erkannten Objekte zu berechnen, betrachtet das Modul Objekt-Eigenschaften, die Objektzugehörigkeit aller Vordergrundpixel und implementiert den jeweiligen Algorithmus.

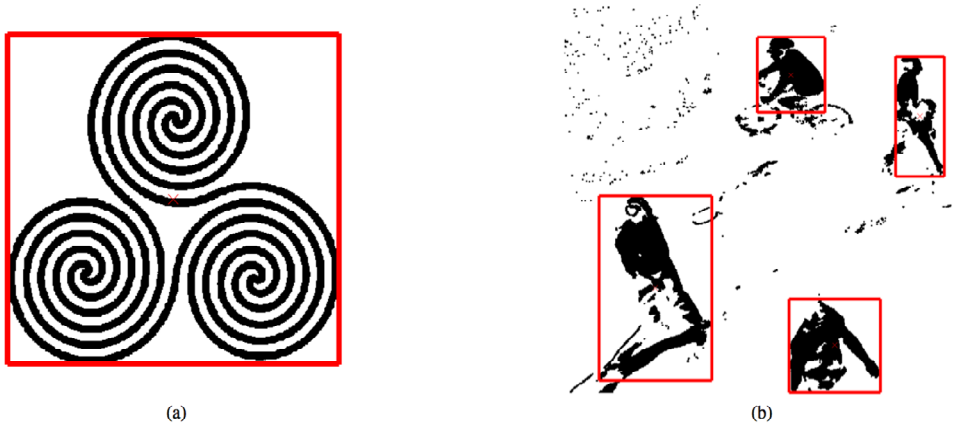
V. Experimentelle Ergebnisse

Um die Vergleichbarkeit der Ergebnisse zu gewährleisten, wird das System ohne die Extraktion von Eigenschaften betrachtet. Zur Evaluation des vorgestellten Ansatzes wurde das System mit VHDL implementiert. Als Testsystem dient ein Xilinx-FPGA[1] der Virtex-5-Reihe. Die Testbilder werden im Block-RAM abgelegt und mit Hilfe eines Bildgenerators ausgegeben. So können definierte und genau bekannte Testbilder reproduzierbar der BLOB-Analyse zugeführt werden. Der Ressourcenbedarf beträgt 5060 Slice-LUTs und 4814 Slice-Register. Bei der Implementierung wurde großer Wert auf die Flexibilität des Systems gelegt. Aus diesem Grund wird der benötigte Speicher mit Slice-Registern realisiert, was hier den relativ großen Ressourcenbedarf erklärt. In einer weiterführenden Implementierung könnten die genannten Speicher problemlos im Block-RAM eines FPGAs untergebracht werden.

Die angegebenen Daten entsprechen einer Implementierung für ein Eingangsbild mit 512 × 512 Bildpunkten und einer Labelbreite von 8 Bit. Für die Implementierung des auf Lauflängenkodierung basierenden Ansatzes wird eine maximale Anzahl von 50 Runs pro Bildzeile festgelegt. Dies genügt, um selbst komplizierte Eingangsbilder, wie in Abbildung 9a, zu verarbeiten.

Mit dem vorgestellten Ansatz kann ein Pixeltakt von 120 MHz verwendet werden, was bei einer Auflösung von 1920 × 1080 zu einer maximalen Bildwiederholrate von 57 Hz führt.

[1] www.xilinx.com



(a) (b)

Abbildung 9: *Testbilder* - Ergebnisse der BLOB-Analyse. (a) Testmuster mit einem einzigen zusammenhängenden Objekt, das in der Bildmitte einen Loch enthält. (b) Mittels Mixture of Gaussians vorverarbeitetes Testbild aus einer Situation im Straßenverkehr mit Schwellwert für die Fläche der erkannten zusammenhängenden Objekte.

Um die vorgestellte Implementierung zu testen, werden das BLOB-Analysemodul mit verschiedenen Testbildern beaufschlagt und die Ergebnisse mit einer MATLAB-basierten Implementierung des Verfahrens verglichen. Hierbei werden sowohl speziell erstellte künstliche Testbilder verwendet als auch Aufnahmen aus konkreten Anwendungsfällen.

Abbildung 9a zeigt eine Figur, die zu Beginn der Übertragung viele unabhängige Objekte enthält, die jedoch alle nach und nach zu einem Objekt verschmolzen werden. Zusätzlich beinhaltet das Muster einen Hohlraum in der Bildmitte. Ein weiteres Testbild in Abbildung 9b zeigt querende Fußgänger in einem an einer Ampelkreuzung aufgenommenen Bild. Das Ursprungsbild wurde zur Vordergrund-/Hintergrundtrennung bereits einer Vorverarbeitung mittels Mixture of Gaussians und anschließender Dilatation unterzogen [9, 10]. Das Ergebnis der BLOB-Analyse zeigt alle zusammenhängenden Objekte mit einer Fläche größer als 1000 Bildpunkte. Damit kann in dieser Anwendung eine Hypothese für eine Fußgängererkennung erstellt werden.

VI. Zusammenfassung

Lauflängenkodierung bietet in der BLOB-Analyse Vorteile bezüglich des Speicherbedarfes und des Bearbeitungsablaufes. So können Eingangsbilder verarbeitet werden, ohne dabei auf eine Pause bei der Übertragung angewiesen zu sein, um Dechaining durchzuführen [3]. Der Speicherbedarf ist vor allem für wenige Runs pro Bildzeile deutlich geringer als bei Verwendung eines gewöhnlichen Zeilenpuffers. Bisher vorgestellte Ansätze zur Erkennung zusammenhängender Objekte unter der Verwendung von Lauflängenkodierung bieten Einschränkungen, da eine gewisse Anzahl von Pixeltakten nach der Beendigung eines Runs benötigt wird, um den Run zu verarbeiten. Der vorgestellte Algorithmus ermöglicht es, eine BLOB-Analyse unter Verwendung von Lauflängenkodierung durchzuführen, ohne dabei bezüglich des Bildinhaltes eingeschränkt zu sein. Das vorgeschlagene Verfahren wurde mittels einer VHDL-Implementierung auf einem FPGA- System getestet. Die Ergebnisse zeigen, dass das Verfahren zur BLOB-Analyse in Echtzeit geeignet ist.

Literaturverzeichnis

[1] A. Rosenfeld, J. L. Pfaltz, “Sequential Operations in Digital Picture Processing”, *Journal of the Association for Computing Machinery* 13 (1966), October, Nr. 4, S. 471–494.

[2] D. R. Lee, S. H. Jin, P. C. Thien, J. W. Jeon, “FPGA based connected component labelling”, *Control, Automation and Systems*, 2007. ICCAS ’07. *International Conference on Control, Automation and Systems* 2007, 2007, S. 2313 – 2317.

[3] D. G. Bailey, C. T. Johnston, “Single Pass Connected Components Analysis”, *Proceedings of Image and Vision Computing* (2007).

[4] W. Rülling, „Realzeit-Bildverarbeitung auf einem FPGA“ *Workshop der Multiprojekt-Chip-Gruppe Baden-Württemberg*, Furtwangen, Juli 2011, Tagungsband 45.

[5] T. Trein, A. T. Schwarzbacher, B. Hoppe, “FPGA Implementation of a Single Pass Real-Time Blob Analysis Using Run Length Encoding”, *Workshop der Multiprojekt-Chip-Gruppe Baden-Württemberg*, 2008.

[6] K. Suzuki, I. Horiba, N. Sugieb, “Linear-time connected-component labeling based on sequential local operations”, *Computer Vision and Image Understanding* 89 (2003), Nr. 1, S. 1 – 23.

[7] K. Wu, E. Otoo, A. Shoshani, “Optimizing Connected Component Labeling Algorithms”, *Medical Imaging* (2005).

[8] C. T. Johnston, D. G. Bailey, “FPGA implementation of a Single Pass Connected Components Algorithm”, *IEEE International Symposium on Electronic Design, Test and Applications* (2008).

[9] C. Stauffer, W.E. Grimson, “Adaptive back- ground mixture models for real-time tracking”, “Computer Vision and Pattern Recognition”, 1999. *IEEE Computer Society Conference* Bd. 2, 1999, S. 246–252.

[10] J. Kempf, K. Doll, „Modulare Hardware-Software Bildverarbeitungsplattform am Beispiel einer Vordergrund-Hintergrundtrennung“, *Workshop der Multiprojekt-Chip-Gruppe Baden-Württemberg*, Furtwangen, Juli 2011, Tagungsband 45.

HOCHSCHULE KONSTANZ
TECHNIK, WIRTSCHAFT UND GESTALTUNG

Generierung von Codekomponenten für BCH-Encodierer

Jens Spinner, Jürgen Freudenberger

***Zusammenfassung*—Bose-Chaudhuri-Hocquenhen (BCH) Codes gehören zu den meist verwendeten Fehlerkorrekturcodes. Sie werden unter anderem bei DVB-T-Empfängern und zur Fehlerkorrektur bei Flash-Speichern eingesetzt. Dabei werden heute zunehmend Codes mit großen Codewortlängen und großer Fehlerkorrekturfähigkeit verwendet, was den Hardware-Entwurf für die Encoder und Decoder vor neue Herausforderungen stellt. In diesem Beitrag wird beschrieben, wie der Entwurf durch Werkzeuge zur Sourcecode-Generierung unterstützt werden kann.**

***Schlüsselwörter*—Codegenerierung, BCH-Codes, Flash-Speicher, Generatorpolynom, Encoder.**

I. Einleitung

Die Bedeutung von Flash-Speicherchips als Permanentspeicher nimmt stetig zu. Flash-Speicher bestehen aus Floating-Gate-Transistoren, in denen Informationen gespeichert werden. Diese Floating-Gates haben die besondere Eigenschaft, dass sie ihren Zustand auch ohne Energieversorgung behalten können. In der Praxis treten jedoch beim Auslesen des Zustands Fehler auf. Die Fehlerwahrscheinlichkeit hängt dabei von der Speicherdichte, der Flash-Technologie (multi-level cell oder singel-level cell) und der Anzahl der Schreib- und Lesezyklen ab [1]. Als Modell für die Fehlerstatistik geht man in der Regel von einem Binär-Symmetrischen-Kanal aus. Für die Kanalcodierung, die die Daten vor Fehlern schützen soll, verwendet man derzeit überwiegend BCH-Codes [1, 2]. Die Fehlerkorrektureinheit im Flash-Controller muss dabei flexibel gestaltet werden, um die Fehlerkorrekturfähigkeit an unterschiedliche Flash-Technologien und Anwendungen anzupassen. In diesem Artikel wird der Entwurf eines konfigurierbaren Encoders beschrieben.

Bei der Hardware-Beschreibung eines BCH-Encoders fällt neben generischem Code, der den Datenfluss steuert, spezifischer Programmcode an, der vom Generatorpolynom des BCH-Codes abhängt.

Jens Spinner, jens.spinner@htwg.konstanz.de und Jürgen Freudenberger, juergen.freudenberger@htwg-konstanz.de, sind Mitglieder der HTWG-Konstanz, Institut für Systemdynamik, Brauneggerstraße 55, 78462 Konstanz.

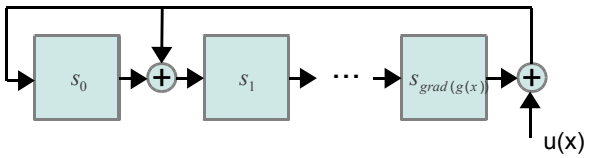


Abbildung 1: Schieberegister zur bitweisen Codierung

Dieser polynomabhängige Teil einer Hardware-Beschreibung kann auf einer höheren Designebene entworfen und mit Hilfe eines Codegenerators in einen konkreten Code wie C, VHDL oder Verilog gewandelt werden. In diesem Artikel wird ein entsprechendes Entwurfsverfahren für eine sehr leistungsfähige parallelverarbeitende Encoder-Einheit vorgestellt.

II. BCH-Encoder

Bei der Codierung zyklischer Codes, zu denen BCH-Codes gehören, wird ein binäres Informationspolynom $u(x)$ mit einem binären Generatorpolynom $g(x)$ multipliziert.

$$v(x) = u(x) \cdot g(x)$$

Dies kann in einem seriellen Verfahren durch ein Schieberegister (Linear Feedback Shift Register, LFSR) realisiert werden, dessen Abgriffe den Koeffizienten des Generatorpolynoms entsprechen (siehe Abbildung 1).

Durch das bitweise Schieben der Information in das LFSR wird codiert. Nach der Encodierung der Information enthält das Register $s(x)$ die Parity-Bits. Die Länge des Generatorpolynoms ist abhängig vom verwendeten Alphabet und der Anzahl der verwendeten „minimalen Polynome“, die wiederum die Anzahl der korrigierbaren bzw. überprüfbaren Bitfehler (Fehlerkorrekturfähigkeit) bestimmt.

Dieser Ablauf kann je nach Anforderung ganz oder teilweise parallelisiert werden. Soll ein möglichst hoher Durchsatz erzielt werden, so wird eine hohe Parallelisierung erzeugt. Je höher wiederum der Parallelisierungsgrad ist, desto höher ist die Anzahl der notwendigen Gatter. Bei einer Teilparallelisierung wird eine gewünschte Anzahl von Informationsbits gleich-

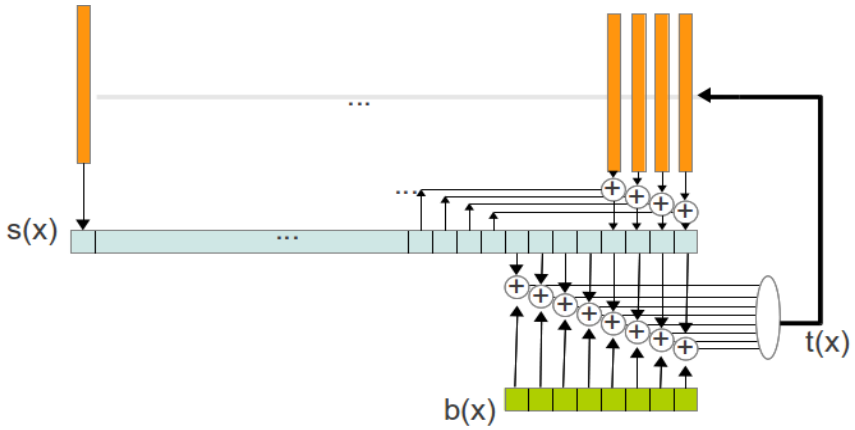


Abbildung 2: Parallele Parity-Berechnung

zeitig bearbeitet. Eine sinnvolle Wahl der Parallelisierung richtet sich unter anderem nach der Größe eines Wortes oder eines Bytes.

In diesem Artikel wird ein Codegenerator vorgestellt, der es ermöglicht, effizient einen parallelen Encoder zu erstellen. Der generische Teil des Codes, der von der Codegenerierung ausgenommen ist, befasst sich hauptsächlich mit der Kontrolle des Datenflusses, also dem Laden der Information aus einer Quelle, sowie dem Zurückgeben der Parity-Bits. Der in diesem Artikel dargestellte generierbare Teil des Verilog-Codes ist vollständig abhängig vom Generatorpolynom und dem Parallelisierungsgrad der Implementierung.

Eine solche Sourcecode-Generierung hat den Vorteil, dass das Generatorpolynom und der Parallelisierungsgrad auf einer höheren Beschreibungsebene entworfen werden können. Einerseits wird hierdurch ermöglicht, bereits bei der Erstellung ohne Mühe längere Generatorpolynome oder Parallelisierungsgrade zu kreieren, andererseits kann der Verilog-Code automatisch für andere Parameter angepasst werden.

Generatorpolynome für BCH-Codes kann man beispielsweise mit Matlab bestimmen. Bekannte Verfahren zur Sourcecode-Generierung für parallele Encoder beschränken sich jedoch bislang auf einfache CRC-Codes [5]. Ohne Codegenerierung ist der Entwickler im Fall einer Neuimplementierung oder bei Änderung der Codeparameter gezwungen, die Konfiguration des Encoders selbst zu bestimmen und die Beschreibung händisch zu erstellen.

III. Parallele Parity-Berechnung

Wie bereits erwähnt, lassen sich die Parity-Bits $s(x)$ durch Multiplikation der Information $u(x)$ mit dem Generatorpolynom $g(x)$ berechnen. In [3] wird gezeigt, wie sich die Berechnung von $s(x)$ in m viele Zwischenergebnisse $s'(x)$ zerlegen lässt.

In Abbildung 2 ist $b(x)$ das parallel zu verarbeitende Informationsfragment, das m viele Bits enthält. In Abbildung 1 ist zu erkennen, wie jedes neu in das LFSR geschobene Informationsbit durch ein Exklusiv-Oder mit dem letzten im Schieberegister befindlichen Bit verknüpft wird. Daher vereinfacht die Funktion

$$t_i(x) = b_i(x) + s_{i+m-1}(x)$$

die Parity-Berechnung [3]. Das neue Informationsfragment kann nun durch folgenden Ausdruck beschrieben werden:

$$s'(x) = R_g(x)\left[t_0(x)x^{grad(g(x))} + \ldots + \ldots + t_{m-1}(x)x^{grad(g(x))+m-1}\right] + R_g(x)\left[s_0x^m + s_1x^{m+1} + \ldots + s_{n-m}x^n\right]$$

Der erste Summand bringt die neuen Informationsbits in die Parity-Berechnung ein während der zweite Summand den Übertrag der bisher berechneten Parity-Bits behandelt. Die Neueinbringung der Informationsbits erfolgt durch die Multiplikation der Informationsstelle mit dem Generatorpolynom. Dies geschieht für jede Informationsstelle mit einem je nach Grad des Informationsbits geschobenen Generatorpolynom. Anschließend werden die Ergebnisse aufsummiert. Die benötigten Teilgeneratorpolynome für jede Teilinformationsstelle sind für ein bestimmtes $g(x)$ statisch. Daher kann für jedes Partiy-Bit eine Bitmaske erstellt werden, die mit der Teilinformation bitweise verundet und anschließend auf alle Bitstellen Modulo 2 aufaddiert wird. Es gibt genau $grad(g(x))$ viele Bitmasken mit der Größe m (siehe Abbildung 3). Jede Bitmaske repräsentiert eine Logikschaltung für eine Speicherstelle im LFSR.

IV. Generierung der Bitmasken

Bitmasken, wie sie in [4] beschrieben sind, können erstellt werden, indem die einzelnen Informationsbits

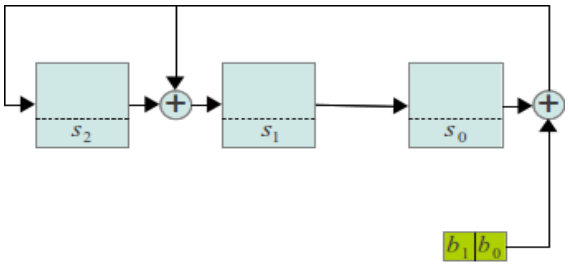


Abbildung 3: Beispiel für Hamming-Code - Initialisierungszustand

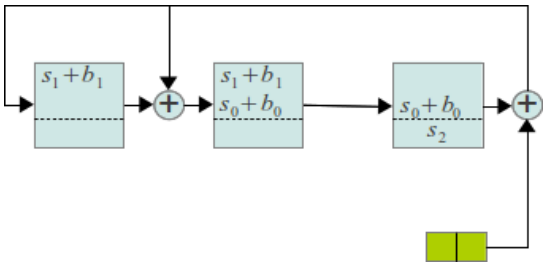


Abbildung 5: Beispiel für Hamming-Code - Schieben von b_1

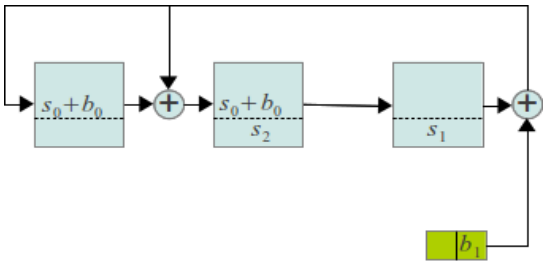


Abbildung 4: Beispiel für Hamming-Code - Initialisierungszustand

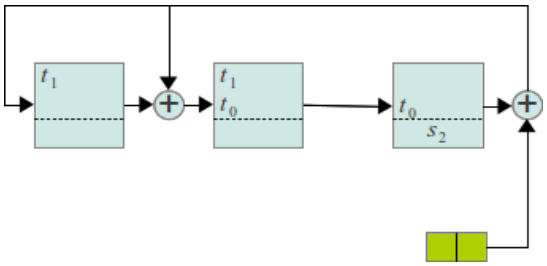


Abbildung 6: Beispiel für Hamming-Code - Substitution

in einem LFSR geschoben werden. Hierzu eignet sich eine Skriptsprache wie TCL, mit der sich leicht Listen verarbeiten lassen. Das LFSR wird durch eine Liste repräsentiert, deren Elemente je eine Speicherstelle darstellen. Eine Funktion, die Kenntnis über das Generatorpolynom hat, schiebt nun jedes Element um eine Stelle weiter, unter Berücksichtigung des Eingangssignals und der generatorpolynomabhängigen Verknüpfungen. In Abbildung 3 ist dies beispielhaft für einen Hamming-Code dargestellt [2]. Das Schieberegister befindet sich im Initialisierungszustand. Die Informationsbits b_0 und b_1 sowie der Initialwert bzw. der Übertrag s_0 bis b_2 wurden noch nicht geschoben.

Zunächst wird b_0 mit s_0 verknüpft und in das LFSR geschoben. Ebenso geschieht dies mit b_1 und s_1 (siehe Abbildung 4 und 5). In einem weiteren Schritt wird die Verknüpfung von $s_x + b_x$ zu t_x vereinfacht (siehe Abbildung 6). Jedes Listenelement des LFSR besteht wiederum aus einer Unterliste von Elementen. Diese Unterliste wächst mit jedem eingeschobenem Element an. Die einzuschiebenden Elemente sind Platzhalter für die einzelnen Bitstellen der Teilinformation.

Daraus ergeben sich LFSR-Listenelemente, deren Platzhalter die Exklusiv-Oder-Logikschaltung (siehe [5]) für jedes Parity-Bit darstellt. In einem weiteren Schritt werden diese Stellen nach den Regeln der Booleschen Algebra vereinfacht.

Die übrigbleibenden Platzhalter werden in Bitmasken transformiert. Jeder Platzhalter repräsentiert eine Bitstelle. Diese Bitmasken werden in eine gewünschte Syntax gewandelt und können mit Hilfe von Include-Anweisungen in C oder Verilog eingebunden werden.

V. Ergebnisse

In einer Beispielimplementierung für verschiedene auswählbare BCH-Codeparameter für eine Fehlerkorrekturfähigkeit von 12, 24, 40, 48, 60, 72 und 96 Bitfehler ist das Verhältnis der Codelänge (Lines of Code) des Codegenerators zur Länge des erzeugten Sourcecodes für die Erstellung der Bitmasken etwa 1:22 und für die Encoder-Einheit 1:28. Der generierte Verilog-Code wurde erfolgreich für einen Xilinx Virtex4 (xc4vlx200) synthetisiert. Es werden für dieses Modul 5542 Slices (3888 Register und 7693 LUTs) benötigt.

VI. Fazit

In diesem Artikel wurde gezeigt, wie man die Logikelemente eines LFSR generiert. Aus einem beliebigen Generatorpolynom können mit Hilfe eines Skripts die benötigten Bitmasken offline erstellt werden. Der dafür notwendige Codegenerator benötigt die zwei Funktionalitäten: „Schieben von Symbolen in einem LFSR“ und „Reduzieren der Gleichung durch Boolesche Algebra“.

Der beschriebene Sourcecode-Generator kann um die Funktionalität einer Generatorpolynom-Erstellung erweitert werden. So können die Eingabeparameter des Codegenerators von einem Generatorpolynom auf die Wahl eines „primitiven Elementes“ und der Anzahl der korrigierbaren Bitfehler geändert werden.

Die Kernvorteile der Codegenerierung liegen in der Verringerung der vom Entwickler zu schreibenden Codezeilen sowie in einem reduzierten Aufwand zur Verifikation bei Änderungen der BCH-Codeparameter.

LITERATURVERZEICHNIS

[1] R. Micheloni, A. Marelli, R. Ravasio, „Error Correction Codes for on-Volatile Memories", *Springer*, 2008.

[2] S. Lin, D. Costello, „Error Control Coding", *Prentice Hall*, 2004.

[3] T. V. Ramabadran, „A Tutorial on CRC Computations" *IEEE*, 1988.

[4] Perez, „Byte-wise CRC-Calculations" *Micro, IEEE*, Juni 1983.

[5] M. Sprachmann, „Automatic Generation of Parallel CRC Circuits", *IEEE*, 2001.

Jens Spinner erhielt den akademischen Grad des BSc in Technischer Informatik im Jahr 2009 sowie den MSc in Informatik im Jahr 2011 von der HTWG-Konstanz. Er ist wissenschaftlicher Mitarbeiter der Hochschule Konstanz.

Dr. Jürgen Freudenberger ist seit 2006 Professor für eingebettete Systeme an der Hochschule Konstanz. Dort leitet er das Institut für Systemdynamik. Seine Forschungsarbeit beschäftigt sich vorrangig mit der Entwicklung von Algorithmen im Bereich der Signalverarbeitung und der Codierung für zuverlässige Datenübertragung sowie mit der effizienten Umsetzung der Verfahren in Hard- und Software.

Development of a Hardware-Software Co-Design for a real-time localization protocol for pedestrian safety

Axel Sikora, Dirk Lill, Manuel Schappacher

***Abstract*—The research project Ko-TAG [2], as part of the research initiative Ko-FAS [1], funded by the German Ministry of Economics and Technologies (BMWi), deals with the development of a wireless cooperative sensor system that shall provide a benefit to current driver assistance systems (DAS) and traffic safety applications (TSA). The system's primary function is the localization of vulnerable road users (VRU) e.g. pedestrians and powered two-wheelers, using communication signals, but can also serve as pre-crash (surround) safety system among vehicles. The main difference of this project, compared to previous ones that dealt with this topic, e.g. the AMULETT project, is an underlying FPGA based Hardware-Software co-design. The platform drives a real-time capable communication protocol that enables highly scalable network topologies fulfilling the hard real-time requirements of the single localization processes. Additionally it allows the exchange of further data (e.g. sensor data) to support the accident prediction process and the channel arbitration, and thus supports true cooperative sensing. This paper gives an overview of the project's current system design as well as of the implementations of the key HDL entities supporting the software parts of the communication protocol. Furthermore, an approach for the dynamic reconfiguration of the devices is described, which provides several topology setups using a single PCB design.**

***Index Terms*—VRU eSafety, localization, system design, hardware-software co-design, time of flight, distance of arrival, driver assistance system.**

I. Introduction

The development of traffic safety and driver assistance systems is a strong objective of the today's automotive industry. Several techniques e.g. based on radar or ultrasonic waves or camera systems have been evaluated and enhanced. For these systems a direct line of sight is a hard requirement for a proper operation.

The Ko-FAS initiative (Kooperative Sensorik und kooperative Perzeption für die Präventive Sicherheit im Straßenverkehr) was launched in 2009, and includes the described subproject Ko-TAG. The Ko-TAG project concentrates on the development of a cooperative sensor network between vehicles and vulnerable road users (VRU). Its objective is to add value to existing passive driver assistance systems such as radar- or camera-based systems. At the one hand, this requires to enable the localization of foreign targets even when losing line of sight or in scenarios with multiple objects requiring prioritization. One the other hand the cooperative sensor network provides valuable additional information about the localized target such as its movement parameters or even entire movement patterns that support the accident prediction algorithms at the higher layers. Since those networks in the Car2X environment are highly dynamic, the actual topology can change within short time periods regarding to the number of network participants and the reaction time of the system the different scenarios are handled by an underlying network communication protocol allowing a prioritization of connected devices and therefore affect the measurement update rate.

The protocol also keeps the security aspects such as the en-/decryption of user sensible data, data integrity and user anonymity. To ease a later integration into existing Car2X technologies or into other currently active projects dealing with this topic such as the sim^{TD} project [3], existing standards are considered during the system design.

The authors' task in the project is to provide a scalable, stable and secure networking platform between vehicles and VRU's that manages the different tasks of the localization process. This platform must meet the strong real-time requirements as well as it has to flexible since the system is still at a development state. Therefore the authors follow an approach of a hardware-software co-design. The general approach of the Ko-TAG project and its communication and measurement protocols are described e.g. in [4] and [5].

Axel Sikora, axel.sikora@hs-offenburg.de; Dirk Lill, dirk.lill@stzedn.de; Manuel Schappacher, manuel.schappacher@stzedn.de are with Steinbeis Innovationszentrum für Embedded Design und Networking, Poststrasse 35, D-79423 Heitersheim, Germany

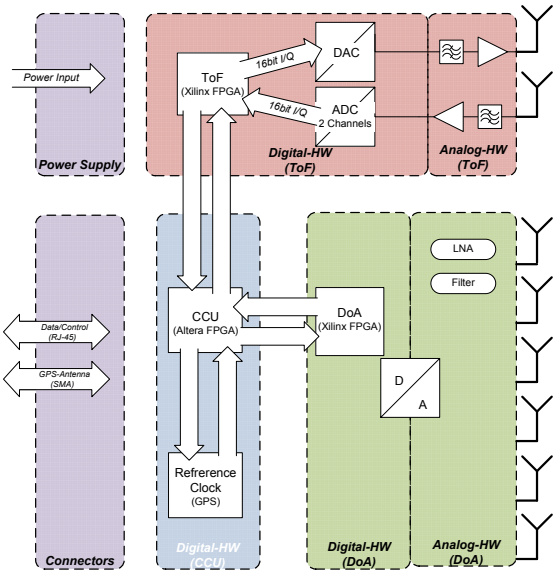


Figure 1: Architecture of the Localization Unit

II. System Architecture

The system architecture consists of several subcomponents to handle the single tasks of he localization and accident prediction process. The system and the underlying protocol design are built on two basic types of nodes. It is anticipated that vehicles get equipped with a Localization Unit (LU) that is able to communicate with as well as to localize SafeTAGs (ST) as their counterpart. An ST is a multifunctioning device that can change its role and behaviour in the network depending on the environment it shall be used in.

A. Architecture of the Localization Unit

The architecture of the LU is shown in Figure 1. It consists of

- A Time of Flight (ToF) system developed by [6], responsible for measuring the distances to the localized objects. This module is implemented in a Xilinx FPGA.
- A Destination of Arrival (DoA) system, developed by [7], to measure the elevation and azimuth to SafeTAGs implemented in a combination of a Xilinx FPGA and a DSP.
- The Communication & Control Unit (CCU), which handles the communication protocol as well as the coordination among the several subcomponents.
- An Ethernet interface connecting the LU to a further component called the fusion unit (FU). The FU is an advanced board computer accepting the sensor and measurement data from the LU. Using this information optionally with data from further connected sensors such as cameras, the FU is able to calculate eventual collision risks. Depending on the determined risks, the LU can reconfigure the LU e.g. to priories endangered SafeTAGs and therefore increasing their measurement update rate.

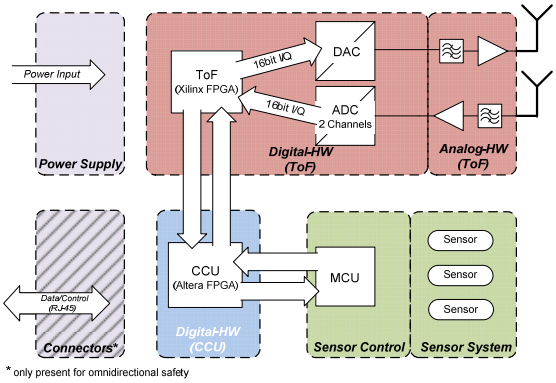


Figure 2: Architecture of the SafeTAG

B. Architecture of the SafeTAG

The SafeTAG's architecture shows some differences compared to the LU since it needs less complexity.

- The ToF subsystem is less complex compared to the LU counterpart since here no measuring has to be performed. The messages are just reflected.
- Since the DoA measurements are performed using the electromagnetic waves of the data communication, the SafeTAG does not contain a DoA module. Instead it is connected to a sensor array developed by [7], which provides important sensor data, which support the accident prediction process.
- The SafeTAG also includes a CCU for the wireless protocol handling and for controlling the sub components.

Depending on the environment, the SafeTAG used in it provides further interfaces to interact with other Car2X elements. The architecture of the SafeTAG is given in figure 2.

III. Communication & Control Unit

A. General Setup of the CCU

The design and the development of the CCU is one of the main tasks of the authors in the Ko-TAG project. The CCU handles the whole data communication including the establishment and configuration of the cooperative sensor network, as well as the exchange of the sensor and meta-data packets. During the communication the CCU has to take care of the security aspects of the system like the encryption/decryption of user sensible data or the integrity of the transmitted and received data. Additionally, the traceability of users has to be prevented.

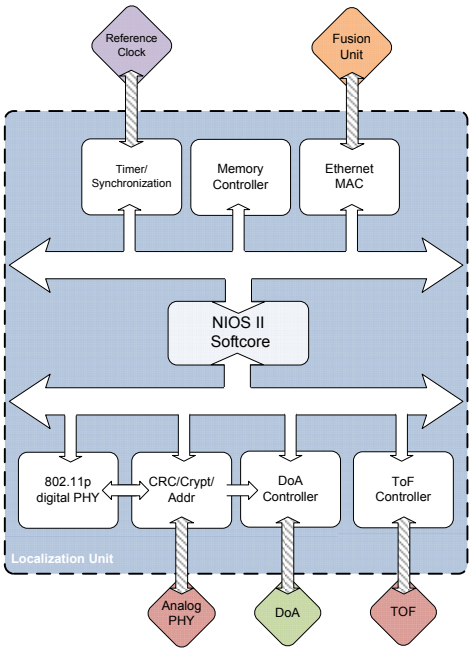


Figure 3: Architecture of the CCU of the localization unit

Besides the exchange of the data packets the single sub modules that take part in the localization process have to be controlled and coordinated by the CCU. To handle the different tasks described above, a hardware-software co-design was implemented in an Altera ARRIA II GX FPGA, including a NIOS II Soft-Core processor to run the appropriate firmware. To support the CPU and to handle the time critical parts such as synchronization, the design includes several application specific IP-Cores (Intellectual Property Core) implemented in VHDL. An overview of the CCU's architecture is shown in Figure .

B. NIOS II Soft-Core

At the centre of the system design a NIOS II Soft-Core is implemented in the FPGA description. The appropriate software primarily manages the Network. Highly scalable networks as in the Car2X environment require a lot of management effort as well as the maintenance of several tables and data structures. Since the system and the protocol design both still are in a development state, a software implementation provides the largest benefit regarding flexibility and efforts to meet future design changes. Furthermore, since all the HDL-modules developed in the project are implemented as Avalon [15] Bus-Slaves, several interfaces are provided to the software implementation allowing a dynamic configuration of the according hardware units.

C. Digital PHY

For the Physical Layer (PHY) of the data communication, an application specific digital PHY (dPHY) IP-Core has been developed and provided [8] that takes care of the baseband modulation. It closely follows the IEEE 802.11p specification what eases the later integration into existing systems and additionally provides application specific features and interfaces to support the localization process. For example it allows seamless appending of a user defined signal sequence to standard 802.11p communication signals e.g. to ease the DoA estimation.

Besides a memory mapped register interface that allows the configuration and the monitoring of the PHY, an external 8 Bit wide FIFO interface gives fast access to its transmit and receive paths. Additional signal lines enable an interconnection to a self-developed Data Chain IP-Core described below.

On the PCB side, the dPHY is interconnected with an analog PHY (aPHY) designed and developed by [6] with a 16Bit wide IQ data interface and further configuration and control lines e.g. to switch the carrier frequency or to support the automatic gain control (AGC).

D. Data Chain

An important part in the design is a multi-functional IP-Core that is placed between the digital PHY's data interface and the CPU's system bus. It performs several tasks regarding the data communication that have been outsourced from software to a hardware description to increase the system performance.

1) Data Access

On top of the IP-Core the data chain implements an Avalon Bus-Master, which enables an autonomous direct memory access (DMA) to the system's included memory both in Tx and Rx mode. This discharges the CPU activity since the software routines for transmitting and receiving data packets equal to simple memory access functions. This gives an important benefit regarding time consumption and implementation effort compared to SPI-Interfaces used in most common transceiver chips.

Besides the general data, further flags indicating different packet states such as CRC or addressing errors or security information are included in the packets meta-information using a special buffer descriptor implementation on a per packet base. The software implements the specified driver similar to standard POSIX (Portable Operating System Interface) interfaces to access those buffer descriptors and therefore enables the lower layers to transmit and receive communication frames.

2) CRC Generation and Validation

To ensure the data integrity, a configurable CRC block is included into the data chain. Checksums are calculated for transmitted frames and verified for incoming packets automatically if configured. This

filters damaged packets and therefore saves CPU and bus time. Since the data chain block is connected to further IP-Cores managing parts of the localization process, it also prevents broken frames to be analysed for measurements.

The Data Chain driver allows the software to configure the CRC hardware block in different manners. Generally, the CRC generation or verification can be en- or disabled dynamically. Furthermore, broken frames can be discarded immediately or forwarded to the software e.g. for debugging purposes with an additional error flag set in the resulting buffer descriptors content.

3) Security & Privacy

The data chain also includes an autonomous en- and decryption block. Configured using the data chain's buffer descriptor interface e.g. regarding key information, incoming packets get decrypted and transmitted packets get encrypted without a further user interaction from software during the process. It is anticipated to use an AES crypto-unit here for providing security and privacy. Since the security design is still subject of on-going development it will be described in a future contribution.

4) Address Checking

Incoming packets will be checked automatically for valid addressing and configured before the reception from software via register accesses. This unloads the system bus as well as the CPU since on the one hand, the address check has not to take place in software and on the other hand, misaddressed frames do not reach the system bus.

The hardware address checking is also important for the DoA estimation process since this is a highly time critical operation. Therefore the data chain is also interconnected to the DoA-Controller via an additional interface to filter unwanted packets from further analysis.

Since the addressing in the communication protocol is not fixed and depends on the according frame type, the HDL entity has to provide several configuration options. Here the data chain driver allows the software configuring several address fields together with their length and their offsets in the communication frames to support an address validation of the several frame types.

E. DoA Controller

The DoA controller serves as interconnection between the CCU and the DoA estimation module. The Controller handles exchange of measurement and control frames as well as it triggers and configures new measurements.

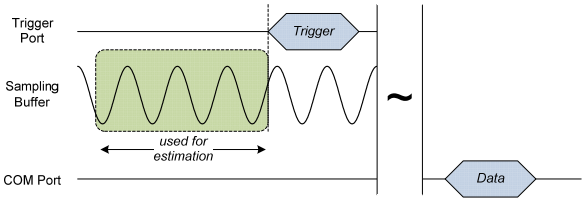


Figure 4: DoA Sampling and Trigger Behaviour

The DoA estimation takes place over regular data packets transmitted during the DoA phase of the network protocol. Through the underlying time slotted channel access mechanism during this phase, the address of the expected data frame is well known. After a successful check of the data integrity and the address information of the received package, which is all done in hardware by the data chain and without software interaction, a new DoA estimation gets triggered by the DoA-Controller. Since the answer to the measurement request is asynchronous, an internal ID, previously defined from software access, is included in the trigger to allow an exact identification of the returned values on reception.

The DoA component itself includes a ring buffer which samples the signals on the communication channel constantly. Once the DoA unit receives a trigger from the DoA-Controller, it analyses the specified window of the data in the sampling buffer for an angle estimation. To get valid measurement results, the offset between the start of the frame and the according trigger needs to be static to guarantee that the correct signal pattern gets used for the following calculation as shown in Figure .

Once the calculation has finished, the DoA module provides several value pairs including their qualities and the device ID of the measurements are associated to via a serial interface to the DoA-Controller. The Software finally collects the data from the DoA Controller and passes it to the LU for further analysis.

F. ToF Controller

The ToF-Controller manages the ToF hardware developed by [6]. Before the start of a new ToF phase it configures the ToF hardware before it triggers a new measurement. The configuration primarily specifies addressing information, which gets included into the following ToF beacon. This additional information is needed for the ST's to identify the vehicle the burst was sent from, as well as for the vehicle for a proper correlation of the ToF replies sent by STs.

During the ToF localization process the ToF-Controller serves as a clock source to indicate the single time slots since their offsets have to be regarded for the appropriate distance calculation. Furthermore, the Controller receives continuous measure data containing the calculated distance and measurement quality via a 16 bit wide parallel interface that have to be associated to the correct device before they get for-

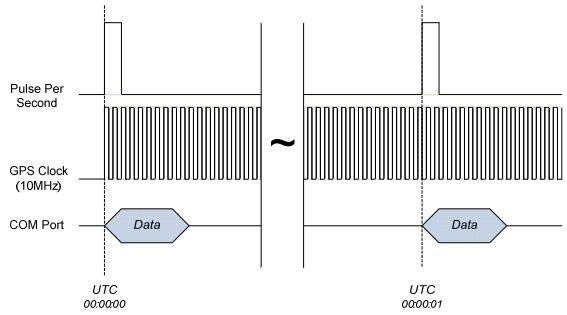


Figure 5: Timing diagram of the GPS synchronization

warded to the FU. To allow a later collection of the measured values at the software layers, an additional FIFO holds the data, together with the according slot indices until the end of the measurement procedure.

G. Synchronization

Since the network protocol uses a time slotted channel access mechanism to provide a deterministic behaviour for time critical processes, the LU's must be synchronized among each other to provide a common grid of the slots to the ST's. A separate synchronization/timer IP-Core takes care about the global synchronization using an external reference clock as time input source to keep given guard times.

Therefore the Ko-TAG project uses a high precision timing GPS module of the u-blox LEA-6T family. The LEA-6T module provides the needed performance regarding the accuracy with an error below 60 ns [9]. Using valid almanac data stored from previously established satellite connections, software can help to accelerate the device start-up procedure and a start-up time of around 1 s can be reached. The module also provides several additional interfaces that finally feed the synchronization IP-Core.

The GPS timing module provides two separate time pulse outputs. One output gives a GPS locked pulse per second (PPS). The second output is a configurable clock with a frequency up to 10 MHz. A combination of these pulse signals and an additional serial protocol implemented in the GPS module enables the IP-Core to synchronize with a maximum error less than 100 ns among other LU's. This synchronization permits the ST's to derive their time slot information from any LU. Since ST's are not equipped with an additional reference clock source they synchronize on the network using the ToF beacons sent by vehicles.

As shown in Figure , the GPS timing module used in the project triggers the start of a new second using the PPS strobe followed by a serial NMEA (National Marine Electronics Association) packet indicating the current time. The time information is interpreted by software and is used to configure the IP-Core via register accesses. In its actual configured state the hardware block uses the following PPS trigger to synchronize according to the previously set configuration. From now on, the GPS locked 10 MHz clock signal is used for the CCUs internal time reference. Once synchronized the IP-Core can be monitored and configured using further register accesses. It provides several hardware timers/counters with a common time reference among all synchronized LU's.

H. Ethernet Interface

The CCU includes a GBIT-Ethernet MAC IP-Core developed by the authors' team and presented in [13]. This interface is used in different manners. According to Figure the CCU is connected with a fusion unit (FU) via an Ethernet interface. This interface is used to provide measurement results and data from the sensor network to the FU as well as to receive control information. The interface uses an extended LocON protocol [15] at application layer.

A further benefit of the Ethernet connection is the possibility to give the user simple access to the system. The user is able to set static configurations such as the wireless network parameters and to observe the current device status. Furthermore, an interface is given to update the device software as well as the FPGA image. An embedded software TCP/IP stack [12] provides the TCP/IP interconnection on the higher layers including an embedded web server.

IV. Software Design

Besides the HDL entities described in chapter III, a firmware was designed and implemented primarily to manage the network and to control and configure the several IP-Cores. Generally, it consists of a wireless communication stack, the application itself and further interface drivers and protocol implementations depending on the actual device type. The main tasks of the software are handled in a wireless communication stack that currently covers the physical (PHY), the data link (DLL) and the application (APL) layers.

At the lower layers e.g. the software part of the PHY, the stack is responsible for the transmission and reception of the communication frames. As described in chapter III, access to frames in Tx- as well as in Rx-mode is implemented through memory access functions wrapped into buffer descriptor interfaces via the data chain IP-Core. Therefore the physical layer software provides the device driver according to access the data chain. Furthermore, the physical layer implements the required driver elements to configure the dPHY, e.g. in its channel and operation mode.

The data link layer (DLL) represents the most complex part of the software since it has to manage and synchronize the complex processes of the network and interacts with several of the IP-Cores described above. E.g. it configures the data chain for address and CRC validation and feeds the security block with different en/decryption keys depending on the communication

partner. The most significant part of the DLL represents the medium access. Since parts of the protocol use a time slotted channel access it also has to take care of a valid distribution and coordination of the single slots. Here it uses interrupts generated by the synchronization IP-Core's timer to meet the timing requirements and to hit the specific time slots within a given guard time.

The APL has its key task at the OBU. Here it is primarily responsible to accept user prioritization requests from the LU. Internal prioritization queues are fed with those requests and enable the APL to configure specific channel access phases of the DLL. This allows a good scalability of the measurement update rates of the single users. I.e. it allows to reserve channel resources for the mostly endangered devices.

V. Device Configuration

During the development process, the single components are very expensive. To provide the possibility of evaluating different network topologies despite of the limited resources, an approach of a single PCB design for the LU and the SafeTAG was chosen. This makes it possible to run and test different network setups without changing the underlying PCB.

The role of the device only depends on the running hardware/firmware combination and can be changed at start-up or even during runtime. To have multiple roles available at a single device, the included flash memory contains several combinations of an FPGA-image and its according software, located in specific areas of the target memory that can be loaded at the device start-up or during runtime. For this task several approaches exist:

- The device can be configured using an EPCS device, an active serial EEProm chip that configures the connected FPGA device at start-up. Usually these devices are limited in their memory size.
- A common method is the configuration via an additional smaller CPLD and an external memory chip. This approach allows the usage of a custom loader firmware, which allows modifying the boot process.
- An almost identical approach uses a microcontroller instead of the CPLD, which helps to reduce costs.

The CCU PCB is designed to support a CPLD configuration as well as the microcontroller approach using the fast passive parallel configuration (FPP) method supported by devices of the Arria II family. The design of the according configuration mechanism is shown in Figure . To support the configuration as different device roles the firmware of the configuration chip accesses a special boot loader section in the external memory device before initiating the configuration process. This section includes information about the base addresses of the different images in memory and provides the possibility to choose the according

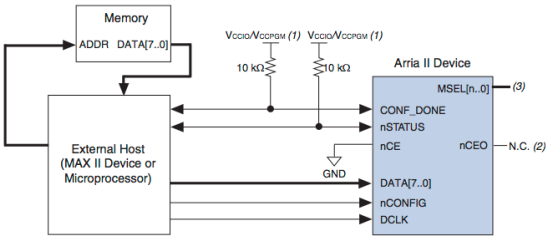


Figure 6: Single Device FPP Configuration Using an External Host [10]

firmware image at start-up time using a DIP panel or a dynamic reconfiguration during runtime.

Since the target devices will be integrated into vehicles at a later development state and thus won't be accessible, a web interface provides easy access to system update and supervision.

VI. Summary and Outlook

The approach of the hardware-software co-design allows the swapping of time-critical components and processes into fast and parallel FPGA implementations. Nevertheless the system remains flexible and configurable by letting software modules handle the complex task such as the network management.

The elements of the system architecture already have been separately tested and proved for their functionality. Currently the integration of the sub components is in progress and will be followed by a complete system test.

Acknowledgement

This work results from the joint project Ko-TAG, which is part of the project initiative Ko-FAS and is partially funded by the German Ministry of Economics and Technologies (BMWi) under contract number 19S9011. The authors are grateful for this support and for the excellent teamwork in the consortium, which consists of BMW Group Forschung und Technik, Continental Safety Engineering International GmbH, Daimler AG, Fraunhofer IIS, TU Munich, and Steinbeis Innovation Centre Embedded Design and Networking.

References

[1] Forschungsinitiative Ko-FAS - Kooperative Sensorik und kooperative Perzeption für die Präventive Sicherheit im Straßenverkehr, http://www.kofas.de/, 19.02.2012.

[2] Forschungsprojekt Ko-TAG – Kooperative Transponder, http://ko-fas.de/deutsch/ko-tag---kooperative-transponder.html, 19.02.2012.

[3] Projekt sim^{TD} (Sichere Intelligente Mobilität Testfeld Deutschland), http://www.simtd.de, 19.02.2012.

[4] D. Lill, M. Schappacher, A. Gutjahr, A. Sikora, "Protocol and System Design of a Cooperative Pedestrian Safety System", 11th Int'l Conference on ITS Communication (ITST2011)", 23.-25.8.2011, St. Petersburg (Russia).

[5] D. Lill, M. Schappacher, S. Islam, A. Sikora, "Wireless Protocol Design for a Cooperative Pedestrian Protection System", 3rd International Workshop on Communication Technologies for Vehicles (Nets4Cars 2011), 23.-24.3.2011, Oberpfaffenhofen.

[6] Technische Universität München, Fachgebiet Höchstfrequenztechnik, http://www.hot.ei.tum.de/

[7] Fraunhofer-Institut für Integrierte Schaltungen IIS, http://www.iis.fraunhofer.de/, 19.02.2012.

[8] Fraunhofer Heinrich-Herz Institut, http://www.hhi.fraunhofer.de/, 19.02.2012.

[9] u-blox, LEA-6, u-blox 6 GPS modules Data Sheet, http://www.u-blox.com/images/downloads/Product_Docs/LEA-6_DataSheet_%28GPS.G6-HW-09004%29.pdf, 19.02.2012.

[10] Configuration, Design Security and Remote System Upgrades in Arria II Devices, AIIGX51009-4.2, Altera Cooperation December 2011.

[11] Experteninformationen und Publikationen aus Ko-TAG, http://ko-fas.de/deutsch/ko-tag---kooperative-transponder/experteninformationen.html, 19.02.2012.

[12] N. Braun, A. Sikora, M. Colling, "High Performance Embedded Ethernet and Internetworking", embedded world 2005 Conference, Nürnberg, S.990-997.

[13] A. Rohleder, S. Jaeckel, A. Sikora, "Design and Test of a Gigabit Ethernet MAC for High-Speed HIL-Support", XLIV. Workshop MPC-Gruppe, Furtwangen, 9.7.2011.

[14] http://www.ict-locon.eu/, 19.02.2012.

[15] Avalon Interface Specifications, Altera Cooperation, May 2011.

Manuel Schappacher studied Computer Engineering at the University of Applied Sciences, Furtwangen and received his Dipl.-Inform. degree in April 2009. After that, he continued to work as project engineer at Steinbeis Innovation Center Embedded Design and Networking (sizedn) mainly in the field of embedded wireless and wired communication, including simulation of networking protocols.

Dipl.-Ing. (FH) Dirk Lill holds a Diploma of University of Cooperative Education, Loerrach, where he studied mechatronics at the Universities in Loerrach (Germany), Muttenz (Switzerland) and Mulhouse (France). He joined Steinbeis Innovation Centre for Embedded Design and Networking (sizedn) in 2002. Since then, he has worked in various projects in the fields of wired and wireless embedded networking, including standard protocols and proprietary wireless protocol design. He is now deputy head of stzedn. He combines excellent experience in PCB hardware design for HF-circuitry with a deep knowledge of communication protocols and their implementation in hard- and software

Axel Sikora holds a diploma of Electrical Engineering and a diploma of Business Administration, both from Aachen Technical University. He has done a Ph.D. in Electrical Engineering at the Fraunhofer Institute of Microelectronics Circuits and Systems, Duisburg, with a thesis on SOI-technologies. After various positions in the telecommunications and semiconductor industry, he became a professor at the Baden-Wuerttemberg Cooperative State University Loerrach in 1999. In 2011, he joined Offenburg University of Applied Sciences, where he holds the professorship of Embedded Systems and Communication Electronics. His major interest is in the field of efficient, energy-aware, autonomous, and value-added algorithms and protocols for wired and wireless embedded communication.

He is founder and head of Steinbeis Innovation Center Embedded Design and Networking (sizedn). Dr. Sikora is author, co-author, editor and co-editor of several textbooks and numerous papers in the field of embedded design and wireless and wired networking, and head and member of numerous steering and program committees of international scientific conferences.

JTAG für SoC

Bernd Adler, Andreas Siggelkow

***Zusammenfassung*—Es wird der Aufbau einer Debug-Umgebung für C/C++-Programme unter Linux mittels JTAG-angebundenen Embedded µC-Systemen beschrieben.**

***Schlüsselwörter*—TAP, JTAG, Debug.**

I. Einleitung

Ein Programm, welches auf einem Embedded uC-System ausgeführt wird, kann nur über wenige Schnittstellen mit der Außenwelt kommunizieren und durch diese gesteuert werden. Eine solche Schnittstelle ist JTAG. Diese Schnittstelle wurde ursprünglich dazu entwickelt um einen einheitlichen Zugang zu Testschaltungen zu erhalten, die dazu benötigt werden, nach der Produktion von Hardwarekomponenten deren Funktionsfähigkeit zu testen. Jedoch wird JTAG inzwischen auch dazu genutzt, um mit dem laufenden System zu kommunizieren, dieses zu programmieren und in erster Linie zu steuern.

Ziel dieser Arbeit ist es, die Kommunikation mit uC-Systemen über JTAG abzuwickeln, dessen Leistungsgrenzen zu ermitteln und eine Debugumgebung für Software aufzubauen, welche über JTAG mit der Hardware kommuniziert und auf dieser das Debugging der Software durchführt. Diese Debugumgebung soll den Anforderungen des Lehrbetriebs genügen, unter Linux lauffähig sein und außerdem aus frei verfügbaren Programmen bestehen.

II. Schematischer Aufbau

Die grundlegenden Komponenten (Abb. 1), die im Zusammenhang mit JTAG beteiligt sind, sind einerseits das Hardwaresystem, das im folgenden als Target bezeichnet wird, das JTAG-Interface, welches den JTAG-Bus mit dem Bus des Computers verbindet, sowie der Computer, von dem aus das Debugging durchgeführt wird. Die für JTAG relevanten Komponenten des Targets sind zunächst der Test Access Port (TAP), der In-Circuit Emulator (ICE) und die einzelnen Bestandteile des Targets, auf die der ICE zugreifen kann. Diese Bestandteile sind beispielsweise der Hauptprozessor (CPU), der Arbeitsspeicher (RAM) und der Flashspeicher.

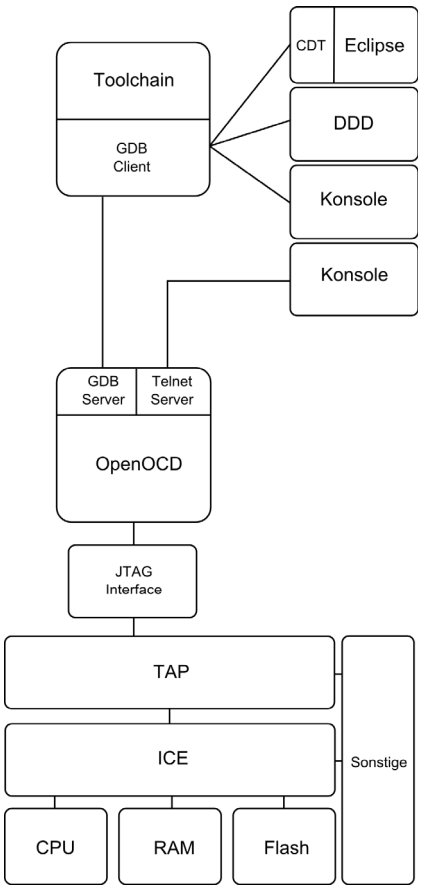


Abbildung 1: Schematischer Aufbau

Der Test Access Port (TAP) ist eine in IEEE 1149.1 standardisierte Schnittstelle, um die einzelnen Leitungen des JTAG-Busses mit den tiefer liegenden Komponenten des Targets zu verbinden. Dabei regelt der TAP nur, wie Daten in das Target hinein und wieder hinaus gelangen. Wie diese übertragenen Daten strukturiert sind, wird durch den TAP nicht festgelegt.

Die über den TAP in das Target übergebenen Daten werden durch den In-Circuit Emulator (ICE) aus dem TAP gelesen. Anhand dieser Informationen führt dieser nun die gewünschten Aktionen aus und steuert die einzelnen Bestandteile des Targets, wie beispielsweise die CPU oder auch die verschiedenen Speicher.

Bernd Adler, bernd.adler@hs-weingarten.de und Andreas Siggelkow, siggelkow@hs-weingarten.de sind Mitglieder der Hochschule Ravensburg-Weingarten, Doggenriedstraße, 88250 Weingarten.

Weiterhin gehören Funktionen wie Breakpoints oder Single Stepping zu den Aufgaben des ICE, sofern diese nicht direkt durch den Hauptprozessor unterstützt werden. Wie ein ICE aufgebaut ist, ob und wie er bestimmte Aktionen unterstützt und wie dieser über JTAG kommuniziert, ist herstellerspezifisch und nicht festgelegt.

Die Aufgabe des JTAG-Interfaces ist es, den JTAG-Bus an einen Bus anzubinden, der an gewöhnlichen Computern vorhanden ist. Diese Anbindung geschieht meist über den Universal Serial Bus (USB) oder über die ältere parallele Schnittstelle (IEEE 1284). Auf dem Computer, über den das Debugging gesteuert wird, werden eine Reihe von Programmen benötigt, um dies zu ermöglichen.

Die Hauptaufgabe von OpenOCD ist es, die Steuerung der Targets über JTAG für die darauf zugreifenden Programme zu abstrahieren und zu vereinheitlichen. Hierbei kommuniziert es einerseits über das JTAG-Interface direkt mit dem Target und bietet andererseits Kommunikationsschnittstellen für die zugreifenden Programme über den Gnu Debugger (GDB) und Telnet. Dazu enthält OpenOCD einen GDB-Server und einen Telnet-Server.

Als Toolchain bezeichnet man Programme, die, um eine bestimmte Aufgabe zu erfüllen, aufeinander aufbauen. In diesem Fall handelt es sich um die Programme, die benötigt werden, um ein Programm aus dem Quellcode in den für Computer verständlichen Maschinencode zu übersetzen. Weiterhin umfasst die Toolchain ein Programm, mit dem man Programme auf Fehler überprüfen kann. Dies ist der Debugger GDB. Während dieser normalerweise als ein Programm ausgeführt wird, kann er auch als ein Verbund aus Clients und einem Server eingesetzt werden. In Verbindung mit OpenOCD übernimmt dieses die Aufgabe des GDB-Servers, während der GDB der Toolchain den Clientanteil übernimmt. Bei der Auswahl der Toolchain muss beachtet werden, dass diese zu der verwendeten Hardwareplattform passt. Für die ARM-Plattform sind verschiedene Toolchains verfügbar, konkret wurde hier die GnuARM Toolchain eingesetzt, jedoch sind generell alle Toolchains nutzbar, solange die Hardwareplattform unterstützt wird.

Eclipse ist eine Entwicklungsumgebung für Software. Als solche wird sie verwendet, um Programme zu schreiben, zu kompilieren und zu debuggen, also die Programme auf Fehler zu testen. Weiterhin kann Eclipse mit Hilfe von Erweiterungen auf die Anforderungen von bestimmten Programmiersprachen angepasst werden. Für die Programmiersprachen C und C++ ist diese Erweiterung das C/C++ Development Toolkit (CDT). Weiterhin wird für das Debugging über JTAG ein zusätzliches Plugin für Eclipse benötigt. Dieses trägt die Bezeichnung „Zylin Embedded CDT“.

Der Data Display Debugger, normalerweise mit DDD abgekürzt, ist eine graphische Oberfläche für kommandozeilenbasierte Debugger. Als solche kann dieses Programm als Alternative zu einer vollständigen Entwicklungsumgebung für das Debuggen von Programmen verwendet werden.

Alternativ zu den graphischen Oberflächen können sowohl OpenOCD- als auch GDB-Befehle über die Kommandozeile erhalten. Dies geschieht bei OpenOCD über eine Telnetverbindung, bei GDB über die Konsole des GDB-Clients.

III. Installation

Im nun Folgenden sind die notwendigen Befehle angegeben, um alle für das Debugging benötigten Programme, mit Ausnahme der Toolchain und des Zylin Plugins, aus der Paketverwaltung von Linux zu installieren. Von den angegebenen Befehlen ist dabei derjenige zu wählen, der zu der verwendeten Linux-Distribution passt.

```
# RPM-based: Fedora
su -c 'yum install eclipse eclipse-cdt ddd openocd libftdi-devel'
# APT-based: Debian, Ubuntu
su -c 'apt-get install eclipse eclipse-cdt ddd openocd libftdi-dev'
```

IV. Development

Das Crosscompiling, welches notwendig ist, um Software für eine andere Hardwareplattform als die, auf der entwickelt wird zu erstellen, stellt gewisse Anforderungen an die Konfiguration der Buildumgebung. In erster Linie ist es erforderlich, die vollständige Kontrolle über den Buildprozess zu besitzen. Hierzu sollten folgende Punkte beachtet werden:

- **Makefile**.
  Es ist empfehlenswert, ein Makefile für den Buildprozess zu verwenden.
- **Linkerscript**.
  Es muss ein eigenes Linkerscript verwendet werden um den Aufbau des Speichers des verwendeten Targets abbilden zu können.
- **Libraries**.
  Bibliotheken müssen zu der Zielplattform und der verwendeten Compilerversion passen.

V. Debugging

Die im Folgenden beschriebenen Schritte beschreiben die einzelnen Aktionen, die notwendig sind, um die Debugumgebung zu starten und in einen einsatzbereiten Zustand zu versetzen. Diese einzelnen Schritte lassen sich mit Hilfe von Scripten größtenteils zusammenfassen und automatisieren. Dies führt dazu, dass bei einer optimal funktionierenden Hardware-Softwarekombination lediglich die Hardwareverbindungen von Hand hergestellt werden müssen. Die Initialisierung der Software beschränkt sich im Ideal-

fall auf das Starten der graphischen Oberfläche, deren Debugmodus, sowie das Starten von OpenOCD.

Der allgemeine Ablauf des Debuggings lässt sich in folgende Abschnitte unterteilen:

- Verbindungsaufbau.
- Laden des Programms und dessen Symboltabelle.
- Reset des Targets.
- Übertragen des Programms in den Speicher des Targets.
- Anfahren der Startposition.
- Debuggen des Programms.

A. Verbindungsaufbau

Der erste Schritt ist das Anschließen der Hardware, dies beinhaltet das Herstellen der Stromversorgung sowie das Verbinden der Datenleitungen des Targets. Außerdem muss das Target über das JTAG-Interface an den Entwicklungscomputer angeschlossen werden.

Der nächste Schritt ist das Starten von OpenOCD mit Hilfe der für das Target und das JTAG-Interface passenden Konfigurationsdateien. Der Befehl, um OpenOCD aufzurufen, stellt sich dann ähnlich wie angegeben dar, wobei die mit dem Parameter -f eingebundenen Konfigurationsdateien zu der verwendeten Hardware passen müssen.

```
openocd -f interface.cfg -f target.cfg
```

Ist die Verbindung zwischen dem Target und OpenOCD hergestellt und arbeitet fehlerfrei, das heißt OpenOCD hat die einzelnen TAPs korrekt erkannt und meldet auch sonst keine weiteren Fehler, kann als nächstes der Debugger, beziehungsweise dessen graphische Oberfläche, gestartet werden.

Alternativ kann an Stelle des Debuggers auch mit Hilfe von Telnet direkt von der Konsole aus auf OpenOCD zugegriffen werden, um damit Befehle abzusetzen. Von OpenOCD wird hierfür standardmäßig der Port 4444 verwendet, jedoch hängt dies von der verwendeten Konfiguration ab. Der Standardport für GDB ist der Port mit der Nummer 3333.

Soll der Debugger verwendet werden, stehen wiederum zwei Möglichkeiten zur Auswahl. Die erste ist, den Debugger direkt über die Konsole zu starten und die zu übertragenden Befehle von Hand einzugeben. Die komfortablere Methode ist, eine graphische Oberfläche wie beispielsweise DDD oder Eclipse zu verwenden.

B. Laden des Programms

Wenn das Target angeschlossen und OpenOCD gestartet wurde, kann das zu debuggende Programm in den GNU Debugger geladen werden. Dies geschieht üblicherweise mit dem Befehl file. Dazu muss das Programm als Executable im „Executable and Linkable Format“ (ELF) vorliegen, also in einer normalen ausführbaren Datei auf unixartigen Systemen. Eine reine Binärdatei ist nicht ausreichend, da der Debugger die Symboltabelle des Programms benötigt, welche in diesem Format dem Programm beiliegt. Das Laden des Programms wird mit dem folgenden Befehl durchgeführt.

```
file program.elf
```

Das Verbinden des GDB-Clients mit dem in OpenOCD enthaltenen GDB-Server ist der nächste Schritt. Hierzu wird über die Kommandozeile, beziehungsweise von der graphischen Oberfläche der Befehl target remote abgesetzt. Das Kommmando target legt fest, welcher GDB-Server verwendet werden soll. Standardmäßig ist dies ein Modul des Clients selbst. Wird jedoch dahinter das Kommando remote angegeben, bedeutet dies, dass für das Debugging ein anderer Server verwendet werden soll. Das Kommando localhost gibt an, dass sich der Server auf dem selben Computer wie der Client befindet, andernfalls könnte man hier auch die IP-Adresse des Zielcomputers angeben. Mit der an localhost angehängten Zeichenkette :3333 gibt man an, dass auf dem Zielcomputer der Port 3333 verwendet werden soll, welcher vorher von OpenOCD geöffnet wurde. Hierdurch wird die Verbindung zwischen dem GNU Debugger und OpenOCD hergestellt.

```
target remote localhost:3333
```

C. Reset

Als nächster Schritt sollte das Target in einen kontrollierten Zustand gebracht werden. Um dazu geeignete Voraussetzungen zu schaffen ist es ratsam, an dieser Stelle das Target zu resetten. Zu beachten ist hierbei dass OpenOCD davon ausgeht zusätzliche Leitungen nutzen zu können, welche nicht im JTAG-Standard definiert sind. Diese Resetleitungen werden oftmals als TRST und SRST bezeichnet. Sind diese nicht verfügbar, muss der Reset manuell erfolgen. Jedoch kann ein Reset auch in begrenztem Umfang durch den Befehl soft_reset_halt simuliert werden, solange das Target normal funktioniert.

Soll aus GDB heraus ein Kommando von OpenOCD ausgelöst werden, kann dies dadurch erreicht werden, dass dem OpenOCD-Befehl das Schlüsselwort monitor vorangestellt wird. Somit ergeben sich folgende Kommandos für den Reset des Targets aus GDB heraus:

- *monitor reset*, der generische Resetbefehl von OpenOCD.
- *monitor reset run*, ein Reset, bei dem das Target explizit weiterläuft.
- *monitor reset halt*, ein Reset, bei dem das Target angehalten wird.
- *monitor reset init*, ein Reset, bei dem eine init-Funktion in der Konfigurationsdatei von OpenOCD ausgeführt wird.
- *monitor soft_reset_halt*, ein von OpenOCD über den ICE simulierter Reset.

D. Übertragen des Programms auf das Target

Nachdem der Reset des Targets ausgeführt wurde, muss dieses nun angehalten werden, um ungehinderten Zugriff auf den Speicher von diesem zu erhalten. Wie auch bei den Resetbefehlen stellt OpenOCD auch für diesen Befehl ein entsprechendes Kommando zur Verfügung, dieses kann aus GDB heraus mit monitor halt ausgelöst werden. Zwar wird das Target normalerweise bereits in der Resetphase angehalten, jedoch kann dieses Kommando hier nochmals abgesetzt werden, um sicherzustellen, dass das Target wirklich angehalten wurde.

Für das Debuggen eines Programms direkt auf dem Target muss das Programm erst auf dieses übertragen werden. Hierbei gibt es verschiedene Verfahrensweisen, die je nach den jeweiligen Anforderungen des Programms und der Hardware ausgewählt werden müssen. Der Grund ist, dass Programme sowohl nur im RAM als auch im RAM und Flash-Speicher des Targets liegen können. Dies ist von der Konfiguration des Linkerscripts abhängig. Im ersten Fall muss damit das Programm nur in den RAM, im zweiten Fall muss es zusätzlich auch in den Flash-Speicher geschrieben werden. Hier gilt es, aus den verfügbaren Methoden die benötigten auszuwählen.

Um ein Programm in den RAM des Targets zu übertragen, gibt es zwei Möglichkeiten. Die erste Möglichkeit ist, die GDB-Kommandos zu verwenden, während die Alternative die OpenOCD-Kommandos sind. Sind die GDB-Kommandos nutzbar, sollten jedoch diese verwendet werden. Die Alternative wird nur dann benötigt, wenn das Programm auf eine andere Adresse geschrieben werden muss als im Linkerscript angegeben wurde.

Um diese Methode zu verwenden, muss das Programm vorher mit dem Kommando file in den Debugger geladen worden sein. War dies erfolgreich, kann man das Programm nun mit dem Befehl load in den RAM des Targets schreiben. Dabei muss man beachten, dass GDB davon ausgeht, den gesamten Adressraum nutzen zu können und es deshalb das Programm standardmäßig auf die kleinste Speicheradresse schreibt. Einige Targets und deren empfohlene Konfigurationen reservieren jedoch Adressraum, wie beispielsweise das Armadeus apf27-Board, das den frei beschreibbaren RAM auf die Adresse 0xA0000000 legt. Dies hat dann zur Folge, dass GDB beispielsweise auf Adresse 0x0 schreiben will, dies aber fehlschlägt, da der Speicher in Wirklichkeit auf 0xA0000000 liegt. In so einem Fall sollte man, wenn möglich, das Linkerscript entsprechend anpassen. Die letzte Alternative sollte sein, auf die OpenOCD-Befehle zurückzugreifen.

Für OpenOCD Methode wird neben dem normalen Executable eine reine Binärversion des Programms benötigt. Diese erhält in der Regel die Dateiendung bin. OpenOCD ermöglicht es zwar auch, andere Formate zu übertragen, diese müssen dann allerdings erst in das Binärformat umgewandelt werden, was eine mögliche Fehlerquelle darstellt. Um eine elf-Datei in eine bin-Datei umzuwandeln, wird die Anwendung objcopy aus der verwendeten Toolchain benötigt.

```
arm-elf-objcopy --output-target=binary program.elf program.bin
```

Das damit erzeugte Binärimage kann nun mit dem OpenOCD Befehl load_image in den RAM des Targets übertragen werden. Im Gegensatz zur GDB-Methode kann hier die Zieladresse explizit angegeben werden. Bei der GDB-Methode ist dies nicht möglich.

```
monitor load_image program.bin 0x0
```

Der Flash-Speicher ist im Gegensatz zum RAM eines Targets ein persistenter Speicher. Das bedeutet, dass er auch nach Trennung der Stromversorgung seine gespeicherten Daten behält. Auf Targets kommen zwei verschiedene Arten von Flash-Speichern vor, die NOR-Flashs und die NAND-Flashs. Außerdem können sich diese direkt auf dem Chip befinden, was als intern bezeichnet wird, als auch extern, wenn sich der Flash auf einem eigenen Chip befindet und über Pins an den Chip des Prozessors angeschlossen ist. In jedem Fall wird aber ein zu dem Flash passender Treiber, ein Programmmodul von OpenOCD, benötigt. Ist kein passender Treiber verfügbar, kann nicht auf den Flash zugegriffen werden. Die hier erläuterten Verfahren lassen sich auch dazu verwenden, die Firmware eines funktionsunfähigen Targets wieder herzustellen.

Ein NOR-Flash ist eine Art von Flash-Speicher, welche aus NOR-Gattern besteht. Im Vergleich mit einem NAND-Flash ist ein aus NOR-Gattern bestehender Flash für gewöhnlich zuverlässiger, jedoch bedeutend teurer. Mit dem flash bank-Befehl wird OpenOCD angewiesen, den entsprechenden Treiber für den konfigurierten TAP zu laden. Dieser Befehl wird für gewöhnlich in der Konfigurationsdatei eingetragen und nur beim Verbinden mit dem Target einmal ausgeführt. Wurde der Treiber erfolgreich geladen, kann nun das Binärimage des Programms mit Hilfe des Befehls flash write_bank in den Flash-Speicher geschrieben werden.

Der NAND-Flash ist eine andere Bauart eines Flash-Speichers. Die allgemeine Vorgehensweise beim schreibenden Zugriff auf einen NAND-Flash, ist im folgenden angegeben.

```
nand device <name> <driver> <target>
nand probe <num>
nand write <num> <filename> <offset> [option...]
```

E. Startposition

Der nächste Schritt ist, den Programcounter auf die Startadresse des zu debuggenden Programms zu setzen. Hierzu muss zuerst die Adresse der Startfunktion, im Allgemeinen ist dies die main()-Funktion, des

Programms ermittelt werden. Wurde der Speicherbereich durch das Linkerscript korrekt gesetzt, kann der GDB-Befehl break verwendet werden. Nachfolgend wird dieser Befehl verwendet um einen Breakpoint auf eine Funktion mit dem Namen main() zu setzen.

Wenn später das Target mit dem Befehl continue gestartet wird, sollte dieses nun an dem gesetzten Breakpoint anhalten.

```
break main
```

Jedoch kann es vorkommen, dass das Programm auf eine andere Adresse geladen werden muss als die durch den Linker vorgesehene. Das in diesem Fall notwendige Vorgehen wird nun erläutert. Wird beim Erzeugen des Programms der Linker verwendet, kann dieser ein Logfile erzeugen, aus dem die betreffende Adresse ermittelt werden kann.

```
arm-elf-ld -v -Map program.map -o main.elf main.o object.o
grep main program.map
```

Jedoch ist die einfachere Methode, dennoch zuerst den GDB-Befehl break main zu verwenden, wobei main der Name der Funktion ist. Ist dieser Befehl erfolgreich, kann sofort mit dem nächsten Schritt fortgefahren werden. Selbst wenn der Befehl fehlschlägt, kann aus der resultierenden Fehlermeldung die Adresse der gewünschten Funktion abgelesen werden. Anhand dieser Adresse kann dann mit OpenOCD manuell ein Breakpoint angelegt werden. Nachfolgend ist der notwendige Befehl dargestellt, wobei in diesem Beispiel ein Breakpoint auf die Adresse 0xAFF01630 angelegt wird.

```
monitor bp 0xAFF01630 4 hw
```

Wenn die Adresse von vornherein schon bekannt ist, kann der Programcounter mit dem Befehl monitor resume 0x0 auf die gewünschte Adresse, in diesem Fall 0x0, gesetzt und das Programm ausgeführt werden.

F. Debuggen

Sobald das Kommando continue, beziehungsweise monitor resume ausgeführt wurde, läuft das Target weiter und sollte an dem gesetzten Breakpoint warten. Es kann nun wie üblich das Debugging durchgeführt werden. Jedoch sollte man beachten, dass es auf den Targets Softwarebreakpoints und Hardwarebreakpoints gibt, GDB jedoch meist so konfiguriert wird, dass Hardwarebreakpoints verwendet werden. Wie dies konfiguriert wird, unterscheidet sich von Target zu Target und auch zwischen den unterschiedlichen Versionen von OpenOCD. Da die Anzahl der Hardwarebreakpoints stark begrenzt ist, sollte man also mit der Verwendung von Breakpoints sehr sparsam umgehen.

VI. Debugging mit DDD

Um den Start von OpenOCD und DDD zu automatisieren, können Startscripte eingesetzt werden. Mit ihnen ist es möglich, die benötigten Programme wie ein gewöhnliches Executable, beispielsweise durch Doppelklick zu starten, ohne auf die Kommandozeile zugreifen zu müssen. Die nachfolgend angegebenen Textzeilenpaare müssen lediglich in jeweils eine Textdatei kopiert werden. Üblicherweise erhält diese die Dateiendung sh für Shell Script. Dieser Datei muss man anschließend Ausführungsrechte mit chmod 755 zuweisen. Der Befehl zum Starten von OpenOCD sollte derjenige sein, der für das jeweilige Target benötigt wird, wobei wenn möglich für die mit dem Parameter -f eingebundenen Konfigurationsdateien relative Pfade verwendet werden sollten. Dies gilt auch für das Startscript für DDD. Die hier verlinkte Datei, hier gdbinit.conf genannt, enthält das zuvor behandelte Initialisierungsscript für das jeweilige Target zum Starten der Debugsession. Die relativen Pfade müssen von der Position des Startscripts in der Verzeichnisstruktur ausgehend definiert werden.

```
#!/bin/sh
openocd -f interface.cfg -f target.cfg
```

```
#!/bin/sh
ddd --debugger arm-elf-gdb -x gdbinit.conf
```

VII. Debugging mit Eclipse

Die Eclipse-Entwicklungsumgebung kann neben der Entwicklung auch das gesamte Debugging verwalten. Dies umfasst neben dem eigentlichen Debugging auch den Start von OpenOCD. Dazu können die External Tools verwendet werden, mit denen Konsolenprogramme aus Eclipse heraus gestartet werden können. Dazu muss zuerst eine entsprechende Debugkonfiguration erstellt werden.

In dem durch die External Tools Configurations geöffneten Dialogfenster kann nun eine neue Konfiguration für OpenOCD angelegt werden. Dabei sollte bei Location der Installationsort von OpenOCD, bei Working Directory der Pfad zu den verwendeten Konfigurationsdateien und bei Arguments der Startbefehl eingetragen werden. Ist dies geschehen, kann die Konfiguration mit einen Klick auf Apply gespeichert werden. Mit einem Klick auf Run kann somit OpenOCD aus Eclipse heraus gestartet werden.

Alternativ kann zum Starten von OpenOCD auch das von DDD verwendete Startscript für OpenOCD verwendet werden. Dies bietet nebenbei den Vorteil, dass das Ausgabefenster der Konsole automatisch größer und somit auch übersichtlicher ist, weiterhin wir dieses auch nicht durch die während des Debuggings geöffnete Debugkonsole unzugänglich, wie dies innerhalb von Eclipse der Fall ist. Ein weiterer Vorteil ist, dass wenn Eclipse abstürzen sollte, die Verbindung mit dem Target nicht automatisch auch abreißt.

Zum Debugging eines Programms muss man nun zuerst eine entsprechende Debugkonfiguration anlegen. Dieser Schritt muss für jedes einzelne Projekt durchgeführt werden, das man debuggen möchte. In dem nun erschienenen Menü können über den Menüpunkt Debug Configurations die verschiedenen Debugkonfigurationen für die einzelnen Projekte verwaltet werden.

In dem durch einen Klick auf Debug Configurations erhaltenen Dialogfenster kann man nun die Konfigurationen verwalten. Um über JTAG direkt auf der Hardware zu debuggen muss hier zunächst eine neue Konfiguration vom Typ Zylin Embedded debug (Native) für das zu debuggende Projekt angelegt werden. Hierzu muss der entsprechende Eintrag zuerst markiert und dann auf den New-Button links oben geklickt werden.

Als nächster Schritt müssen nun die Konfigurationseinstellungen bearbeitet werden. Hier sind die Karteikarten Main, Debugger und Commands von Bedeutung. Auf der Karteikarte Main wird zunächst bei Project das Projekt ausgewählt und bei C/C++ Application der Name des Executables eingegeben. Auf der Karteikarte Debugger muss bei GDB Debugger der GDB der verwendeten Toolchain eingetragen werden. Auf der Karteikarte Commands kann nun das Initialisierungsscript direkt in das Textfeld Initialize commands eingetragen werden. Mit einem Klick auf Apply kann die Konfiguration nun gespeichert werden. Mit einem Klick auf Debug wird die Debugsession gestartet.

Es folgen einige der wichtigsten Kommandos:

- *monitor*: Absetzen eines OpenOCD-Befehls aus GDB heraus.
- *target*: Zur Kopplung mit OpenOCD.
- *file*: Lädt ein Programm in den GNU Debugger.
- *add-symbol-file*: Lädt eine Programm bei alternativer Speicheradresse.
- *load*: Lädt ein Programm in den RAM des Targets. Dieser Befehl ist mit dem OpenOCD-Befehl *load_image* vergleichbar.
- *break*: Setzt einen Breakpoint, vergleichbar mit dem OpenOCD-Befehl bp.
- *continue*: Führt die Programmausführung fort, vergleichbar mit dem OpenOCD-Befehl *resume*.
- *step*: Lässt das Programm eine Codezeile weiterlaufen.

VIII. Weitere Arbeiten

Aktuell wird ein FPGA entwickelt, der eine Debug-Hardware emulieren soll. Ein funktionales memory-mapped Register wird an einen Wishbone-Bus angeschlossen, ein TAP-Controller wird integriert und parallel zum funktionalen Register wird ein JTAG-Datenregister eingebaut. Somit ist es möglich, das Peripheral-Register über den Wishbone-Bus zu beschreiben und unabhängig davon dieses Register

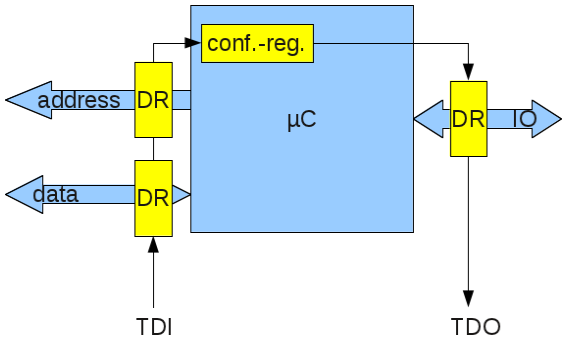


Abbildung 2: Debug-Hardware – JTAG-Dataregister.

mit Hilfe des JTAG-Datenregisters zu verändern. Damit einher geht, einen OpenOCD-Treiber für unseren TAP-Controller zu schreiben (Abb. 2). Unser Ziel ist, einen Hardwaredebugger zu implementieren, in dem Daten, Adressen, IOs und einige prozessorinterne Register via JTAG manipulierbar sind. Dieses Prinzip wird bei ARM-Prozessoren und auch bei Infineon-Chips angewendet.

Literaturverzeichnis

[1] IEEE Std 1149.1-1990 “IEEE Standard Test Access Port and Boundary-Scan Architecture”.

[2] D. Rath, “Open On-Chip Debugger” http://openocd.sourceforge.net/thesis.pdf, Jan. 2012.

[3] OpenOCD User's Guide, http://openocd.sourceforge.net/doc/html/index.html, Jan. 2012.

[4] C. Valens, “Zepter: Debuggen über JTAG”, Elektor 3-2011.

[5] J. Lynch, “Using Open Source Tools for AT91SAM7 Cross Development” http://www.atmel.com/dyn/resources/prod_documents/atmel_tutorial_source.zip, Jan. 2012.

Bernd Adler erhielt den akademischen Grad B.Sc. in Informatik im Jahr 2009 von der Hochschule Ravensburg-Weingarten, wo er sein Masterstudium in Informatik voraussichtlich im Jahr 2012 beendet.

Andreas Siggelkow erhielt den akademischen Grad Dipl.-Ing. Elektrotechnik im Jahr 1988 von der Universität Karlsruhe. Im Jahr 1996 promovierte er an der Universität Stuttgart zum Dr.-Ing. Seit dem Jahr 2007 ist er Professor für ASIC-Design und Rechnerarchitekturen an der Hochschule Ravensburg-Weingarten.

Kryptografische USB-Schnittstelle

Camille Marbach, Timo Pfander, Norbert Reifschneider

***Zusammenfassung*—Das vorliegende Projekt - realisiert als Studienarbeit an der Hochschule Heilbronn - beschreibt eine kryptografisch gesicherte USB-Schnittstelle in einem FPGA. Hierfür werden die Kryptographieverfahren RSA zum Schlüsseltausch sowie TDES bzw. AES für den Datenaustauschverwendet. Die Hardware wird in einem Xilinx Spartan 3E FPGA implementiert. Die kryptografischen Algorithmen werden in VHDL entwickelt und zu IP-Cores zusammengefasst. Für die USB-Schnittstelle kommt der LogiCore XPS USB2 der Firma Xilinx zum Einsatz. Die IP-Cores werden in Verbindung mit einem MircoblazeEmbedded-Processor zu einem System-on-Chip zusammengefasst. Der Embedded-Processor übernimmt die Kommunikationssteuerung und dient als Schnittstelle für zukünftige Systemerweiterungen.**

***Schlüsselwörter*—FPGA, TDES, AES, RSA, Kryptografie, USB, Spartan-3E.**

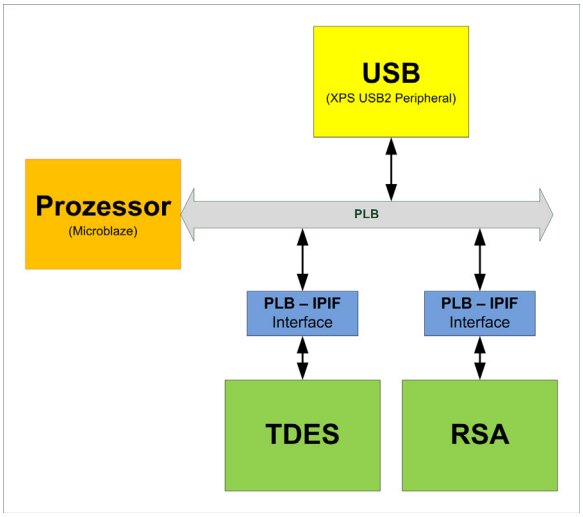


Abbildung 1: Blockschaltbild kryptografische USB-Schnittstelle

I. Einleitung

Die Verschlüsselung von Daten spielt heute eine große Rolle in unserem Alltag. Erst eine Verschlüsselung trägt zu dem hohen Maß an Sicherheit bei, die wir gewöhnt sind. Verschlüsselt wird heute in nahezu allen Bereichen der elektronischen Datenverarbeitung, sei es beim Bezahlen mit der Kreditkarte, dem Funkschloss am Auto oder der Datenübertragung zwischen Computern und in Netzwerken.

Der Aufwand für eine Verschlüsselung hängt dabei grundsätzlich von den Anforderungen ab. Beispielsweise wird die Übertragung von Bankdaten viel kritischer gesehen als das Öffnen eines elektrischen Garagentors. Für die heute üblichen Algorithmen stellt sich immer die Frage: Hardware oder Software? Die beiden Varianten werden im Folgenden kurz erläutert.

Camille Marbach, info@camille-marbach.de, Timo Pfander, t.pfander@web.de und Norbert Reifschneider, reifschneider@hs-heilbronn.de, Max-Planck-Str. 39, 74081 Heilbronn.

II. Aufgabenstellung

Das Ziel des Projekts ist die Entwicklung einer kryptografisch gesicherten USB-Schnittstelle. Diese soll über einen gesicherten USB-Kanal zunächst unverschlüsselte Rohdaten von einem Computer empfangen, diese nochmals verschlüsseln und dann z.B. über das Internet versenden können. Ebenso soll der Vorgang in umgekehrter Richtung möglich sein. Ferner soll zwischen Computer und Schnittstelle eine Authentifizierung stattfinden, so dass weder die Software auf dem Computer durch eine fremde Hardware getäuscht noch eine z.B. entwendete Schnittstelle an einen nicht autorisierten Computer angeschlossen werden kann.

Die Daten müssen innerhalb der Schnittstelle nicht mittels eines angreifbaren Software-Algorithmus verschlüsselt werden, sondern werden in einem externen Gerät via Hardware verschlüsselt. Diese Form bietet neben der Entlastung des PC und der höheren Geschwindigkeit auch Sicherheitsvorteile, da die Hardware nicht durch Hacker angegriffen werden kann. Die Hardware im externen Gerät besteht aus einem FPGA, in dem die Verschlüsselungstechnologie auf Basis von VHDL-Code realisiert wird. Die Implementierung von Verschlüsselungsalgorithmen kann auf programmierbaren Logikbausteinen besonders effi-

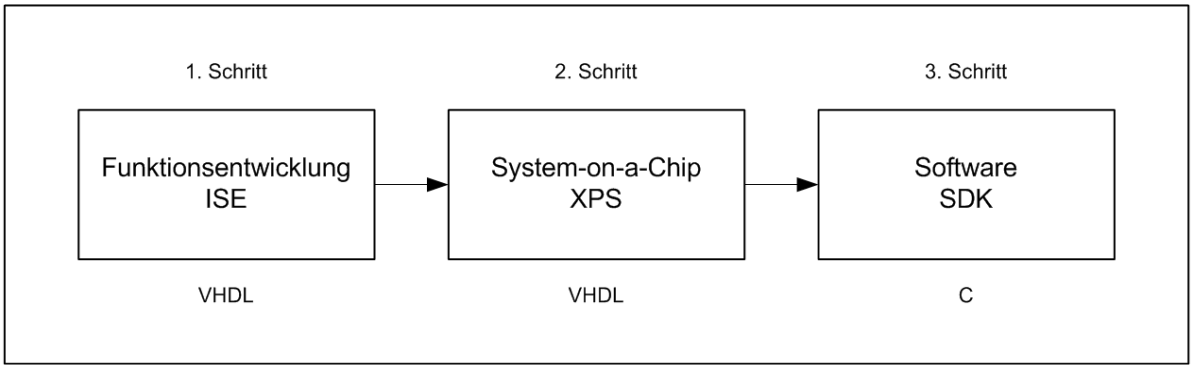


Abbildung 2: Entwicklungsschritte für ein System-on-Chip-Projekt

zient erfolgen, da sie aus vielen sich wiederholenden Rechenaufgaben bestehen. Neben der Tabellentransformation (Table Look-Up) sind diese Rechenaufgaben hauptsächlich Addition, Multiplikation und Bit-Rotation.

III. Konzept

Im Blockschaltbild (Abbildung 1) ist die Struktur des Gesamtsystems ersichtlich. Der Microblaze-Prozessor, das PLB-IPIF-Interface und der USB-Core sind fertige IP-Cores, die von Xilinx zur Verfügung gestellt werden. Für die Cores TDES, AES und RSA werden die Schnittstellen entwickelt und angepasst, der RSA-Algorithmus wird in VHDL selbst entworfen. Der Datenaustausch zwischen den Cores wird vom Microblaze-Prozessor gesteuert.

A. Betriebsablauf Gesamtsystem

Die Steuerung der Kommunikation zwischen den IP-Cores erfolgt mittels eines Microblaze Embedded Prozessors. Durch das Erzeugen eines Workspace im Plattformstudio (XPS) wird die Projektumgebung vorkonfiguriert. In der Xilinx Software-Entwicklungsumgebung (SDK) wird nach den üblichen Designstrategien für die Programmiersprache C das Programm für den Microblaze entwickelt.Eine Besonderheit stellt die Datei '`xparameters.h`' dar. Diese vom XPS beim Export erzeugte Datei enthält die Adressbereiche der IP-Cores für die Adressierung in der Software.

IV. Verschlüsselungstechnologie

A. TDES

Das TDES-Verfahren existiert bereits seit 1976 und wurde von IBM mit Unterstützung vom amerikanischen Geheimdienst NSA entwickelt. Die Sicherheit von DES wurde schon vor einiger Zeit in Frage gestellt, deshalb hat man sich dazu entschieden, das Verfahren drei Mal nacheinander mit verschiedenen Schlüsseln anzuwenden und damit die Sicherheit um ein Vielfaches zu erhöhen. Bei der Entwicklung des Verfahrens wurde darauf geachtet, dass ein gutes Synthetisieren (Realisierung mit logischen Grundelementen) in Hardware möglich ist, was dem VHDL-FPGA-Entwurf entgegen kommt.

Beim DES-Verfahren handelt es sich um ein symmetrisches Verschlüsselungsverfahren. Dabei wird die Ver- und Entschlüsselung mit dem gleichen Schlüssel durchgeführt. Die Längen von Klartext und Geheimtext sowie die Schlüssellänge sind identisch. Das Verfahren basiert im Wesentlichen auf drei Rechenoperationen:

- Permutation (Vertauschen von Bits)
- Exklusiv-Oder Funktion (EXOR)
- Substitution (ersetzen von Bitfolgen)

Die Rundenfunktion F bildet den wichtigsten Teil des Algorithmus. Sie verarbeitet den Schlüssel und führt zu einer als zufällig annehmbaren Bitfolge. Nach einer Anfangspermutation wird der Eingangswert mit der Rundenfunktion EXOR-Verknüpft, dieser Prozess wird sechzehn Mal nacheinander durchlaufen. Danach wird das Ergebnis noch einmal permutiert.

Die Implementierung des TDES-Algorithmus wird auf Basis des zur Verfügung gestellten Projekts "Des-Hard" von Prof. Dr. Norbert Reifschneider realisiert. Die Top-Levels von DES und TDES wurden von der Schaltplanprogrammierung (Schematic) in VHDL umgeschrieben und in die Xilinx-Entwicklungsumgebung in einer separaten Bibliothek eingebunden. Die Funktion von Algorithmus und Testdaten zur Verifizierung sind in den Quellen [1] und [2] dargestellt.

B. RSA

1) Grundlagen RSA-Verfahren

Das RSA-Verfahren ist das bekannteste Public-Key-Verschlüsselungsfahren (asymmetrisches Kryptographieverfahren). Das Verfahren wurde 1977 von R. Rivest, A. Shamir und L. Adleman entwickelt, die gleichzeitig die Namensgeber sind.

Der große Vorteil der Public Key Verschlüsselung liegt im Bereich der Schlüssel. Es werden immer zwei Schlüsselpaare erzeugt, die jeweils einen öffentlichen und einen privaten Schlüssel enthalten. Der öffentliche Schlüssel ist jedem Nutzer frei zugänglich und dient der Verschlüsselung. Der private Schlüssel dient nur

der Entschlüsselung und wird geheim gehalten. Mit dem öffentlichen Schlüssel kann eine verschlüsselte Nachricht nicht decodiert oder der private Schlüssel berechnet werden. Dies ermöglicht daher ein gefahrloses Austauschen der Schlüssel und ist somit ein großer Vorteil gegenüber der symmetrischen Verschlüsselung.

Die Grundlage des RSA-Verfahrens bildet die Einwegfunktion mit Falltür. Eine Einwegfunktion ist definiert durch folgende zwei Eigenschaften:

- Es gibt eine effiziente Methode für die Berechnung von $f(x)$ für alle $x \in X$.
- Es gibt keinen „schnellen“ Algorithmus für die Berechnung von x aus der Beziehung von $y = f(x)$ für alle $y \in f[x]$.

Die Falltürfunktion erlaubt jedoch eine Einwegfunktion umzukehren, allerdings nur mittels einer zusätzlichen Information, die im Allgemeinen geheim gehalten wird. Die Einwegfunktion beim RSA-Verfahren bildet die Multiplikation zweier Primzahlen miteinander, da die Faktorisierung eines solchen Produkts und damit die Zerlegung in die ursprünglichen Primzahlen viel Zeit beansprucht. Die Falltür ist die Berechnung der modularen Inversen, die für die Entschlüsselung benötigt wird. Diese lässt sich sehr einfach unter der Voraussetzung berechnen, dass man die Primzahlen (oder eine davon) kennt.

Das Ver- und Entschlüsseln beruht auf den beiden mathematischen Operationen:

- Potenzieren
- Division mit Rest (modulo)

2) Implementierung in VHDL

Die Verschlüsselung erfolgt über die Gleichung:

$$C = m^e \bmod n .$$

Die Umkehrung, d.h. die Entschlüsselung erfolgt nach der Vorschrift

$$m = C^d \bmod n ,$$

wobei m dem Klartext und C dem codierten Klartext (Codetext) entspricht. Der öffentliche Schlüssel besteht aus (n,e) und der private Schlüssel aus (n,d).

Zuerst wird der öffentliche und private Schlüssel berechnet. Hierbei gilt, dass n das Ergebnis der Multiplikation der beiden Primzahlen ist. Nun wird noch die Eulersche Funktion von n berechnet. Diese gibt an, wie viele teilerfremde Zahlen n hat und wird für die Berechnung von e und d benötigt. Da die Zahl n aus zwei Primfaktoren berechnet wurde ergibt sich die Eulersche Funktion zu:

$$\varphi(x) = (\text{Primzahl}1 - 1) \cdot (\text{Primzahl}2 - 1)$$

Hier ergibt sich nun auch die Falltür. Wie unter Punkt 1) beschrieben, besteht die Falltür darin, dass man die modulare Inverse berechnen muss. Dies geschieht über die Eulersche Funktion, die man mit Kenntnis der Primzahlen sehr leicht und ohne Kenntnis nicht in angemessener Zeit bestimmen kann.

Mit der Eulerschen Funktion kann man dann die Zahl e berechnen oder eine konstante Zahl definieren für die gilt:

$$1 < e < \varphi(n) \text{ und } ggT(\varphi(n), e) = 1$$

Die Zahl d hingegen ist die modulare Inverse zu der Zahl e. Diese wird mit Hilfe des erweiterten Euklidischen Algorithmus berechnet.

Die Ver- und Entschlüsselung einer Zahl erfordert, wie aus den Gleichungen ersichtlich, das Potenzieren des Klartextes bzw. des Codetextes mit einer großen Zahl. Damit die Zahlen nicht zu groß geraten, nimmt man den Square-and-Multiply Algorithmus in Verbindung mit der Modulo Rechnung zur Hilfe. Ein angenehmer Nebeneffekt des Square-and-Multiply Algorithmus ist, dass bei einer Rechnung der Form x^k anstatt k Multiplikationen weniger als $2 \cdot \log_2(k)$ Multiplikationen nötig sind.

3) Sicherheit des RSA-Verfahrens

Das Ziel der Kryptoanalyse ist klar. Der Angreifer möchte aus dem verschlüsselten Text den Klartext berechnen. Es gibt verschiedene Methoden um dies zu versuchen. Beim RSA-Verfahren sind viele Parameter frei wählbar, unter anderem die beiden Primzahlen und die Zahl e für den öffentlichen Schlüssel. Eine falsche oder zu kleine Wahl dieser Parameter kann dem Angreifer sein Vorhaben drastisch erleichtern. Je komplizierter die mathematische Beschreibung des Algorithmus ist, desto mehr Fehler können eingebaut werden und bieten somit mehr Angriffsfläche.

Die Angriffe auf Verschlüsselungsverfahren können in mehrere Gruppen aufgeteilt werden:

- Faktorisierungsangriff
- Brute-Force-Methode (Vollständige Schlüsselsuche)
- Low-Exponent-Attacke
- Seitenkanalangriff

Beim Faktorisierungsangriff wird versucht die Zahl n zu faktorisieren, d. h. in die beiden Primzahlen zu zerlegen. Darum sollten die Primzahlen groß genug gewählt werden. Eine Empfehlung lautet hierbei, dass die Primzahlen etwa 512 Bit lang sein sollten, was einen Schlüssel von 1024 Bit ergibt. Dies ist für Anwendung mit nicht allzu hohen Sicherheitsanforderungen ausreichend. Für besonders sicherheitsrelevante Daten wird ein Schlüssel von mindestens 2048 Bit Länge empfohlen.

Die Brute-Force-Methode ist bei den empfohlenen Schlüssellängen keine große Gefahr mehr, da bei dieser Methode alle nur möglichen Schlüssel nacheinander ausprobiert werden. Dies würde eine nahezu unendlich lange Zeit in Anspruch nehmen.

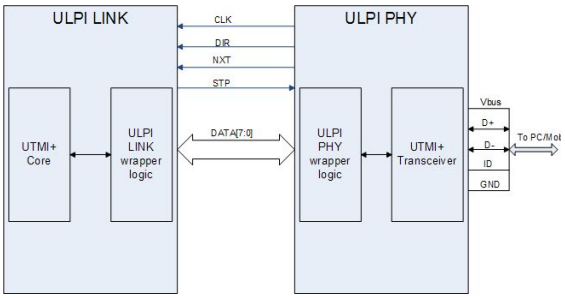


Abbildung 3: Blockschaltbild ULPI und UTMI Interface (Quelle: www.ulpi.org)

Die Low-Exponent-Attacke kann es dem Angreifer bei einer zu klein gewählten Zahl *e* ermöglichen, den privaten Schlüssel sehr schnell zu errechnen. Wird beispielsweise die gleiche Nachricht oder die annähernd gleiche Nachricht (die ersten 1.000 Zeichen sind gleich) an genau oder mehr als *e* Empfänger gesendet, kann der Angreifer mit dem Chinesischen Restsatz rasch den privaten Schlüssel berechnen.

Die Seitenkanalangriffe sind nicht nur speziell auf das RSA Verfahren ausgerichtet, können dieses aber bei falscher Implementierung auch treffen. Hierbei wird entweder über den Stromverbrauch oder die Berechnungszeit die Zahl *d* 'gemessen'. Dies geschieht bei der Verwendung des Square-and-Multiply Algorithmus, da dieser den Exponenten in binärer Darstellung verarbeitet. In Abhängigkeit der Werte von 2^0 bis 2^{k-1} werden dabei verschiedene mathematische Operationen ausgeführt. Bei einer Null führt der Algorithmus eine Multiplikation und Quadrierung durch, bei einer Eins nur eine Quadrierung. Diese beiden unterschiedlichen Behandlungen sind anhand der Berechnungszeit bzw. Stromaufnahme ersichtlich, woraus man dann die Binärdarstellung der Zahl *d* erhalten kann.

V. USB-Schnittstelle

Das Spartan-3E-Starter-Kit verfügt über keine USB-Schnittstelle zur Datenkommunikation, deshalb ist eine externe Hardware an die Erweiterungsschnittstelle des Starter-Kits anzuschließen. Der USB-Port auf dem Starter Kit dient lediglich zur Programmierung und zum Debuggen[1], mit ihm können keine Daten ausgetauscht werden. Aufgrund der Komplexität der USB-Schnittstelle wird die Anbindung eines passenden Bausteins in Verbindung mit einem Soft-Core realisiert.

A. Xilinx IP-Core

Das in der Xilinx-Entwicklungsumgebung verfügbare IP-Core für die USB-Funktionalität, LogiCore XPS USB2, muss käuflich erworben werden. Für dieses Projekt soll eine kostenlose 120 Tage-Evaluierungslizenz von Xilinx verwendet werden.

Das Core kommuniziert intern über den "Processor Local Bus" (PLB) mit dem Microblaze, nach außen wird der Treiberbaustein über die ULPI-Schnittstelle angesprochen. Bei diesem Interface werden die Daten, die vom FPGA über den USB übertragen werden sollen, in definierte Register des Cores geschrieben. Umgekehrt werden die zu lesenden Daten aus den entsprechenden Registern gelesen. Das Core verfügt außerdem noch über eine Vielzahl von Registern zur Konfiguration der FPGA internen Busanbindung, der USB-Funktionalität und der ULPI-Schnittstelle [7]. Die Registerbeschreibung kann dem Datenblatt [3] des USB-Cores entnommen werden.

B. Hardware

Für die Hardwareanbindung wird ein Physical-Layer-Baustein (PHY) mit der beschriebenen ULPI-Schnittstelle angeschlossen. Die Entscheidung fiel auf den USB3300 von SMSC, da hier eine ausführliche Schnittstellenbeschreibung und ein Demoboard (USB3300) für Xilinx-FPGAs existieren. Andere PHY-Bausteine für die ULPI-Schnittstelle hätten zum Eigenbau einer Platine geführt, weitere fertig aufgebaute Alternativen waren zum Zeitpunkt der Auswahl nicht verfügbar.

VI. Implementierung

A. TDES

1) TDES-Module

Das Modul (engl. module) stellt den Top-Level des erstellten ISE-Projekts dar. Die Ports der Entity, die vorher nach außen führten, werden nun in der nächsthöheren Ebene, dem Kernel, weiterverarbeitet.

2) TDES-Kernel

Im Kernel werden die Signale aus dem Top-Level des TDES-Algorithmus mit den Registern des IPIC-Interface verbunden. Weiterhin werden im Kernel die Register auf den IPIC-Bus, der die Zwischenstufe zwischen User-Core und PLB-Bus bildet, geschrieben bzw. gelesen. Die Anzahl der Register wurde auf die erforderliche Anzahl von 14 reduziert.

Für den Zugriff auf die Register des User-Cores über den PLB-Bus muss für jedes Register eine eindeutige Adresse festgelegt werden (memory map). Für jeden Kern, der im System-on-Chip vorhanden ist, wird eine Basisadresse (C_BASEADDR) festgelegt. Die Berechnung der Registeradresse zur Verwendung im Microblaze oder anderen Systemkomponenten läuft folgendermaßen ab:

Registeradresse = C-BASEADDR + Adresse relativ

[1] Fehlersuche und -Diagnose in Programmen.

Um also beispielsweise das "lower double word" des Datenausgangs vom TDES-Core zu lesen, muss bei einer Basisadresse von 0x90000000 und einem Adress-Offset von 0x30 die Registeradresse 0x9000 0030 adressiert werden.

3) TDES-IO

Im Top-Level des TDES, dem tdes.vhd wurde nichts verändert. Diese Datei wird durch das Erzeugen von Templates mit dem "Create or Import Periphial"-Assistent im XPS automatisch mit den passenden Einstellungen für das Core (Busschnittstelle, Interrupts, Adressierung) erzeugt. Da das User-Core keine externe Schnittstelle bzw. Anschlüsse besitzt wie z.B. das USB-Core, stellen diese Ports nur den PLB-Bus zur FPGA-internen Verbindung dar.

Die PLB-Bus-Parameter sind Defaultwerte und werden bei der Erstellung des IPIF-Interface gesetzt. Die Ports und Parameter des PLB-Bus werden auf der Kernel-Ebene an den Kernel des PLB-Interface übergeben. Der PLB-Kernel 'plbv46_slave_single_v1_01' besitzt neben den PLB-Ports auch die Ports für die IPIC-Schnittstelle: Hier wird der IPIC-Bus auf den PLB-Bus umgesetzt.

B. USB

Der USB-Kern (XPS-USB2-Peripheral) wird im Plattformstudio aus dem IP-Core Katalog ausgewählt und zum Projekt hinzugefügt. Anschließend muss der Kern mit dem PLB Bus des Microblaze (mb_plb) verbunden werden und ein Adressbereich bzw. Startadresse eingestellt werden.

Bei einer Anbindung an Hardware, wie dem USB3300 von SMSC müssen die Anschlüsse eingestellt und die entsprechenden Portpins in die Zuordnungsdatei eingetragen werden. Zu dem USB-Kern gibt es ein Datenblatt [3] und ein Application Sheet [4].

C. Microblaze

Die Software wird, wie in Kapitel III.A. beschrieben, entwickelt. Das Flussdiagramm (Abbildung 3) beschreibt die im Microblaze implementierte Funktion der Steuerung zwischen USB-Schnittstelle und Verschlüsselungsalgorithmus.

D. Software Microblaze-Embedded-Prozessor

Im SDK kann auf Bibliotheksfunktionen zur Kommunikation mit den Kernen zurückgegriffen werden. Im vorliegenden Fall stellen diese Treiber dar.

- USB: Für die Initialisierung und Kommunikation mit dem USB-Kern stellt Xilinx das Headerfile '`xusb.h`' zur Verfügung. Hier werden zahlreiche Funktionen definiert, die zu Initialisierung, Betrieb und Datenaustausch benötigt werden.
- TDES: Für die Kommunikation mit dem TDES-Core werden die universellen Schnittstellenfunktionen mittels der Xilinx-Bibliotheksfunktionen "General Purpose Input/Output" ('`xgpio.h`') verwendet. Hier muss ebenfalls eine Initialisierung erfolgen, danach kann auf die in der Memory Map beschriebenen Register direkt zugegriffen werden.

Mit Hilfe der automatisch erstellten Headerdatei '`parameters.h`' können die Adressen der Kerne mittels Variablen an die Bibliotheksfunktionen für USB und TDES übergeben werden. Empfehlenswerte Literatur hierfür ist die Einführung zur Programmierung einer Software-Applikation [5].

VII. ZUSAMMENFASSUNG

Das IP-Core für die Ver- und Entschlüsselung mittels TDES sowie der Microblaze Embedded Prozessor wurden erfolgreich implementiert. Das TDES-Core wurde mittels der Tabellen in den Quellen [1] und [2] verifiziert. Der RSA-Algorithmus wurde entwickelt und für die Implementierung als IP-Core vorbereitet. Das USB-Core konnte aufgrund der Lizenzproblematik nicht getestet werden, die Implementierung erfolgte nach Quelle [3]. Der Quellcode für die Ablaufsteuerung mit dem Microblaze Embedded Prozessor wurde in der Simulation verifiziert.

VIII. AUSBLICK

Die nächsten Schritte zur Umsetzung des Gesamtprojekts sind die Einbindung des RSA-Projekts als IP-Core in das System-on-Chip, die Schlüsselerzeugung bzw. die Generierung von dazu benötigten Primzahlen aus echten Zufallszahlen, die Verifizierung der USB-Schnittstelle inklusive externer Hardwarekomponenten, die Entwicklung einer Benutzeroberfläche für ein PC-Betriebssystem sowie die Integration weiterer Verschlüsselungsverfahren und Schnittstellen (Abb. 6). Im nächsten Schritt sollte die Gesamtarchitecktru verifiziert werden. Für weitere Studien- oder Diplomarbeiten könnte man sich vorstellen, das Gesamtprojekt in einem kleineren FPGA oder einem Spartan-3E-Entwicklungsboard umzusetzen.

IX. DANKSAGUNG

Die Autoren bedanken sich bei der Hochschule Heilbronn für die Ausgabe und Betreuung der interessanten Studienabeit. Ferner gilt unser Dank Herrn Thomas Finke, wissenschaftlicher Mitarbeiter der Fakultät Technik 1, für die kompetente Unterstützung bei offenen Fragen.

LITERATURVERZEICHNIS

[1] S. S. Keller, “Modes of Operation Validation System for the Triple Data Encryption Algorithm (TMOVS): Requirements and Procedures”, *NIST Special Publication* 800-20*, US. Department of Commerce/National Institute of Standards and Technology*, 2000.

[2] W. Mehuron, “Data Encryption Standard (DES)”, *US. Department of Commerce/National Institute of Standards and Technology*, 1999.

[3] Xilinx, “LogiCORE IP XPS Universal Serial Bus 2.0 Device (v5.00a)”, December 2010. (DS639).

[4] Xilinx, “Reference Design: LogiCORE™ OPB USB 2.0 Device”, Tutorial zur Anwendung des USB-Cores (XPS USB2 Device), June 2010.

[5] http://www.fpgadeveloper.com/2011/06/write-a-software-application-with-sdk.html, Tutorial zur Entwicklung einer Anwendung mit Xilinx SDK, (14.02.2012).

Camille Marbach ist eingeschriebener Student der Hochschule Heilbronn im Studiengang Elektronik und Informationstechnik. Die Hauptinteressen und Studienschwerpunkte liegen in der Digital- und Informationstechnik, im besonderen FPGA und Mikrocontroller.

Timo Pfander war zum Zeitpunkt des Projekts Student an der Hochschule Heilbronn im Studiengang Elektronik und Informationstechnik. Gegenwärtig studiert er an der Fachhochschule Lübeck im Studiengang Kommunikations-, Informations- und Mikrotechnik. Die Hauptinteressen und Studienschwerpunkte liegen vor allem im Bereich der Informationstechnik und Embedded Systems.

Prof. Dr. Norbert Reifschneider ist Professor für Elektronik und Informationstechnik der Fakultät Technik 1 an der Hochschule Heilbronn.

Hunting PCI Jitter – A Real World Example

Hermann Ruckerbauer

***Zusammenfassung*—Um die Funktionalität von seriellen Schnittstellen sicherzustellen ist es notwendig, die Signalqualität mittels „Compliance Tests“ zu bewerten. Diese zeigen, ob die Kombination aus den verwendeten Bausteinen und dem PCB Design/Layout ein funktionsfähiges System ergeben. Im vorliegenden Beispiel war das Ergebnis des Compliance Tests „Fail“ und die Ursache dafür musste gefunden werden.**

***Schlüsselwörter*—PCIe, Compliance Test, Eye Diagram, Datenauge, Jitter, Time Interval Error, TIE.**

I. Einleitung

Die Geschwindigkeiten der heute eingesetzten seriellen Schnittstellen werden mit jeder Generation höher. Beispiele hierfür sind:

- USB 1/2/3: 0.01 Gb/s, 0.48 Gb/s, 5 Gb/s
- PCIe 1/2/3: 2.5 Gb/s, 5 Gb/s, 8 Gb/s
- SATA 1/3/3: 1.5 Gb/s, 3 Gb/s, 6 Gb/s

Diese Geschwindigkeitssteigerung führt zu immer höheren Anforderungen an das Design und vor allem das Layout auf der Leiterplatte. Meist beinhalten diese „digitalen“ Schnittstellen die Möglichkeit, verschiedene Geräte anzuschließen. Auf einem Computersystem ist meist ein Controller oder Prozessor die Quelle für diese Busse. Der Benutzer kann dann verschiedenste Geräte über Steckverbinder anschließen. Im Falle von USB erfolgt dies über Kabel, bei PCIe sind es Einsteckkarten. Um sicherzustellen, dass ein System auch mit allen Partnern funktioniert, definieren die Spezifikationen der Busse sehr genau welche Eigenschaften das Signal am Stecker aufzuweisen hat. Dies beinhaltet z. B. die Länge eines Bits (UI = Unit Intervall), Jitter, Spannungspegel und Augenweiten. Diese Messungen werden häufig mit einem Oszilloskop vorgenommen. Hierzu bieten die Hersteller vordefinierte Testabläufe an, mit denen man diese Messungen einfach durchführen kann. Im vorliegenden Fall zeigte die Compliance Messung einen Fehler bei der Bestimmung des „Unit Interval“. Bei 2,5 Gb/s ist die Übertragungsdauer eines einzelnen Bits 400 ps. Die Spezifikation legt fest, dass eine maximale Abweichung von ± 300 ppm erlaubt ist. Dies entspricht einer Abweichung von 120 fs vom idealen UI.

Hermann Ruckerbauer, hermann.ruckerbauer@eyeknowhow.de, EKH-EyeKnowHow, Veilchenstraße 1, 94554 Moos.

II. Detaillierte Untersuchungen

Ausgehend von der Fehlermeldung der Compliance Tests (Tabelle 1) muss man detailliertere Analysen vornehmen, um auf den Fehlermechanismus schließen zu können.

A. Analyse der Augendiagramme

Ein wichtiges Werkzeug zur Bewertung der Qualität eines Signals ist das Augendiagramm. Dieses wird gebildet, indem man die einzelnen Unit Intervalls eines Signals übereinander legt. Solch ein Augendiagramm kann man mit verschiedenen Rahmenbedingungen generieren:

- Transition Eye
- Non-Transition Eye
- Bandbreite der PLL der CDR (Clock Data Recovery)

Der Vergleich Transition Eye vs. Non-Transition Eye erlaubt die Bewertung der Pre-Emphasis des Senders. In PCIe (und allen anderen schnellen Schnittstellen) wird diese Technik verwendet, um das Low-Pass Verhalten einer Leitung auf einer Leiterplatte zu kompensieren. Dazu wird ein erstes Bit nach einer Schaltflanke mit normaler Treiberleistung getrieben. Für weitere identische Bits wird die Treiberstärke abgesenkt. Sind die Einstellungen im Sender falsch gewählt, dann zeigt sich dies am unterschiedlichen Aussehen der Datenaugen bei Transition und Non-Transition bits. In diesem Fall war der Jitter im Auge für beide Diagramme identisch. Damit konnte eine falsche Pre-Emphasis-Einstellung des Treibers ausgeschlossen werden (Abbildung 1).

Ein Signal bei diesen Datenraten weist Jitter in verschiedenen Frequenzbereichen auf: Niederfrequente Anteile, die sich bei Messungen über längere Zeiträume im Mikrosekundenbereich auswirken und hochfrequenten Jitter, der jede einzelne Flanke verschieben kann. Um Signal über längere Zeit stabil zu erkennen,

Tabelle 1: „Fail“-Ergebnis eines PCIe 1.1 Compliance Tests

Margin Thresholds	
Warning	< 2 %
Critical	< 0 %

Pass	# Failed	# Trials	Test Name	Actual Value	Margin	Spec Range
✗	1	1	System Board Tx, Unit Interval (PCIE 1.1)	400.1650ps	-18.8 %	399.8800ps <= VALUE <= 400.1200ps
✓	0	1	System Board Tx, Template Tests (PCIE 1.1)	0.000	50.0 %	Zero Mask Failures
✓	0	1	System Board Tx, Median to Max Jitter (PCIE 1.1)	66.75ps	13.3 %	VALUE <= 77.00ps
✓	0	1	System Board Tx, Eye-Width (PCIE 1.1)	266.28ps	8.2 %	VALUE >= 246.00ps
✓	0	1	System Board Tx, Peak Differential Output Voltage (Transition)(PCIE 1.1)	464.5mV	20.6 %	274.0mV <= VALUE <= 1.2000V
✓	0	1	System Board Tx, Peak Differential Output Voltage (NonTransition)(PCIE 1.1)	495.5mV	25.6 %	253.0mV <= VALUE <= 1.2000V

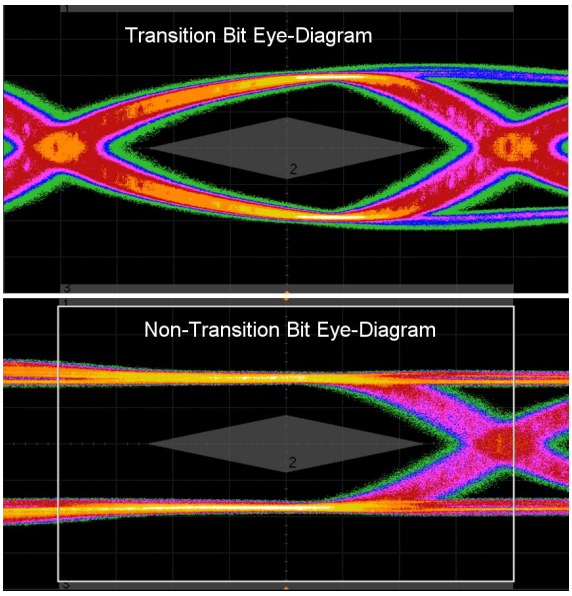


Abbildung 1: Augendiagramm-Vergleich zur Bewertung der Pre-Emphasis Einstellung im Controller

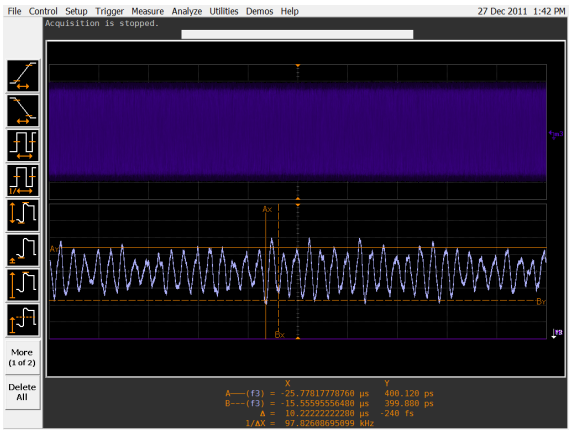


Abbildung 2: Time Intervall Error (TIE)-Messung

muss ein Empfänger dem niederfrequenten Jitter folgen, während er den hochfrequenten Jitter ausfiltern sollte. Diese Funktion wird in der physikalischen Schaltung in einer CDR (Clock-Data-Recovery) implementiert. Damit wird aus dem empfangenen Signal die zugehörige Phaseninformation (Clock) erzeugt. Innerhalb der CDR wird dies mittels einer PLL erreicht. Diese hat eine bestimmte Bandbreite, in der sie Jitter durchlässt. Für PCIe ist diese Bandbreite mit 1,5 MHz festgelegt. Unterhalb dieser Bandbreite sollte Jitter (wie z. B. Spread Spectrum Clocking SSC mit 30 kHz) durch die PLL durchgereicht werden. Damit wird erreicht, dass bei der Erzeugung des Augendiagramms die Grenzen der Bits leicht verschoben werden können, um dem Signal zu folgen. Auf dem Scope kann man dieses Verhalten einer PLL mit verschiedenen Parametern in die Messung einbeziehen. Erhöht man die Bandbreite der PLL, stellt man eine deutliche Verbesserung der Qualität des Datenauges fest. Dies deutet darauf hin, dass der Test aufgrund eines niederfrequenten Jitter-Effekts ausgefallen ist. Basierend auf dieser Erkenntnis kann man als nächste Messung genauere Jitter-Messungen festlegen.

B. Time Intervall Error (TIE) Messung

Eine interessante Variante der Jitter Messung ist die Time Intervall Error (TIE) Messung. Diese trägt die Abweichung der Flanken des gemessenen Signals von einem idealen Zeitpunkt über der Zeit auf.

Die TIE-Messung zeigte die deutliche Signatur einer Störung. Eine Modulation mit 100 kHz ist in Abbildung 2 deutlich erkennbar. Mit diesem Hinweis kann man die Signalwege zurückverfolgen und damit den Ursprung der Störung finden.

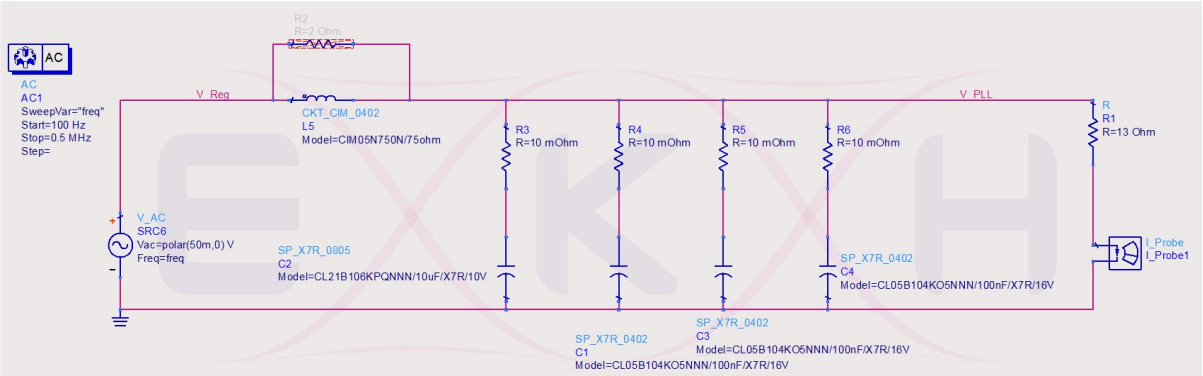


Abbildung 3: Schaltplan Power Filter Simulation mit Ferrite Bead und Widerstand

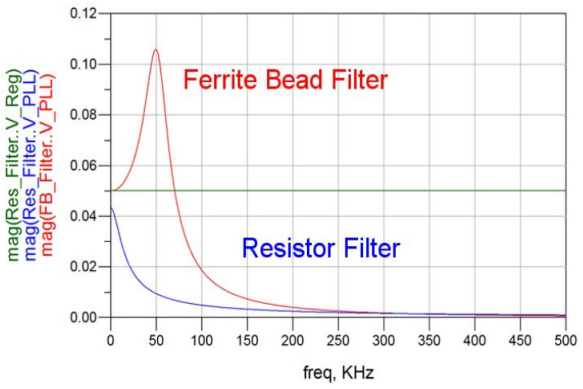


Abbildung 4: Power Filter Simulation mit Ferrite Bead und Widerstand

Weitere TIE Messungen zeigten die Signatur des Fehlers (Modulation des Jitters mit 100 kHz) am Transmit-Signal wie auch auf der 100 MHz-Referenz-Clock. Bereits die Input-Signale am Controller zeigten das Verhalten, was nahelegt, dass es bereits aus dem Clock Buffer, der den Controller speist, kommt. Der wahrscheinlichste Grund, dass der Clockbuffer eine schlechte Clock erzeugt, ist eine Störung der Versorgungsspannung am Baustein.

C. Power Noise-Messung

Um dies zu prüfen, wurden die verschiedenen Versorgungsspannungen des Clockbuffers überprüft. An einem Versorgungspin VDD_Ref konnten Spannungsschwankungen von 30 mV mit einer 100 kHz-Modulation nachgewiesen werden. Um zu zeigen, ob diese Schwankungen von der PLL erzeugt wurden oder schon auf der Versorgungsspannung lagen, wurde die Messung am Ausgang der Spannungserzeugung gemessen. Auch hier waren 100 kHz nachweisbar. Eine Prüfung des Datenblatts des Spannungsreglers zeigte eine Einstellung zur Verringerung der Leistungsaufnahme. Im DCM (Discontinous Current Mode) verringerte der Regler seine Schaltfrequenz abhängig von der Last. In diesem Falle lag diese Frequenz bei 100 kHz. Um solche Effekte zu vermeiden, ist in digitalen Schaltungen häufig ein Ferrite Bead zum Filtern der Versorgungsspannung vorgesehen, so auch in diesem Falle. Dieser erzeugt mit den dahinter verwendeten Kapazitäten einen LC-Filter. Solche Schaltungen werden oft von einem Design ins nächste übernommen ohne sie genau zu charakterisieren oder anzupassen. Eine kurze AC-Simulation mit einer Spannungsquelle und einer 50 mV sinusförmigen Störung zeigte, dass diese Schaltung im Bereich bis 100 kHz die eingangsseitige Störung sogar verstärkte (Abbildungen 3 und 4).

D. Optimierungsmöglichkeiten

Um die Versorgung der PLL zu verbessern, gibt es verschiedene Möglichkeiten:

- Glättung der Ausgangsspannung durch Anpassung der Ausgangs-LC-Kombination
- Optimierung der Versorgungsspannungsfilterung an der PLL
- Abschalten der stromsparenden DCM-Funktion des Spannungsreglers

In separaten Versuchen wurde bei jeder dieser Maßnahmen gezeigt, dass das Verhalten des Systems deutlich verbessert werden konnte. Bei der Optimierung des Filters der PLL-Versorgung muss beachtet werden, dass bei einem reinen RC Filter ein DC-Spannungsabfall erfolgt. Deshalb muss man sicherstellen, dass die PLL mit der verringerten Spannung funktioniert. Ebenso muss man sicherstellen, dass der Widerstand für die entsprechende Leistung ausgelegt ist und auch aus Zuverlässigkeitsgründen nicht heiß wird.

Bei der Glättung der Ausgangsspannung aus dem Spannungsregler muss man die komplette Beschaltung neu dimensionieren. Einfach nur Kapazitäten zu addieren, stört den Regelkreis der Schaltung.

Das Abschalten der stromsparenden DCM Funktion geschieht auf Kosten des Leistungsverbrauches im Leerlauf.

Tabelle 2: Compliance Test-Ergebnis nach der Optimierung

Margin Thresholds	
Warning	< 2 %
Critical	< 0 %

Pass	# Failed	# Trials	Test Name	Actual Value	Margin	Spec Range
✓	0	1	System Board Tx, Unit Interval (PCIE 1.1)	400.0750ps	18.8 %	399.8800ps <= VALUE <= 400.1200ps
✓	0	1	System Board Tx, Template Tests (PCIE 1.1)	0.000	50.0 %	Zero Mask Failures
✓	0	1	System Board Tx, Median to Max Jitter (PCIE 1.1)	48.97ps	36.4 %	VALUE <= 77.00ps
✓	0	1	System Board Tx, Eye-Width (PCIE 1.1)	294.87ps	19.9 %	VALUE >= 246.00ps
✓	0	1	System Board Tx, Peak Differential Output Voltage (Transition)(PCIE 1.1)	505.8mV	25.0 %	274.0mV <= VALUE <= 1.2000V
✓	0	1	System Board Tx, Peak Differential Output Voltage (NonTransition)(PCIE 1.1)	511.0mV	27.2 %	253.0mV <= VALUE <= 1.2000V

Das System bestand die Tests und wurde von -19 % auf +19 % Abstand zur Spezifikation für die UI-Messung verbessert. Auch Augenweite und Jitter konnten deutlich verbessert werden (Tabelle 2).

III. Fazit

Durch die genaue Analyse konnte die Ursache des Fehlers lokalisiert, bestimmt und beseitigt werden. Notwendig war dazu die genaue Kenntnis von verschiedensten Dingen:

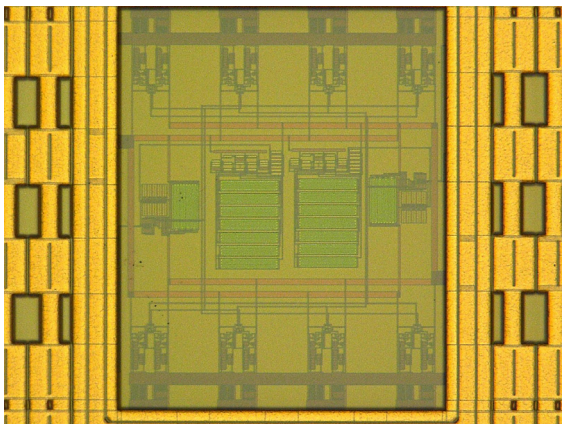
- Spezifikation des PCIe-Busses
- Benutzung eines Oszilloskops für High Speed-Messungen
- Mess- und Analysefunktionen des Oszilloskops
- Funktionalität einer PLL (und CDR)
- Design von Spannungsreglern
- Funktion von Filtern einer Versorgungsspannung
- Analogsimulation
- Digital System Design

Diese Liste zeigt, dass speziell die Fehleranalyse ein breites Wissen voraussetzt und vor allem auch die Bereitschaft, sich in neue Themen einzuarbeiten und zu lernen. Selbst bei sehr gutem Basiswissen ist es aufgrund der Komplexität heutiger Designs nicht möglich, alle Fachgebiete abzudecken.

Hermann Ruckerbauer erhielt den akademischen Grad des Dipl.-Ing. (FH) in Mikrosystemtechnik im Jahr 1994 von der Fachhochschule Regensburg. Danach arbeitete er bei Siemens/Infineon/Qimonda in der DRAM-Entwicklung. Er hält mehr als 50 Patente. Seit 2009 ist er als freiberuflicher Berater in allen Fragen zu Signal/Power Integrity und Speicherschnittstellen tätig.

hochschule mannheim

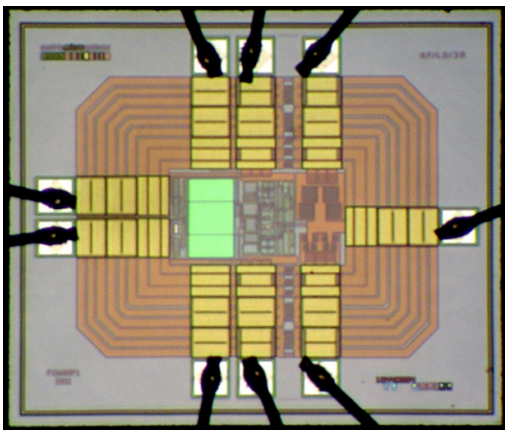
High-speed Multiplexer-/Demultiplexer-IC



ENTWURF	Christoph Rahnke, Josua Arndt, Bernd Vettermann, Jürgen Giehl Hochschule Mannheim, Institut für Entwurf integrierter Schaltkreise, Paul-Wittsack-Straße 10, 68163 Mannheim
ENTWURFSVERFAHREN	Full Custom Design
TECHNOLOGIE	AMS S35D4M5 0,35 µm CMOS 4M/2P
CHIPFLÄCHE	1,4 mm x 1,3 mm
GEHÄUSE	SOIC16
FUNKTIONSBLÖCKE	Komparatoren, Peaking Current Source, Logikblöcke, Buffer, Schalter
FUNKTION	Der Chip enthält einen Double Balanced Mischer (jeweils 1:4) mit gemeinsamer Ansteuerlogik und kann sowohl als MUX (Sender) als auch als DEMUX (Empfänger) betrieben werden. Das Eingangssignal wird periodisch (jeweils 1/4 Periode) auf die 4 Ausgänge gelegt (für DEMUX Betrieb, bei MUX umgekehrt). Die minimale MUX-Periode beträgt 8 ns für das UKW-Band (DAB). Der Durchgangswiderstand der Schalter beträgt weniger als 10 Ohm. Die Ansteuerung erfolgt durch 2 differentielle 90° phasenverschobene Cosinusschwingungen (IQ) der MUX-Frequenz, die durch eine externe DDS erzeugt werden. Die internen Rechtecksignale werden über Komparatoren daraus erzeugt. Die Referenzströme werden durch eine Peaking Current Source erzeugt.
HERSTELLDATUM	II. Quartal 2011
KOSTENTRÄGER	MPC-Gruppe Baden-Württemberg
VERÖFFENTLICHUNG	C. Rahnke, J. Giehl, B. Vettermann, „Entwurf eines High-Speed Multiplexers/Demultiplexers für einen Mischer in 0,35µm Technologie“, *Workshop der Multiprojekt-Chip-Gruppe Baden-Württemberg*, Karlsruhe, Tagungsband ISSN 1862-7102, S. 29 – 34, Ausgabe 42, Juli 2009.

Hochschule Ulm

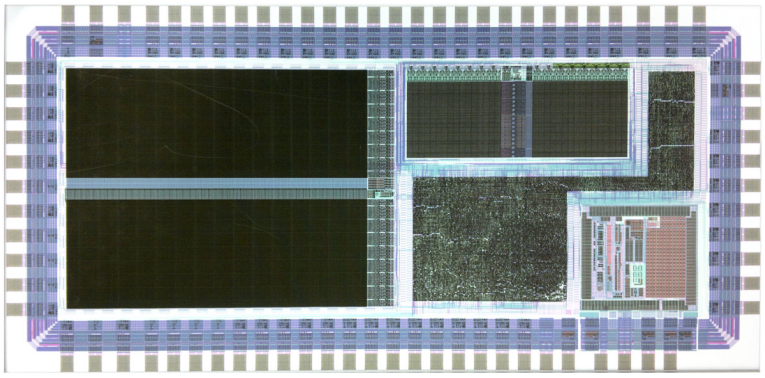
CMOS-Leistungsverstärker für niedrige Versorgungsspannungen



ENTWURF	Lukas Schumm, Gerhard Forster Hochschule Ulm, Institut für Kommunikationstechnik, Prittwitzstraße 10, 89075 Ulm
ENTWURFSVERFAHREN	Full Custom Design
TECHNOLOGIE	AMS C35B4C3 0,35 µm CMOS 4M/2P/HR
CHIPFLÄCHE	1,19 mm x 1,00 mm
GEHÄUSE	DIL 16
FUNKTIONSBLÖCKE	Rail-to-Rail-Operationsverstärker mit Versorgungseinheit
FUNKTION	Der Chip enthält den IP-Core eines CMOS-Operationsverstärkers, der sowohl am Ausgang als auch am Eingang bis an die Versorgungsspannung aussteuerbar ist. Bei der Versorgungsspannung 3,3 V erreicht die Endstufe eine Ausgangsspannung von 3,0 Vpp an 50 Ohm. Die Schleifenverstärkung beträgt 82 dB, die Transitfrequenz 10 MHz und die Offsetspannung liegt unter 2 mV.
HERSTELLDATUM	IV. Quartal 2011
KOSTENTRÄGER	MPC-Gruppe Baden-Württemberg
VERÖFFENTLICHUNG	L. Schumm, G. Forster, „Ein CMOS-Verstärker für niedrige Versorgungsspannungen“, *Workshop der Multiprojekt-Chip-Gruppe Baden-Württemberg*, Furtwangen, Tagungsband ISSN 1868-9221, S. 7 – 15, Ausgabe 46, Juli 2011.

Hochschule Offenburg
University of Applied Sciences

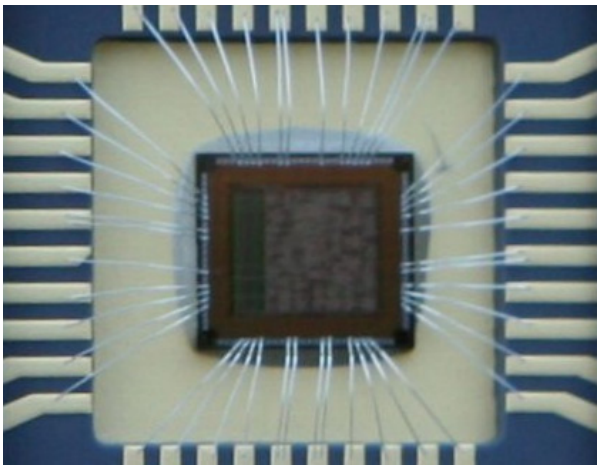
SIRIUS-JANUS 16/32-bit Prozessorkern für einen PDA mit JTAG - Schnittstelle



ENTWURF	Benjamin Dusch, Sebastian Stickel, Andreas Kreker, Dirk Jansen Hochschule Offenburg, Institut für Angewandte Forschung, Badstraße 24, 77652 Offenburg
ENTWURFSVERFAHREN	Full Custom Design
TECHNOLOGIE	UMC L180 1P6M MM/RFCMOS
CHIPFLÄCHE	3,16 mm x 1,52 mm
GEHÄUSE	QFN80
FUNKTIONSBLÖCKE	Prozessorkern SIRIUS-JANUS, 32 kByte RAM (IP von UMC), 36 kByte ROM (IP von UMC), Displaysteuerung, Steuerung für einen extern anzuschließenden RAM, AHI Audioeinheit, SPI Controller, drei Timer, PLL (IP von UMC) mit Steuerlogik, Boundary Scan Testlogik (JTAG) mit integrierter Scan Chain.
FUNKTION	Der Chip enthält den Prozessorkern SIRIUS-JANUS mit Peripherie sowie ein 32 kByte statisches RAM und ein 36 kByte statisches ROM. Darüber hinaus verfügt der Chip über ein Interface und Wait State Generatoren, um ein OLED-Display mit Touchscreen direkt anzusprechen. Im ROM sind mehrere Softwareroutinen einschließlich eines Open FAT Dateisystems zur Ansteuerung einer SD-Karte enthalten. Mit Synopsys Synthesetechniken wurde der Chip auf maximale Performance von 241 MIPS (berechnet auf Basis des Layouts) bei bestmöglicher Flächennutzung optimiert. Gedacht für den Einsatz in einem PDA, der an der HS Offenburg entwickelt wird, ist der Chip aber auch als Prozessor in zahlreichen anderen Applikationen anwendbar. Der ASIC ist ein Redesign mit weiteren Elementen des schon im Jahre 2009 hergestellten und erfolgreich erprobten ersten PDAs.
HERSTELLDATUM	I. Quartal 2012
KOSTENTRÄGER	MPC-Gruppe Baden-Württemberg
VERÖFFENTLICHUNG	B. Dusch, S. Stickel, A. Kreker, D. Jansen, „Entwicklung eines 16/32-bit Prozessorkerns für einen PDA mit JTAG-Schnittstelle und Implementierung in einer 0,18µm-CMOS Technologie“, *Workshop der Multiprojekt-Chip-Gruppe Baden-Württemberg*, Furtwangen, Tagungsband ISSN 1868-9221, S. 25 – 34, Ausgabe 46, Juli 2011.

Hochschule Ravensburg-Weingarten

Frequenzzähler x05t



ENTWURF	Erstes Semester, Masterstudiengang Electrical Engineering (MEE), WS 10/11 Betreuer: Walter Ludescher, Andreas Siggelkow, Christoph Weber Hochschule Ravensburg-Weingarten, Doggenriedstraße, 88250 Weingarten
ENTWURFSVERFAHREN	Gate Array Design
TECHNOLOGIE	IMS 0,5 µm CMOS, Institut für Mikroelektronik Stuttgart
CHIPFLÄCHE	3,4 mm x 3,1 mm
GEHÄUSE	DIL 40
FUNKTIONSBLÖCKE	Zeitbasis für Referenzsignal 10 MHz, 10 dekadische Zählstufen, Prozessorschnittstelle in zwei Varianten, I/O-Port-Expander. Komplexität circa 2000 CMOS-Zellen.
FUNKTION	Der Chip dient zur Messung von Frequenz, Periodendauer und Phase an 3 Eingangssignalen mit Bandbreiten von DC bis 300 MHz. Er stellt eine flexible Prozessorschnittstelle zum Ein- und Auslesen der Messdaten bereit und enthält einen Port-Expander zum Anschluss eines externen LCD-Displays, zur Steuerung externer HF-Koaxrelais, HF-Dämpfungsglieder und Multiplexer.
HERSTELLDATUM	I. Quartal 2011
KOSTENTRÄGER	MPC-Gruppe Baden-Württemberg
VERÖFFENTLICHUNG	

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@hs-offenburg.de

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

Hochschule Ravensburg-Weingarten
Prof. Dr. Siggelkow, (0751) 501-9633
siggelkow@hs-weingarten.de

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

Hochschule Ulm
Prof. Dipl.-Phys. Forster, (0731) 50-28338
forster@hs-ulm.de

www.mpc.belwue.de