Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Masterthesis

Michael Gäth

FPGA-Implementierung eines Blind-Source-Separation-Algorithmus mittels Stochastic Computing

Fakultät Technik und Informatik
Department Informations- und Elektrotechnik

Faculty of Engineering and Computer Science
Department of Information and Electrical Engineering

Michael Gäth

FPGA-Implementierung eines Blind-Source-Separation-Algorithmus mittels Stochastic Computing

Masterthesis eingereicht im Rahmen der Masterprüfung
im Masterstudiengang Informations- und Kommunikationstechnik
am Department Informations- und Elektrotechnik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr.-Ing. Karl-Ragmar Riemschneider
Zweitgutachter : Prof. Dr.-Ing. Jürgen Vollmer

Abgegeben am 16. Oktober 2014

Michael Gäth

Thema der Masterthesis

FPGA-Implementierung eines Blind-Source-Separation-Algorithmus mittels Stochastic Computing

Stichworte

Blind Source Separation, blinde Quellentrennung, Independent Component Analysis, Unabhängigkeitsanalyse, Hérault-Jutten-Algorithmus, Stochastic Computing, stochastische Rechentechnik, stochastischer Bitstrom, FPGA

Kurzzusammenfassung

In dieser Arbeit wird die Eignung der stochastischen Rechentechnik für die FPGA-Implementierung eines Algorithmus zur blinden Quellentrennung diskutiert. Dazu werden zunächst die Grundlagen betrachtet und anschließend der Hérault-Jutten-Algorithmus mittels stochastischer Rechentechnik simuliert und auf einem Spartan-6-FPGA implementiert. Die Implementierung wird getestet, die Ergebnisse werden an einen PC übertragen und bewertet.

Michael Gäth

Title of the paper

FPGA Implementation of a Blind Source Separation Algorithm Using Stochastic Computing

Keywords

Blind Source Separation, Independent Component Analysis, Hérault-Jutten Algorithm, Stochastic Computing, Stochastic Bitstream, FPGA

Abstract

In this thesis the applicability of Stochastic Computing for an FPGA implementation of a Blind Source Separation algorithm is discussed. First the basic principles are considered and then the Hérault-Jutten algorithm is simulated with Stochastic Computing elements and subsequently implemented on a Spartan-6-FPGA. The implementation is tested an the results are transmitted to and evaluated on a PC.

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

1 Einleitung

Ein klassisches Problem der Signalverarbeitung ist das Cocktailparty-Problem. Befinden sich viele Menschen in einem Raum und reden durcheinander, dann ist das menschliche Gehirn in der Lage, sich auf den eigenen Gesprächspartner zu konzentrieren. Alle anderen Geräuschquellen werden ausgeblendet. Angenommen, zur technischen Lösung des Problems sind in jenem Raum mehrere Mikrofone platziert, die diese Geräuschkulisse aufnehmen. Im vereinfachten Fall sind die aufgenommenen Signale Mischungen der verschiedenen Schallquellen. Das technische Verfahren zur Rekonstruktion der Quellen aus solchen vermischten Signalen wird als Blind Source Separation[1] (BSS) bezeichnet. Kenntnis über den konkreten Verlauf dieser Quellen und Informationen über die Gewichtungsfaktoren der Mischung sind dabei nicht vorhanden, die Trennung erfolgt blind. Durchgeführt wird die Separation basierend auf der Annahme, die Quellsignale seien statistisch unabhängig. Ziel bei der BSS ist es demzufolge, die unabhängigen Signalkomponenten aus den vermischten Signalen zu extrahieren. Im Idealfall sind diese mit den Quellsignalen identisch. Die Suche nach den Komponenten wird als Independent Component Analysis[2] (ICA) bezeichnet. Prinzipiell geht es bei der ICA darum, ein Entmischungssystem zu finden, dessen Ausgangssignale so statistisch unabhängig wie möglich sind. Die Unabhängigkeit wird allgemein durch eine Kostenfunktion beschrieben. Im Falle von ICA werden hierfür zumeist sogenannte Higher-Order Statistics gewählt, welche die Gaußähnlichkeit der Signale angeben. Statistisch unabhängige Signale sind nach dem Grenzwertsatz der Statistik nicht-gaußähnlich. Deshalb ist das Entmischsystem dann optimal eingestellt, wenn die Gaußähnlichkeit der Ausgangsignale minimiert ist. Die Optimierung des Entmischungssystems kann je nach Problemstellung in Echtzeit adaptiv oder nicht echtzeitfähig per Stapelverarbeitung durchgeführt werden [3].

Für eine echtzeitfähige multivariate Signalverarbeitung bietet sich die Implementierung auf einem FPGA (Field Programmable Gate Array) an. Im Gegensatz zu einem digitalen Signalprozessor (DSP) können Daten massiv parallel verarbeitet werden. Im Vergleich zu einer anwendungsspezifischen integrierten Schaltung (ASIC) ist die Implementierungszeit kurz und eine eventuelle Korrektur einfach und schnell durchführbar. In der Vergangenheit wurden mehrere ICA-basierte BSS-Algorithmen auf einem FPGA implementiert. In [4] wurde eine vierkanalige EEG-Signaltrennung (Elektroenzephalogramm) implementiert. Mit vier synthetischen Signalen wurde eine BSS-Implementierung in [5] erfolgreich getestet. Auf besonderes

[1] deutsch: blinde Quellentrennung [1]
[2] deutsch: Unabhängigkeitsanalyse [2]

Interesse stößt die Blind Source Separation beispielsweise im Bereich der Biomedizin, denn dort ist es weiterhin ein Problem, bei der multisensorischen Aufnahme der Gehirnströme (EEG) die Nutz- von den Störsignalen zu trennen [6, 7].

Ein ungewöhnlicher Ansatz in der Signalverarbeitung, der im Rahmen dieser Thesis betrachtet werden soll, ist die Informationsverarbeitung mit zufälligen Bitströmen. Dies ist eine Rechentechnik und wird Stochastic Computing[3] (SC) genannt. Sie ermöglicht die Durchführung arithmetischer Operationen mit einem im Vergleich zur konventionellen Festkomma-Arithmetik sehr viel geringeren Hardwareaufwand [9]. Die Grundlagen für SC wurden von von Neumann 1956 definiert, der damit die Erforschung der Rechentechnik Ende der 60er Jahre beeinflusste. Allerdings wurde zu dieser Zeit die geringe Bandbreite der Rechentechnik als Schwachpunkt identifiziert, und durch den stetigen Fortschritt bei den konventionellen binären Schaltungen verringerte sich das Interesse an SC [10]. In den letzten zwei Jahrzehnten wurde SC hauptsächlich im Kontext von Spezialanwendungen wie etwa neuronalen Netzen betrachtet.

Stochastic Computing wurde bisher selten mit BSS kombiniert. Fanghänel et al. implementierten in [11] den Hérault-Jutten-Algorithmus komplett mit SC-Elementen auf einem Chip mit 6 CPLDs (Complex Programmable Logic Device). Die Hardwarearchitektur des Chips war jedoch für neuronale Netze ausgelegt [12]. Das korrekte Konvergenzverhalten war bei ihrer Implementierung nicht gegeben. Den sogenannten Cichocki-Unbehauen-Algorithmus implementierten Hori et al. in [13] auf einem FPGA. Allerdings wurden wichtige Komponenten nicht mit SC-Komponenten realisiert. Es handelt sich dort um eine hybride Lösung. Eine dokumentierte Implementierung eines BSS-Algorithmus ausschließlich mit SC-Komponenten und gegebener Funktionalität wurde in der Literatur nicht gefunden.

1.1 Zielstellung

In dieser Thesis soll analysiert werden, inwiefern Stochastic Computing für eine echtzeitfähige Blind Source Separation geeignet ist. Ziel ist es, einen BSS-Algorithmus mit SC-Komponenten auf einem FPGA zu implementieren. Im Speziellen soll der Hérault-Jutten-Algorithmus, der historisch erste BSS-Algorithmus, betrachtet werden. Zielhardware ist der Spartan-6-FPGA der Firma Xilinx auf einem Nexys3-Board der Firma Digilent.

Im Mittelpunkt soll die Bewertung des Potenzials von Stochastic Computing stehen.

Im Rahmen der Bewertung sollen die Vor- und Nachteile im Vergleich zur Festkomma-Arithmetik untersucht werden. Besonderheiten der einzelnen Komponenten sind aufzuzeigen, Probleme sollen erläutert und eventuelle Lösungsmöglichkeiten sollen erarbeitet wer-

[3] deutsch: stochastische Rechentechnik [8]

den. Das Verhalten des Hérault-Jutten-Algorithmus mit und ohne Stochastic Computing soll verglichen und bewertet werden. Eine Spezialisierung auf ein konkretes Anwendungsszenario ist nicht Teil dieser Arbeit.

Die VHDL-Beschreibung der einzelnen SC-Komponenten soll die FPGA-Architektur effizient ausnutzen. Der Hardwareaufwand ist mit einer Festkomma-Implementierung zu vergleichen.

1.2 Gliederung

In Kapitel 2 werden die wichtigsten Grundlagen zur Blind Source Separation aufgeführt. Die Problemstellung sowie das Lösungsverfahren werden mathematisch modelliert. Der Hérault-Jutten-Algorithmus wird vorgestellt, getestet und mit weiteren Algorithmen verglichen.

Anschließend werden in Kapitel 3 die relevanten Grundlagen zu Stochastic Computing erläutert. Das Prinzip der Informationsverarbeitung wird erklärt. Die Komponenten werden vorgestellt und analysiert, verschiedene Codierungsverfahren werden gegenübergestellt. Zuletzt werden Vor- und Nachteile im Kontext der gängigen Festkomma-Arithmetik diskutiert.

In Kapitel 4 wird die Blind Source Separation mittels Stochastic Computing betrachtet. Simulationsergebnisse werden vorgestellt und evaluiert.

Nach der funktionalen Verifikation des Algorithmus wird in Kapitel 5 dessen FPGA-Implementierung dargestellt. Die Ergebnisse der Hardware-Synthese der SC-Komponenten werden vorgestellt und das implementierte Gesamtsystem beschrieben und verifiziert. Der Hardwareaufwand wird mit einer Festkomma-Implementierung verglichen.

Im letzten Kapitel 6 werden die Ergebnisse zusammengefasst und bewertet. Abschließend wird ein Ausblick auf mögliche weiterführende Arbeiten gegeben.

2 Blind Source Separation

In diesem Kapitel soll das Verfahren Blind Source Separation vorgestellt werden. Zu Beginn wird auf die Problemstellung eingegangen und das Lösungsverfahren erläutert. Anschließend wird der Hérault-Jutten-Algorithmus vorgestellt und mit anderen Algorithmen verglichen.

2.1 Problemstellung

Das Cocktailparty-Problem wurde bereits in der Einleitung angeführt. Zur Veranschaulichung soll die Problemstellung anhand dieses Beispiels modelliert werden.

Die Anzahl der Geräuschquellen sei auf zwei Sprachsignale begrenzt. Vermischte Signale werden durch zwei in unterschiedlichen Entfernungen aufgestellte Mikrofone aufgenommen. Das Problem wird in Abb. 2.1 illustriert.

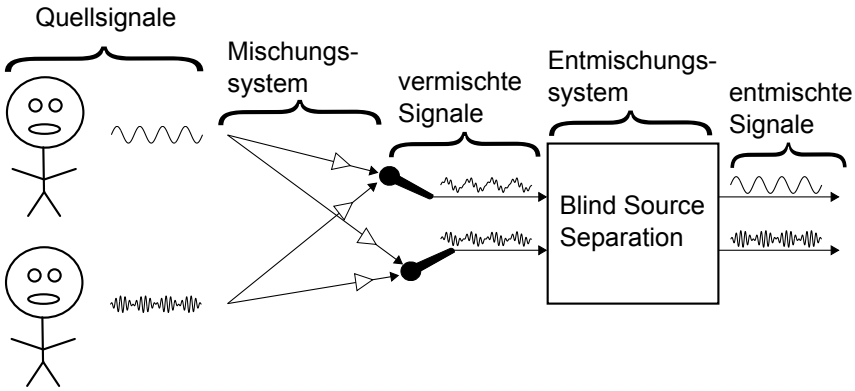

Abbildung 2.1: Schema des Cocktailparty-Problems

Der Übertragungskanal wird als ein Mischungssystem durch die Gewichtungsfaktoren a_{ij} charakterisiert. Frequenzabhängigkeit, Zeitverzögerung und -varianz sowie additives Rauschen werden vernachlässigt. Es handelt sich also um ein stark vereinfachtes Kanalmodell. Die Gewichtungsfaktoren sind abhängig von der Entfernung von Mikrofon und Quelle. Die

vermischten Signale $x_1[n]$ und $x_2[n]$ können durch folgendes lineares Gleichungssystem ausgedrückt werden:

$$x_1[n] = a_{11} \cdot s_1[n] + a_{12} \cdot s_2[n] \tag{2.1}$$
$$x_2[n] = a_{21} \cdot s_1[n] + a_{22} \cdot s_2[n] \tag{2.2}$$

Die Quellsignale sind hier $s_1[n]$ und $s_2[n]$, deren gewichtete Summen ergeben die Mischungssignale. Aus diesen müssen die Quellsignale extrahiert werden. Das geschieht durch ein Entmischungssystem, welches durch seine Entmischungskoeffizienten w_{ij} charakterisiert wird. Für die Ausgangssignale des Entmischungssystems gilt:

$$y_1[n] = w_{11} \cdot x_1[n] + w_{12} \cdot x_2[n] \tag{2.3}$$
$$y_2[n] = w_{21} \cdot x_1[n] + w_{22} \cdot x_2[n] \tag{2.4}$$

Die Entmischungskoeffizienten sind so einzustellen, dass im Idealfall $y_1[n] = s_1[n]$ und $y_2[n] = s_2[n]$ ist.

Wie die Ideallösung für die Entmischungskoeffizienten lautet, wird durch die Betrachtung des Mischungs- bzw. Entmischungsvorgangs in der Matrix-Vektorform deutlich.

Aus Gl. (2.1) und Gl. (2.2) ergibt sich mit der Mischungsmatrix $\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}$ sowie den Signalvektoren $\mathbf{s}[n] = \begin{pmatrix} s_1[n] & s_2[n] \end{pmatrix}^T$ und $\mathbf{x}[n] = \begin{pmatrix} x_1[n] & x_2[n] \end{pmatrix}^T$ für die Mischung

$$\mathbf{x}[n] = \mathbf{A} \cdot \mathbf{s}[n]. \tag{2.5}$$

Für den Entmischungsvorgang gilt mit der Entmischungsmatrix $\mathbf{W} = \begin{pmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{pmatrix}$ und dem Ausgangsvektor $\mathbf{y}[n] = \begin{pmatrix} y_1[n] & y_2[n] \end{pmatrix}^T$:

$$\mathbf{y}[n] = \mathbf{W} \cdot \mathbf{x}[n] \tag{2.6}$$

An dieser Stelle sei darauf hingewiesen, dass das Mischungsmodell nicht nur für den hier angeführten zweidimensionalen Fall gilt. Der allgemeine Fall mit N Quellsignalen und M Sensoren ist in Abb. 2.2 dargestellt [1].

Setzt man Gl. (2.5) in Gl. (2.6) ein,

$$\mathbf{y}[n] = \mathbf{W} \cdot \mathbf{A} \cdot \mathbf{s}[n], \tag{2.7}$$

so wird die Ideallösung der Entmischungsmatrix ersichtlich. Ist die Entmischungsmatrix gleich dem Kehrwert der Mischungsmatrix, dann sind die Ausgangssignale gleich den Quellsignalen.

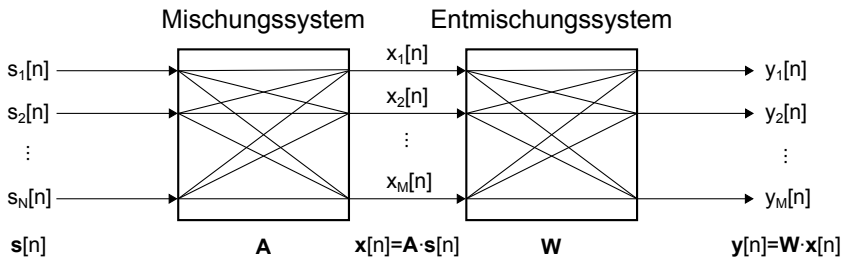

Abbildung 2.2: Allgemeines System zur Blind Source Separation mit N Quellsignalen und M Sensoren

Mathematisch ausgedrückt muss die Systemmatrix $\mathbf{C} = \mathbf{W}\cdot\mathbf{A}$ idealerweise die Einheitsmatrix $\mathbf{I}$ sein, was bei $\mathbf{W} = \mathbf{A}^{-1}$ der Fall ist. Dann ist $\mathbf{y}[n] = \mathbf{s}[n]$, die Quellen werden perfekt rekonstruiert.

Die Mischungsmatrix ist jedoch nicht bekannt. Deren Inverse muss geschätzt werden. Dies wird bei der Independent Component Analysis getan. Dort wird davon ausgegangen, dass die Quellsignale statistisch unabhängig sind [14].

2.2 Independent Component Analysis als Lösungsverfahren

Die Independent Component Analysis ist eine Methode, um eine Blind Source Separation durchzuführen. In diesem Abschnitt wird gezeigt, was als statistische Unabhängigkeit bezeichnet wird und wie diese gemessen werden kann. Ausgangspunkt ist erneut ein Beispiel mit zwei Quellsignalen $s_1[n]$ und $s_2[n]$. Diese sind statistisch unabhängig, wenn die Verbundwahrscheinlichkeit $p(s_1 \cap s_2)$ gleich dem Produkt der Einzelwahrscheinlichkeiten ist [3]:

$$p(s_1 \cap s_2) = p(s_1) \cdot p(s_2) \tag{2.8}$$

Das bedeutet, der Wert $s_1[n]$ hängt in keiner Weise vom Wert $s_2[n]$ ab und umgekehrt. Für N Quellsignale gilt entsprechend

$$p(s_1 \cap s_2 \cap \cdots \cap s_N) = \prod_{i=1}^{N} p(s_i). \tag{2.9}$$

In Abb. 2.3 links sind zwei Quellsignale in Abhängigkeit voneinander aufgetragen. Die Signale $s_1 \in (-0.5, 0.5)$ und $s_2 \in (-1, 1)$ sind dort zwei gleichverteilte Rauschprozesse. Es ist zu sehen, dass durch die Kenntnis eines Abtastwertes des einen Signals kein Rückschluss

auf den Abtastwert des anderen Signal gezogen werden kann. Gl. (2.8) ist also gültig, die Signale sind statistisch unabhängig.

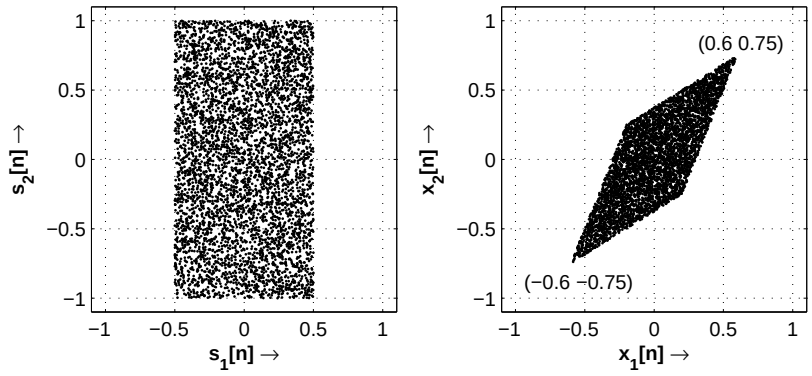

Abbildung 2.3: Veranschaulichung der statistischen Unabhängigkeit anhand einer Mischung; links Quellsignale, rechts vermischte Signale [3, 14]

In dem rechten Diagramm werden die vermischten Signale $x_1[n]$ und $x_2[n]$ gezeigt. Da sie aus den selben beiden Quellsignalen $s_1[n]$ und $s_2[n]$ bestehen, die lediglich durch die Mischungsmatrix, hier $\mathbf{A} = \begin{pmatrix} 0.8 & 0.2 \\ 0.5 & 0.5 \end{pmatrix}$, unterschiedlich gewichtet sind, sind die Signale nicht mehr unabhängig voneinander. Gl. (2.8) gilt hier nicht mehr. Wird beispielsweise $x_1[n] = 0.6$ gemessen, dann ist stets $x_2[n] = 0.75$.

Der Ansatz muss also sein, aus diesen Mischungssignalen Komponenten zu extrahieren, die jene Eigenschaft aufweisen, welche die Mischungssignale nicht aufweisen - die statistische Unabhängigkeit. Die Entmischungsmatrix **W** ist demzufolge so einzustellen, dass diese Unabhängigkeit maximiert wird.

Bevor gezeigt wird, wie die statistische Unabhängigkeit gemessen werden kann, wird hier kurz auf das Mehrdeutigkeitsproblem der ICA eingegangen. Abb. 2.4 zeigt nochmals die gleichverteilten Rauschsignale sowie drei mögliche Ergebnisse nach der Entmischung. Für alle Ergebnisse gilt die Definition aus Gl. (2.8). Im Diagramm oben rechts wird die ideale Lösung gezeigt, die Entmischungsmatrix ist gleich der inversen Mischungsmatrix, die Systemmatrix ist $\mathbf{C} = \mathbf{I}$, die Quellsignale werden perfekt rekonstruiert. Jedoch *kann* sowohl eine vorzeichenbehaftete Skalierung (unten links) auftreten als auch eine Permutation (unten rechts). Mit der Skalierungsmatrix **S** und der Permutationsmatrix **P** gilt folglich für eine erfolgreiche Rekonstruktion der Quellsignale:

$$\mathbf{C} = \mathbf{S} \cdot \mathbf{P} \cdot \mathbf{I} \tag{2.10}$$

Gl. (2.10) sagt aus, dass das Ziel bei der ICA erreicht ist, wenn die Entmischungsmatrix **W** so eingestellt ist, dass die Systemmatrix $\mathbf{C} = \mathbf{W} \cdot \mathbf{A}$ pro Zeile und Spalte nur einen Wert ungleich Null aufweist.

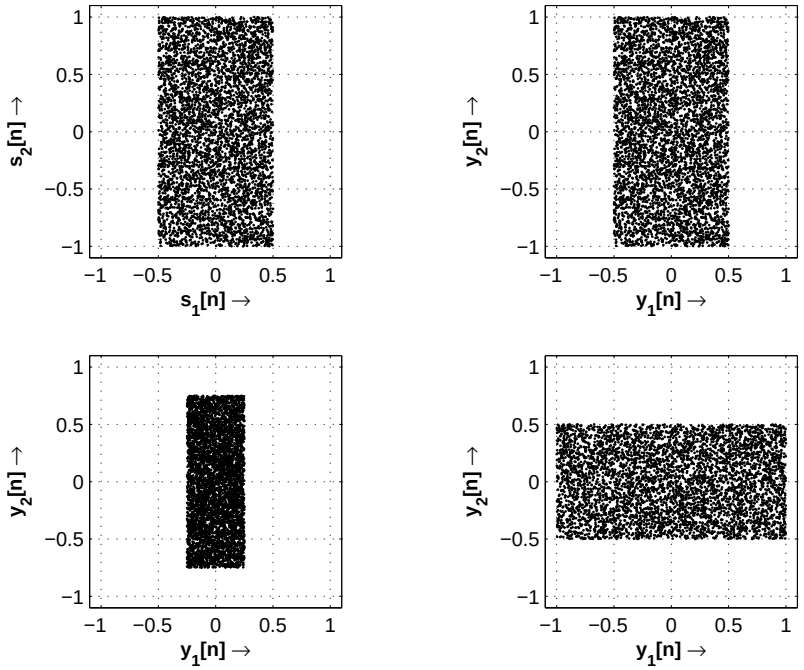

Abbildung 2.4: Veranschaulichung des Mehrdeutigkeitsproblems; Quellsignale (oben links), Ergebnis der Entmischung bei idealer Systemmatrix (oben rechts), skalierter Systemmatrix (unten links), permutierter Systemmatrix (unten rechts)

Nun stellt sich die Frage, wie diese statistische Unabhängigkeit gemessen werden kann.

Der zentrale Grenzwertsatz der Statistik besagt, dass die Wahrscheinlichkeitsdichtefunktion einer Summe von N nicht-gaußförmigen Zufallsvariablen für $N \to \infty$ gegen die Normalverteilung konvergiert [15]. Die Wahrscheinlichkeitsdichtefunktion der Summe von zwei oder mehr Quellsignalen ist demnach gaußähnlicher als die Dichtefunktionen der einzelnen Signale. Abb. 2.5 zeigt die Wahrscheinlichkeitsdichtefunktionen eines Prozesses x bestehend aus der Summe von N Zufallsvariablen. Grundsätzlich können Zufallsvariablen abhängig von ihrer Dichtefunktion in zwei Klassen unterteilt werden. Die Dichtefunktion von subgaußförmigen Prozessen ist flacher als die Normalverteilung (linkes Diagramm), die Dichtefunktion von supergaußförmigen Prozessen ist spitzer (rechtes Diagramm). Auf der Abbildung zu erkennen ist die Konvergenz zur Normalverteilung für steigendes N. Die *Nicht-Gaußähnlichkeit* ist also ein Maß für die statistische Unabhängigkeit eines Signals.

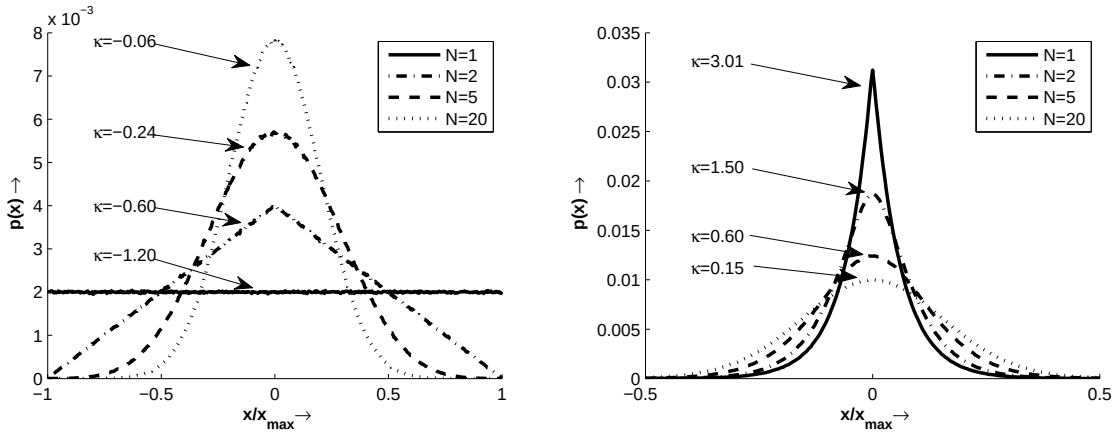

Abbildung 2.5: Verlauf von Wahrscheinlichkeitsdichtefunktionen für verschiedene Anzahl N an überlagerten Quellsignalen; links subgaußförmige, rechts supergaußförmige Quellsignale

Und diese Nicht-Gaußähnlichkeit kann gemessen werden. Das vierte zentrale statistische Moment[1], die Kurtosis κ, beschreibt die Abweichung der Dichtefunktion eines Zufallsprozesses von der Normalverteilung. Das unnormierte vierte Moment ist definiert als:

$$\hat{\kappa} = E\{(x - \mu_x)^4\} \tag{2.11}$$

Normiert auf die vierte Potenz der Standardabweichung σ_x ist das Moment aus Gl. (2.11) bei einem normalverteilten Prozess genau 3. Es gilt dann $\hat{\kappa}/\sigma_x^4 = 3$. Deshalb wird die Kurtosis auch definiert als

$$\kappa = \frac{E\{(x - \mu_x)^4\}}{\sigma_x^4} - 3 \tag{2.12}$$

Die Kurtosis nach Gl. (2.12) wurde in Abb. 2.5 eingetragen. Für subgaußförmige Prozesse ist sie negativ, für supergaußförmige Prozesse positiv. Mit steigendem N wird die Kurtosis betragsmäßig kleiner.

Folglich kann im Prinzip die statistische Unabhängigkeit der Ausgangssignale des Entmischungssystems in Form der Nicht-Gaußähnlichkeit durch die Messung der Kurtosis bestimmt und maximiert werden. Da die Kurtosis sehr sensibel gegenüber Änderungen der Wahrscheinlichkeitsdichtefunktion ist, werden in der Praxis andere Maße für die Unabhängigkeit genutzt [17]. Diese sind allgemein eng verwandt mit der Kurtosis, messen die Nicht-Gaußähnlichkeit jedoch robuster.

[1] in der angelsächsichen Literatur wird für Momente der Ordnung 3 und höher der Begriff Higher-Order Statistics verwendet [16], die bei der ICA eine wichtige Rolle spielen

2.3 Hérault-Jutten-Algorithmus

In dem vorangegangenen Abschnitt wurde erläutert, dass bei der ICA die statistische Unabhängigkeit als Kriterium zur Extrahierung der Quellsignale aus den vermischten Signalen genutzt werden kann. Das Entmischungssystem, charakterisiert durch die Entmischungsmatrix **W**, muss adaptiert werden, sodass die Ausgangsignale des Systems maximal unabhängig sind. Die Adaption, das Anlernen der Entmischungsmatrix, wird mittels Optimierungsalgorithmen durchgeführt. Das Schema ist in Abb. 2.6 dargestellt. Der Zeitindex n verdeutlicht, dass die Entmischungsmatrix iterativ angelernt wird.

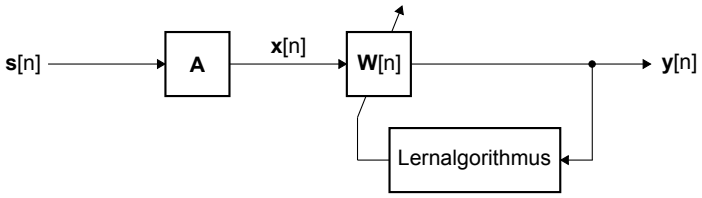

Abbildung 2.6: Schema der adaptiven Independent Component Analysis

Bei den adaptiven Optimierungsalgorithmen wird prinzipiell, ausgehend von einer Startmatrix $\mathbf{W}[0]$, für jeden Zeitpunkt n die Richtung berechnet, bei der die Nicht-Gaußähnlichkeit der Ausgangssignale $\mathbf{y}[n] = \mathbf{W}[n] \cdot \mathbf{x}[n]$ am stärksten wächst. Dazu wird ein Optimierungskriterium als Kostenfunktion verwendet und beispielsweise mit Hilfe des Gradientenverfahrens optimiert [3]. Verschiedene Algorithmen mit unterschiedlichen Optimierungskriterien wurden entwickelt. Der erste Algorithmus zur Lösung des BSS-Problems, der Hérault-Jutten-Algorithmus, wird im Folgenden vorgestellt, getestet und mit anderen Algorithmen verglichen.

2.3.1 Optimierungskriterium

Für die Lösung des BSS-Problems haben Hérault und Jutten das Konzept der Korrelation erweitert [18]. Sind zwei Prozesse x_1 und x_2 statistisch unabhängig, dann sind sie auch unkorreliert. Die Unabhängigkeit impliziert die Unkorreliertheit der Prozesse. Umgekehrt reicht die Schätzung der Korrelation jedoch nicht aus, um deren statistische Unabhängigkeit festzustellen. Die Unkorreliertheit impliziert nicht die Unabhängigkeit.

Die Prozesse sind unkorreliert, wenn die Kreuz-Kovarianz $C_{x_1x_2}$ gleich 0 ist:

$$C_{x_1x_2} = E\{(x_1 - \mu_{x_1}) \cdot (x_2 - \mu_{x_2})\} = 0 \tag{2.13}$$

Statistisch unabhängig sind sie nur, wenn zusätzlich gilt [19]:

$$E\{(x_1 - \mu_{x_1}) \cdot (x_2 - \mu_{x_2})\} = E\{x_1 - \mu_{x_1}\} \cdot E\{x_2 - \mu_{x_2}\} \tag{2.14}$$

Das Kriterium aus Gl. (2.13) wird verschärft, indem verlangt wird, dass die Kovarianz immer noch gleich 0 sein muss, nachdem mindestens einer der Prozesse nichtlinear transformiert wurde. Mit $y_1 = g_1(x_1)$ und $y_2 = g_2(x_2)$, wobei mindestens eine der beiden Funktionen **g** nichtlinear ist, muss für die nichtlineare Korrelation gelten:

$$C_{y_1 y_2} = E\{(y_1 - \mu_{y_1}) \cdot (y_2 - \mu_{y_2})\} = 0 \tag{2.15}$$

Sind die nichtlinear transformierten Prozesse unkorreliert, sind sie auch statistisch unabhängig. Die Nichtlinearitäten müssen ungerade sein [3]. Implizit werden bei der nichtlinearen Dekorrelation Higher-Order Statistics verwendet. Somit ist sie eng verbunden mit der Kurtosis.

2.3.2 Lernregel und Struktur

Um aus zwei vermischten Signalen die Quellsignale zu extrahieren, schlugen Hérault und Jutten die rückgekoppelte Struktur in Abb. 2.7 vor [3].

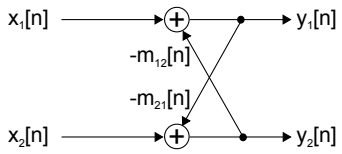

Abbildung 2.7: Schema der Rückkopplung des Hérault-Jutten-Algorithmus

Aus der Struktur ergibt sich für den allgemeinen Fall von N Quellsignalen und Sensoren:

$$\mathbf{y}[n] = \mathbf{x}[n] - \mathbf{M}[n] \cdot \mathbf{y}[n] \tag{2.16}$$

Hierbei sind $m_{ij}[n] = 0$ für $i = j$, die Elemente der Hauptdiagonalen der Matrix $\mathbf{M}[n]$ sind stets gleich 0. Aufgelöst nach dem Ausgangsvektor $\mathbf{y}[n]$ ergibt sich

$$\mathbf{y}[n] = (\mathbf{I} + \mathbf{M}[n])^{-1} \cdot \mathbf{x}[n]. \tag{2.17}$$

Da eine Matrixinvertierung numerisch sehr aufwendig sein kann, wurde für den Algorithmus folgende Approximation vorgeschlagen:

$$\mathbf{y}[n] = (\mathbf{I} - \mathbf{M}[n]) \cdot \mathbf{x}[n] \tag{2.18}$$

Analog zu der in den vorangegangenen Abschnitten vorgestellten Entmischungsmatrix ist hier eine Rekonstruktion der Quellsignale erfolgreich, wenn gilt (vgl. Gl. (2.10)):

$$\mathbf{C} = (\mathbf{I} - \mathbf{M}[n]) \cdot \mathbf{A} = \mathbf{S} \cdot \mathbf{P} \cdot \mathbf{I} \tag{2.19}$$

Zur Anpassung der Entmischungskoeffizienten $m_{ij}[n]$ wird der Momentanwert der nichtlinearen Korrelation nach Gl. (2.15) verwendet. Zu jedem Zeitpunkt n werden die Koeffizienten angepasst:

$$m_{ij}[n+1] = m_{ij}[n] + \Delta m_{ij}[n] \tag{2.20}$$

$\Delta m_{ij}[n]$ wird wie folgt berechnet:

$$\Delta m_{ij}[n] = \begin{cases} \mu g_1(y_i[n]) \cdot g_2(y_j[n]) & i \neq j \\ 0 & i = j \end{cases} \tag{2.21}$$

Im Mittel ist $\Delta m_{ij}[n]$ für nichtlinear dekorrelierte Ausgangssignale gleich 0.

Das Blockschaltbild des Hérault-Jutten-Algorithmus für $N = 2$ wird in Abb. 2.8 gezeigt.

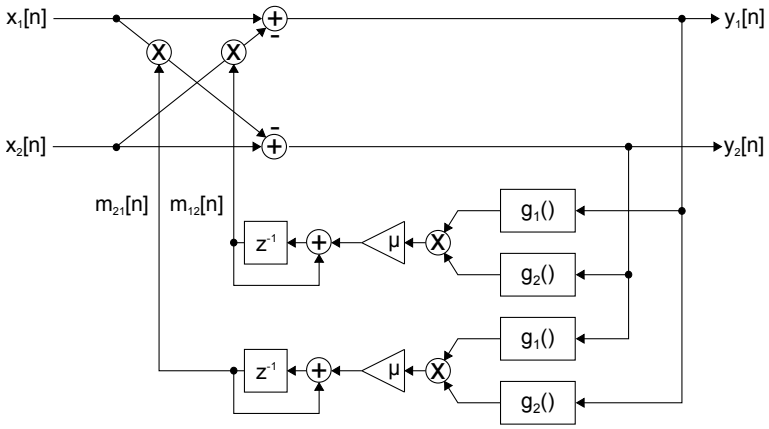

Abbildung 2.8: Blockschaltbild des Hérault-Jutten-Algorithmus für $N = 2$ Quellsignale

Die Wahl der nichtlinearen Funktionen wurde von Hérault und Jutten heuristisch durchgeführt. Sie empfahlen $g_1(y) = y^3$ und $g_2(y) = tanh(y)$ [11]. In anderen Quellen wurden die Funktionen $g_1(y) = y^2 sign(y)$ oder $g_1(y) = y$ und $g_2(y) = sign(y)$ empfohlen [20, 21, 3]. Mit Hilfe von Matlab-Simulationen wurde der Algorithmus mit verschiedenen Parametern getestet. Die Testcases und die optimalen Parameter sind im nächsten Abschnitt dokumentiert.

2.3.3 Test des Konvergenzverhaltens

Um die optimalen Parameter des Hérault-Jutten-Algorithmus zu bestimmen, wurde das Konvergenzverhalten für unterschiedliche nichtlineare Funktionen getestet.

Zur Bewertung des Konvergenzverhaltens wurde der oft verwendete Performance Index E_1 genutzt [3, 22]:

$$E_1[n] = \left[\sum_{i=1}^{N}\sum_{j=1}^{N}\left(\frac{|c_{ij}[n]|}{max_k|c_{ik}[n]|} + \frac{|c_{ji}[n]|}{max_k|c_{ki}[n]|}\right)\right] - 2N \tag{2.22}$$

Dieser gibt den Separationsfehler mit Hilfe der Elemente c_{ij} der Systemmatrix $\mathbf{C}[n] = \mathbf{W}[n]\mathbf{A}$ an. In Abschnitt 2.2 wurde im Kontext des Mehrdeutigkeitsproblems erwähnt, dass die Separation erfolgreich ist, wenn die Systemmatrix pro Zeile und Spalte nur ein Element ungleich 0 aufweist (vgl. auch Gl. (2.19)). Dies ist der Fall, wenn Gl. (2.22) gleich 0 ist. Je kleiner der Separationsfehler E_1 ist, desto besser wurden die Quellsignale rekonstruiert.

Die getesteten Funktionen sind in Tab. 2.1 zusammengestellt.

Abk.	$g_1(y)$	$g_2(y)$	Abk.	$g_1(y)$	$g_2(y)$	Abk.	$g_1(y)$	$g_2(y)$
y3s	y^3	$sgn(y)$	y2s	$y^2 sgn(y)$	$sgn(y)$	y1s	y	$sgn(y)$
y3t8	y^3	$tanh(8y)$	y2t8	$y^2 sgn(y)$	$tanh(8y)$	y1t8	y	$tanh(8y)$
y3t3	y^3	$tanh(3y)$	y2t3	$y^2 sgn(y)$	$tanh(3y)$	y1t3	y	$tanh(3y)$
y3t1	y^3	$tanh(y)$	y2t1	$y^2 sgn(y)$	$tanh(y)$	y1t1	y	$tanh(y)$

Tabelle 2.1: Getestete Kombinationen von (nichtlinearen) Funktionen für den Hérault-Jutten-Algorithmus (Reihenfolge für subgaußförmige Quellsignale)

Bei dem Hérault-Jutten-Algorithmus ist die Reihenfolge der beiden Funktionen abhängig von der Beschaffenheit der Quellsignale zu wählen. Bei anderen Algorithmen ist dies auch der Fall. Auch wenn nur eine nichtlineare Funktion verwendet wird, muss diese in deren Abhängigkeit gewählt werden. Die Beschaffenheit ist in die zwei Kategorien *subgaußförmig* und *supergaußförmig* zu unterteilen. Die Anordnung in der obigen Tabelle beispielsweise ist für subgaußförmige Signale gewählt, für supergaußförmige Signale sind g_1 und g_2 zu vertauschen. Deshalb wurden zum Test des Hérault-Jutten-Algorithmus und der weiteren Algorithmen zwei Testcases konzipiert. Getestet werden die Algorithmen separat für

1. subgaußförmige Quellsignale
2. supergaußförmige Quellsignale

Die Anzahl der Quellsignale ist stets gleich der Anzahl der Sensoren, eventuelle Störquellen und Verzögerungen werden nicht berücksichtigt. Als subgaußförmige Signale wurden gleichverteilte Rauschprozesse mit der Wahrscheinlichkeitsdichte $p(s) = \begin{cases} 0.5, & s \in (-1,1) \\ 0, & \text{sonst} \end{cases}$ gewählt. Als supergaußförmige Signale wurden Laplace-verteilte Zufallsprozesse mit der Dichtefunktion $p(s) = \frac{1}{\sqrt{2}\sigma_s} e^{-\frac{\sqrt{2}s}{\sigma_s}}$ verwendet (mit $\sigma_s = 0.15$).

Die Mischungsmatrix **A** besteht bei den Tests aus zufälligen Koeffizienten a_{ij}, für die folgende Restriktion gilt:

$$\sum_{j=1}^{N} |a_{ij}| = 1 \tag{2.23}$$

Dadurch wird sichergestellt, dass die vermischten Signale den Wertebereich von ± 1 nicht verlassen, was für den späteren Vergleich mit der stochastischen Rechentechnik wichtig ist. Desweiteren sind die vermischten Signale dadurch maximal ausgesteuert, weshalb die Anforderungen an die Algorithmen bei diesen Testszenarios so gering wie möglich sind.

Abb. 2.9 zeigt die Simulationsergebnisse für zwei sub- bzw. supergaußförmige Signale. Bei keiner Kombination ist ein deutlich bestes Konvergenzverhalten zu erkennen.

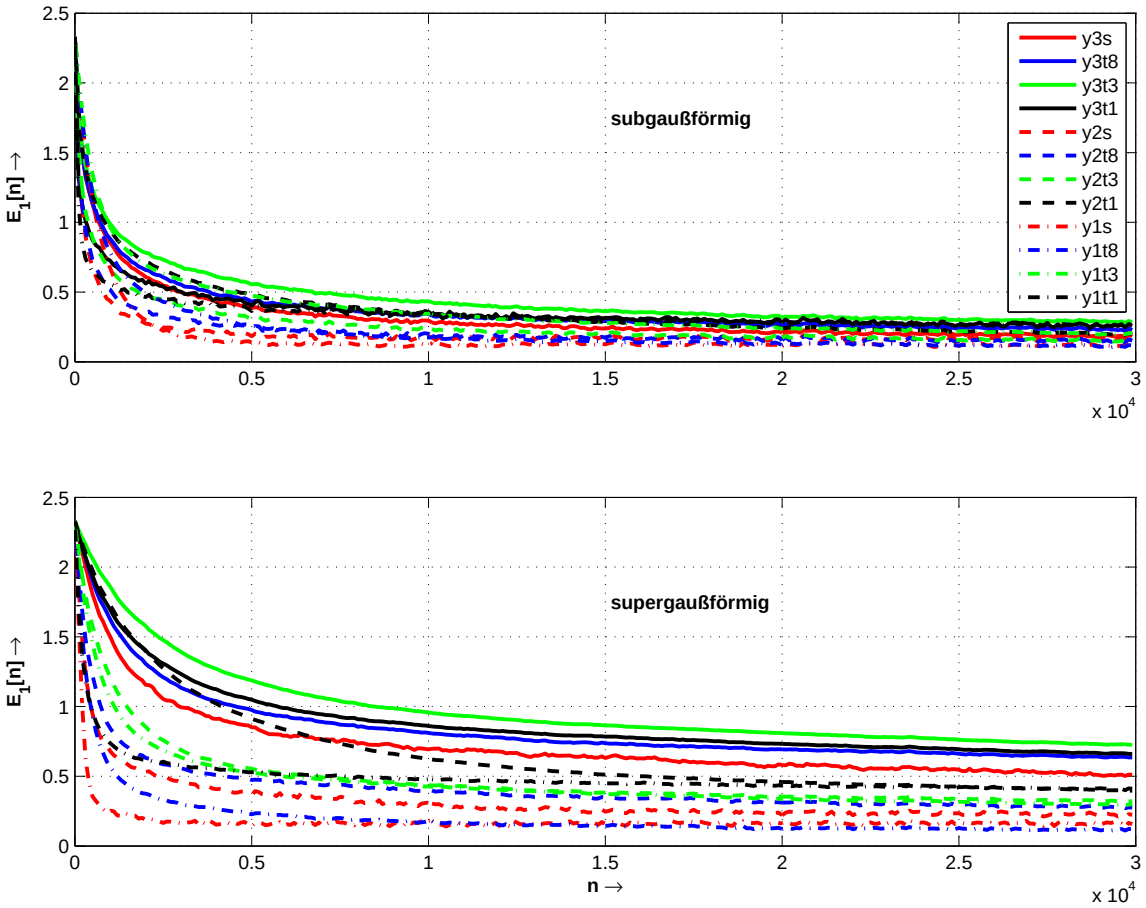

Abbildung 2.9: Test des Hérault-Jutten-Algorithmus mit verschiedenen nichtlinearen Funktionen für $N = 2$ Quellsignale

Die Wahl der Kombination muss demzufolge mit Blick auf die Implementierung mit einer begrenzten Auflösung getroffen werden. Die kubische Funktion sowie die Funktion $y^2 sgn(y)$

sind in solchen Fällen weniger geeignet. Deshalb wird als guter Kompromiss zwischen Separationsfehler und Konvergenzgeschwindigkeit die Kombination $y1s$ gewählt. Denn aus den obigen Abbildungen wird nicht ersichtlich, dass die Stabilität nicht bei allen Kombinationen gegeben ist. Bei $y3t1$ beispielweise konvergiert der Algorithmus zumeist nicht. Bei $y1s$ hingegen konvergierte er in allen 100 Durchläufen. Bei mehr als zwei Quellsignalen ist jedoch auch mit $y1s$ keine Konvergenz mehr garantiert. Dies soll im nächsten Abschnitt betrachtet werden.

2.3.4 Experimenteller Vergleich mit weiteren Algorithmen

Um die Leistungsfähigkeit des Hérault-Jutten-Algorithmus bewerten zu können, soll in diesem Abschnitt ein Vergleich mit zwei weiteren ICA-basierten BSS-Algorithmen durchgeführt werden.

Verglichen wird der Hérault-Jutten-Algorithmus mit dem Cichocki-Unbehauen-Algorithmus [20] und dem EASI-Algorithmus (Equivariant Adaptive Separation via Independence) [23].

Getestet wurde auch hier das Konvergenzverhalten bei sub- und supergaußförmigen Quellsignalen. Die Ergebnisse für zwei Quellsignale sind in Abb. 2.10 zu sehen. Die Separationsfehler der drei Algorithmen unterscheiden sich im Mittel unwesentlich. Auch die Konvergenzgeschwindigkeit ist nahezu gleich.

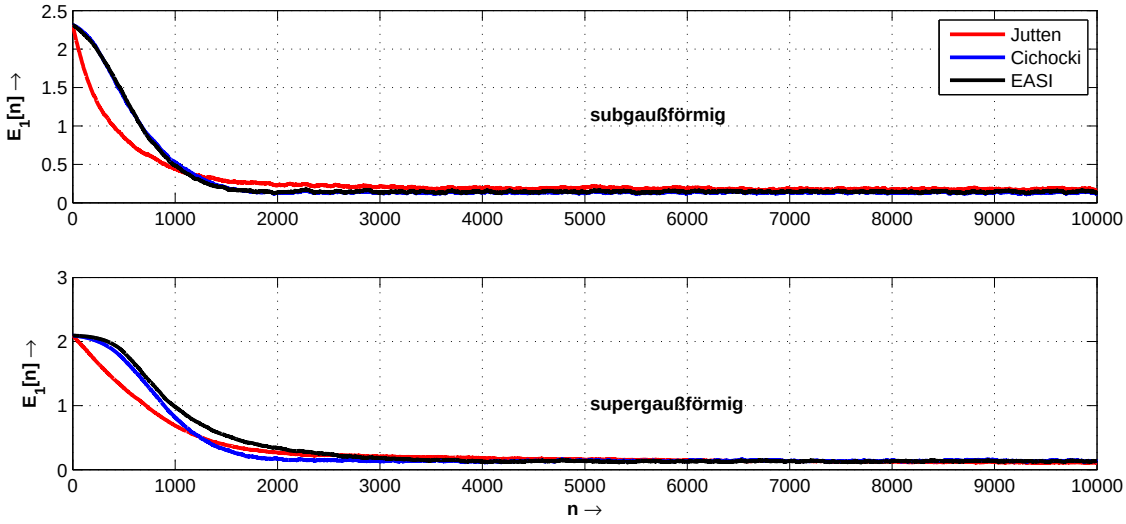

Abbildung 2.10: Vergleich der drei BSS-Algorithmen für $N = 2$ Quellsignale

Bei drei Quellsignalen zeigt sich jedoch die Schwäche des Hérault-Jutten-Algorithmus (vgl. Abb. 2.11). Nur für zwei Quellsignale konvergiert dieser Algorithmus stabil [3]. Die anderen

beiden Algorithmen hingegen konvergieren auch bei drei Quellsignalen stets. 1000 Durchläufe wurden simuliert, die Rauschprozesse und Mischungsmatrix bei jedem Durchlauf variiert.

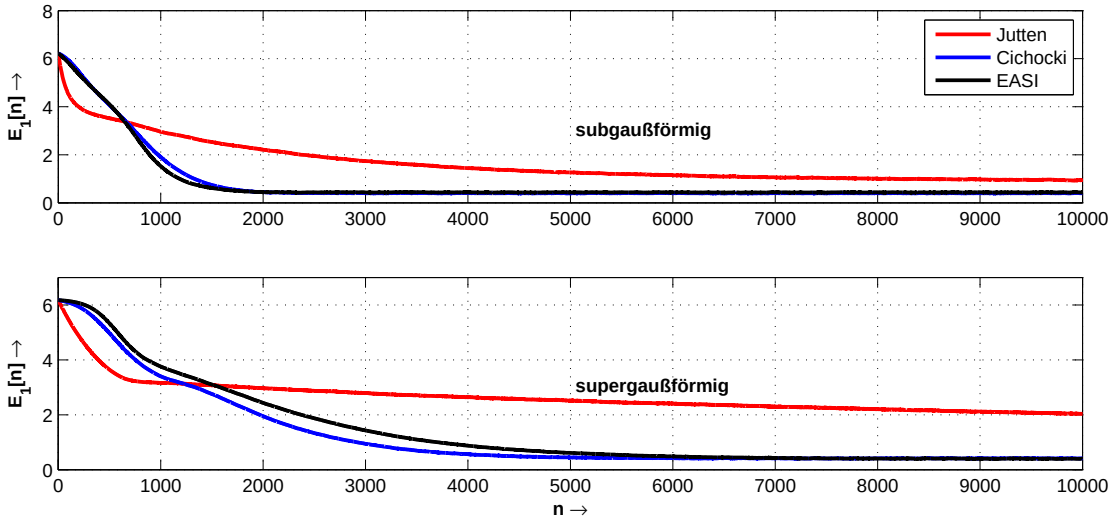

Abbildung 2.11: Vergleich der drei BSS-Algorithmen für $N = 3$ Quellsignale

Abb. 2.12 zeigt die 10 Durchläufe mit dem höchsten Separationsfehler des Hérault-Jutten-Algorithmus bei supergaußförmigen Quellsignalen. Auch abgebildet sind die entsprechenden Verläufe des Cichocki-Unbehauen- und des EASI-Algorithmus. Es ist zu erkennen, dass der Hérault-Jutten-Algorithmus in den abgebildeten Fällen nicht konvergiert.

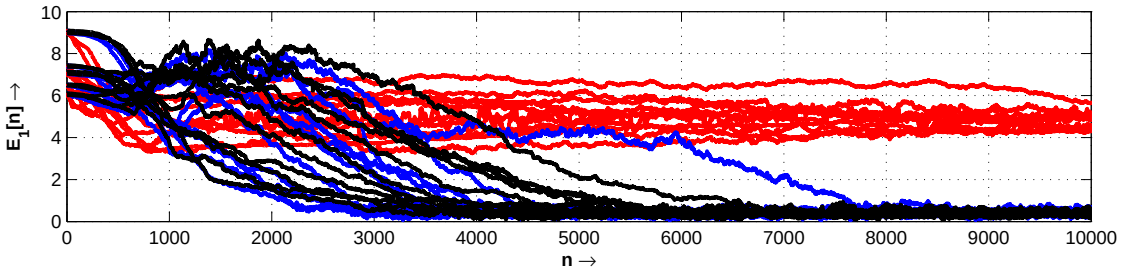

Abbildung 2.12: Fehlende Stabilität des Hérault-Jutten-Algorithmus für $N = 3$ Quellsignale

Für den Test mit supergaußförmigen Signalen wurde das Konvergenzverhalten des Hérault-Jutten-Algorithmus ausgewertet. Als maximaler Separationsfehler für eine erfolgreiche Trennung wurde das Doppelte des mittleren Separationsfehlers der 500 Durchläufe mit dem geringsten Separationsfehler des Hérault-Jutten-Algorithmus festgelegt. Nur die letzten 1000 Samples wurden berücksichtigt. Die Obergrenze ergibt sich zu 2.0773. Die anderen beiden Algorithmen weisen dagegen für alle 1000 Durchläufe im Mittel einen Separationsfehler von

0.4 auf. In 47.2% der Durchläufe unterschreitet der Hérault-Jutten-Algorithmus die Obergrenze nicht.

Die Tests haben gezeigt, dass der Hérault-Jutten-Algorithmus nur für zwei Quellsignale zuverlässig konvergiert.

Um die Leistungsfähigkeit der anderen beiden Algorithmen zu verdeutlichen, wurde das Verhalten bei 20 Quellsignalen getestet.

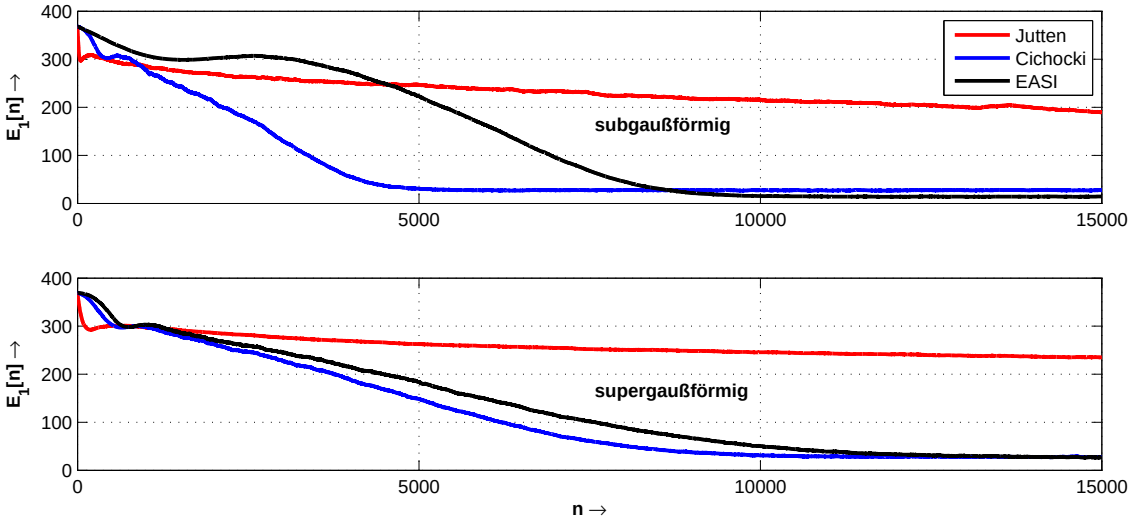

Abbildung 2.13: Vergleich der drei BSS-Algorithmen für $N = 20$ Quellsignale

Auch hier konvergieren die beiden Algorithmen in allen Durchläufen.

In diesem Kapitel wurde das Blind-Source-Separation-Problem modelliert. Der Lösungsansatz Independent Component Analysis wurde erläutert, der Hérault-Jutten-Algorithmus vorgestellt. Mit Hilfe von Matlab-Simulationen konnte festgestellt werden, dass die Funktionen $g_1(y) = y$ und $g_2(y) = sgn(y)$ den besten Kompromiss zwischen Konvergenzgeschwindigkeit und Separationsfehler liefern. Durch den Vergleich mit weiteren Algorithmen wurde deutlich, worin die Schwäche des Hérault-Jutten-Algorithmus liegt: Für mehr als zwei Quellsignale ist die Konvergenz nicht immer gewährleistet.

3 Stochastic Computing

In diesem Kapitel soll die stochastische Rechentechnik vorgestellt werden. Zu Beginn wird gezeigt, wie ein Dezimalwert in eine Wahrscheinlichkeit codiert wird. Danach werden die Komponenten beschrieben, mit denen Rechenoperationen durchgeführt oder Funktionen approximiert werden können. Anschließend wird auf den wichtigen Aspekt der Zufallszahlengenerierung eingegangen. Abschließend werden alternative Codierungsverfahren vorgestellt und die Vor- und Nachteile der stochastischen Rechentechnik gegenüber der Festkomma-Arithmetik diskutiert.

3.1 Codierung

Bei der stochastischen Rechentechnik wird eine Dezimalzahl[1] $x \in (-1, 1]$ in eine Wahrscheinlichkeit $p_X \in (0, 1)$ umgesetzt. Dabei besteht folgender linearer Zusammenhang:

$$p_X = \frac{x+1}{2} \tag{3.1}$$

Der kleinste Wert $x = -1$ entspricht der Wahrscheinlichkeit $p_X = 0$, der Wert $x = 0$ der Wahrscheinlichkeit $p_X = 0.5$ und der Maximalwert der Wahrscheinlichkeit $p_X = 1$. Abb. 3.1 zeigt die Übertragungskennlinie.

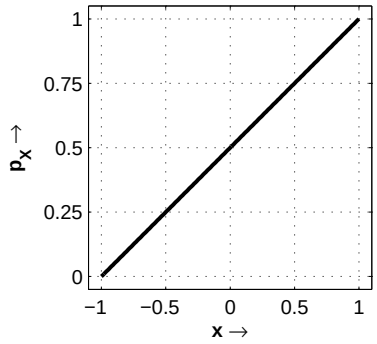

Abbildung 3.1: Übertragungskennlinie bei der stochastischen Codierung

[1] In dieser Arbeit werden ausschließlich digitale Signale verarbeitet. Zur einheitlichen Darstellung von N-Bit-Festkommazahlen wurde das $Q(N-1)$-Format gewählt, sodass gilt: $x = -x_{N-1} + \sum_{i=0}^{N-2} x_i \cdot 2^{-i-1}$

Um den rationalen Wert x in die Wahrscheinlichkeit p_X zu codieren, wird ein stochastischer Bitstrom X erzeugt. Die Wahrscheinlichkeit, dass ein beliebiges Bit dieses Bitstroms gleich Eins ist, repräsentiert die Wahrscheinlichkeit p_X:

$$p(X = 1) = p_X \tag{3.2}$$

Besteht der Bitstrom X beispielsweise zu 0% aus Einsen, so entspricht dies der Wahrscheinlichkeit $p_X = 0$. Bei 100% Einsen in dem Bitstrom ist $p_X = 1$, bei 50% Einsen ist $p_X = 0.5$, wobei die Anordnung von Einsen und Nullen zufällig ist. So repräsentieren z.B. die Bitströme $(1, 0, 0, 1, 1, 1, 0, 0)$, $(0, 1, 0, 0, 0, 1, 1, 1)$ und $(1, 1, 0, 0)$ dieselbe Wahrscheinlichkeit $p_X = 0.5$.

Erzeugt wird der Bitstrom X mit einem Digital-Stochastik-Umsetzer (siehe Bild 3.2). Bei der Umsetzung wird der Wert x mit L Zufallszahlen r verglichen.

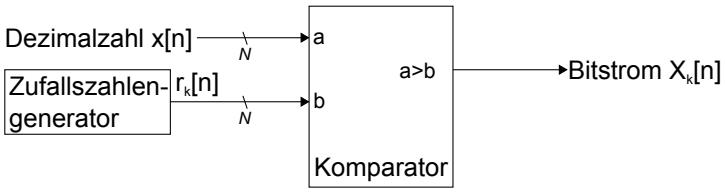

Abbildung 3.2: Digital-Stochastik-Umsetzer

Dabei ist das k-te Bit des Bitstroms eine Eins, wenn der Wert x größer als die Zufallszahl r_k ist:

$$X_k = x > r_k, \qquad mit \;\; 0 \leq k \leq L - 1 \tag{3.3}$$

Die Zufallszahlen sind gleichverteilt, sodass gemäß Gl. (3.2) die Wahrscheinlichkeit einer Eins am Ausgang des Komparators $p(X = 1)$ gleich der Wahrscheinlichkeit p_X ist [9]. Mit der Anzahl an Einsen K_X in einem Bitstrom der Länge L gilt demzufolge:

$$p_X = \frac{K_X}{L} \tag{3.4}$$

Daraus ergibt sich die Struktur des Stochastik-Digital-Umsetzers in Bild 3.3, der die Decodierung des Bitstromes ermöglicht. Die Anzahl der Einsen repräsentiert die Wahrscheinlichkeit, weshalb für die Decodierung ein Zähler bei jeder Eins um $\Delta p_X = \frac{1}{L}$ inkrementiert wird.

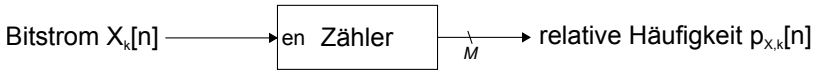

Abbildung 3.3: Stochastik-Digital-Umsetzer

Das Beispiel einer Codierung ist in Abb. 3.4 dargestellt. $L = 8$ Zufallszahlen pro Abtastwert werden erzeugt. Der Bitstrom entsteht durch die Komparation eines Abtastwertes mit L Zufallszahlen.

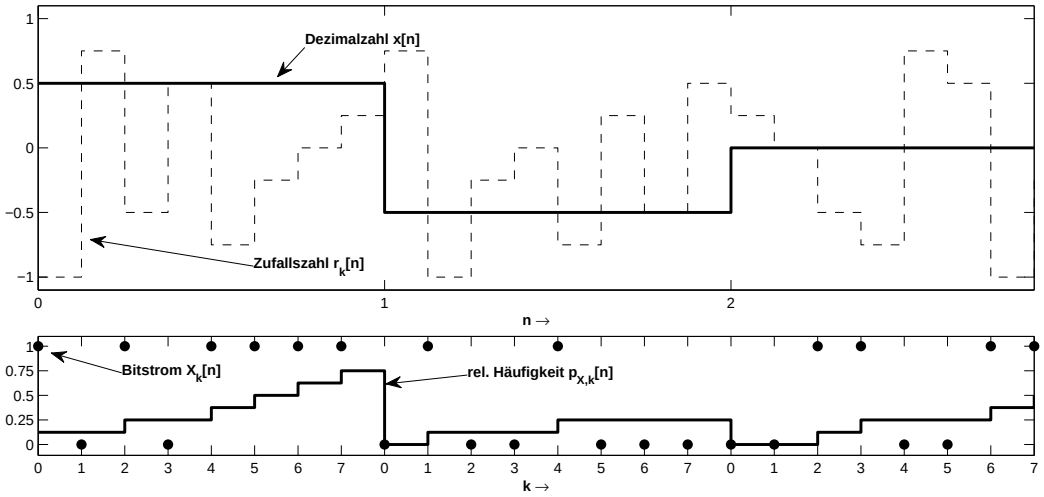

Abbildung 3.4: Beispielhafter Verlauf einer Codierung; oben Dezimalzahl und Zufallszahl, unten Bitstrom und dekodierte relative Häufigkeit

Da Zufallszahlen über L Werte nicht ideal gleichverteilt sind, kommt es zu einer Varianz bei der Codierung. Die Anzahl der Einsen bei der Digital-Stochastik-Umsetzung stellt nur eine Schätzung der Wahrscheinlichkeit dar:

$$\hat{K}_X = \sum_{k=0}^{L-1} X_k \quad \rightarrow \quad \hat{p}_X = \frac{\hat{K}_X}{L} \tag{3.5}$$

Erst der Erwartungswert $E\{\hat{p}_X\}$ ist gleich der Wahrscheinlichkeit. Abb. 3.5 veranschaulicht die Streuung bei der Codierung. Für den linken Plot wurde jeder Dezimalwert x 100 mal in einen Bitstrom codiert. Die Bitstromlänge betrug $L = 512$. Es ist zu sehen, dass die Streuung zu betragsmäßig kleinen Dezimalwerten steigt.

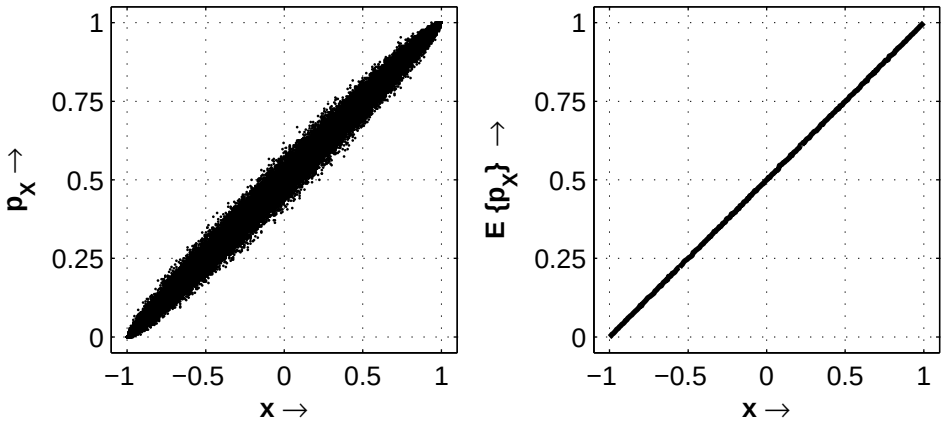

Abbildung 3.5: Streuung bei der Digital-Stochastik-Umsetzung; links relative Häufigkeit, rechts Mittelwert

Rechts ist der Mittelwert abgebildet, der näherungsweise der Transformationsgeraden aus Abb. 3.1 entspricht. Für die Varianz bei der Digital-Stochastik-Umsetzung gilt [9]:

$$Var\{\hat{p}_X\} = \frac{p_X(1-p_X)}{L} \tag{3.6}$$

Die Varianz ist also antiproportional zu der Bitstromlänge L. Durch Verlängerung des Bitstroms wird die Schätzung der Wahrscheinlichkeit genauer, die Verzögerung bei der Verarbeitung eines Dezimalwertes jedoch entsprechend größer.

Zwei Nachteile bei der stochastischen Rechentechnik werden hier ersichtlich:

- unpräzise Codierung durch Varianz der Zufallszahlen
- Verzögerung um L Takte bei der Verarbeitung eines Dezimalwertes

In den folgenden Abschnitten wird gezeigt, dass dafür mit sehr geringem Hardwareaufwand Rechenoperationen durchgeführt und nichtlineare Funktionen approximiert werden können und sich zusätzlich die Störanfälligkeit deutlich verringert.

3.2 Komponenten mit kombinatorischer Logik

Durch die Repräsentation einer Wahrscheinlichkeit als Bitstrom ist es möglich, mit einfachen logischen Verknüpfungen der Bits eine Multiplikation, Addition oder Subtraktion durchzuführen. Das soll in diesem Abschnitt gezeigt werden.

3.2.1 Multiplizierer

Aufgrund der statistischen Unabhängigkeit aller Zufallszahlen sind alle Bitströme unabhängig. Dementsprechend gilt für die Verbundwahrscheinlichkeit zweier Bitströme A und B:

$$p(A = 1 \wedge B = 1) = p_A \cdot p_B \tag{3.7}$$

Übertragen auf die einzelnen Bits der Ströme gilt:

$$A_k \wedge B_k \quad \Leftrightarrow \quad p_A \cdot p_B \tag{3.8}$$

Durch die UND-Verknüpfung zweier Bitströme ergibt sich das Produkt der beiden Wahrscheinlichkeiten. Gesucht ist aber das Produkt zweier Dezimalzahlen $c = a \cdot b$. Um dieses

aus den Bitströmen zu erhalten, müssen die Zahlen als Wahrscheinlichkeiten dargestellt werden. Gemäß Gl. (3.1) gilt allgemein für eine Dezimalzahl x:

$$x = 2p_X - 1 \tag{3.9}$$

Daraus folgt:

$$\begin{array}{ccccc} \overbrace{2p_C - 1} & = & \overbrace{(2p_A - 1)} & \cdot & \overbrace{(2p_B - 1)} \\ c & = & a & \cdot & b \end{array} \tag{3.10}$$

Gl. (3.10) kann nun nach p_C umgestellt werden:

$$p_C = p_A \cdot p_B + (1 - p_A) \cdot (1 - p_B) \tag{3.11}$$

Daraus ergibt sich für die Multiplikation eine einfache XNOR-Verknüpfung der Bitströme:

$$\begin{aligned} C_k &= (A_k \wedge B_k) \vee (\overline{A}_k \wedge \overline{B}_k) \\ &= A_k \leftrightarrow B_k \end{aligned} \tag{3.12}$$

Abb. 3.6 zeigt den stochastischen Multiplizierer.

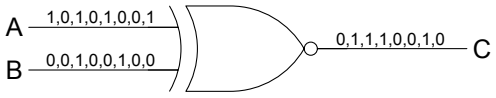

Abbildung 3.6: Stochastischer Multiplizierer

Als Beispiel werden in der Abbildung die Werte $a = 0$ und $b = -0.5$ multipliziert. Die Wahrscheinlichkeiten sind nach Gl. (3.1) $p_A = 4/8$ und $p_B = 2/8$. Als Ergebnis erhält man erwartungsgemäß $p_C = 4/8$, also $c = 0$. Wenn Gl. (3.13) erfüllt ist, sind die Bitsequenzen statistisch unabhängig [24]. Die Bitströme im Beispiel wurden bewusst genauso gewählt, in der Praxis muss die Bitstromlänge viel größer sein.

$$\sum_{k=0}^{L-1} A_k B_k = \frac{1}{L} \left(\sum_{k=0}^{L-1} A_k \cdot \sum_{k=0}^{L-1} B_k \right) \tag{3.13}$$

Ist Gl. (3.13) nicht erfüllt, z.B. wenn in obiger Abbildung $B = (0, 0, 1, 0, 1, 0, 0, 0)$ ist, dann kommt trotz präziser Codierung aufgrund von Korrelationen zwischen den Bitströmen ein falsches Ergebnis heraus (in diesem Fall $p_C = 6/8$, $c = 0.5$).

Wichtig bei der Verknüpfung von Bitströmen ist also die statistische Unabhängigkeit zueinander.

Das Varianz bei der Multiplikation wird in Abb. 3.7 veranschaulicht.

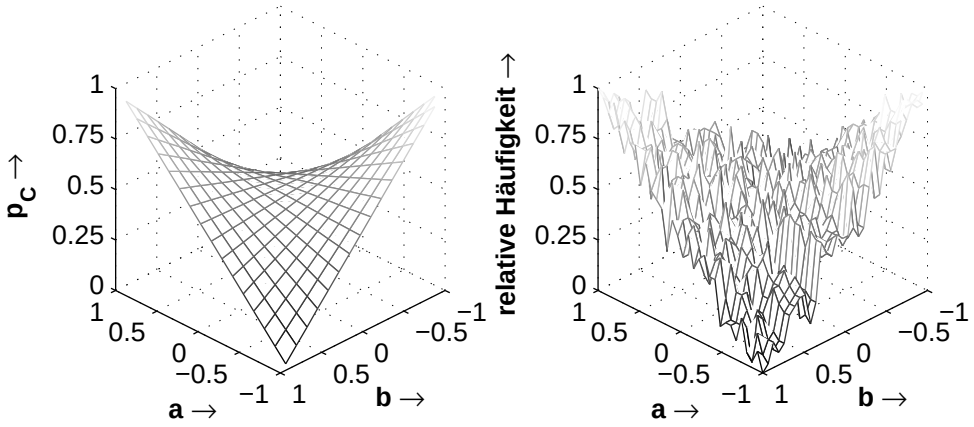

Abbildung 3.7: Stochastische Multiplikation; links ideal, rechts mit $L = 512$

3.2.2 Addierer und Subtrahierer

Bei der stochastischen Addition muss berücksichtigt werden, dass die maximale Summe zweier Wahrscheinlichkeiten $p_{C,max} = 1$ sein kann. Deswegen ist die stochastische Addition eine mittelnde Addition:

$$c = \frac{a + b}{2} \tag{3.14}$$

Durch Einsetzen der Wahrscheinlichkeiten und Umstellen nach p_C erhält man

$$p_C = 0.5p_A + 0.5p_B. \tag{3.15}$$

Die Gewichtung mit 0.5 wird durch eine zusätzliche binäre Zufallszahlenfolge R_b mit $p_R = 0.5$ erzeugt:

$$p_C = p_R p_A + (1 - p_R)p_B \tag{3.16}$$

Abgebildet auf die logischen Verknüpfungen der Bitströme ergibt sich ein Multiplexer als Element zur mittelnden Addition:

$$C_k = (R_{b,k} \wedge A_k) \vee (\overline{R}_{b,k} \wedge B_k) \tag{3.17}$$

Für eine mittelnde Subtraktion muss der Bitstrom B invertiert werden. Der stochastische Addierer und der stochastische Subtrahierer sind in Abb. 3.8 zu sehen. Der beispielhafte Verlauf einer Addition ist in Abb. 3.9 abgebildet.

Wichtig ist, dass eine Wahrhscheinlichkeit stets ausschließlich eine Dezimalzahl im Wertebereich ± 1 darstellt und nur unter dieser Voraussetzung die stochastischen Komponenten funktionieren.

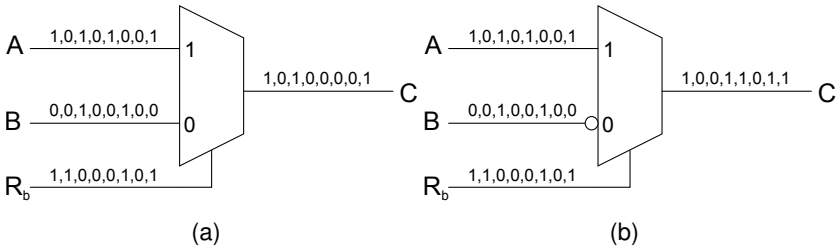

Abbildung 3.8: Stochastischer Addierer (a) und stochastischer Subtrahierer (b)

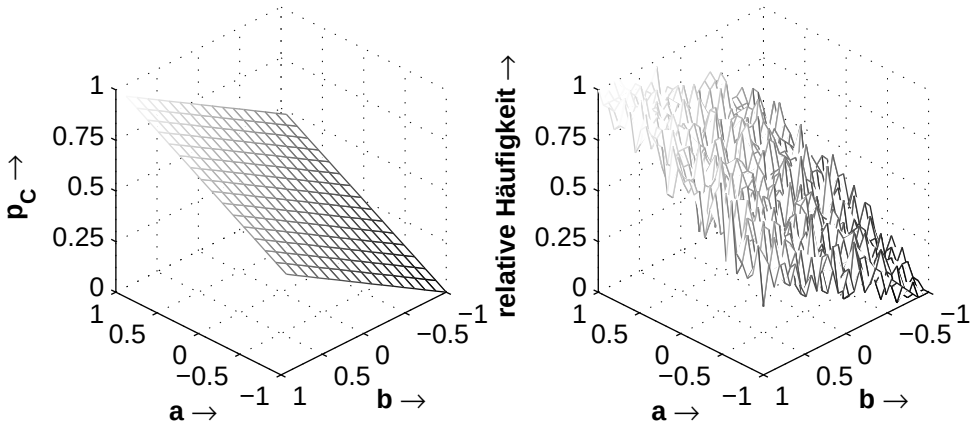

Abbildung 3.9: Stochastische Addition; links ideal, rechts mit $L = 512$

3.3 Komponenten mit sequentieller Logik

In Kapitel 2 wurde gezeigt, dass zur Quellentrennung nichtlineare Funktionen benötigt werden und die Koeffizienten iterativ adaptiert werden. Nichtlineare Funktionen können bei der stochastischen Rechentechnik approximiert werden und auch Integratoren für die Koeffizientenadaption können realisiert werden. Dies geschieht mit Hilfe von sequentieller Logik.

3.3.1 Integrierer

Zu jedem Zeitpunkt soll der Dezimalwert $a[n]$ akkumuliert werden:

$$w[n] = w[n-1] + a[n] \tag{3.18}$$

Zur Verdeutlichung der Zeitabhängigkeit wird nun der Zeitindex n genutzt.

Um zur Struktur des Integrierers zu gelangen, wird an dieser Stelle lediglich der zu akkumulierende Wert mit seiner Wahrscheinlichkeit $a[n] = 2p_A[n] - 1$ dargestellt:

$$w[n] = w[n-1] + (2p_A[n] - 1) \tag{3.19}$$

Formt man den rechten Term um,

$$w[n] = w[n-1] + (p_A[n] - (1 - p_A[n])), \tag{3.20}$$

und ersetzt die Wahrscheinlichkeit einer Eins im Bitstrom $p_A[n] = \frac{K_A[n]}{L}$ durch die Anzahl an Einsen $K_A[n] = \sum_{k=0}^{L-1} A_k[n]$, normiert auf die Bitstromlänge L, und die Wahrscheinlichkeit einer Null $1 - p_A[n] = \frac{L-K_A[n]}{L}$ durch die Anzahl an Nullen $L - K_A[n] = \sum_{k=0}^{L-1} \overline{A}_k[n]$, normiert auf die Bitstromlänge L, dann ergibt sich folgender Ausdruck für die Integration:

$$w[n] = w[n-1] + \frac{1}{L} \sum_{k=0}^{L-1} \left(A_k[n] - \overline{A}_k[n] \right) \tag{3.21}$$

Das Integral $w[n]$ ergibt sich demzufolge aus der mit $\frac{1}{L}$ gewichteten Differenz von Einsen und Nullen des Bitstroms A. Realisiert werden kann die Integration nach Gl. (3.21) mit einem Up/Down-Zähler, der bei einer Eins im Eingangsbitstrom um $1/L$ inkrementiert und bei einer Null um $1/L$ dekrementiert wird. In Abb. 3.9 ist das Blockschaltbild eines stochastischen Integrierers zu sehen.

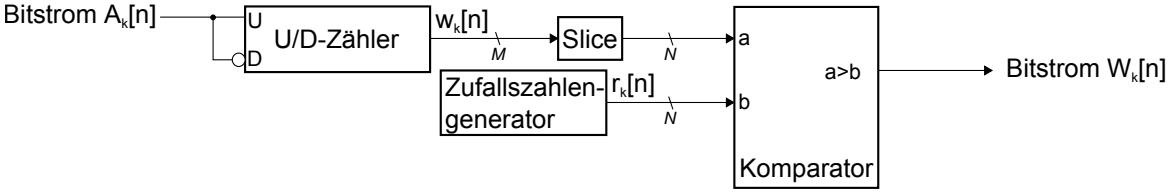

Abbildung 3.10: Stochastischer Integrierer

Das Ausgangssignal des Zählers ist der Dezimalwert $w_k[n]$. Erst am Ende des Bitstromes ist die Integration für den Zeitpunkt n abgeschlossen und somit $w_{L-1}[n] = w[n]$. Der Zähler muss mindestens $M = \lceil ld(L) \rceil + 1$ Bit breit sein, um alle Werte von $-L/L$ bis L/L darstellen zu können. Die obersten N Bit des Integrierers entsprechen dem Dezimalwert des Integrals, wenn $M \geq (N+1)$ ist, die Bitstromlänge L also so gewählt wurde, dass zumindest theoretisch alle 2^N Dezimalwerte eindeutig codiert werden können. Dieser N-Bit-Dezimalwert muss zur Weiterverarbeitung in einen Bitstrom codiert werden. Der stochastische Integrierer besteht dementsprechend aus dem Zähler sowie einem D/S-Umsetzer.

Durch Erhöhen der Bitbreite M ist es möglich, eine gewichtete Integration durchzuführen. Mit $M > \lceil ld(L) \rceil + 1$ ergibt sich für die Integration

$$w[n] = w[n-1] + \mu \cdot a[n] \tag{3.22}$$

mit dem Gewichtungsfaktor $\mu = 2^{-(M-ld(L)-1)}$. Abb. 3.11 zeigt den Verlauf der Integration gemäß Gl. (3.22) für $N = 12$ und $a[n] = 1 - 2^{-(N-1)}$. Auf der Abzisse gekennzeichnet wurden die Kehrwerte der Gewichtungsfaktoren. Zu diesen Zeitpunkten erreicht der Integrator das Maximum. Ein Überlaufschutz verhindert den Überlauf hin zu negativen Zahlen. Ebenso wird ein Unterlauf verhindert.

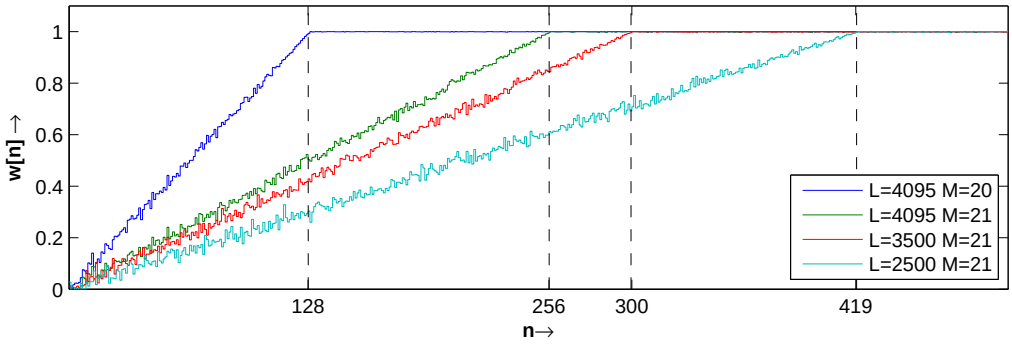

Abbildung 3.11: Verlauf der gewichteten stochastischen Integration mit unterschiedlichen Gewichtungsfaktoren μ abhängig von Integrierer-Bitbreite M und Bitstromlänge L; Bitbreite des Eingangssignals ist stets $N = 12$

Vorteil durch diese gewichtete Integration ist vor allem das Einsparen eines Digital-Stochastik-Codierers beim Anlernen der Entmischungskoeffizienten.

3.3.2 Approximation von Funktionen

Auf Basis eines linearen Zustandsautomaten können verschiedene Funktionen approximiert werden. Die grundlegende Struktur ist in Abb. 3.12 abgebildet. Die lineare Anordnung der Zustände entspricht der eines Up/Down-Zählers mit Sättigung. Aus Sicht der Digitaltechnik handelt es sich um einen Moore-Automaten mit $2n$ Zuständen. Das Eingangssignal X ist der stochastische Bitstrom des Dezimalwertes x, das Ausgangssignal Y der Bitstrom des transformierten Signals $y = fsm(x)$. Abhängig von der Wahl des Moore-Ausgangswertes s_i je Zustand S_i können verschiedene Funktionen approximiert werden.

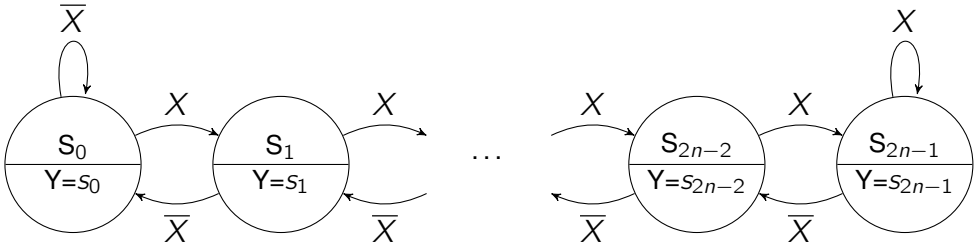

Abbildung 3.12: Zustandsdiagramm eines linearen Zustandsautomaten mit dem Eingangssignal X und dem Ausgangssignal Y

Die theoretischen Grundlagen wurden von Gaines erarbeitet und basieren auf der Modellierung des Zustandsautomaten als homogene irreduzible und aperiodische Markov-Kette [9]. Brown et al. entwickelten mit dem Ansatz von Gaines die zwei hier vorgestellten Automaten [25], deren Funktionalität sie jedoch nur empirisch nachwiesen. Li et al. bewiesen nachträglich mathematisch die Korrektheit dieser Zustandsautomaten [26, 27].

Im Folgenden soll gezeigt werden, dass der Zustandsautomat in Abb. 3.13 die Funktion $fsm(x) = tanh(n \cdot x)$ approximiert. Ausgangspunkt sind folgende Annahmen, die sich aus der Modellierung als Markov-Kette ergeben:

1. Die Wahrscheinlichkeit im stationären Zustand im Zustand S_i zu sein, ist $p_i = P(S[n] = S_i)$ und es muss gelten: $\sum_{i=0}^{2n-1} p_i = 1$.
2. Die Übergangswahrscheinlichkeit von einem Zustand S_{i-1} in den Zustand S_i, $P(S[n] = S_i|S[n-1] = S_{i-1})$ ist gleich der Übergangswahrscheinlichkeit vom Zustand S_i nach S_{i-1}, $P(S[n] = S_{i-1}|S[n-1] = S_i)$, sodass mit der Wahrscheinlichkeit einer Eins im Eingangsbitstrom p_X gilt: $p_i \cdot (1 - p_X) = p_{i-1} \cdot p_X$.
3. Die Wahrscheinlichkeit einer Eins im Ausgangsbitstrom Y ergibt sich mit den Ausgangswerten $s_i \in (0, 1)$ je Zustand S_i zu $p_Y = \sum_{i=0}^{2n-1} s_i \cdot p_i$.

Aus Punkt 1 und 2 folgt für die Wahrscheinlichkeit im Zustand S_i zu sein:

$$p_i = \frac{\left(\frac{p_X}{1-p_X}\right)^i}{\sum_{j=0}^{2n-1}\left(\frac{p_X}{1-p_X}\right)^j} \tag{3.23}$$

tanh-Approximation

Das Zustandsdiagramm für die Approximation zeigt Abb. 3.13.

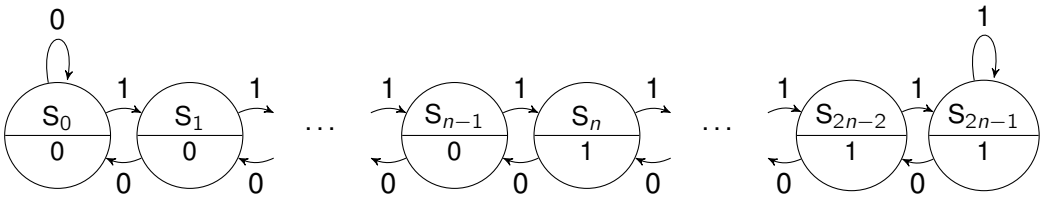

Abbildung 3.13: Zustandsautomat der tanh-Approximation

Für das Ausgangssignal gilt demzufolge:

$$s_i = \begin{cases} 0, & 0 \leq i \leq n-1 \\ 1, & n \leq i \leq 2n-1 \end{cases} \tag{3.24}$$

Für die Wahrscheinlichkeit einer Eins im Ausgangsbitstrom folgt damit

$$p_Y = \sum_{i=n}^{2n-1} p_i. \tag{3.25}$$

Gl. (3.25) eingesetzt in Gl. (3.23) führt zu

$$p_Y = \frac{\sum_{i=n}^{2n-1} \left(\frac{p_X}{1-p_X}\right)^i}{\sum_{j=0}^{2n-1} \left(\frac{p_X}{1-p_X}\right)^j} = \frac{\left(\frac{p_X}{1-p_X}\right)^n - \left(\frac{p_X}{1-p_X}\right)^{2n}}{1 - \left(\frac{p_X}{1-p_X}\right)^{2n}}, \tag{3.26}$$

was durch Ausklammern und Umformen,

$$p_Y = \frac{\left(\frac{p_X}{1-p_X}\right)^n \cdot \left(1 - \left(\frac{p_X}{1-p_X}\right)^n\right)}{\left(1 + \left(\frac{p_X}{1-p_X}\right)^n\right) \cdot \left(1 - \left(\frac{p_X}{1-p_X}\right)^n\right)}, \tag{3.27}$$

gekürzt werden kann zu

$$p_Y = \frac{\left(\frac{p_X}{1-p_X}\right)^n}{1 + \left(\frac{p_X}{1-p_X}\right)^n}. \tag{3.28}$$

Mit der bekannten Transformation $p_Y = \frac{y+1}{2}$, eingesetzt in Gl. (3.28), ergibt sich folgender Ausdruck:

$$\frac{y+1}{2} = \frac{\left(\frac{1+x}{1-x}\right)^n}{1 + \left(\frac{1+x}{1-x}\right)^n} \tag{3.29}$$

Dieser aufgelöst nach y

$$y = \frac{\left(\frac{1+x}{1-x}\right)^n - 1}{\left(\frac{1+x}{1-x}\right)^n + 1} \tag{3.30}$$

ergibt mit der Taylor-Reihe $(1 \pm x)^n \approx e^{\pm n \cdot x}$ die Approximation des Tangens Hyperbolicus:

$$y = \frac{e^{2n \cdot x} - 1}{e^{2n \cdot x} + 1} = tanh(n \cdot x) \tag{3.31}$$

Durch die Anzahl an Zuständen $2n$ kann die Steilheit der Approximation variiert werden. Abb. 3.14 zeigt diese Abhängigkeit.

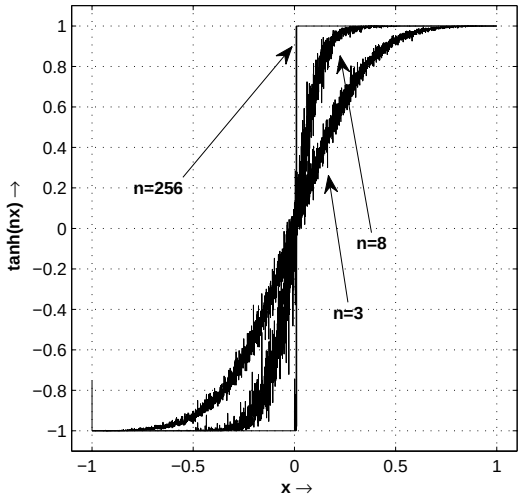

Abbildung 3.14: Verlauf der tanh-Approximation abhängig von der Anzahl an Zuständen

Für $n \to \infty$ wird aus der Approximation die Signum-Funktion: $\lim_{n \to \infty} tanh(n \cdot x) = sgn(x)$.

Gain-Funktion

Die Gain-Funktion benötigt zusätzlich eine binäre Zufallszahlenfolge R_b. Der Zustandsautomat ist in Abb. 3.15 zu sehen. Für den mathematischen Beweis sei auf [27] verwiesen. Die Steilheit der Gain-Funktion wird durch die Wahrscheinlichkeit einer Eins p_{R_b} bestimmt [25, 27].

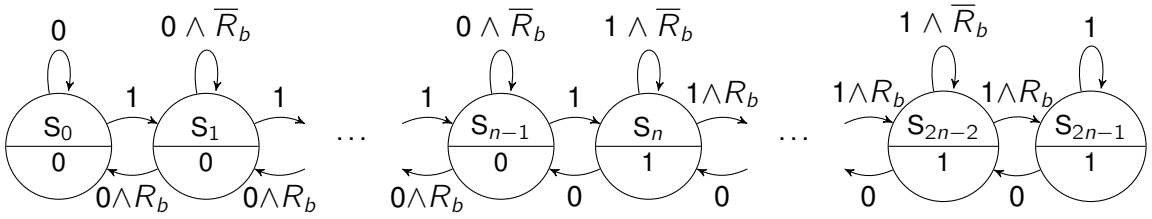

Abbildung 3.15: Zustandsautomat der Gain-Funktion

Für die Ausgangswahrscheinlichkeit p_Y gilt:

$$p_Y = \begin{cases} 0, & 0 \leq p_X < \frac{p_{R_b}}{1+p_{R_b}} \\ \frac{1+p_{R_b}}{1-p_{R_b}} \cdot p_X - \frac{p_{R_b}}{1-p_{R_b}}, & \frac{p_{R_b}}{1+p_{R_b}} \leq p_X \leq \frac{1}{1+p_{R_b}} \\ 1, & \frac{1}{1+p_{R_b}} < p_X \leq 1 \end{cases} \tag{3.32}$$

Mit Bezug auf die Skalierung bei der stochastischen Addition und Subtraktion soll hier der Gain auf den Faktor 2 festgelegt werden. Dies wird mit Gl. (3.32) bei $p_{R_b} = 0.33$ erreicht.

Die notwendige Anzahl an Zuständen muss empirisch ermittelt werden. Abb. 3.16 zeigt die Simulationsergebnisse für verschiedene n. Bei $n = 4$ ist die Steilheit hin zu den Maxima deutlich geringer als bei $n = 8$ und $n = 16$. Dort sind kaum Unterschiede zu erkennen.

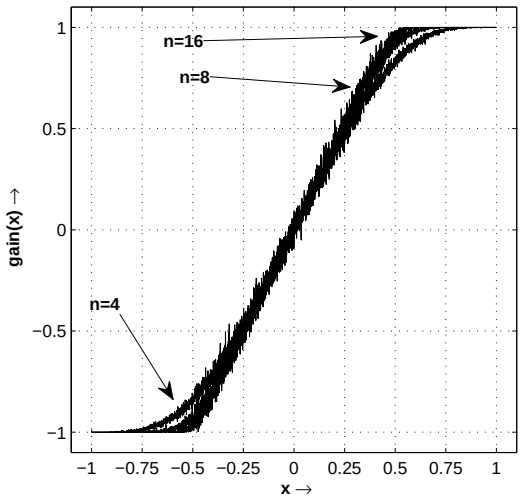

Abbildung 3.16: Verlauf der Gain-Funktion abhängig von der Anzahl an Zuständen; $p_{R_b} = 0.33$

Zur Festlegung der Anzahl der Zustände wurde die mittlere Fehlerleistung e^2 nach K zufälligen Eingangswerten x für verschiedene n ermittelt. Unter der Annahme, die Gain-Funktion

wird eingesetzt, wenn die Eingangssignale betragsmäßig klein sind, wurde der Wertebereich auf $|x| \leq 0.5$ beschränkt.

$$e^2 = \frac{1}{K} \sum_{n=0}^{K-1} (sat(2x[n]) - gain(x[n]))^2 \tag{3.33}$$

Hierbei wird die ideale Gain-Funktion als $sat(2x)$ bezeichnet. Die Ergebnisse sind für $K = 100000$ Werte und $L = 4095$ in Tab. 3.1 zusammengefasst.

n	4	8	16	32	64
e^2	0.0051	0.0014	0.0012	0.0012	0.0013

Tabelle 3.1: Mittlere Fehlerleistung der Gain-Approximation

Mit $n = 8$, also 16 Zuständen, ist ein guter Kompromiss aus Genauigkeit und Hardwareaufwand gefunden. Für $n = 64$ ist die Fehlerleistung sogar höher, da die Anzahl an Zuständen für die Dynamik des Eingangssignals zu hoch ist.

Ein wichtiger Aspekt bei der stochastischen Rechentechnik ist die *Runlängenverteilung*. Die Zufallsfolgen müssen die Runlängenverteilung einer Bernoulli-Sequenz aufweisen, damit die Komponenten funktionieren [8]. Als Runlänge definiert wird die Anzahl aufeinanderfolgender gleicher Bits. Die Bitfolge $(0, 1, 1, 1, 0)$ weist eine Einser-Runlänge $rl_1 = 3$ auf, die Bitfolge $(1, 0, 0, 1)$ eine Nuller-Runlänge $rl_0 = 2$. Die ideale Verteilung dieser Runlängen resultiert aus der statistischen Unabhängigkeit der einzelnen Bits einer Zufallsfolge. Für die Wahrscheinlichkeit einer Runlänge i eines Bitstroms X mit der Wahrscheinlichkeit einer Eins p_X und der Wahrscheinlichkeit einer Null $1 - p_X$ gilt:

$$p(rl_1 == i) = (1 - p_X)^2 \cdot p_X^i \tag{3.34}$$

$$p(rl_0 == i) = p_X^2 \cdot (1 - p_X)^i \tag{3.35}$$

Für die stochastische Multiplikation, Addition und Subtraktion muss mindestens eine Eingangsfolge diese Runlängenverteilung aufweisen. Die Theorie der Zustandsautomaten basiert auf Eingangsfolgen mit idealer Runlängenverteilung. Dies ist auch gewährleistet, solange die Zufallszahlen bei der Codierung aus Bernoulli-Sequenzen erzeugt werden, was generell näherungsweise der Fall ist. Die Ausgangsfolgen der Zustandsautomaten sind jedoch keine Bernoulli-Sequenzen. Dadurch können zwei Zustandsautomaten nicht ohne weiteres kaskadiert werden.

Zur Veranschaulichung zeigt Abb. 3.17 das Simulationsergebnis einer Kaskade aus Gain-Automat und tanh-Automat. Die resultierende Funktion lautet $y = tanh(3 \cdot gain(x))$.

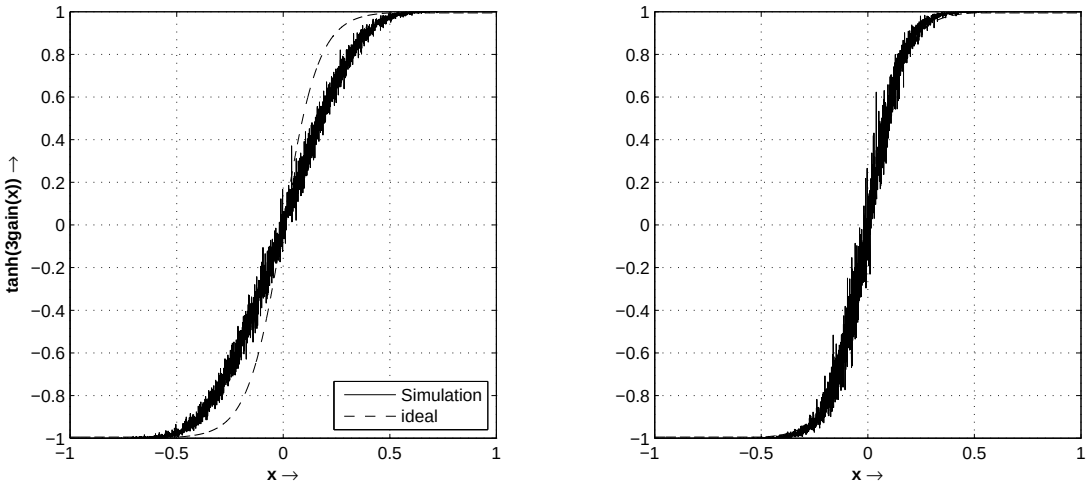

Abbildung 3.17: Verhalten von kaskadierten Zustandsautomaten; links fehlerhaftes Ausgangssignal, rechts korrektes Ausgangssignal durch 4x8 Interleaver

Auf dem linken Diagram ist zu sehen, dass die Kaskadierung fehlerhafte Ergebnisse nach sich zieht. Als Lösungsansatz vorgeschlagen wird ein Interleaver, der zwischen der Automaten-Kaskade eingefügt wird. Dadurch wird die Runlängenverteilung des Gain-Ausgangssignals der idealen Bernoulli-Verteilung angenähert. Abb. 3.18 zeigt diesen Effekt für den Eingangswert $x = 0.25$. Das Ausgangssignal ist dementsprechend $y = 0.5$, die Wahrscheinlichkeit einer Eins im Ausgangsbitstrom ist $p_Y = 0.75$. Daraus kann mit Gl. (3.34) und (3.35) die ideale Runlängenverteilung berechnet werden, die in die Diagramme eingezeichnet wurde.

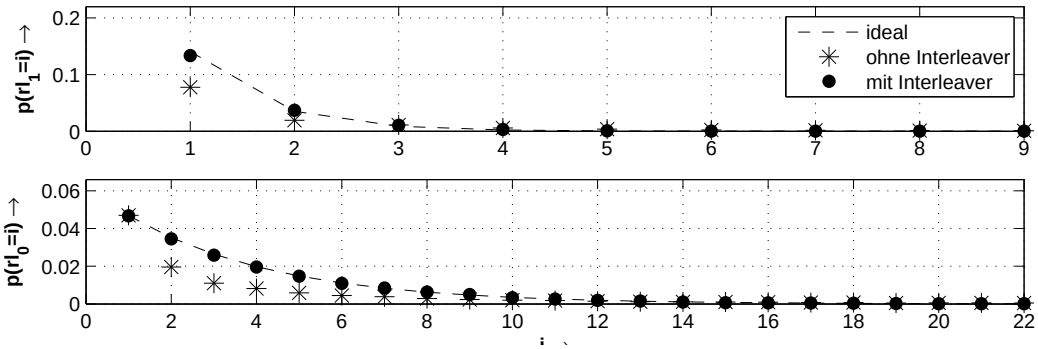

Abbildung 3.18: Runlängenverteilung des Gain-Ausgangssignals bei $x = 0.25$

Die Annäherung an die ideale Verteilung durch den Interleaver ist deutlich zu erkennen. Empirisch wurde für $L = 4095$ ermittelt, dass ein 4×8-Interleaver den besten Kompromiss aus Fehlerleistung und Hardwareaufwand liefert. Das Schema eines 2×1-Interleavers ist in

Abb. 3.19 zu sehen. Allgemein besteht ein $m \times n$-Interleaver aus $m + 1$ Stufen, wobei jede Stufe $i \in (0, m)$ aus $i \cdot n$ Flipflops besteht.

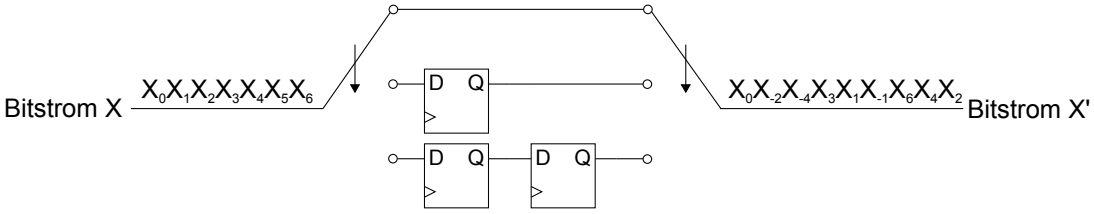

Abbildung 3.19: Schema eines 2×1-Faltungsinterleavers

Es handelt sich um einen Faltungsinterleaver, der prinzipiell den Eingangsbitstrom verlustlos durchmischt [28]. Dadurch wird die unweigerlich aus der Struktur der Zustandsautomaten resultierende Korrelation des Ausgangsbitstroms verringert.

Alle relevanten Basis-Komponenten der stochastischen Rechentechnik sind an dieser Stelle vorgestellt worden. Ein weiterer wichtiger Faktor bei der Rechentechnik ist die Generierung der Zufallszahlen.

3.4 Zufallszahlen

In der Praxis reicht es häufig, Pseudozufallszahlen zu generieren. Diese weisen die wichtigsten Eigenschaften von Zufallszahlen auf, werden jedoch deterministisch erzeugt und können somit reproduziert werden. Generiert werden kann eine binäre Pseudozufallszahlenfolge mit einem rückgekoppelten Schieberegister. Abb. 3.20 zeigt ein solches Schieberegister mit $M = 3$ Bit. Die Periodizität der Folge wird bei geeigneter Rückkopplung maximal und ist $m = 2^M - 1$. Im Fall von $M = 3$ Bit ist die Folge $m = 7$ Bit lang.

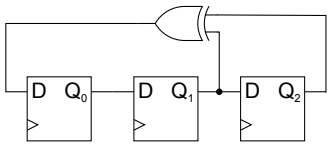

Abbildung 3.20: Rückgekoppeltes Schieberegister mit $M = 3$ Bit

Die binäre Zufallsfolge kann z.B. am Ausgang des ersten Flipflops bei Q_0 abgegriffen werden. Am zweiten Flipflop kann die um einen Takt verzögerte Folge, am dritten die entsprechend um zwei Takte verzögerte Folge abgegriffen werden.

Um für SC geeignete Zufallszahlen zu generieren, ist eine im Vergleich zur Bitstromlänge L viel größere Periodizität notwendig. Für die Codierung gemäß Abb. 3.2 werden im einfachsten Fall N Schieberegister mit verschiedenen Startwerten konkateniert. Das Schema eines solchen Zufallszahlengenerators ist in Abb. 3.21 zu sehen.

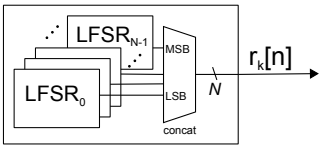

Abbildung 3.21: N-Bit-Pseudozufallszahlengenerator bestehend aus rückgekoppelten Schieberegistern

Diese Struktur mit N Schieberegistern für jeden Codierer ist jedoch nur mit einem hohen Hardwareaufwand realisierbar, was den Vorteil der Ressourcen sparenden Struktur des SC-Gesamtsystems zunichte machen kann. Deshalb wurde von Alspector et al. die Struktur in Abb. 3.22 vorgeschlagen [29]. Das Problem bei der Struktur aus Abb. 3.20 ist die geringe Verschiebung der Folgen von jeweils einem Bit pro Flipflop, weshalb das direkte Abgreifen der Folgen zur Codierung nicht geeignet ist.

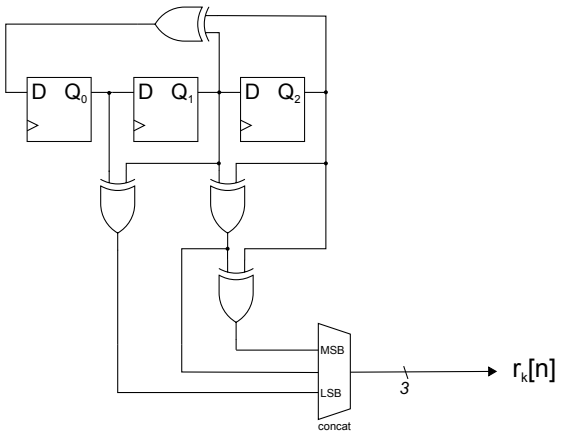

Abbildung 3.22: Effektive Generierung von 3-Bit-Pseudozufallszahlen

Durch xor-Verknüpfungen der Bits eines einzigen Schieberegisters ist es möglich, weit verschobene binäre Zufallsfolgen zu generieren. Diese weit verschobenen Folgen können dann konkateniert werden. Eine Implementierung mit einem 28-Bit-Schieberegister ist in [12] dokumentiert. 59 Folgen wurden aus diesem Schieberegister erzeugt, die annähernd statistisch unabhängig voneinander sind, da deren Mindestverschiebung zueinander generell $\gg L$ ist.

3.5 Alternative Codierungsverfahren

Um den Hardwareaufwand bei der Codierung weiter zu reduzieren, wurden alternative Codierungsverfahren entwickelt. Auch der Ansatz, die Streuung bei der Codierung zu reduzie-

ren bzw. zu eliminieren, wurde in der Vergangenheit verfolgt. In diesem Abschnitt werden diese Codierungsverfahren vorgestellt.

3.5.1 Bitslice-Technik

Die Bitslice-Technik wurde in [12] vorgestellt und erlaubt die Codierung einer Dezimalzahl mit nur einer binären Zufallsfolge. Dazu muss die Dezimalzahl in dem Offset-Binary-Zahlenformat vorliegen. Dadurch stellt jede Binärstelle ein Wahrscheinlichkeitsgewicht dar. Dies wird für $N = 3$ Bit in Tab. 3.2 veranschaulicht.

Das Offset-Binary-Format erhält man aus dem gängigen 2er-Komplement-Format, indem das MSB invertiert wird (vgl. Spalte 2 und 3). In der vierten Spalte sind die zu der Dezimalzahl gehörenden Wahrscheinlichkeiten aufgelistet. Fett gekennzeichnet sind jene, die das Gewicht einer einzigen Binärstelle darstellen.

Dezimalzahl	2er Komplement	Offset Binary	p_X
-1.00	100	000	0
-0.75	101	00**1**	**0.125**
-0.50	110	0**1**0	**0.25**
-0.25	111	011	0.375
0.00	000	**1**00	**0.5**
0.25	001	101	0.625
0.50	010	110	0.75
0.75	011	111	0.875

Tabelle 3.2: Zusammenhang zwischen der Offset-Binary-Codierung und der Wahrscheinlichkeit p_X

Ist nur das MSB gesetzt, so sind 50% der Bits im Bitstrom gleich Eins, ist nur das LSB gesetzt, sind es 12.5%. Somit kann die Dezimalzahl x wie folgt in eine Wahrscheinlichkeit p_X transformiert werden:

$$p_X = \sum_{i=0}^{N-1} x_i \cdot 2^{-(N-i)} \tag{3.36}$$

Daraus resultiert der Codierer in Abb. 3.23. Die Binärstellen - die Bitslices - der Dezimalzahl werden mit einem Multiplexer verbunden, der von einer binären Zufallsfolge geschaltet wird. Durch Kaskadierung der Bitslices wird die Gewichtung nach Gl. (3.36) realisiert. Mit $p_{R_b} = 0.5$ wird zu 50% das invertierte MSB durchgeschaltet. Die Wahrscheinlichkeit, dass bei der Zufallsfolge auf eine Eins eine Null folgt, ist $p_{R_b} \cdot (1 - p_{R_b}) = 0.25$, sodass zu 25% Bit 1 durchgeschaltet wird. Damit das LSB durchgeschaltet wird, müssen zwei Nullen auf

eine Eins folgen, weshalb das LSB mit der Wahrscheinlichkeit $p_{R_b} \cdot (1 - p_{R_b})^2 = 0.125$ durchgeschaltet wird.

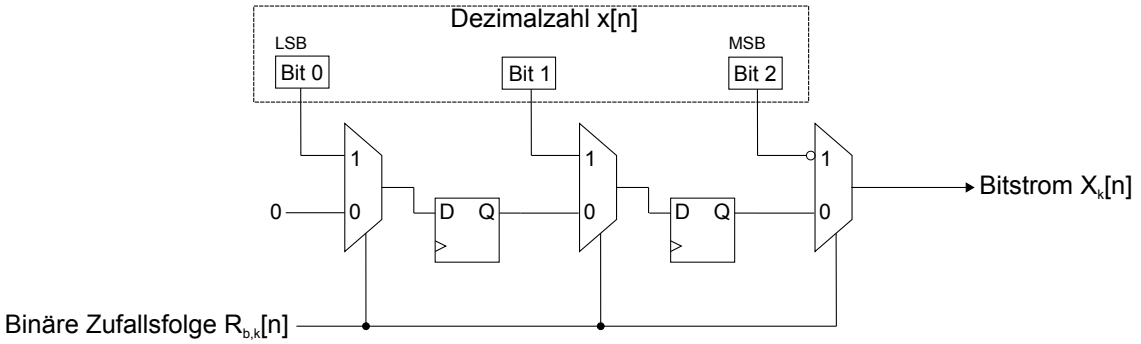

Abbildung 3.23: Codierung mit Bitslice-Technik

Die Verwendung nur einer binären Zufallsfolge bei dieser Codierung führt jedoch zu einem Problem, das in Abb. 3.24 veranschaulicht wird. Der erzeugte Bitstrom weist nicht immer eine Bernoulli-Verteilung der Runlängen auf, sodass es bei der Verwendung von Zustandsautomaten zu einer fehlerhaften Funktionsapproximation kommt.

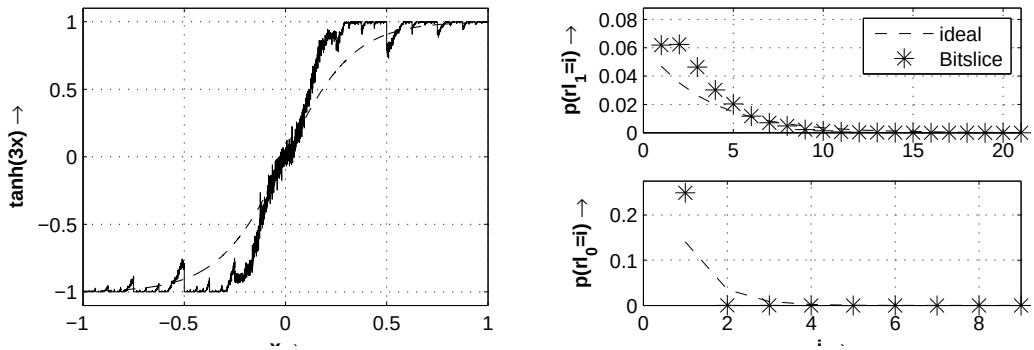

Abbildung 3.24: Fehlerhaftes Verhalten der tanh-Approximation bei der Bitslice-Technik; links Approximation, rechts Runlängenverteilung für $x = 0.5$

Links abgebildet ist der Verlauf der tanh-Approximation, es sind deutliche Sprünge zu sehen. Rechts ist die Einser-Runlängenverteilung für den codierten Wert $x = 0.5$ zu sehen.

Als Abhilfe wurden zwei Varianten erarbeitet. Genau wie bei der Gain-Funktion kann auch hier dem Codierer ein Interleaver nachgeschaltet werden, der durch verlustloses Mischen des Bitstroms dessen Runlängenverteilung verbessert. Nachteil ist hier der zusätzliche Aufwand, den der Interleaver nach sich zieht, wodurch sich die Ressourceneinsparung durch die weniger benötigten Zufallszahlen verringert.

Die zweite Variante zur Abhilfe ist in Abb. 3.25 zu sehen. Für eine N Bit breite Dezimalzahl werden auch N näherungsweise unkorrelierte Zufallsfolgen genutzt. Dadurch bleibt der Bedarf an Zufallsfolgen im Vergleich zur Komparator-Technik gleich. Jedoch verlangt die Bitslice-Technik wenig kombinatorische Logik und weist eine Pipelining-Struktur auf, die den kombinatorischen Pfad kurz hält.

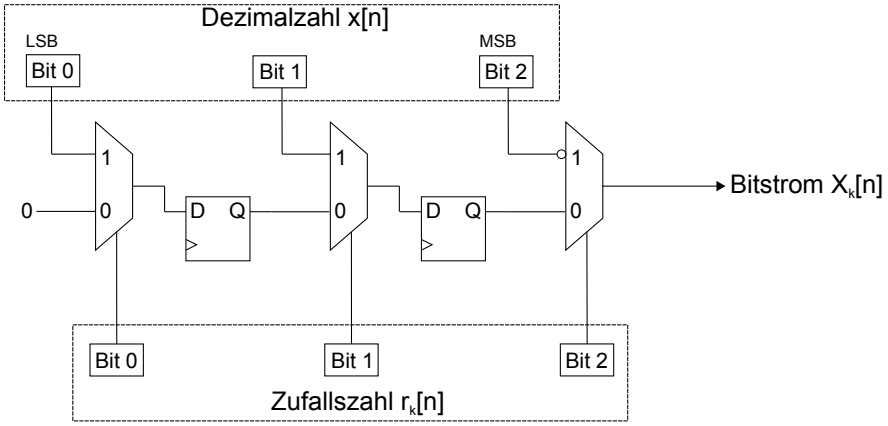

Abbildung 3.25: Modifizierte Codierung mit Bitslice-Technik

Die Runlängenverteilung bei der modifizierten Codierung entspricht wieder annähernd der Bernoulli-Verteilung. Das Verhalten der beiden vorgeschlagenen Varianten in Kombination mit der tanh-Approximation wird in Abb. 3.26 gezeigt.

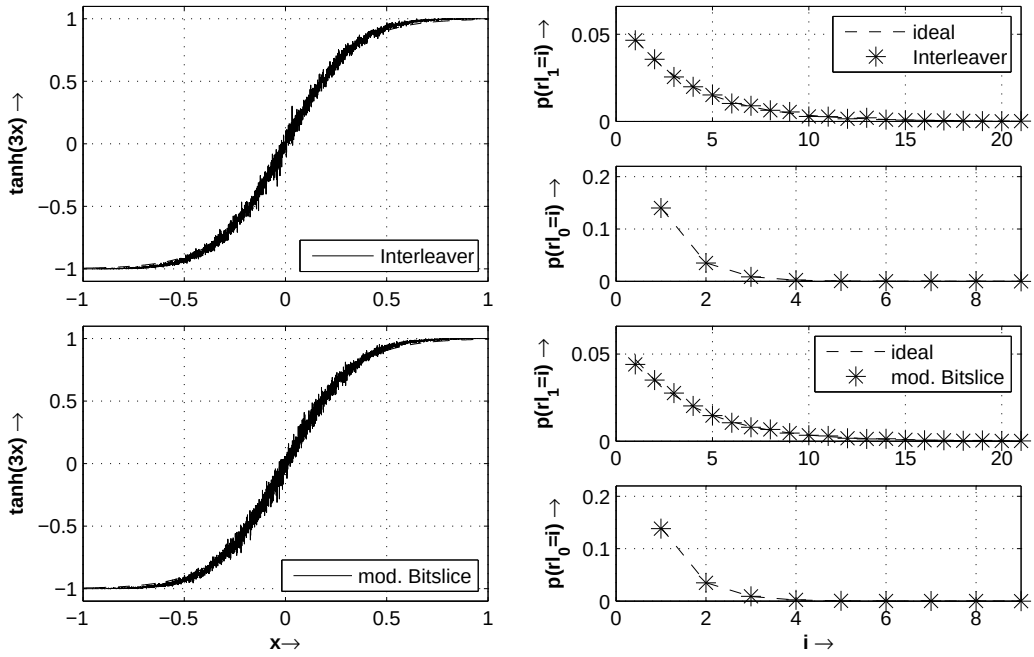

Abbildung 3.26: Korrekte Approximation bei modifizierter Bitslice-Technik

3.5.2 Bitstream-Modulator

Jeavons et al. schlugen in [24] eine der Bitslice-Technik ähnliche Codierung vor. Die Struktur entspricht der modifizierten Bitslice-Codierung aus Abb. 3.25, anstatt des Multiplexers wird das Modul aus Abb. 3.27 eingesetzt.

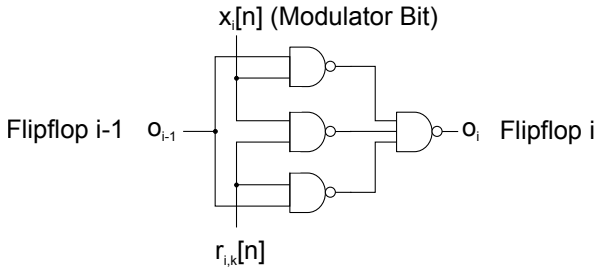

Abbildung 3.27: Bitstream-Modulator-Modul

Als Modulator-Bit wird die Binärstelle x_i definiert. Durch Betrachtung der Wahrheitstabelle wird deutlich, wie das Modul funktioniert.

x_i	o_{i-1}	r_i	o_i	x_i	o_{i-1}	r_i	o_i
0	0	0	0	1	0	0	0
0	0	1	0	1	0	1	1
0	1	0	0	1	1	0	1
0	1	1	1	1	1	1	1

Tabelle 3.3: Wahrheitstabelle des Bitstream-Modulator-Moduls

Wenn das Modulator-Bit $x_i = 0$ ist, dann realisiert das Modul eine UND-Verknüpfung zwischen der vorherigen Stufe und der binären Zufallsfolge r_i. Ist das Modulator-Bit $x_i = 1$, dann wird eine ODER-Verknüpfung realisiert. Daraus folgt mit der Wahrscheinlichkeit einer Eins in der vorherigen Stufe $p_{o_{i-1}}$ für die Wahrscheinlichkeit einer Eins in der betrachteten Stufe p_{o_i}:

$$p_{o_i} = \begin{cases} x_0/2, & i = 0 \\ p_{o_{i-1}}/2 + x_i/2, & i > 0 \end{cases} \tag{3.37}$$

Die durch den Bitstrom abgebildete Wahrscheinlichkeit p_X ist gleich der Wahrscheinlichkeit der letzten Stufe $p_{o_{N-1}}$. Die Überlegungen zu Gl. (3.37) führen unmittelbar zu folgendem Polynomring:

$$p_X = \frac{x_{N-1}}{2} + \sum_{i=N-2}^{0} \frac{x_i}{2} \cdot \left(\frac{1}{2}\right)^{N-1-i} \tag{3.38}$$

Gl. (3.38) ist nichts anderes als die kanonische Form von Gl. (3.36), für $N = 3$ ergibt sich beispielsweise erwartungsgemäß $p_X = \frac{x_2}{2} + \frac{x_1}{2} \cdot \frac{1}{2} + \frac{x_0}{2} \cdot \frac{1}{2^2} = \frac{x_2}{2} + \frac{x_1}{4} + \frac{x_0}{8}$.

Das Prinzip ist demzufolge genau das Gleiche wie bei der Bitslice-Technik, durch den Einsatz der Bitstream-Modulator-Module anstelle der Multiplexer sind die Bitströme jedoch nicht identisch.

3.5.3 Weighted Binary Sequence Generator

Eine besondere Technik der Codierung entwickelten Gupta et al. [30]. Sie verwarfen die Forderung der statistischen Unabhängigkeit der Zufallszahlen und entwickelten einen Codierer, der mit nur einem Schieberegister einen *rauschfreien* Bitstrom erzeugt. Ausgangspunkt ist erneut das Offset-Binary-Format. Der Weighted Binary Sequence Generator (WBSG) genannte rauschfreie Codierer ist in Abb. 3.28 dargestellt.

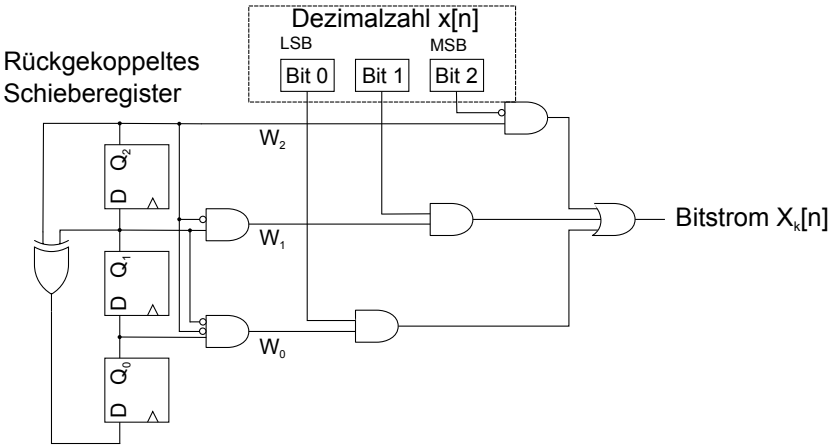

Abbildung 3.28: Struktur eines Weighted Binary Sequence Generators

Die Bitstromlänge muss hier stets gleich der Periode des Schieberegisters sein. Im gezeigten Fall ist $L = 7$. Tab. 3.4 veranschaulicht die Funktionsweise des Codierers. Durch die UND-Verknüpfungen der Register Q_i wird gewährleistet, dass stets nur ein Bit der Dezimalzahl in den Bitstrom geschaltet wird. Das negierte MSB wird 4-mal geschaltet, die anderen Bits entsprechend 2- bzw. 1-mal. Die Gewichtung gemäß Tab. 3.2 ist hier gegeben. Durch die kurze Bitstromlänge entsteht ein Fehler, der sich mit steigender Länge verringert.

Genau wie bei der Bitslice-Technik liegt bei dem WBSG eine verfälschte Runlängenverteilung vor. Die Verschiebung der benachbarten Bitfolgen ist zu gering.

k	0	1	2	3	4	5	6	Einsen
Q_2	1	0	1	1	1	0	0	
Q_1	0	1	1	1	0	0	1	
Q_0	1	1	1	0	0	1	0	
W_2	1	0	1	1	1	0	0	4
W_1	0	1	0	0	0	0	1	2
W_0	0	0	0	0	0	1	0	1
X	$\overline{x}_2$	x_1	$\overline{x}_2$	$\overline{x}_2$	$\overline{x}_2$	x_0	x_1	

Tabelle 3.4: Rauschfreie Codierung mit dem WBSG

3.5.4 Plessmann-Methode

Bei der Plessmann-Methode werden N verschiedene N-Bit breite Schieberegister mit unterschiedlichen Startwerten konkateniert, sodass sich für eine Bitlänge $L = 2^N - 1$ eine rauschfreie Codierung mit korrekter Runlängenverteilung ergibt [31]. Ein Beispiel zeigt Tab. 3.5. Dort erzeugen drei konkatenierte Schieberegister alle 7 Werte bis auf den Wert 0.

Startwert	Bitfolge						
100	**1**	0	1	1	1	0	0
111	0	0	**1**	0	1	1	1
010	1	1	1	0	0	**1**	0
Zufallszahl	-0.75	-1	-0.25	0.25	0.75	-0.5	0.5

Tabelle 3.5: Ideal verteilte Zufallszahlen

Um Zufallszahlen zu erzeugen, die jeden Wert bis auf 0 pro Periode nur einmal annehmen, können nicht beliebige Startwerte genutzt werden. Deshalb wurde nach passenden Startwerten gesucht. Abb. 3.29 zeigt den Ablauf des Suchverfahrens. Die Folge eines beliebigen Startwerts wird als Referenzfolge verwendet. Die Kreuzkorrelation zu allen anderen Folgen ist stets maximal, wenn der Verschiebungsindex der Korrelation gleich der Verschiebung der beiden Folgen ist. Diese werden gespeichert, sodass ein definierter minimaler Abstand zweier Folgen eine zu starke Korrelation verhindert. Aus jenen Folgen, die den Mindestabstand aufweisen, werden zufällig N Folgen zu einer Zufallsfolge konkateniert.

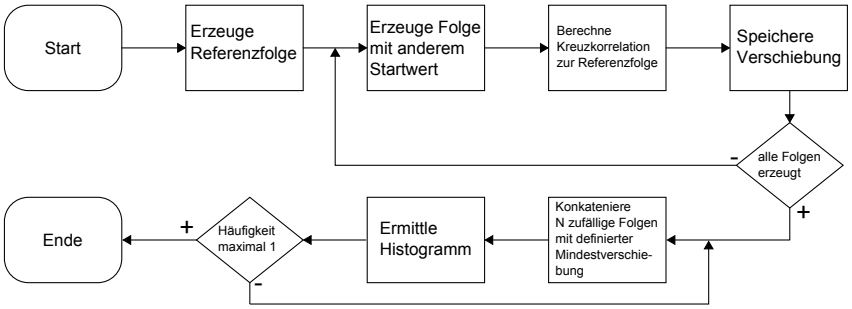

Abbildung 3.29: Flussdiagramm des Verfahrens zur Findung von Zufallszahlen für die Plessmann-Methode

Das Histogramm wird ermittelt und auf das maximale Aufkommen der Werte geprüft. Kommt jeder Wert höchstens einmal vor, so wird die Konstellation gespeichert. Mit einer solchen Konstellation kann die angestrebte rauschfreie Codierung durchgeführt werden. Dazu wird die Komparatortechnik genutzt.

Abb. 3.30 zeigt die Approximation der tanh-Funktion. Da die Runlängenverteilung näherungsweise einer Bernoulli-Sequenz gleicht, ist der Fehler bei der Approximation gering.

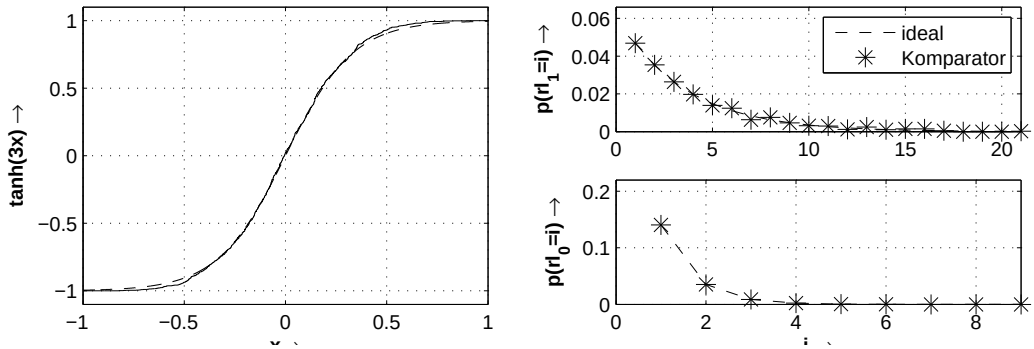

Abbildung 3.30: Rauschfreie Approximation durch Komparator-Codierung mit ideal gleichverteilten Zufallszahlen

3.6 Gegenüberstellung der Rechentechniken

Mit den Kenntnissen über die SC-Komponenten kann ein erster Vergleich zur konventionellen Zahlendarstellung vollzogen werden. Tab. 3.6 fasst die wichtigen Aspekte zusammen.

	SC	Festkomma	rauschfreies SC
Wertebereich	limitiert auf ± 1	erweiterbar	limitiert auf ± 1
Genauigkeit	gering	fix durch Bitbreite	zumeist hoch
Störsicherheit	robust	anfällig	robust
Verzögerung	hoch	zumeist niedrig	hoch
Operationen	wenige	viele	wenige
Hardwareaufwand	sehr gering	z.T. hoch	sehr gering

Tabelle 3.6: Vergleich der Rechentechniken

Ein schwerwiegender Nachteil bei der stochastischen Rechentechnik ist der begrenzte Wertebereich. Mit den bekannten Komponenten ist es nicht möglich, Zahlen betragsmäßig größer als Eins zu verarbeiten. Das ist jedoch notwendig, damit potentere Algorithmen als der Hérault-Jutten-Algorithmus implementiert werden können. Denn bei solchen konvergieren die Entmischungskoeffizienten stets gegen Werte außerhalb des SC-Wertebereichs.

Ein weiterer Nachteil ist die Genauigkeit der Zahlendarstellung bei SC. Bei Festkommazahlen bestimmt die Bitbreite die Genauigkeit, bei SC ist stets eine gewisse Varianz bei der Codierung vorhanden. Auch wichtig ist die Verstärkung der Varianz durch arithmetische Operationen. Hier schafft das vorgestellte rauschfreie SC Abhilfe, wobei dessen Funktionalität in komplexeren Anordnungen mit Rückkopplung im nächsten Kapitel getestet wird.

Der Informationsgehalt der einzelnen Bits eines Bitstroms ist gleich. Daraus folgt eine hohe Störsicherheit, denn das Umkippen eines oder mehrerer Bits eines Stromes hat nur geringfügige Auswirkungen auf Operationen oder Approximationen. Bei der Festkommazahl hingegen kann es durch nur einen Bitfehler beim Vorzeichenbit zu starkem Fehlverhalten eines Systems kommen.

Um die Auflösung bei SC zu verdoppeln muss die Länge des Bitstromes verdoppelt werden. Das führt zu einer signifikanten Verzögerung und entsprechend zu einer um ein Vielfaches längeren Rechenzeit im Vergleich zur Berechnung mit Festkommazahlen.

Auch gibt es Nachteile bei arithmetischen Operationen. Gerade die mittelnde Addition und Subtraktion sind in Hinblick auf das zu kleineren Werten steigende Rauschen suboptimal. Die vorgeschlagene Kaskadierung der Gain-Funktion wirkt der Mittelung entgegen, führt aber zu einem höheren Hardwareaufwand.

Der insgesamt jedoch deutlich geringere Hardwareaufwand ist eines der größten Vorteile der stochastischen Rechentechnik.

In diesem Kapitel wurde die stochastische Rechentechnik vorgestellt. Stärken und Schwächen wurden aufgezeigt, Lösungsansätze diskutiert. Die einzelnen Komponenten wurden

getestet und funktionieren. Wie sich die Vor- und Nachteile der Rechentechnik auf einen Lernalgorithmus auswirken, soll im nächsten Kapitel geprüft werden.

4 Blind Source Separation mittels Stochastic Computing

In den vorangegangenen Kapiteln wurden die relevanten Grundlagen zu Blind Source Separation und Stochastic Computing diskutiert. Der Hérault-Jutten-Algorithmus wurde getestet, die Vor- und Nachteile der stochastischen Rechentechnik wurden dargestellt. Die Vorteile, der geringe Hardwareaufwand und die Störsicherheit, sind für die Funktionalität von BSS-Algorithmen mit SC-Komponenten irrelevant. Denn der Hardwareaufwand ist erst bei der Implementierung wichtig und die Störsicherheit ist ausschließlich bei der Betrachtung des Spezialfalls eines fehlerbehafteten Systems relevant. Manche Nachteile hingegen wirken sich signifikant auf die Eignung von SC aus. Besonders die Begrenzung des Wertebereichs wiegt schwer. Sie verhindert die SC-Umsetzung von BSS-Algorithmen, deren Koeffizienten diesen Wertebereich verlassen. Diese bieten allgemein jedoch ein stabileres und schnelleres Konvergenzverhalten im Vergleich zum Hérault-Jutten-Algorithmus für $N \geq 3$ Signale (vgl. Abschnitt 2.3.4). Für die Bewertung der Eignung von SC für BSS sollen dementsprechend zu Beginn dieses Kapitels die Auswirkungen des begrenzten Wertebereichs analysiert werden. Anschließend wird das Verhalten des Hérault-Jutten-Algorithmus mit SC-Komponenten getestet. Abschließend wird das Verhalten des Algorithmus mit rauschfreiem Stochastic Computing betrachtet.

4.1 Auswirkungen des begrenzten Wertebereichs

In Abschnitt 2.3.4 wurde gezeigt, dass der Hérault-Jutten-Algorithmus lediglich für $N = 2$ Quellsignale zuverlässig konvergiert. Der Cichocki-Unbehauen- sowie der EASI-Algorithmus dagegen konvergieren auch bei einer höheren Zahl an Quellsignalen zuverlässig.

Zur Analyse der Auswirkungen des begrenzten Wertebereichs soll nun kurz der Cichocki-Unbehauen-Algorithmus betrachtet werden. Die Lernregel des Algorithmus lautet:

$$\Delta \mathbf{W}[n] = \mu \left(\mathbf{\Lambda} - \mathbf{g}_1(\mathbf{W}[n]\mathbf{x}[n]) \cdot \mathbf{g}_2(\mathbf{W}[n]\mathbf{x}[n])^T \right) \mathbf{W}[n] \tag{4.1}$$

Im Gegensatz zum Hérault-Jutten-Algorithmus werden hier stets $N \cdot N$ Koeffizienten adaptiert, die Entmischungsmatrix entspricht demzufolge der in Kap. 2 allgemein betrachteten

Matrix. Abb. 4.1 zeigt beispielhaft den Verlauf der Entmischungskoeffizienten des Cichocki-Unbehauen-Algorithmus für $N = 2$ Signale. Zu sehen sind die Ergebnisse einer Floating-Point-Simulation. Als Funktionen verwendet wurden $g_1(y) = tanh(10y)$ und $g_2(y) = y$.

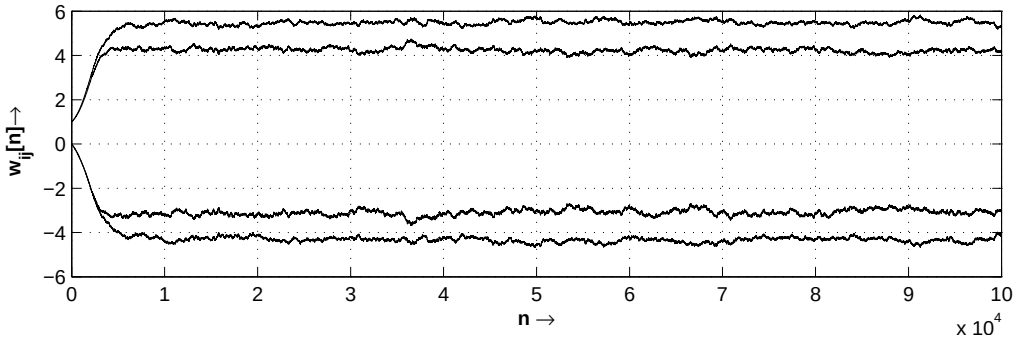

Abbildung 4.1: Verlauf der Entmischungskoeffizienten des Cichocki-Unbehauen-Algorithmus bei $N = 2$ Quellsignalen

Die Koeffizienten verlassen den Wertebereich der stochastischen Rechentechnik. Dies ist kein Sonderfall. Für $N = 2, 3, 4, 5$ Quellsignale wurden jeweils 1000 Durchläufe mit zufälligen Mischungsmatrizen durchgeführt. Sowohl für subgaußförmige als auch für supergaußförmige Quellsignale konvergierte *immer* mindestens ein Koeffizient gegen einen Wert außerhalb des Wertebereichs. Dieses Problem lösten Hori et al. für ihre hybride FPGA-Implementierung des Algorithmus mit stochastischer Rechentechnik [13]. Sie führten eine Adaptionsgewichtung $\alpha = 0.5$ ein, mit der alle Koeffizienten der Zeile i multipliziert werden, wenn einer der Koeffizienten $w_{ij}[n] = \pm 1$ ist. Zusätzlich wird dann die i-te Zeile der Skalierungsmatrix $\mathbf{\Lambda}$, eine Diagonalmatrix mit den Elementen λ_{ii}, mit α gewichtet. Der Verlauf der Adaption nach diesem Verfahren ist in Abb. 4.2 für das gleiche Beispiel wie in Abb. 4.1 zu sehen. In den ersten ca. 10000 Samples wird 4-mal eine Gewichtung durchgeführt. Danach konvergieren die Koeffizienten aufgrund der zusätzlichen Gewichtung der Skalierungsmatrix, ohne in die Begrenzung zu gelangen. Im Vergleich zum unmodifizierten Algorithmus dauert in diesem speziellen Fall die Adaption nur deutlich länger. Denn bereits nach etwas mehr als 5000 Samples sind die Koeffizienten bei der unmodifizierten Variante adaptiert. Die Konvergenz ist hier aber auch bei dem modifizierten Algorithmus gegeben.

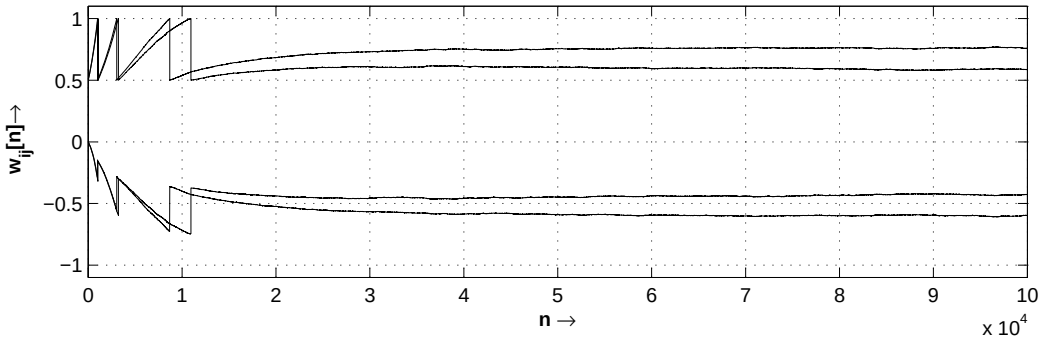

Abbildung 4.2: Verlauf der Entmischungskoeffizienten des Cichocki-Unbehauen-Algorithmus mit Adaptionsgewichtung bei $N = 2$ Quellsignalen

Die Gewichtung hat jedoch häufig einen größeren Einfluss auf das Konvergenzverhalten als im obigen Beispiel. Abb. 4.3 zeigt das mittlere Konvergenzverhalten des Algorithmus für 2 subgaußförmige Quellsignale nach 1000 Durchläufen.

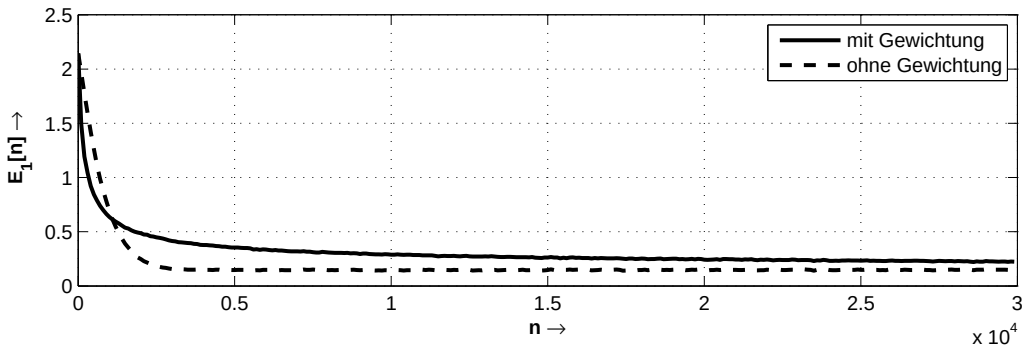

Abbildung 4.3: Vergleich des Cichocki-Unbehauen-Algorithmus mit dessen Modifikation

Die Konvergenzgeschwindigkeit ist im Mittel geringfügig niedriger. Betrachtet man jedoch alle Durchläufe separat, so zeigt sich das instabile Verhalten infolge der Gewichtung. Während der unmodifizierte Algorithmus stets konvergiert, verliert der modifizierte Algorithmus diese Stabilität. Das Doppelte des mittleren Separationsfehlers der letzten 5000 Samples aller 1000 Durchläufe wurde als Kriterium für eine erfolgreiche Separation gewählt. Bei 8.3% der Durchläufe wurde nach 30000 Samples dieses Kriterium nicht erfüllt. Variation des Konvergenzparameters und der Funktionen $g_1(y)$ und $g_2(y)$ führten zu keiner Stabilität. In Abb. 4.4 sind oben die 100 Durchläufe mit den höchsten Separationsfehlern des modifizierten Algorithmus zu sehen, unten das Konvergenzverhalten des normalen Algorithmus bei den gleichen Durchläufen. Dort ist die Stabilität stets gegeben. Abb. 4.5 zeigt die Verteilung des mittleren Separationsfehlers. Während der unmodifizierte Algorithmus in allen Durchläufen

mit einer geringen Streuung gegen den gleichen Wert konvergiert, ist bei der Fehlerverteilung der Modifikation ein maximaler Fehler von ca. 3 zu erkennen.

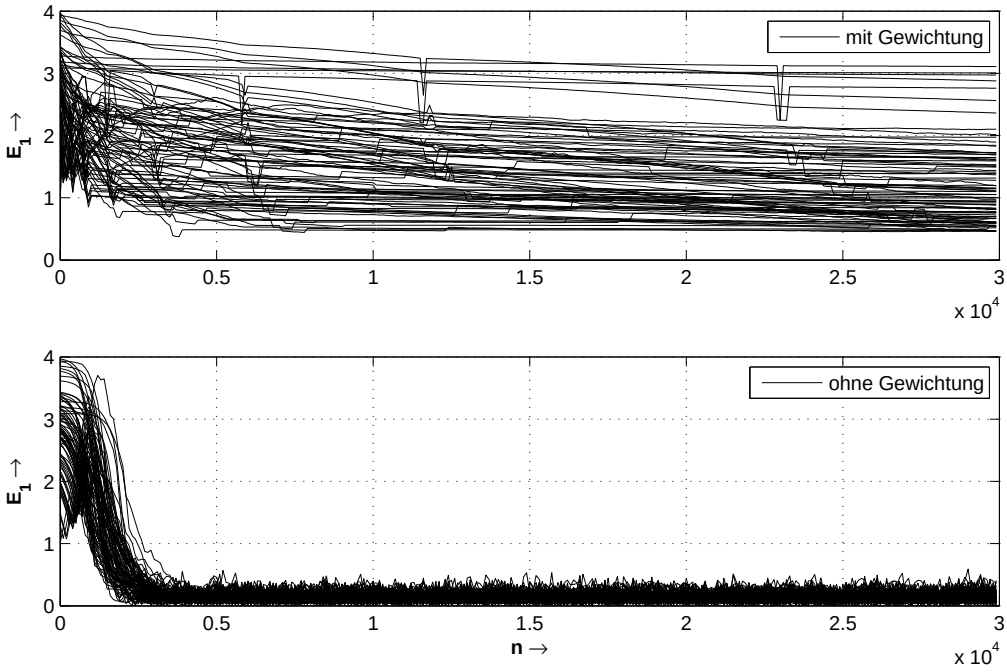

Abbildung 4.4: Vergleich des Cichocki-Unbehauen-Algorithmus mit dessen Modifikation; Darstellung der 100 Durchläufe mit den größten Separationsfehlern

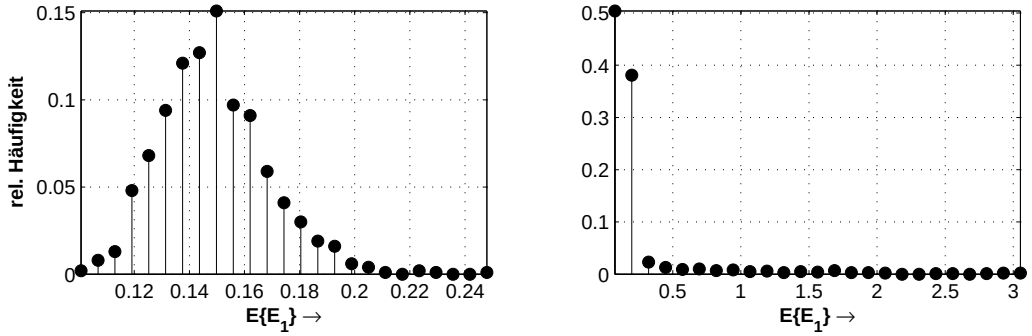

Abbildung 4.5: Mittlerer Separationsfehler; links Cichocki-Unbehauen-Algorithmus, rechts Modifikation

Die Simulation hat gezeigt, dass die Modifizierung des Cichocki-Unbehauen-Algorithmus bereits bei der Floating-Point-Simulation zu unzulässigem Verhalten führt. In [13] wurde die Modifikation für einen simplen Fall wie in Abb. 4.2 mit SC getestet. Zusätzlich wurden dort vereinfachend die nichtlinearen Funktionen mit Lookup-Tabellen und Festkommazahlen statt

mit Zustandsautomaten und stochastischer Rechentechnik realisiert. Desweiteren wurde die Funktionalität ausschließlich mit stationären Quellsignalen und einer einzigen Mischungsmatrix getestet. Die Probleme werden jedoch erst ersichtlich, wenn wie hier unterschiedliche Fälle untersucht werden.

Für den EASI-Algorithmus wurde eine ähnliche Modifizierung getestet. Auch hier ist bereits für zwei Quellsignale keine Stabilität mehr gewährleistet (vgl. Abb. 4.6). Bei 7% der 100 Durchläufe wurde nach 30000 Samples der mittlere Konvergenzfehler aller Durchläufe nicht erreicht. Auch hier konnte durch Variation der Parameter keine Stabilität erreicht werden.

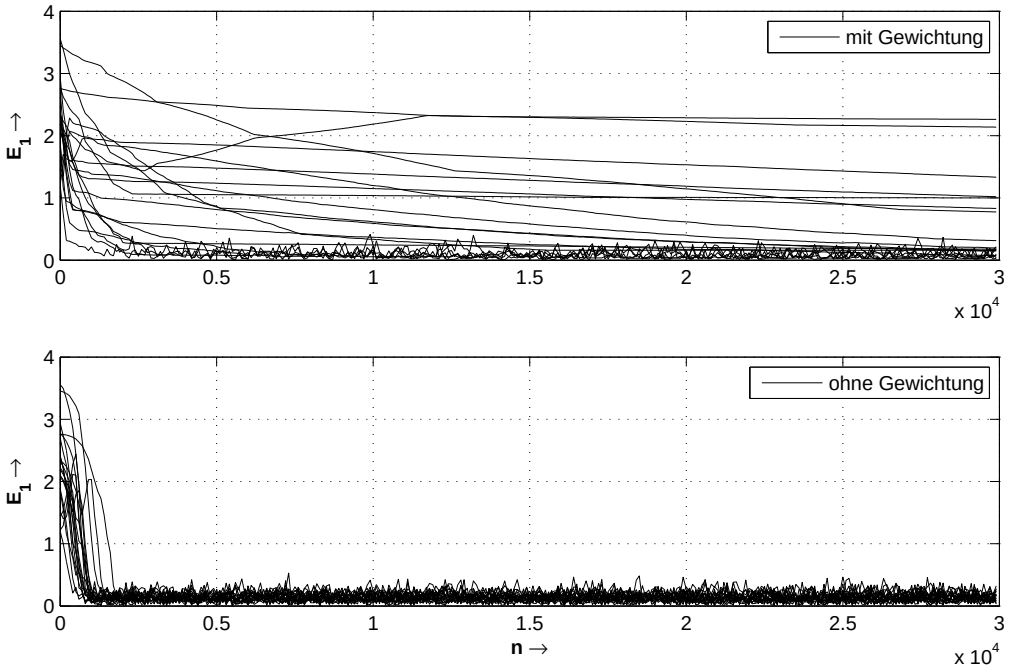

Abbildung 4.6: Vergleich des EASI-Algorithmus mit dessen Modifikation

Demzufolge sind die hier betrachteten und im Vergleich zum Hérault-Jutten-Algorithmus performanten Algorithmen mit den aktuell bekannten SC-Komponenten aufgrund des begrenzten Wertebereichs für eine Implementierung mit Stochastic Computing ungeeignet. Die in [8] und [9] dokumentierte Theorie basiert auf der Annahme, dass alle Dezimalwerte betragsmäßig kleiner oder gleich Eins sind. Die vorgestellten SC-Komponenten funktionieren nur unter dieser Annahme. Angenommen die Dezimalzahlen $-D \leq a \leq D$ und $-D \leq b \leq D$ mit $D > 1$ sollen in stochastische Bitströme codiert und multipliziert werden. Dann sind die Abbildung und die Codierung unproblematisch. Die Wahrscheinlichkeit der Dezimalzahl $-D \leq x \leq D$ kann mit $p_X = \frac{x+D}{2D}$ durchaus mit einem Bitstrom realisiert werden. Das Problem entsteht bei der Verarbeitung der Ströme. Für das Produkt $c = a \cdot b$ gilt:

$$2Dp_C - D = (2Dp_A - D) \cdot (2Dp_B - D) \tag{4.2}$$

Die Wahrscheinlichkeiten müssten mit dem Faktor D multipliziert werden. Eine einfache Umsetzung, mit Berücksichtigung von Überläufen, wenn $Dp_X > 1$ wird, ist nicht bekannt. Die Verarbeitung von Zahlen außerhalb des Wertebereichs ist in der stochastischen Rechentechnik nicht vorgesehen.

Das Besondere an dem Hérault-Jutten-Algorithmus ist, dass für den Fall von $N = 2$ Quellsignalen die optimalen Entmischungskoeffizienten stets $|w_{ij}| \leq 1$ sind. Unabhängig davon, dass die Stabilität dieses Algorithmus für $N \geq 2$ nicht gegeben ist, verlassen in korrekt konvergierenden Fällen die Koeffizienten auch dort teilweise den Wertebereich der stochastischen Rechentechnik. Deshalb wird im folgenden Abschnitt das Verhalten des Hérault-Jutten-Algorithmus mit SC-Komponenten für $N = 2$ Signale untersucht.

4.2 Simulation des Hérault-Jutten-Algorithmus

Das Blockschaltbild des Algorithmus mit SC-Komponenten ist in Abb. 4.7 zu sehen. Die D/S- und S/D-Blöcke stellen die Digital-Stochastik- und Stochastik-Digital-Umsetzer dar.

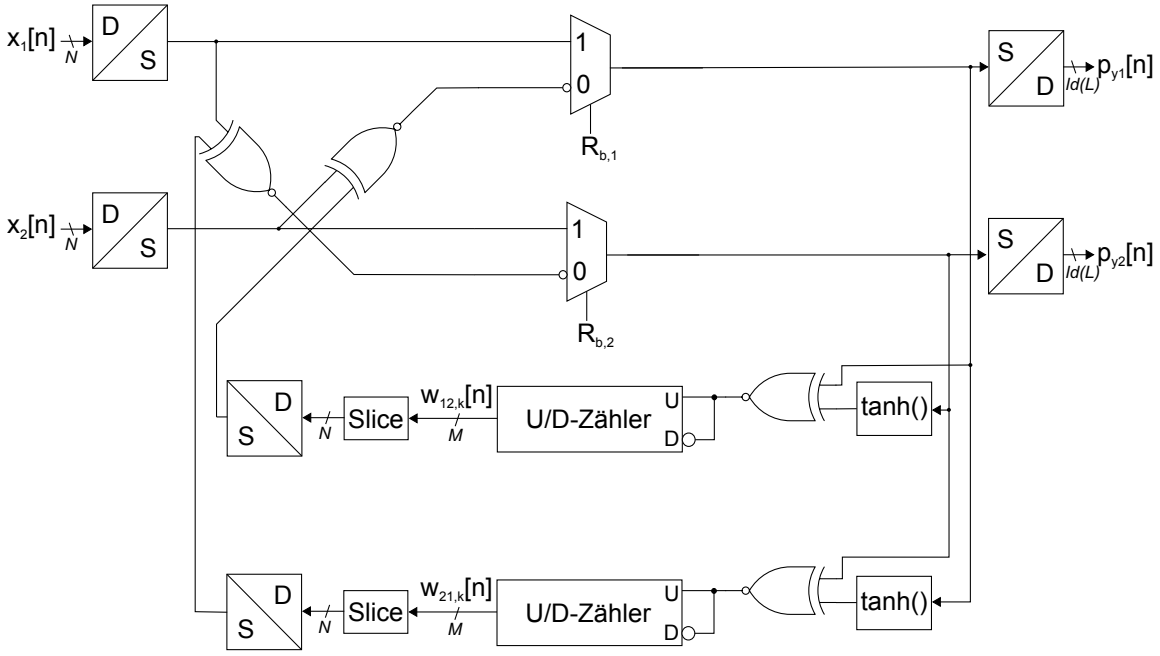

Abbildung 4.7: Blockschaltbild des Hérault-Jutten-Algorithmus mit SC-Komponenten

Eine Simulation mit der Software Matlab/Simulink war zu langsam. Deshalb wurden die Komponenten in C programmiert. Über mex-Dateien (Matlab executable) ermöglicht es die

Software Matlab, C-Programme aus der Matlab-Umgebung heraus zu starten [32]. Die Verläufe der Ausgangssignale sowie der Entmischungskoeffizienten werden in eine csv-Datei (Comma-separated values) gespeichert, die nach Beenden der Simulation in Matlab ausgelesen werden kann. Ein Simulationsergebnis für zwei stationäre Quellsignale ist in Abb. 4.8 dargestellt. Die Quellsignale, die Mischungsmatrix sowie die SC-Parameter werden in einem Matlab-Script definiert und beim C-Programmaufruf übergeben. Dadurch ist es möglich, ohne weiteres den Verlauf der SC-Simulation mit dem Verlauf der Floating-Point-Simulation zu vergleichen. In blau dargestellt sind die Verläufe der Floating-Point-Simulation. Die mittelnde Subtraktion sowie die sgn-Approximation durch die tanh-Funktion wurden dabei berücksichtigt.

Es ist deutlich zu sehen, dass die Verläufe sehr gut übereinstimmen. Durch die stochastische Rechentechnik weisen die Signale erwartungsgemäß eine vergleichsweise hohe Streuung auf. Die Konvergenz zu den optimalen Entmischungskoeffizienten ist auch gegeben. Dies war bei der SC-Implementierung desselben Algorithmus in [11] nicht der Fall. Hauptunterschied zu jener Implementierung ist die Verwendung einer anderen tanh-Approximation. Für die obige Simulation wurden 12-bit-Dezimalzahlen, 4095 Bit lange Bitströme und $tanh(8y)$ als Approximation der sgn-Funktion genutzt. Über die Bitbreite des Integrierers wurde der Konvergenzparameter auf $\mu = 2^{-6}$ festgelegt. Als Pseudozufallszahlengenerator wurde das 28-Bit-LFSR aus [12] genutzt, zur Codierung die Bitslice-Technik mit den Modulen von Jeavons.

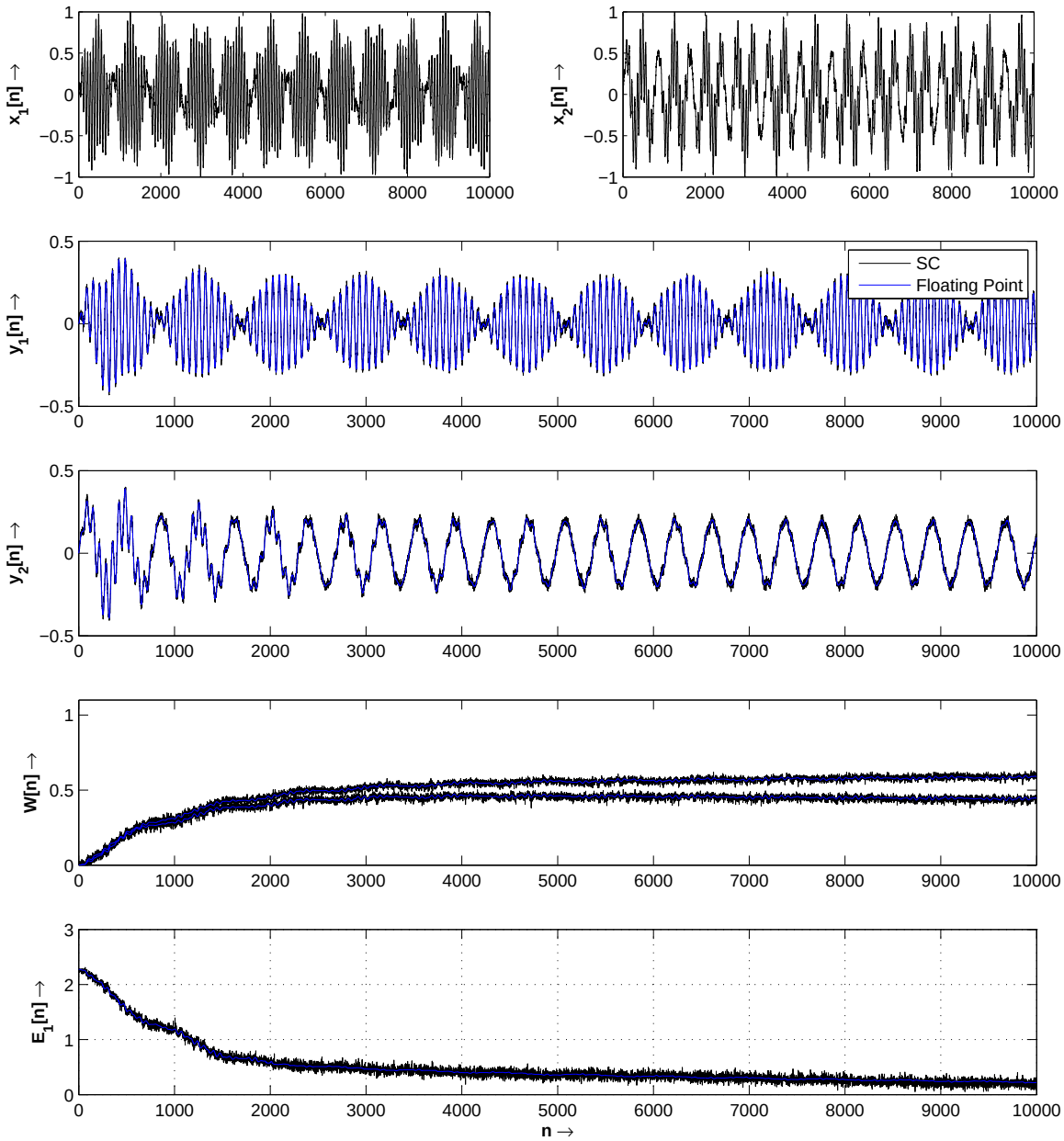

Abbildung 4.8: Verlauf des Hérault-Jutten-Algorithmus mit SC-Komponenten; schwarz SC-Simulation, blau Floating-Point-Simulation

Abb. 4.9 zeigt den mittleren Separationsfehler nach 1000 Durchläufen für subgaußförmige Signale. Der prinzipielle Verlauf ist annähernd gleich. Im Mittel verursacht die SC-Variante jedoch einen höheren Separationsfehler.

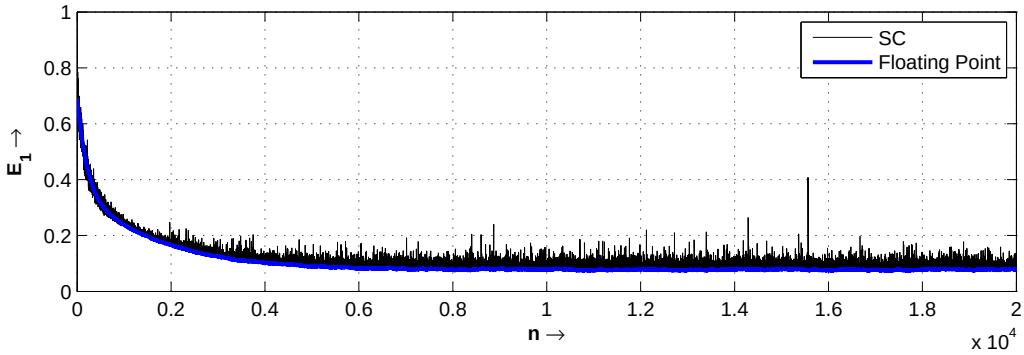

Abbildung 4.9: Separationsfehler des Hérault-Jutten-Algorithmus mit SC-Komponenten bei subgaußförmigen Quellsignalen

Abb. 4.10 verdeutlicht, dass auch die Stabilität durch Stochastic Computing nicht verloren geht. Die Verteilung der mittleren Separationsfehler ist nahezu gleich.

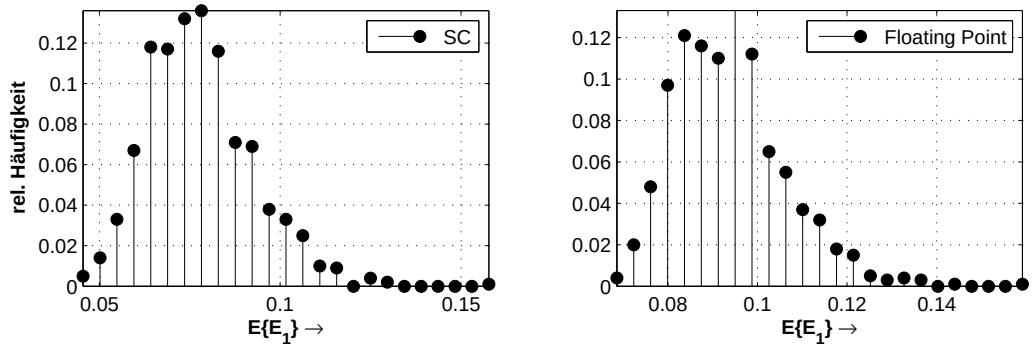

Abbildung 4.10: Mittlerer Separationsfehler; links mit Stochastic Computing, rechts Floating-Point-Simulation

Für supergaußförmige Signale ist die Approximation der sgn-Funktion mit $tanh(8y)$ zu ungenau. Die Aussteuerung ist zu gering, sodass der Zustandsautomat auf $n = 32$, also 64 Zustände, dimensioniert werden muss. Abb. 4.11 zeigt den mittleren Separationsfehler für supergaußförmige Signale. Auch hier ist das Konvergenzverhalten nahezu gleich. Der mittlere Separationsfehler ist bei der SC-Variante erneut höher.

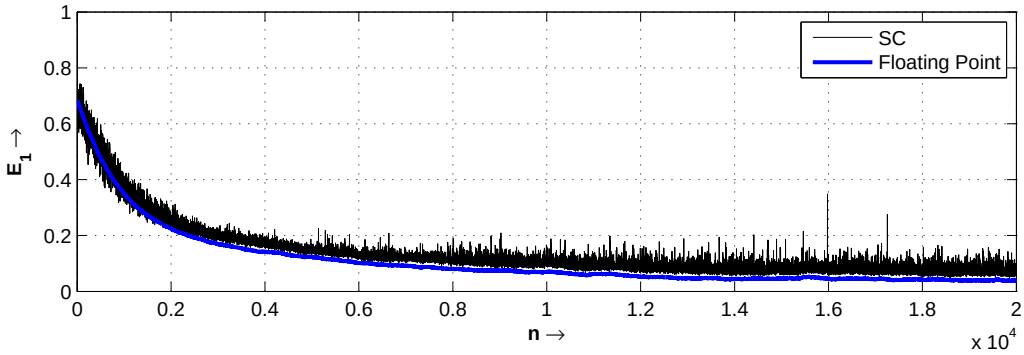

Abbildung 4.11: Separationsfehler des Hérault-Jutten-Algorithmus mit SC-Komponenten bei supergaußförmigen Quellsignalen

Die Frage ist, ob sich die mittelnde Subtraktion negativ auf das Konvergenzverhalten auswirkt. Zur Klärung wird eine modifizierte stochastische Subtraktion vorgeschlagen. Abb. 4.12 zeigt das Blockschaltbild des Subtrahierers. Er besteht aus dem Multiplexer kaskadiert mit der Gain-Funktion und dem Interleaver aus Abschnitt 3.3.2.

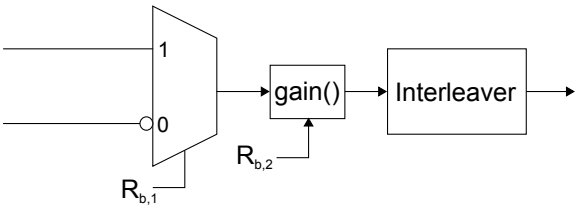

Abbildung 4.12: Blockschaltbild des modifizierten stochastischen Subtrahierers

Die gleichen Mischungsmatrizen und Quellsignale wurden für die Simulation mit dem modifizierten Subtrahierer genutzt. Das Ergebnis ist in Abb. 4.13 zu sehen. Der mittlere Separationsfehler ist sogar höher, da sich die Varianz durch die Gain-Funktion erhöht. Eine deutliche Verbesserung, die den höheren Hardwareaufwand rechtfertigt, ist nicht zu sehen.

Der modifizierte Subtrahierer kann die Qualität des Algorithmus mit SC-Komponenten nicht nachhaltig aufwerten. Die Funktionalität ist aber weiterhin gegeben, sodass diese Modifikation bei mehrfach kaskadierten Addierern oder Subtrahierern die resultierende Mittelung ausgleichen könnte.

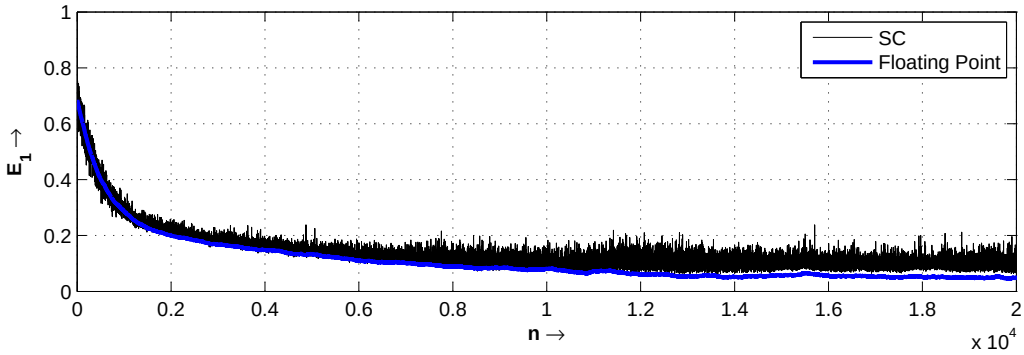

Abbildung 4.13: Separationsfehler des Hérault-Jutten-Algorithmus mit SC-Komponenten bei supergaußförmigen Quellsignalen (modifizierte stochastische Subtraktion)

Die Simulationen haben gezeigt, dass der Hérault-Jutten-Algorithmus mit der stochastischen Rechentechnik ein nahezu gleiches Konvergenzverhalten aufweisen kann. Bei schwach ausgesteuerten Signalen verschlechtert die Varianz der Rechentechnik das Konvergenzverhalten, die mittelnde Subtraktion hat dabei aber keinen merklich negativen Einfluss.

4.3 Verhalten bei rauschfreier Codierung

Um den Nachteil der Varianz bei der stochastischen Rechentechnik zu eliminieren, wurde im Rahmen dieser Thesis auch untersucht, wie sich der Hérault-Jutten-Algorithmus mit L-periodischen Zufallszahlen verhält (vgl. Abschnitt 3.5.4). Bei diesem Ansatz wird eine wichtige Grundregel der stochastischen Rechentechnik ignoriert. Die Zufallszahlen weisen nun Muster auf, sie wiederholen sich für jeden Abtastwert, sie sind nicht mehr über einen Zeitraum $\gg L$ unkorreliert. Dieser Ansatz wurde erstmals in Verbindung mit dem WBSG dokumentiert [30]. Miao et al. nutzten diesen WBSG in [33], um einen FIR-Filter (Finite Impulse Response) zu entwickeln. Mehrere Codierer wurden dort parallel eingesetzt und deren Bitströme vermischt. Es kamen jedoch keine Zustandsautomaten zum Einsatz, sodass die Runlängenverteilung keine Schwierigkeiten bereitete. Deren Ziel war es, das Rauschen zu minimieren, nicht zu eliminieren. Ansonsten wurde in der Literatur nichts bezüglich rauschfreier Codierung gefunden.

In Abschnitt 3.5.3 wurde erwähnt, dass der WBSG keine Sequenzen mit Bernoulli-Verteilung erzeugt. Hier wurden Zufallszahlen entwickelt, sodass Dezimalwerte rauschfrei in Bitströme mit einer Runlängenverteilung codiert werden können, die von Zustandsautomaten korrekt verarbeitet werden können.

Zum Test der rauschfreien Codierung ändert sich an der Anordnung aus Abb. 4.7 nichts. Lediglich der Zufallszahlengenerator wird getauscht. Das Ergebnis der Simulation mit stationären Signalen ist in Abb. 4.14 zu sehen.

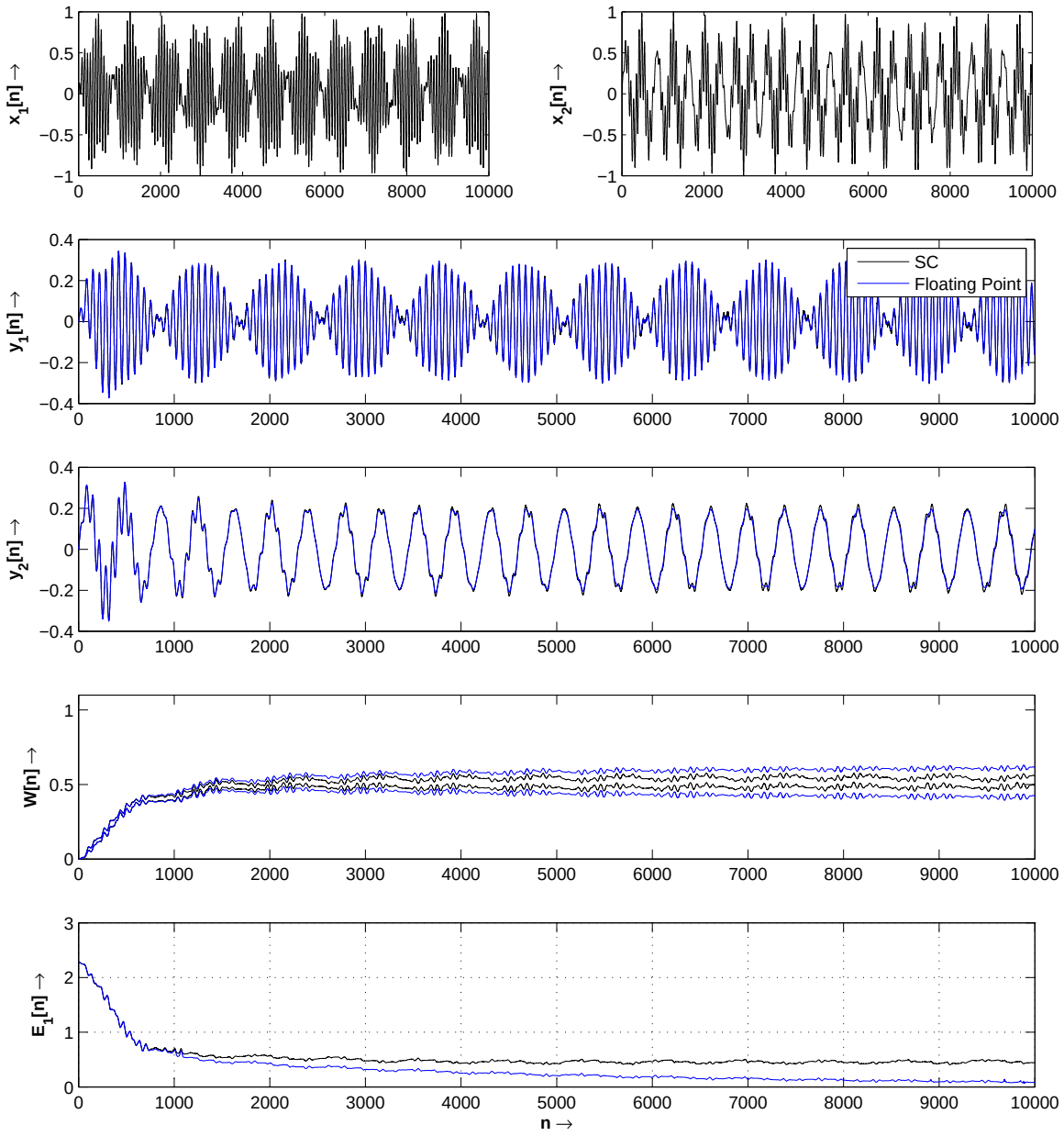

Abbildung 4.14: Verlauf des Hérault-Jutten-Algorithmus mit rauschfreier Codierung

Die Signale rauschen zwar nicht mehr, jedoch konvergieren die Entmischungskoeffizienten nicht gegen die optimalen Werte. Um die allgemeine Funktionalität der rauschfreien Codierung zu testen, wurden für eine feste Mischungsmatrix die optimalen Entmischungskoeffizi-

enten $w_{12,id}$ und $w_{21,id}$ ermittelt und, wie in Abb. 4.15 zu sehen, das Zusammenwirken der Arithmetik-Komponenten ohne Lernprozess geprüft.

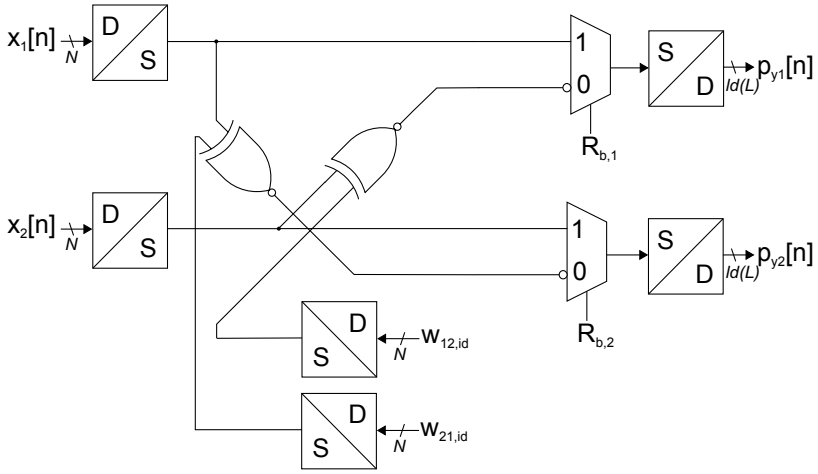

Abbildung 4.15: Blockschaltbild der Anordnung für den Test der rauschfreien Codierung

Das Ergebnis dieses Tests wird in Abb. 4.16 gezeigt. Die entmischten Signale entsprechen denen der Floating-Point-Simulation für optimale Koeffizienten.

Demzufolge entsteht das Problem durch den Lernprozess. Durch heuristisches Vorgehen konnte festgestellt werden, dass sich das Verhalten bei $g(y) = tanh(16y)$ statt der sonst verwendeten Funktion $g(y) = tanh(8y)$ der Floating-Point-Variante annähert. Bei einer Parametrierung des Zustandsautomaten für einen noch steileren Funktionsverlauf verschlechtert sich das Konvergenzverhalten im Vergleich zur Floating-Point-Variante wieder. Es scheint nicht mit der Runlängenverteilung zusammenzuhängen, da durch den Einsatz von Interleavern vor den nichtlinearen Funktion keinerlei Besserung folgt. Abb. 4.17 zeigt den Verlauf des Separationsfehlers gemittelt über 10 Durchläufe mit subgaußförmigen Signalen.

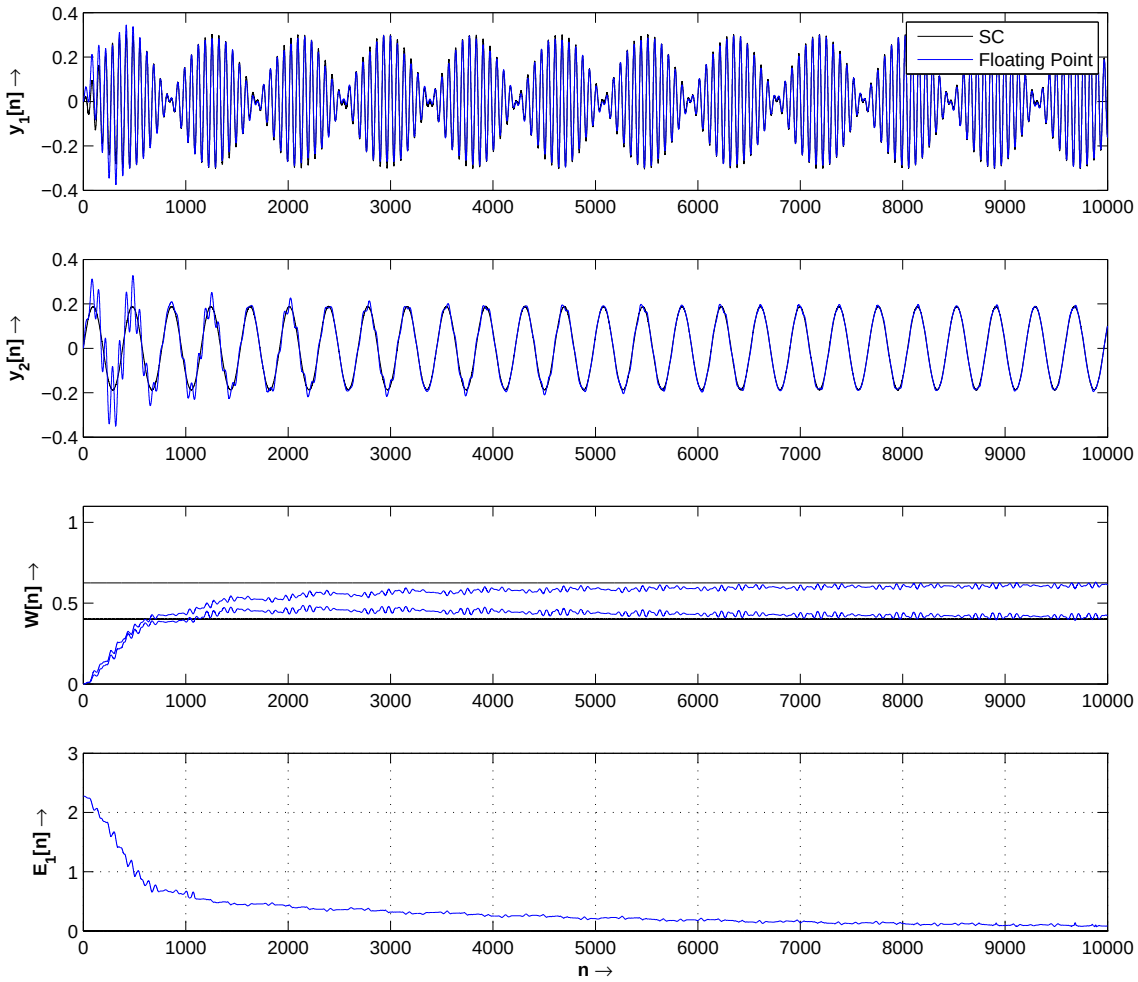

Abbildung 4.16: Test der rauschfreien Codierung mit optimalen Entmischungskoeffizienten

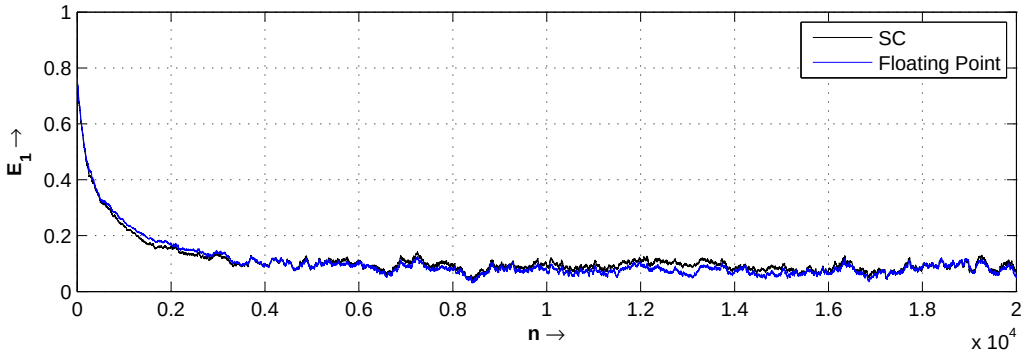

Abbildung 4.17: Separationsfehler mit rauschfreier Codierung

Der höhere Separationsfehler infolge des Rauschens kann also mit den hier vorgeschlage-

nen Zufallszahlen und einer Erhöhung der Steilheit der tanh-Approximation eliminiert werden.

Für supergaußförmige Quellsignale wurde allerdings keine funktionierende Konstellation gefunden. Ursache scheint die geringe Aussteuerung zu sein. Die Berechnung der Produkte vor den Integrierern sind dann das Problem. Wird in dem C-Programm statt des Zustandsautomaten die tanh-Funktion der mathematischen C-Bibliothek (math.h) verwendet und werden die Produkte mit Gleitkommazahlen berechnet, so ist die Funktionalität häufiger gegeben. Auch bei subgaußförmigen Signalen, die nicht voll ausgesteuert sind, treten diese Probleme auf.

Die rauschfreie Codierung weist demzufolge kein zum klassischen SC identisches Verhalten auf. Durch die Codierung kann ein neues, zur Zeit unkontrollierbares Fehlverhalten auftreten. Deshalb wird diese im Weiteren nicht mehr betrachtet.

In diesem Kapitel wurde gezeigt, dass lediglich der historisch erste Blind-Source-Separation-Algorithmus ohne Anpassungen mit Stochastic Computing realisiert werden kann. Durch den begrenzten Wertebereich müssen andere Algorithmen modifiziert werden. Es wurde gezeigt, dass diese Modifikationen das Konvergenzverhalten signifikant verschlechtern.

Die Funktionalität des Hérault-Jutten-Algorithmus mit SC-Komponenten wurde durch C-Simulationen verifiziert. Auch die rauschfreie Codierung wurde getestet. Sie funktioniert lediglich für subgaußförmige Signale zuverlässig. Die geringe Aussteuerung der supergaußförmigen Signale scheint Probleme zu bereiten, das Verhalten ist mit dem heutigen Wissensstand nicht beherrschbar.

5 FPGA-Implementierung

In diesem Kapitel wird beschrieben, wie der Hérault-Jutten-Algorithmus auf einem FPGA implementiert wurde. Zu Beginn wird auf die verwendete Hardware eingegangen. Anschließend wird das Syntheseergebnis der SC-Komponenten betrachtet. Zur VHDL-Beschreibung wurde das Xilinx ISE Webpack verwendet. Danach wird das implementierte Gesamtsystem vorgestellt, was eine Auswertung des Algorithmus von einem Computer aus ermöglicht. Zuletzt wird der Hardwareaufwand der SC-Implementierung mit dem einer Festkomma-Variante verglichen.

Implementiert wurde der Algorithmus für 12-Bit-Werte mit einer Bitstromlänge von 4095 und einem Konvergenzparameter $\mu \approx 2^{-7}$, sodass der Integrierer 20 Bit breit sein muss. Der Zustandsautomat der tanh-Approximation hat 16 Zustände, approximiert also wie in der C-Simulation die Funktion $tanh(8y)$. Die Funktionen sind so angeordnet, dass subgaußförmige Signale getrennt werden können.

5.1 Verwendete Hardware

Da der Algorithmus auf einem Spartan-6-FPGA realisiert wurde, soll zu Beginn dieses Abschnitts der allgemeine Aufbau von FPGAs an seinem Beispiel kurz beschrieben werden.

Die Logikschaltungen auf einem FPGA werden mit rekonfigurierbaren Logikblöcken (CLB) aufgebaut. Diese bestehen beim Spartan-6 aus jeweils zwei Slices. In einem Slice befinden sich 4 Look-Up-Tabellen (LUT) mit jeweils 6 Eingängen sowie 8 Flipflops und weiterer Logik. Mit den LUTs wird die kombinatorische Logik realisiert. Insgesamt 9112 LUTs stehen in dem Spartan-6-FPGA zur Verfügung. Drei verschiedene Typen an Slices sind vorhanden:

1. *SLICEM*: Deren LUTs können neben Logikfunktionen als 64-Bit Distributed RAM oder als 32-Bit- bzw. 16-Bit-Schieberegister genutzt werden (25% aller Slices)
2. *SLICEL*: Bieten die Funktionalität der SLICEMs bis auf die Speicher- und Schieberegister-Funktion (25% aller Slices)
3. *SLICEX*: Bieten die Funktionalität der SLICELs bis auf spezielle Logik-Optionen (50% aller Slices)

Die CLBs sind in Schaltmatrizen angeordnet und können flexibel programmabhängig miteinander verbunden (geroutet) werden. Neben den CLBs befinden sich zusätzlich festverdrahtete, eingebettete Ressourcen auf dem FPGA:

- Block RAM
- Hardwaremultiplizierer und -addierer
- Digital Clock Manager
- Takttreiber

Ein Block RAM ist ein Speicher, der fest auf dem FPGA platziert ist. Durch Nutzung dieses Speichers kann vermieden werden, dass die LUTs in Form von Distributed RAM als Speicherressourcen genutzt werden müssen. Der Spartan-6 bietet 576kBit an Block RAM. Durch Verwendung der vorgefertigten Arithmetik-Schaltungen können ressourcenintensive Operationen ohne Beanspruchung der Logikblöcke realisiert werden. Der Spartan-6 bietet 32 Arithmetik-Blöcke, die jeweils aus einem 18x18 Multiplizierer, Addierer und Akkumulator bestehen [34].

Für den späteren Vergleich des Hardwareaufwands der Implementierungen mit stochastischer Rechentechnik und mit Festkommazahlen sind diese beiden Ressourcen wichtig. Abb. 5.1 zeigt die geschilderte Struktur anhand eines Ausschnitts aus dem Programm PlanAhead der Firma Xilinx. Die einzelnen Ressourcen wurden gekennzeichnet.

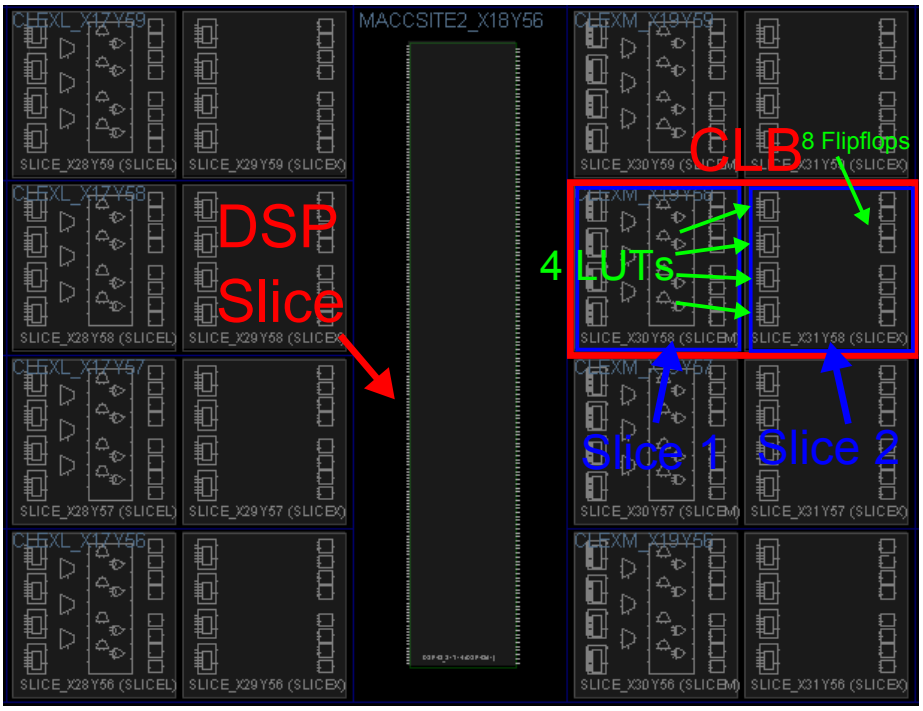

Abbildung 5.1: Ausschnitt des Spartan-6-Floorplans zur Veranschaulichung der FPGA-Struktur

Ein Spartan-6-FPGA XC6SLX16 ist auf dem Nexys3-Board der Firma Digilent verbaut. Auf diesem Board soll der Algorithmus implementiert werden. Abb. 5.2 zeigt jenes Board.

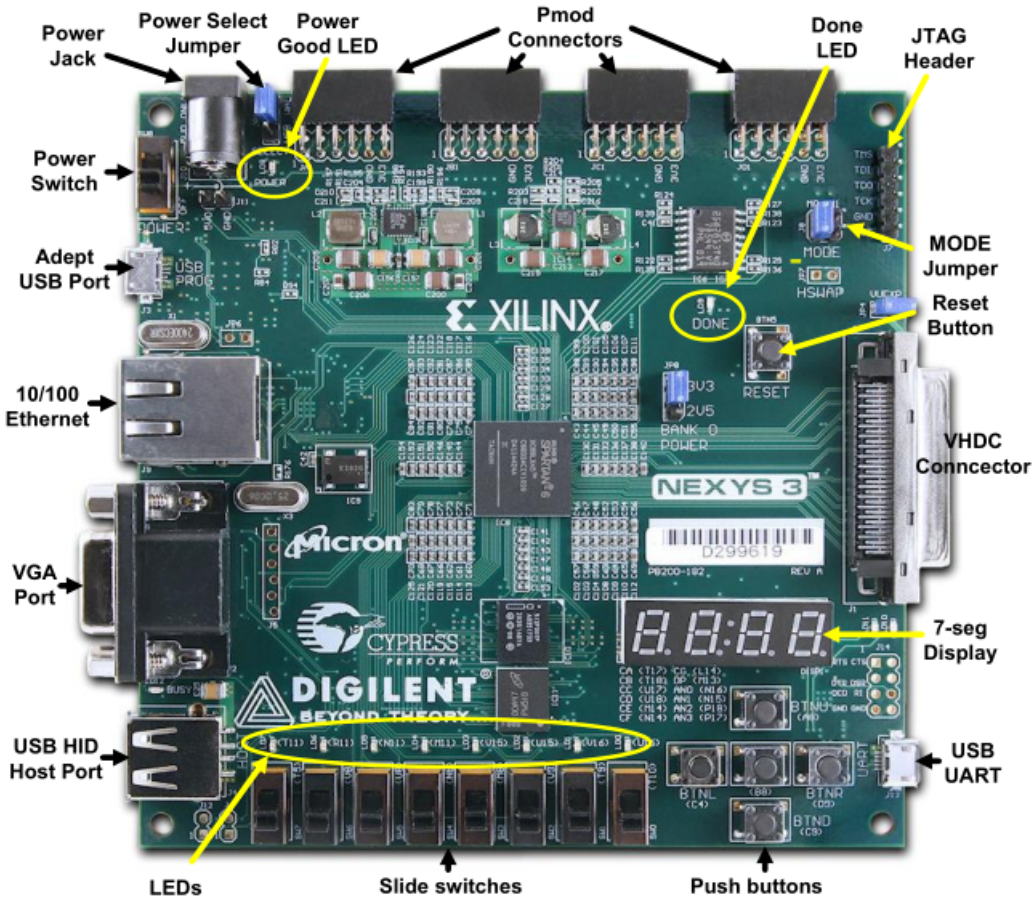

Abbildung 5.2: Nexys3-Board von Digilent [35]

Das Board verfügt über 16MB Cellular RAM von Micron, Ethernet-, USB- und UART-Schnittstellen sowie Pmod-Buchsen (Peripheral Modification) zur Erweiterung mit externer Peripherie.

Peripherie, um Testsignale vom PC aus zu empfangen, ist auf dem Board vorhanden. Der RAM-Baustein ermöglicht die Speicherung der relevanten Signale des Algorithmus zu jedem Abtastzeitpunkt.

Unter Berücksichtigung der FPGA-Struktur kann nun die Beschreibung der einzelnen SC-Komponenten diskutiert werden.

5.2 Hardwareaufwand der Komponenten

In diesem Abschnitt sollen die Syntheseergebnisse der einzelnen Komponenten betrachtet werden, um deren Hardwareaufwand einschätzen zu können. Zum Teil soll durch einen Vergleich des Hardwareaufwands von verschiedenen Implementierungen die beste Variante ausgewählt werden (z.B. bei den unterschiedlichen Codierungsverfahren). Als Metrik für den Hardwareaufwand wird die Anzahl an verwendeten LUTs genutzt [36].

5.2.1 Zufallszahlengenerator

Der verwendete Pseudozufallszahlengenerator aus [12] besteht aus einem rückgekoppelten 28-Bit-Schieberegister und XNOR- und XOR-Verknüpfungen, mit denen insgesamt 59 binäre Zufallsfolgen erzeugt werden. Für den zu implementierenden Algorithmus reichen 50 Folgen aus, sodass 51 LUTs benötigt werden. Die Nutzung von 50 separaten 28-Bit-Schieberegistern würde ohne weitere Maßnahmen 1400 LUTs beanspruchen. Durch Nutzung der SLICEMs als Schieberegister kann der Aufwand minimiert werden, jedoch bleibt er deutlich höher als bei der Implementierung nur eines rückgekoppelten Schieberegisters.

5.2.2 Codierer

In Abschnitt 3.5 wurden verschiedene Codierungsverfahren vorgestellt. Sowohl die Komparator-Technik als auch die modifizierte Bitslice-Technik und der Bitstream-Modulator wurden zum Vergleich implementiert. Die letzten beiden Techniken unterscheiden sich lediglich durch das Modul, welches das Ausgangsbit einer Stufe erzeugt. Abb. 5.3 zeigt eine entsprechende Stufe. Das Modul M ist bei der Bitslice-Technik ein Multiplexer, bei dem Bitstream-Modulator das Modulator-Modul (vgl. Abschnitt 3.5.2).

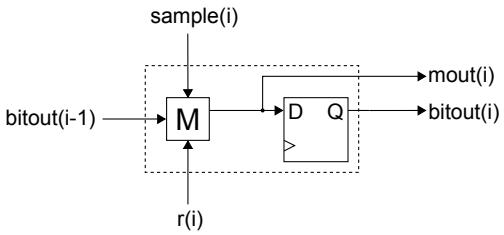

Abbildung 5.3: Bitslice-Stufe

Pro Modul wird stets eine LUT benötigt. Die Syntheseergebnisse sind in Abb. 5.4 zu sehen. Links ist der Multiplexer zu sehen, rechts das Ergebnis des Modulator-Moduls. Betrachtet

man die Wahrheitstabelle von dieser Implementierung, so ist zu erkennen, dass es sich um die NAND-Logik aus Abschnitt 3.5.2 handelt.

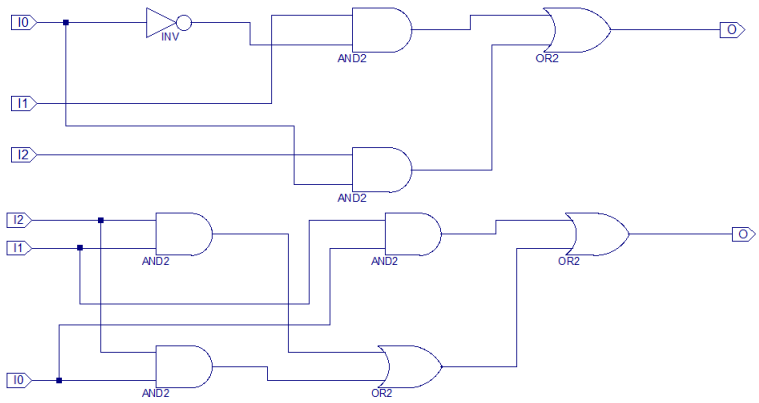

Abbildung 5.4: Syntheseergebnisse für das Modul; Multiplexer oben (I0=r(i), I1=sample(i), I2=bitout(i-1)), Modulator-Modul unten (I0=r(i), I1=bitout(i-1), I2=sample(i))

Für einen N-Bit-Codierer werden N dieser Stufen instantiiert, sodass N LUTs und $N-1$ Flipflops verbraucht werden. Für die letzte Stufe (Stufe des MSBs) wird der ungespeicherte Ausgang genutzt.

Aber auch die Komparator-Technik kann mit wenig LUTs implementiert werden. Abb. 5.5 zeigt, wie zwei 2-Bit-Werte mit einer einzigen LUT verglichen werden können. Werte mit einer höheren Bitbreite werden durch das Synthese-Tool in 2-Bit-Paare aufgeteilt, sodass für eine N-Bit-Komparation $N-1$ LUTs benötigt werden (für $N > 9$).

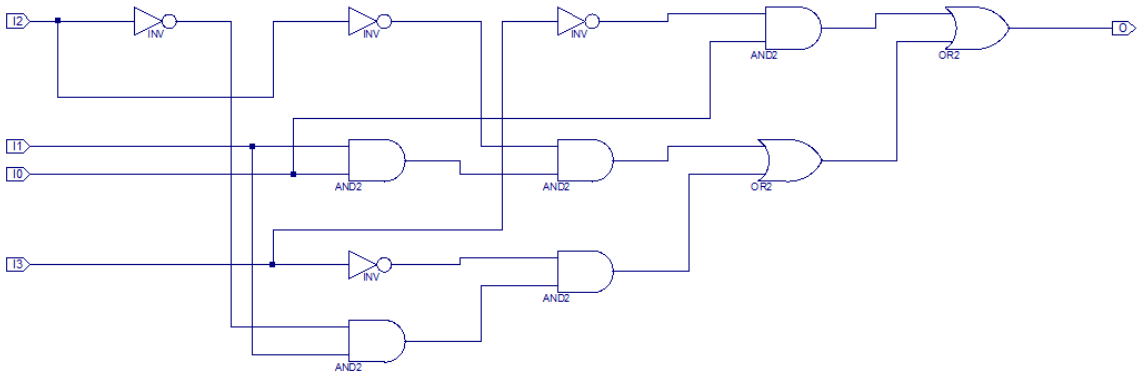

Abbildung 5.5: Syntheseergebnis für den Vergleich von 2-Bit-Werten (I0=r(0), I1=r(1), I2=sample(1), I3=samples(0)); Vergleich sample < r

Ein geringerer Hardwareaufwand im Vergleich zur klassischen Komparator-Technik ergibt sich bei der Verwendung eines FPGAs durch die Bitslice-Technik also nicht. Im Gegenteil,

der Aufwand ist sogar um eine LUT höher. Die Pipelining-Struktur der Bitslice-Technik ist jedoch der Struktur des Komparators vorzuziehen, weshalb der Bitstream-Modulator nach Jeavons implementiert wurde.

5.2.3 Decodierer

Für den Algorithmus werden zwei Decodierer für die zwei Ausgangssignale benötigt. Für $L = 4095$ ergibt sich ein 12-Bit-Zähler. Dieser benötigt eine spezielle Reset- und Set-Logik. Denn beim Übergang von zwei aufeinanderfolgenden Bitströmen muss der Zähler auf 1 gesetzt werden, wenn das Eingangsbit gleich 1 ist. Daraus resultiert ein Bedarf an 15 LUTs pro Decodierer.

5.2.4 Arithmetische Operationen

Die arithmetischen Operationen, die Multiplikation per XNOR sowie die Subtraktion per Multiplexer, benötigen jeweils nur eine LUT. Bei dem Algorithmus werden 4 Multiplikationen und 2 Subtraktionen durchgeführt, die mit 6 LUTs realisiert werden. Abb. 5.6 zeigt die LUT, die für die Multiplikation synthetisiert wird. Es ist die bekannte Form des XNORs als Sum of Products.

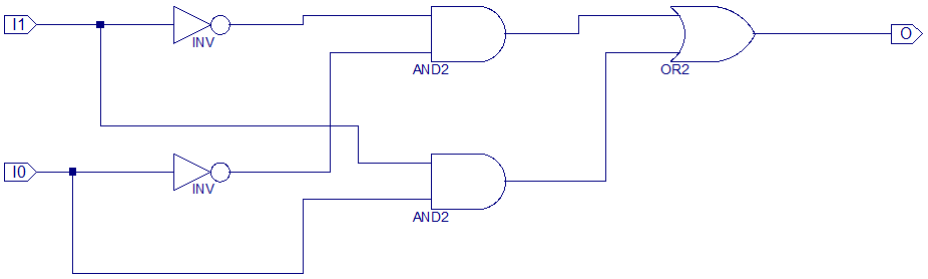

Abbildung 5.6: Syntheseergebnis für die stochastische Multiplikation

5.2.5 Integrierer

In [12] wurde folgende Struktur für den Integrierer in Bitslice-Technik vorgeschlagen.

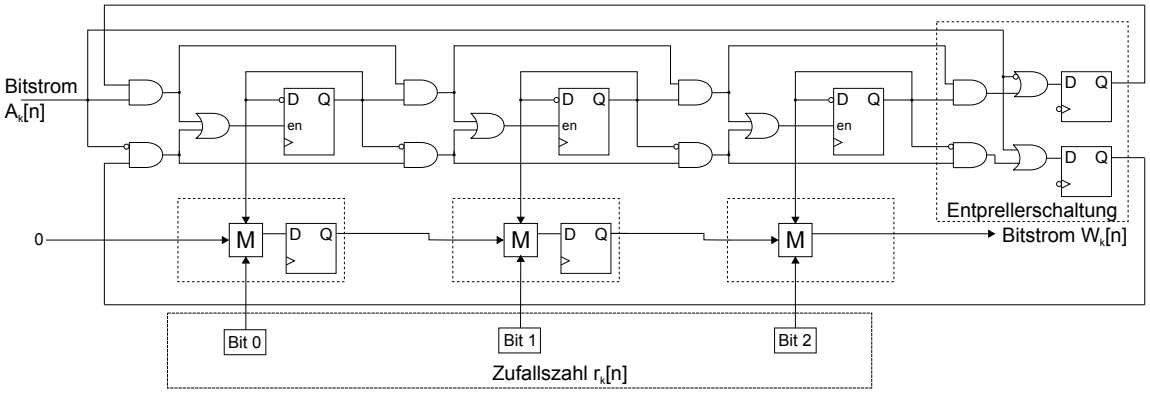

Abbildung 5.7: Struktur eines stochastischen 3-Bit-Integrierers nach [12]

Die Entprellerschaltung war nicht weiter dokumentiert und wurde im Rahmen dieser Thesis entwickelt. Durch diese Anordnung kann auf Komparatoren für den Über- und Unterlaufschutz verzichtet werden. Allerdings kosten die logischen Verknüpfungen stets eine LUT, weil die UND-Verknüpfungen durchgeschliffen werden, sodass für einen 20-Bit breiten Integrierer 72 LUTs benötigt werden. Desweiteren muss die Entprellerschaltung mit fallender Flanke betrieben werden, was zu Problemen bei der Einhaltung der Timing-Constraints führen kann. Die Implementierung eines 20-Bit-Integrierers mit Komparatoren für Über- und Unterlaufschutz beansprucht nur 46 LUTs und ist deswegen der oben abgebildeten Variante vorzuziehen. Die zwei Integrierer verbrauchen trotzdem insgesamt die meisten Hardwareressourcen aller SC-Komponenten.

5.2.6 Nichtlineare Funktion

Der Zustandsautomat für die tanh-Approximation benötigt für die 16 Zustände 4 Register. Die Zustände werden mit dem Gray-Code encodiert. Für die Übergangs- und Ausgangslogik werden 5 LUTs verbraucht.

Insgesamt werden nach der Synthese 213 LUTs für den Hérault-Jutten-Algorithmus verbraucht. Durch weitere Optimierungen beim Mapping wird der Aufwand noch geringfügig verringert (siehe Abschnitt 5.4).

5.3 Gesamtsystem

In diesem Abschnitt wird das Gesamtsystem, das auf dem FPGA implementiert wurde, vorgestellt und getestet.

5.3.1 Konzeption

Das System ist so zu realisieren, dass der Hérault-Jutten-Algorithmus vom PC aus mittels eines MATLAB-Scripts getestet werden kann. Die Mischsignale sollen auf dem PC erzeugt und zu dem FPGA übertragen werden. Die Verarbeitung muss durch das System auf dem FPGA gesteuert werden. Die Ausgangssignale sowie die Entmischungskoeffizienten sollen zur Auswertung an den Computer gesendet werden. Folgende Aufgaben ergeben sich:

1. Empfangen und Speichern der 2 Mischsignale
2. Verarbeiten der Mischsignale
3. Speichern der Ausgangssignale und der Entmischungskoeffizienten
4. Senden der Signale und Koeffizienten an den Computer

Ziel ist es dabei, die Implementierung des Algorithmus zu verifizieren. Dementsprechend reicht es aus, einen einfachen UART zu implementieren und zum Senden und Empfangen der Daten zu verwenden. Zum Speichern der Daten kann der externe RAM-Baustein genutzt werden, der sich wie ein SRAM-Baustein ansteuern lässt [37]. Für das Empfangen, Speichern, Verarbeiten und Senden muss eine Steuereinheit implementiert werden, welche die Steuerung der Abläufe übernimmt.

5.3.2 Aufbau

Das entwickelte System ist in Abb. 5.8 zu sehen. Es besteht aus einem UART-Core, der die Daten empfängt und sendet. Über die Steuereinheit wird das Speichern und Auslesen der Daten im RAM-Baustein gesteuert.

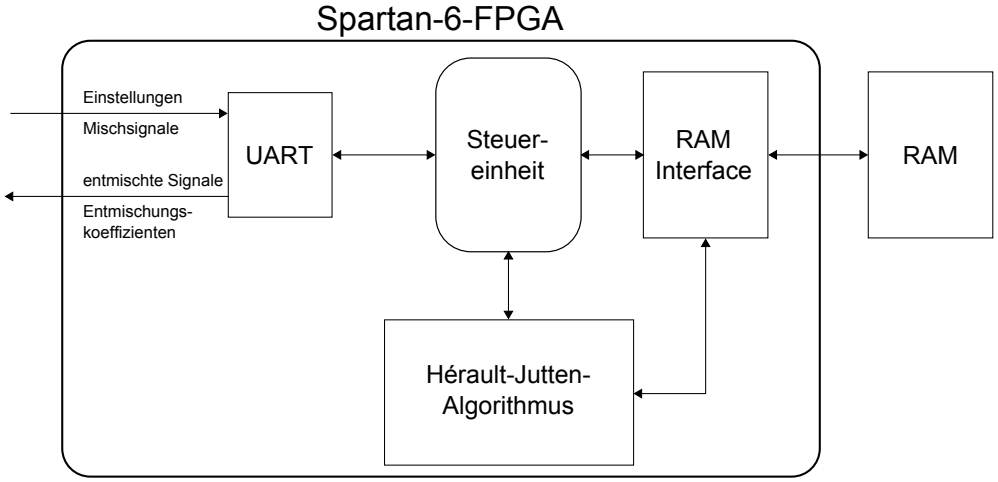

Abbildung 5.8: Hardware-Architektur des Gesamtsystems

Für die Kommunikation mit dem Speicher wurde ein RAM-Interface geschrieben. Der Algorithmus ist in einer separaten Komponente geschrieben und wird auch von der Steuereinheit gestartet und gestoppt. Die Steuereinheit selbst besteht aus drei Komponenten, die entsprechend die Steuerung des Empfangens, der Verarbeitung sowie des Sendens übernehmen. Abb. 5.9 zeigt die Anordnung der geschriebenen VHDL-Komponenten.

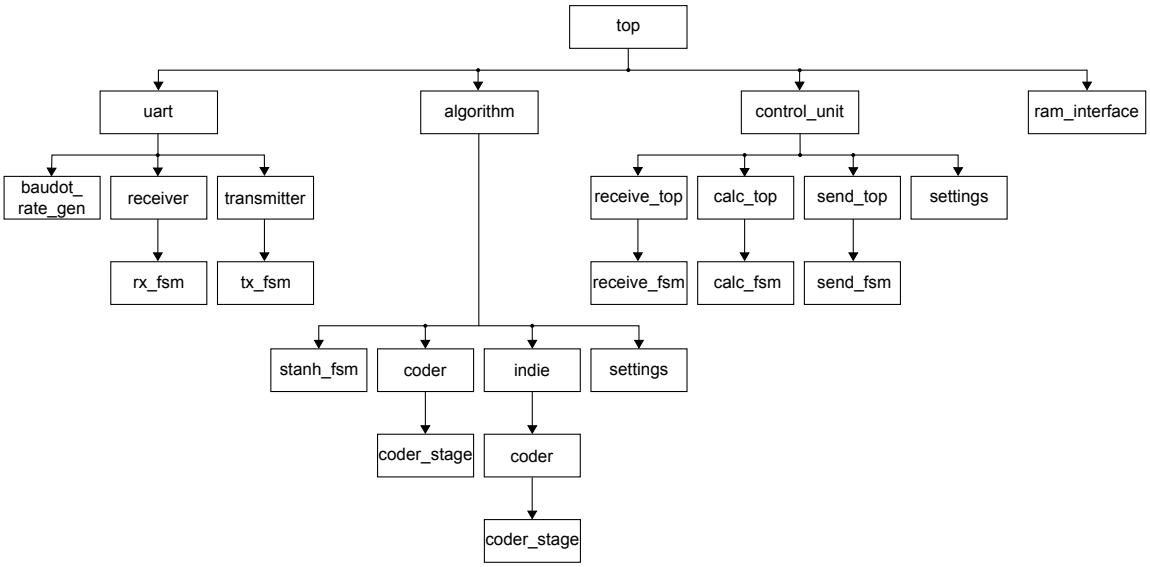

Abbildung 5.9: Hierarchie des Gesamtsystems

Im Folgenden soll das System getestet werden.

5.3.3 Test des Systems

In diesem Abschnitt soll zu Beginn die Funktionalität der Hauptkomponenten mit Hilfe von Testbenches und Testimplementierungen verifiziert werden. Zum Schluss dieses Abschnitts, nachdem das Ergebnis einer Übertragung die korrekte Arbeitsweise des Gesamtsystems gezeigt hat, wird der Hardwareaufwand der implementierten SC-Variante mit dem einer Festkomma-Variante des Algorithmus verglichen.

RAM-Interface

Für das Schreiben und Lesen des RAM-Bausteins wurde ein Interface geschrieben (vgl. Abb. 5.10). Der Speicher des Nexys3-Boards wurde hierbei als asynchrones SRAM mit einer Zugriffszeit von 70ns verwendet.

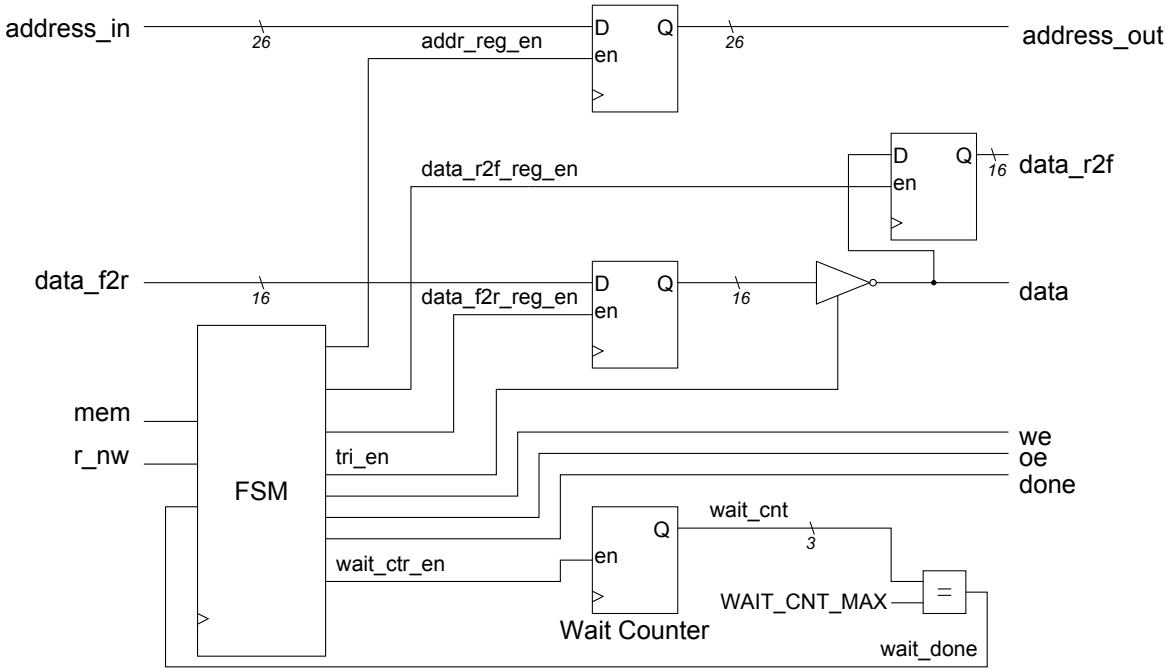

Abbildung 5.10: Blockschaltbild des RAM-Interfaces

Das Schema des Testsystems, welches ausschließlich die korrekte Funktionalität des RAM-Interfaces sicherstellen soll, wird in Abb. 5.11 gezeigt.

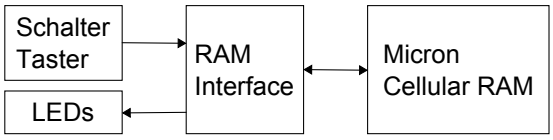

Abbildung 5.11: Schema des RAM-Interface-Tests

Bei diesem Test werden zu Beginn mehrere Werte in den RAM-Baustein geschrieben. Mit Hilfe der Taster und Schalter des Boards können die Daten ausgelesen und neue Daten geschrieben werden. Mit einer Testbench wurde das System geprüft.

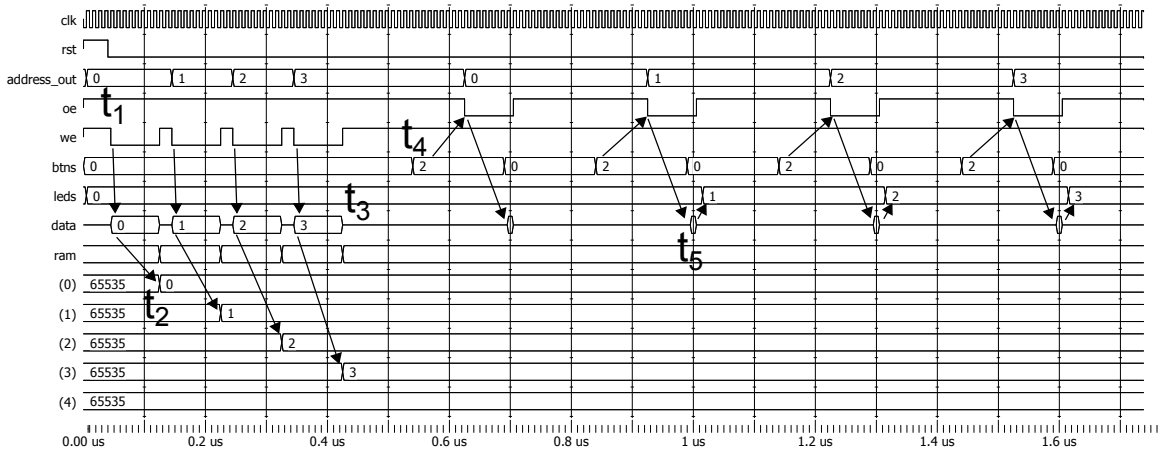

Abbildung 5.12: Testbench für den RAM-Interface-Test

Zum Zeitpunkt t_1 beginnt die Initialisierung des RAMs durch die Testkomponente. Vier Werte von 0 bis 3 werden in den Baustein geschrieben. Das RAM Interface wird aktiviert, setzt das low-aktive Write Enable we auf 0 und legt die Daten auf den Tri-State Treiber. Diese müssen mindestens 70 ns nach dem Nullsetzen von we anliegen, damit sie in dem RAM-Baustein gespeichert werden (t_2). Nach 3 weiteren Schreibvorgängen bis t_3 wird durch Drücken des Buttons $btns(1)$ ein Lesevorgang initiiert (t_4). Die Daten werden aus dem Speicher gelesen und auf den $leds$ ausgegeben (z.B. bei t_5).

Das System wurde auf dem FPGA implementiert und funktioniert fehlerfrei. Damit konnte die korrekte Funktionalität des RAM-Bausteins sowie des geschriebenen Interfaces verifiziert werden.

UART

Um die Funktionalität des geschriebenen UARTs zu testen, wurde ein Echoprogramm implementiert, das mit einem PC kommuniziert. Abb. 5.13 zeigt die Testbench zu diesem Programm.

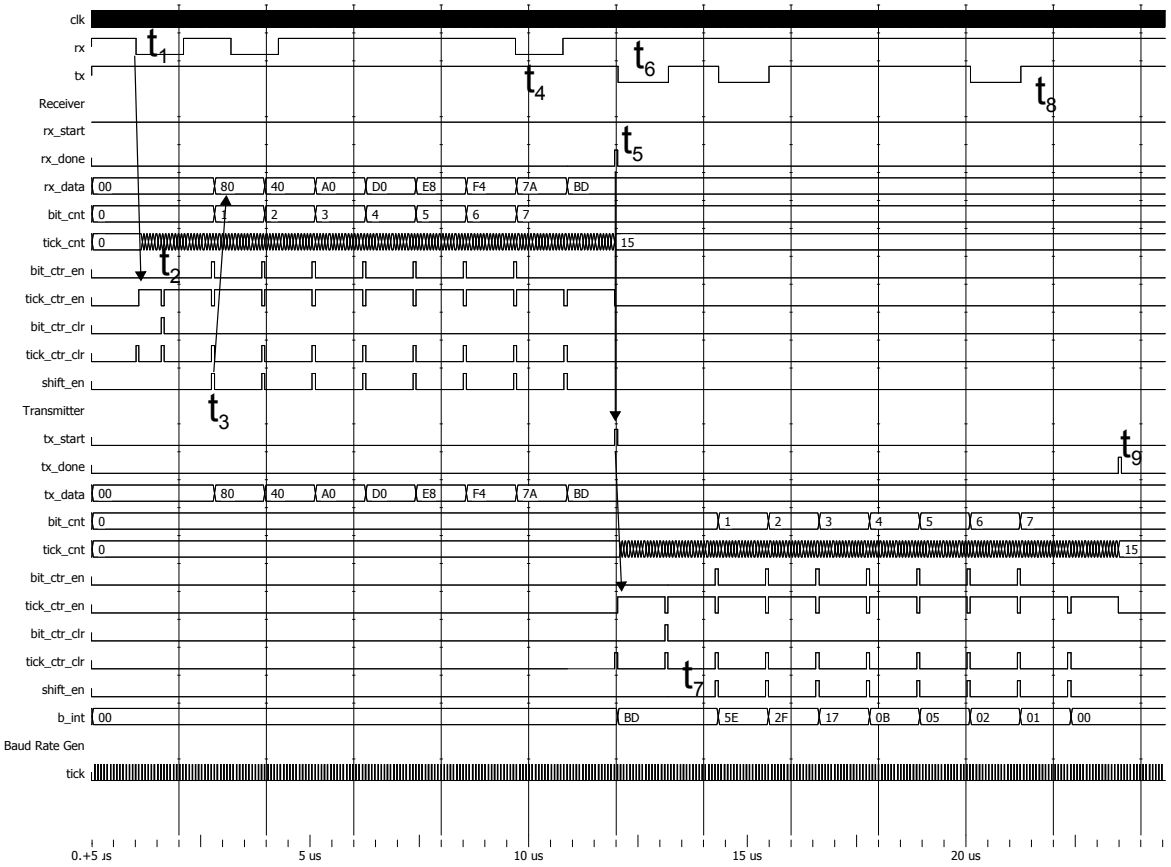

Abbildung 5.13: Testbench für den UART-Echo-Test

Das Steuersignal *rx_start* ist dauerhaft auf 1, zum Zeitpunkt t_1 wird die Empfangsleitung *rx* auf 0 gesetzt, was den Start einer Übertragung einleitet. Ein Symbol wird an den FPGA gesendet. Das erkennt der Receiver und startet den Tick Counter, der den Abtastzeitpunkt signalisiert. Zum Zeitpunkt t_2 sind 16 Ticks vergangen, sodass der Tickzähler zurückgesetzt wird. Das Startbit wurde dadurch berücksichtigt, nach weiteren 8 Ticks wird mit dem Speichern der Datenbits begonnen. Hierzu wird das Schieberegister aktiviert (t_3). Das erste Datenbit ist eine 1 und wird auf das MSB des Schieberegisters gespeichert und in den folgenden Takten zum LSB hingeschoben. Zum Zeitpunkt t_4 wird das letzte Datenbit, das MSB, übertragen und gespeichert. Anschließend wird das Statussignal *rx_done* gesetzt, was das Ende eines Symbolempfangs signalisiert (t_5). Da dieses Signal und das Empfangssymbol bei diesem Echoprogramm zum Transmitter durchgeschliffen wurden, beginnt unmittelbar die Übertragung des Symbols. Gestartet wird mit dem Startbit (t_6), anschließend werden ab t_7 alle empfangenen Bits sukzessive übertragen, bis zum Zeitpunkt t_8 das Stoppbit gesendet wird. Das Ende der Übertragung wird durch das Statussignal *tx_done* gekennzeichnet (t_9).

Die Simulation zeigt die korrekte Funktionalität des geschriebenen UARTs. Das empfangene Datenwort $0xBD$ wird korrekt zurückgesendet. Zur Hardware-Verifikation wurde das System auf dem FPGA implementiert und mit dem Terminal-Programm PUTTY erfolgreich getestet. Treiberseitig kann eine maximale Baudrate von $921600bit/s$ eingestellt werden, die bei diesem einfachen Echoprogramm fehlerfrei genutzt werden kann.

Um die Funktionalität des UARTs auch bei Burst-Daten, wie sie im Gesamtsystem vorkommen, testen zu können, wurde ein weiterer Test durchgeführt. Bei diesem Test werden bis zu 2^{23} 16-Bit-Symbole an den FPGA gesendet, im RAM-Baustein abgespeichert und anschließend wieder zurückgeschickt. Die Testbench, für eine bessere Übersicht werden dort nur 8 Symbole übertragen, ist in Abb. 5.14 zu sehen.

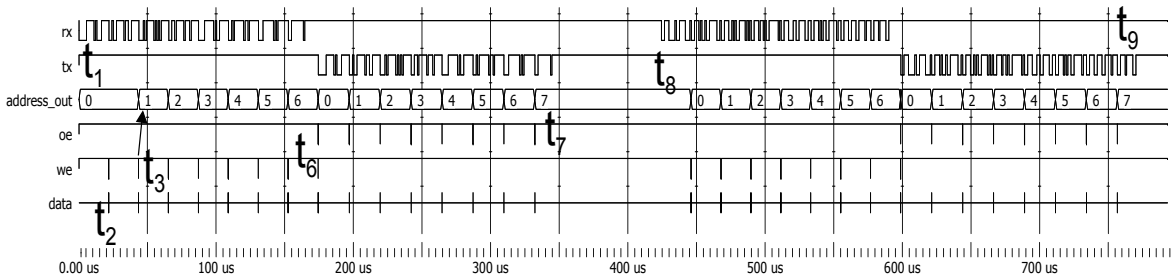

Abbildung 5.14: Testbench für den UART-RAM-Test

Zum Zeitpunkt t_1 startet der Empfang von den acht 16-Bit-Symbolen. Bei t_2 wird das erste empfangene 16-Bit-Symbol in den RAM-Baustein geschrieben, weshalb das low-aktive Write Enable auf 0 gesetzt wird. Dadurch verlässt der Tri-State Buffer $data$ kurz seinen hochohmigen Zustand, sodass die Daten in der Adresse $address\_out$ des RAM-Bausteins gespeichert werden. Anschließend wird bei dem nächsten Nullsetzen von we die Adresse inkrementiert (z.B. bei t_3), um das RAM zu beschreiben. Zum Zeitpunkt t_6 wurden alle Symbole empfangen, sodass der Sendevorgang beginnt, der Adresszähler zurückgesetzt und das low-aktive Output Enable oe auf 0 gesetzt werden. Bei t_7 wurden die empfangen Daten korrekt aus dem RAM gelesen und versendet. Zum Zeitpunkt t_8 werden neue Daten empfangen und im RAM gespeichert (die alten Daten werden korrekt überschrieben). Bei t_9 sind beide Empfangs- und Sendevorgänge korrekt abgeschlossen. Die funktionale Simulation liefert korrekte Ergebnisse.

Das Testsystem wurde auf dem FPGA implementiert und getestet. Die maximale Baudrate für eine zuverlässige Kommunikation ohne Fehler ist $R_{max} = 234000kbit/s$. Bei höheren Raten kam es bereits bei 2^{16} Symbolen zu Übertragungsfehlern, mit R_{max} wurden 2^{23} 16-Bit-Werte fehlerfrei gesendet und wieder empfangen.

Control Unit

Die Steuereinheit wurde in drei Komponenten unterteilt, die nacheinander das Empfangen der Daten, die Verarbeitung und das Senden steuern. Abb. 5.15 zeigt die Testbench für die Empfangssteuerung.

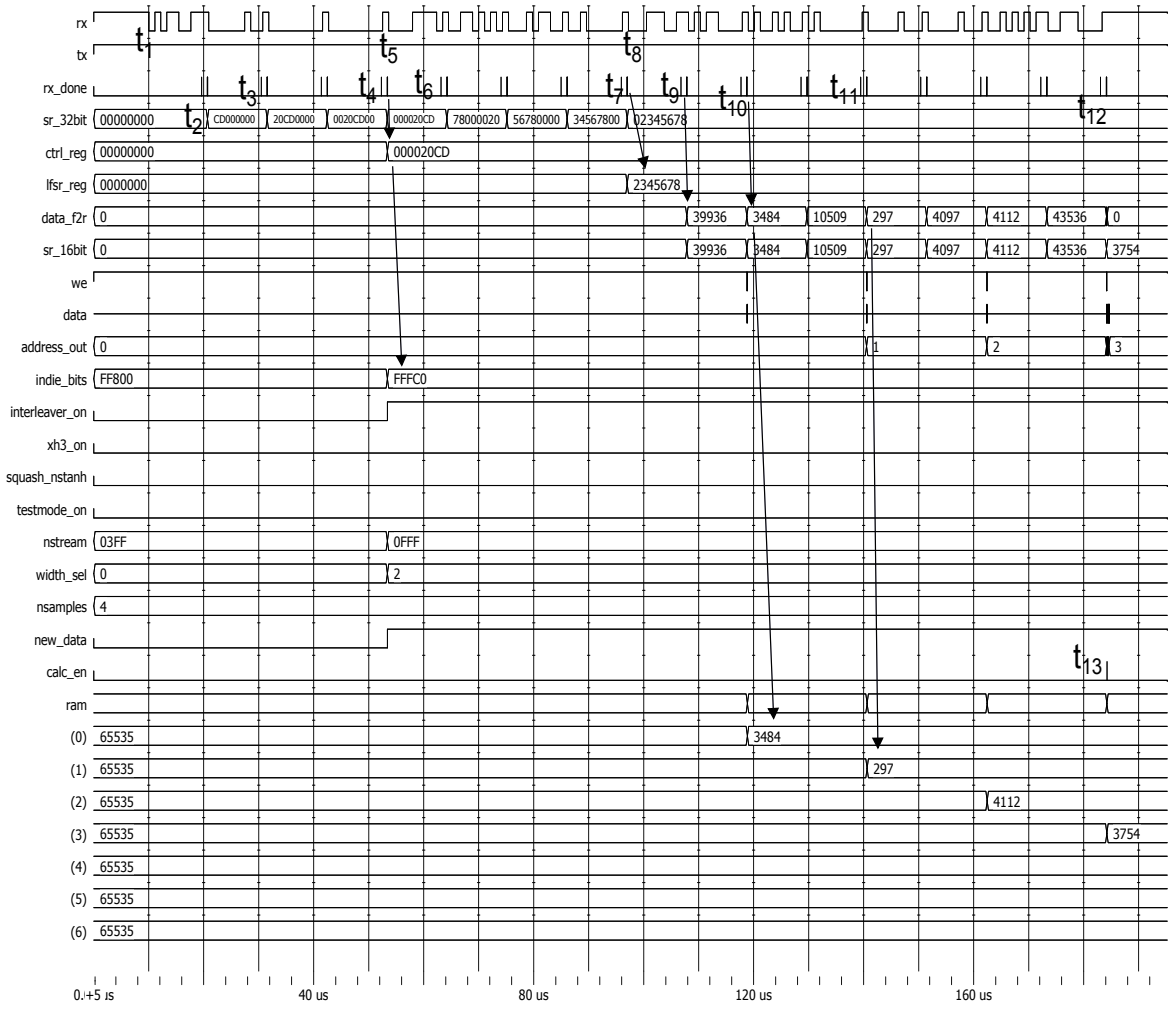

Abbildung 5.15: Testbench für die Empfangs-Steuereinheit

Zuerst wird ein 32-Bit-breites Steuersymbol $0x000020CD$ empfangen, welches in das Register $ctrl\_reg$ gespeichert werden soll.

Zum Zeitpunkt t_1 wird die Übertragung der Daten gestartet. Die Empfangsleitung rx wird auf 0 gesetzt und ein Startbit detektiert. Nach dem Empfang der 8 Datenbits wird zum Zeitpunkt t_2 das erste Byte $0xCD$ in das 32-Bit-Schieberegister gespeichert, bei t_3 das zweite Byte.

Nachdem alle 4 Bytes des Steuersymbols empfangen wurden, wird der Inhalt des Schieberegisters in das Steuerregister $ctrl\_reg$ geschrieben (t_4). Dadurch werden die Einstellungen aktualisiert. Als nächstes wird der Startwert des Pseudozufallszahlengenerators übertragen (t_5). Die Bytes werden erneut in dem 32-Bit Schieberegister zwischengespeichert (t_6) und in das $lfsr\_reg$ geschrieben (t_7). Daraufhin werden ab t_8 die 4 Abtastwerte übertragen. Hier wird nun jedes Byte in das 16-Bit Schieberegister geschoben (t_9) und als 16-Bit-Symbol in den RAM-Baustein geschrieben. Zum Zeitpunkt t_{10} ist zu erkennen, dass das Write Enable we auf 0 gesetzt wird, die Datenleitung kurz den hochohmigen Zustand verlässt und bei der Adresse 0 der erste Abtastwert 3484 gespeichert wird. Der zweite Abtastwert wird bei t_{11} gespeichert und zum Zeitpunkt t_{12} der letzte Wert. Der Empfang ist abgeschlossen, das Signal $calc\_en$ wird gesetzt (t_{13}). Dadurch wird die Verarbeitung der Abtastwerte durch den Algorithmus initiiert.

Die Testbench der Steuerungskomponenten für die Signalverarbeitung wird in Abb. 5.16 gezeigt.

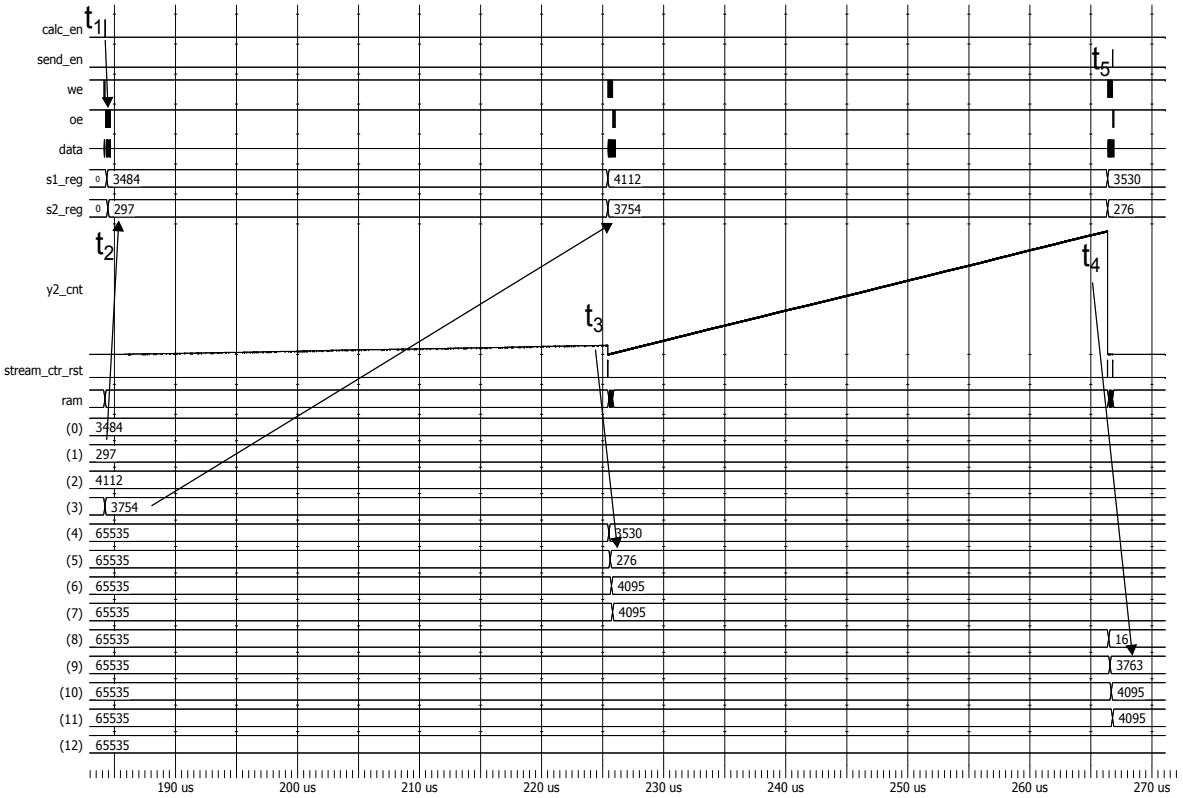

Abbildung 5.16: Testbench für die Verarbeitungs-Steuereinheit

Zur leichten Verifikation werden die Eingangssignale s_1 und s_2 lediglich D/S- und direkt wieder S/D-umgesetzt. Zum Test der präzisen Dekodierung bestehen die Bitströme für die Um-

setzer der Gewichte nur aus Einsen, sodass jeweils der Zählerstand 4095 gespeichert werden sollte. Zum Zeitpunkt t_1 wird der *calc*-Zustandautomat gestartet, *oe* wird auf 0 gesetzt und die ersten beiden Werte aus dem RAM in die Eingangsregister *s1_reg* und *s2_reg* geschrieben (t_2). Die Umsetzung wird gestartet, der Bitstrom-Counter zählt bis 4095, und beispielhaft anhand von *y2_cnt* wird die Dekodierung von s_2 gezeigt. Zum Zeitpunkt t_3 ist der Bitstrom komplett erzeugt, die Zählerwerte werden in die Adressen 4-7 geschrieben. Es ist zu erkennen, dass die ersten beiden dekodierten Bitströme (Adresse 4 und 5) die Eingangswerte darstellen (Adresse 0 und 1) und die letzten beiden Werte (Adresse 6 und 7) erwartungsgemäß gleich 4095 sind. Da in diesem Test 2 Abtastwerte verarbeitet werden, wird bis t_4 die zweite Umsetzung durchgeführt. Die umgesetzten Werte werden erneut an den korrekten Adressen gespeichert, die Verarbeitung ist beendet. Durch diesen Test konnte neben der Steuerung der Verarbeitung zusätzlich die korrekte Funktionsweise des Pseudozufallszahlengenerators sowie der Dekodierer verifiziert werden. Bei t_5 wird der Sende-Zustandsautomat aktiviert.

Die Testbench für die Steuerung des Sendens der Daten zeigt Abb. 5.17.

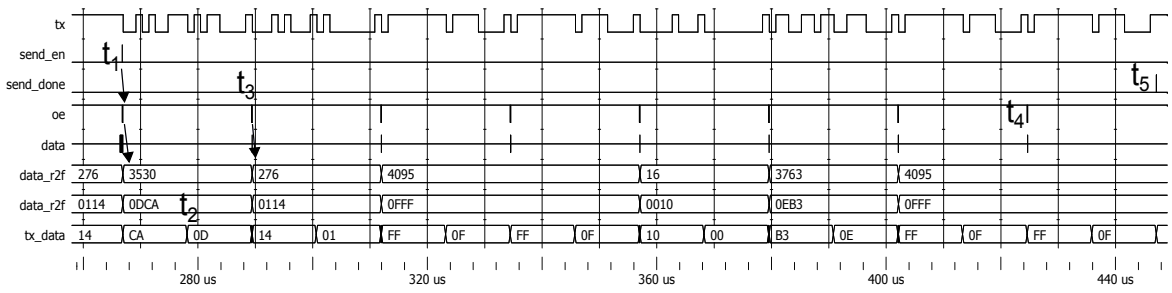

Abbildung 5.17: Testbench für die Sende-Steuereinheit

Zu Beginn wird der erste verarbeitete Wert aus dem RAM gelesen (t_1). Zuerst wird das untere Byte dieses 16-Bit-Wortes gesendet, danach das obere (t_2). Bei t_3 wurden die beiden Bytes gesendet und es wird der nächste Wert aus dem RAM gelesen. Zum Zeitpunkt t_4 startet die Übertragung des letzten 16-Bit-Werts. Nachdem dieser gesendet wurde, wird mit *send_done* das Ende der Übertragung signalisiert.

Gesamtsystem

Das Verhalten der Steuerungskomponenten wurde geprüft, Datenübertragung und Datenspeicherung funktionieren. Um zu zeigen, dass das Gesamtsystem funktioniert, wurden Testdaten in MATLAB erzeugt und an den FPGA gesendet. Das Ergebnis, die empfangenen Ausgangssignale sowie die Entmischungskoeffizienten, wurde mit der Floating-Point-Simulation verglichen. Abb. 5.18 zeigt, dass die Daten korrekt verarbeitet werden. Die Koeffizienten

rauschen im Vergleich zur C-Simulation nicht, weil bei der Implementierung direkt die Zählerstände statt der dekodierten Werte, die vom Algorithmus verarbeitet werden, gespeichert werden.

Die Implementierung des Hérault-Jutten-Algorithmus auf dem FPGA funktioniert nachweislich.

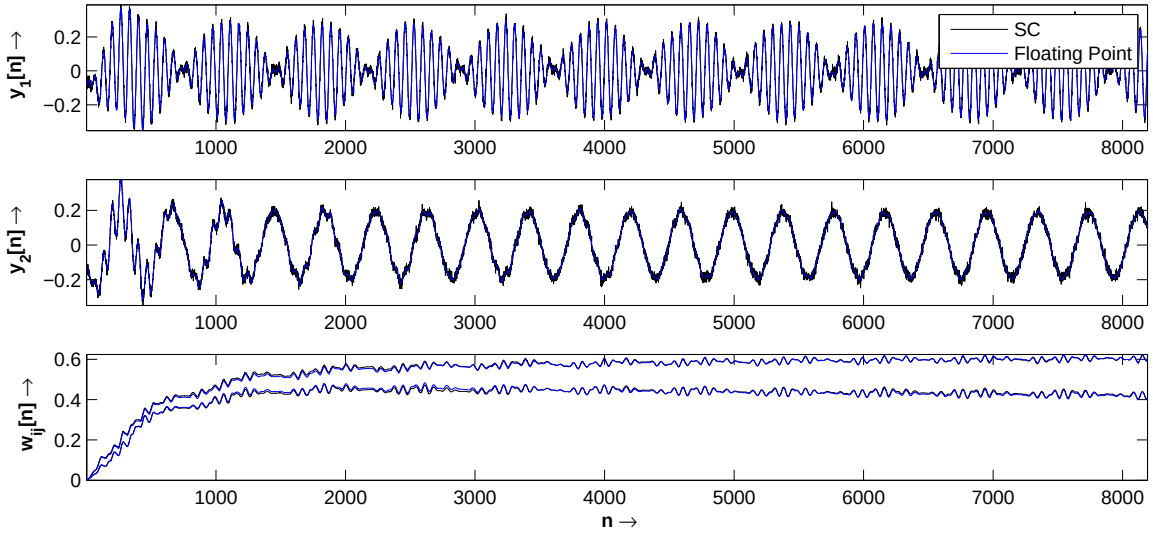

Abbildung 5.18: Test der FPGA-Implementierung

5.4 Vergleich mit Festkomma-Arithmetik

Zum Vergleich des Hardwareaufwands mit einer Festkomma-Variante des Algorithmus wurde mit der System Generator Toolbox von Xilinx ein synthesefähiges Modell in der Simulink-Umgebung erstellt (siehe Abb. 5.19). Die Parameter sind identisch. Die tanh-Funktion wurde mit Hilfe zweier 16-Bit breiten und 4096 Werte tiefen Lookup-Tabellen realisiert, der Konvergenzparameter $\mu = 2^{-7}$ wurde durch siebenmaliges Rechtsschieben umgesetzt.

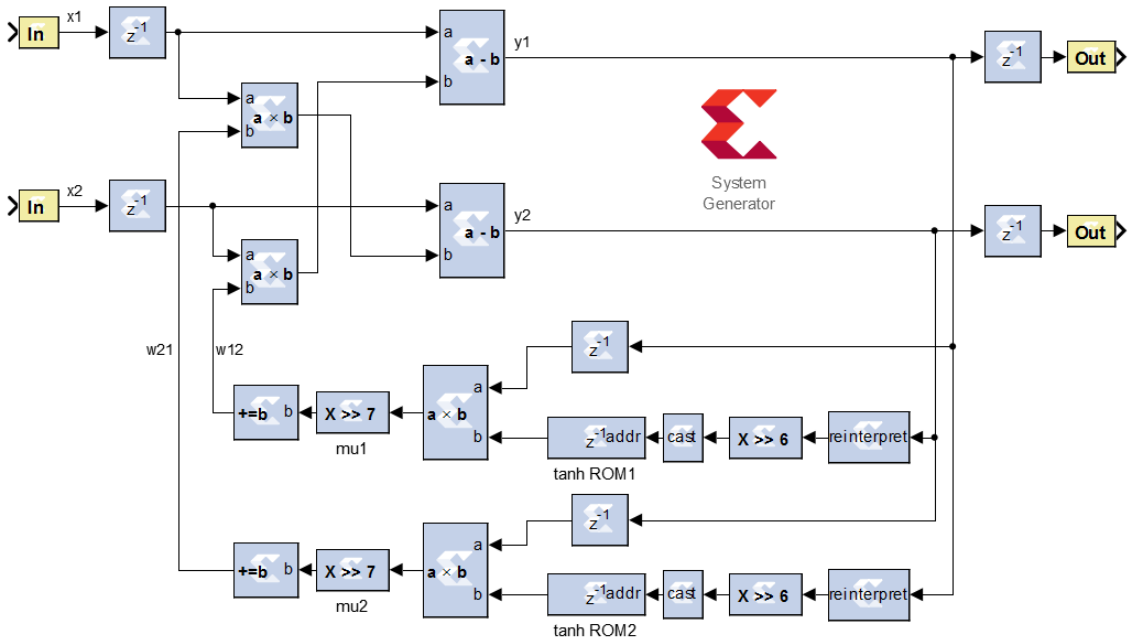

Abbildung 5.19: Simulink-Modell des Hérault-Jutten-Algorithmus

Die Funktionalität des Modells wurde in der Matlab/Simulink-Umgebung getestet. Abb. 5.20 zeigt, dass das Konvergenzverhalten korrekt ist. Im oberen Diagramm ist der Verlauf der Koeffizienten dargestellt. Auch durch Hineinzoomen ist kaum ein Unterschied zwischen der Festkomma- und der Gleitkommasimulation zu erkennen. Durch die Verwendung von Block RAM ist die Adaption um einen Takt verzögert.

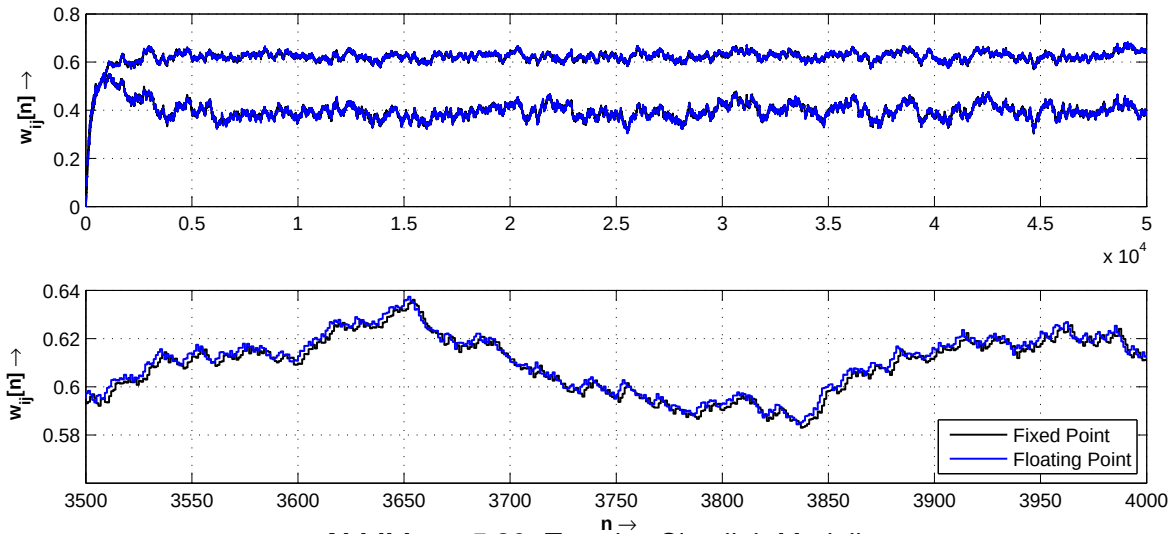

Abbildung 5.20: Test des Simulink-Modells

Das abgebildete Modell wurde in ein synthetisierbares VHDL-Modell übersetzt [38]. Das Modell wurde so konfiguriert, dass für die arithmetischen Operatoren die eingebetteten 18x18 DSP-Blöcke des Spartan-6-FPGAs verwendet werden und für die tanh-Lookup-Tabellen eingebettetes Block RAM. Dadurch ergibt sich sogar ein geringerer Hardwareaufwand als bei der SC-Implementierung. Tab. 5.1 stellt die Ergebnisse der Map Reports gegenüber.

Slice Logic Utilization	Stochastic Computing	Festkomma	vorhanden
Slice Registers	170	36	18224
Slice LUTs	200	19	9112
RAMB16BWERs	0	8	32
DSP48A1s	0	8	32

Tabelle 5.1: Verbrauchte Ressourcen nach dem Mapping

Die RAMB16BWERs-Blöcke sind 16kBit-Block-RAM-Elemente, die hier als 8 16Bit breite und 1024 Werte tiefe Blöcke die beiden tanh-Tabellen implementieren. Die 8 DSP48A1s-Blöcke werden für die 4 Multiplikationen, 2 Additionen und 2 Integrationen verwendet [39].

Hieraus folgt, dass der Vorteil des geringen Hardwareaufwands der SC-Komponenten erst vorhanden sein kann, wenn das Signalverarbeitungssystem so groß ist, dass die festverdrahteten DSP-Slices und Block-RAM-Speicher eines FPGAs für die Festkomma-Implementierung nicht mehr ausreichen. Dann bestünde jedoch immer noch die Möglichkeit, die Signalverarbeitung sequentiell mit einem kleineren Datenpfad, der mit den eingebetteten Blöcken umsetzbar ist, zu realisieren.

In diesem Kapitel wurde gezeigt, dass es möglich ist, den Hérault-Jutten-Algorithmus ausschließlich mit SC-Komponenten auf einem FPGA zu implementieren. Es wurde deutlich, dass durch die Struktur eines FPGAs andere Varianten einer SC-Komponente effizienter implementiert werden können als auf einem ASIC. Weiter wurde allerdings ersichtlich, dass der Vorteil des geringen Hardwareaufwands hier nicht zum Tragen kommt, da ein FPGA über spezielle Hardwareressourcen verfügt, die bei der Festkomma-Arithmetik genutzt werden können und somit eine aufwandsgünstigere, präzisere Implementierung mit Festkommazahlen ermöglichen.

6 Bewertung und Ausblick

In dieser Arbeit wurde erstmals nachweislich ein Blind-Source-Separation-Algorithmus mittels Stochastic Computing erfolgreich auf einem FPGA implementiert. Es wurde gezeigt, dass das Konvergenzverhalten der SC-Variante jenem der Floating-Point-Variante gleicht. Der SC-Algorithmus verhält sich also genauso. Die Qualität der Separation wird zwar durch die der Rechentechnik zugrundeliegende Varianz im Mittel verschlechtert, die Stabilität ist jedoch gleich. Darüber hinaus wurde im Rahmen dieser Arbeit die Plessmann-Methode untersucht, die zumindest für gut ausgesteuerte Eingangssignale funktioniert. Dadurch könnte ein Nachteil der stochastischen Rechentechnik, die Varianz, eliminiert werden. Das Verhalten ist allerdings nicht vollständig nachvollziehbar, für schlecht ausgesteuerte Signale ist die Funktionalität nicht gegeben. Die Plessmann-Methode kann dagegen nicht den schwerwiegendsten Nachteil der stochastischen Rechentechnik eliminieren. Es wurde gezeigt, dass die Limitierung des Wertebereichs bei Stochastic Computing verhindert, im Vergleich zum Hérault-Jutten-Algorithmus zuverlässige Blind-Source-Separation-Algorithmen zu implementieren. Diese Limitierung führt dazu, dass lediglich der Hérault-Jutten-Algorithmus ausschließlich für zwei vermischte Signale mit SC-Komponenten implementiert werden kann. Ein Vorteil von Stochastic Computing ist dabei die aufwandsgünstige Realisierung der Komponenten. Es musste indes festgestellt werden, dass dieser Vorteil bei einer FPGA-Implementierung nicht vorhanden ist. Denn der Algorithmus benötigt nur wenige arithmetische Operatoren und nichtlineare Funktionen, wofür bei einer Implementierung mit Festkommazahlen eine geringe Zahl der eingebetteten Ressourcen des FPGAs verwendet werden kann. Dadurch ist der Verbrauch an Logik-Ressourcen bei der SC-Variante deutlich höher.

Vorteile bei der Implementierung können erst dann entstehen, wenn deutlich mehr als zwei vermischte Signale verarbeitet werden und dementsprechend eine massiv parallele Signalverarbeitung durchgeführt wird. Dann würden die eingebetteten Ressourcen des FPGAs für eine parallele Verarbeitung mit Festkommazahlen nicht mehr ausreichen, sodass ein Teil der ressourcenintensiven Operationen mit programmierbarer Logik umgesetzt werden müsste. Allerdings besteht bei einem solchen Sachverhalt die Möglichkeit, anstatt einer vollparallelen Verarbeitung sequentiell einen kleineren Datenpfad, der mit den eingebetteten Ressourcen implementiert werden kann, zu nutzen. Dadurch wäre der Hardwareaufwand bei Festkommazahlen immer noch geringer, weil hauptsächlich die Steuerung der sequentiellen Verarbeitung Logik-Ressourcen beanspruchen würde. Ein solcher Fall wurde für die Aufgabenklasse

Blind Source Separation hier nicht im Detail betrachtet, da die Funktionalität mit SC bei mehr als zwei Signalen, mit den aktuell bekannten Komponenten, nicht gegeben ist.

Aus den Ergebnissen der hier angestellten Untersuchungen folgt, dass ausschließlich die Störsicherheit der stochastischen Rechentechnik als Vorteil bei der Implementierung einer blinden Quellentrennung auf einem FPGA zu werten ist. Diese Störsicherheit, die zu einer hohen Fehlertoleranz führt, ist einzig bei fehlerbehafteten Systemen vorteilhaft. Sartori et al. schildern in [40], dass die Anzahl an Fehler in Systemen aufgrund einer immer höheren Intergrationsdichte in Zukunft steigen werde. Erst wenn diese Annahmen eintreten und die Fehlerhäufigkeit hoch ist, ist die Störsicherheit der stochastischen Rechentechnik ein Vorteil. Aktuell ist in der Literatur kein Fall einer Blind Source Separation dokumentiert, die auf einem fehlerbehafteten System implementiert wurde oder in einer Umgebung eingesetzt wird, wo z.B. durch kosmische Strahlung die Funktionalität mit Festkommazahlen nicht mehr gegeben ist.

Die Vorteile der Rechentechnik greifen bei der blinden Quellentrennung nicht, die Nachteile führen dazu, dass nur ein Algorithmus, limitiert auf den Spezialfall von zwei Signalen, implementiert werden kann. Demzufolge muss gefolgert werden, dass der Lösungsansatz des Stochastic Computing, mit den aktuell bekannten Komponenten, für die Aufgabenklasse Blind Source Separation nicht geeignet ist. Es sind aber durchaus Anwendungen möglich, bei denen die Wertebereich-Begrenzung nicht existiert und die Signalverarbeitung so massiv parallel ist, dass SC vorteilhaft genutzt werden kann. Die parallele Dekodierung von u.a. beim DVB-S2-Standard genutzten LDPC-Codes (Low-Density Parity-Check) ist eine solche Anwendung [41, 42].

Mit den Erkenntnissen dieser Arbeit sind folgende weitere Arbeiten denkbar:

- Die Entwicklung neuer SC-Komponenten für einen erweiterten Wertebereich ist notwendig, damit performante BSS-Algorithmen, die auch mehr als zwei Signale zuverlässig trennen können, mittels SC implementiert werden können. Ob dies möglich ist und inwiefern der Hardwareaufwand steigen würde, ist zu klären.
- Die Auswirkungen von Korrelationen in den stochastischen Bitströmen sind in der stochastischen Rechentechnik immer noch weitestgehend unverstanden. Die Untersuchung der Rolle der Zufallszahlen ist wichtig für das Verständnis von Stochastic Computing und sollte erforscht werden.
- Mit der hier betrachteten Plessmann-Methode ist es möglich, rauschfrei zu rechnen. Die negativen Auswirkungen einer geringen Aussteuerung sowie das Verhalten der nichtlinearen Funktion müssen untersucht werden, damit die Methode nicht nur in Spezialfällen funktioniert. Auch ist die Frage zu klären, wie die Zufallszahlen effizient implementiert werden können.

A Inhalt des Datenträgers

Die hier aufgelisteten Anhänge sind in elektronischer Form auf einer CD in dem Ordner *Anhang* abgelegt. Diese CD ist bei dem Erstprüfer Prof. Dr.-Ing. Karl-Ragmar Riemschneider einzusehen.

1. **Aufgabenstellung** (/Anhang/1_Aufgabenstellung): Dieser Ordner enthält die Aufgabenstellung als pdf-Datei, die vom Erstprüfer verfasst und im Original dem Prüfungsausschussvorsitzenden bei der Anmeldung der Thesis überreicht wurde.
2. **Matlab-Scripte** (/Anhang/2_Matlab): Hier sind sowohl die Floating-Point Simulationsprogramme der Blind-Source-Separation-Algorithmen als auch die Skripte zur Generierung von Plessmann-Zufallszahlen zu finden.
3. **Stochastic-Computing-Programme** (/Anhang/3_C): In diesem Ordner befinden sich die geschriebenen C-Programme samt Matlab-Programmen, mit denen die verschiedenen SC-Codierer getestet wurden und der Hérault-Jutten-Algorithmus mit SC-Komponenten simuliert wurde.
4. **FPGA-Implementierung** (/Anhang/4_Implementierung): Hier sind die ModelSim-Projekte für die funktionalen Tests der Hauptkomponenten sowie die ISE-Projekte für deren Hardware-Tests zu finden. Desweiteren sind in diesem Ordner das ISE-Projekt des Gesamtsystems sowie das Simulink-Modell inklusive ISE-Projekt der Festkomma-Variante des Algorithmus.

Literaturverzeichnis

[1] KÖHLER, Bert-Uwe: *Konzepte der statistischen Signalverarbeitung*. Springer-Verlag Berlin Heidelberg, 2005. – ISBN 3–540–23491–8

[2] STEINMÜLLER, Johannes: *Vorlesungsskript Bildverstehen - Kapitel 10*. https://www.tu-chemnitz.de/informatik/KI/edu/biver/ss2013/bild12_10.pdf. Version: SS 2013. – [Online; Aufruf am 18.03.2014]

[3] HYVÄRINEN, Aapo ; KARHUNEN, Juha ; OJA, Erkki: *Independent Component Analysis*. Wiley-Interscience, 2001. – ISBN 978–3–642–62579–4

[4] HUANG, Wei-Chung ; HUNG, Shao-Hang ; CHUNG, Jen-Feng ; CHANG, Meng-Hsiu ; VAN, Lan-Da ; LIN, Chin-Teng: FPGA implementation of 4-channel ICA for on-line EEG signal separation. In: *Biomedical Circuits and Systems Conference, 2008. BioCAS 2008. IEEE*, 2008, S. 65–68

[5] SHYU, Kuo-Kai ; LEE, Ming-Huan ; WU, Yu-Te ; LEE, Po-Lei: Implementation of Pipelined FastICA on FPGA for Real-Time Blind Source Separation. In: *Neural Networks, IEEE Transactions on* 19 (2008), June, Nr. 6, S. 958–970. http://dx.doi.org/10.1109/TNN.2007.915115. – DOI 10.1109/TNN.2007.915115. – ISSN 1045–9227

[6] VIGARIO, R. ; SARELA, J. ; JOUSMIKI, V. ; HÄMÄLÄINEN, M. ; OJA, E.: Independent component approach to the analysis of EEG and MEG recordings. In: *Biomedical Engineering, IEEE Transactions on* 47 (2000), May, Nr. 5, S. 589–593. http://dx.doi.org/10.1109/10.841330. – DOI 10.1109/10.841330. – ISSN 0018–9294

[7] REPOVŠ, Grega: Dealing with Noise in EEG Recording and Data Analysis. (2010)

[8] MASSEN, Robert: *Stochastische Rechentechnik - Eine Einführung in die Informationsverarbeitung mit zufälligen Pulsfolgen*. Carl Hanser Verlag München Wien, 1977. – ISBN 3–446–12150–1

[9] GAINES, B. R.: Stochastic Computing Systems. In: *Plenum Press* Vol. 2 (1969), S. 37–172

[10] ALAGHI, Armin ; HAYES, John P.: Survey of stochastic computing. In: *ACM Transactions on Embedded computing systems (TECS)* 12 (2013), Nr. 2s, S. 92

[11] FANGHANEL, K. ; KOLLMANN, K. ; ZEIDLER, H.C. ; PLESSMANN, R. ; RIEMSCHNEIDER, K.-R.: Parallel bit-stream neurohardware for blind separation of sources. 5 (1997), Apr, S. 4149–4152 vol.5. http://dx.doi.org/10.1109/ICASSP.1997.604860. – DOI 10.1109/ICASSP.1997.604860. – ISSN 1520–6149

[12] RIEMSCHNEIDER, Karl-Ragmar: *Parallele Hardware für Backpropagation-Netze auf der Basis stochastischer Rechenwerke*, Universität der Bundeswehr Hamburg, Dissertation, 1996

[13] HORI, M. ; UEDA, M.: FPGA implementation of a blind source separation system based on stochastic computing. In: *Soft Computing in Industrial Applications, 2008. SMCia '08. IEEE Conference on*, 2008, S. 182–187

[14] STONE, James V.: *Independent Component Analysis - A Tutorial Introduction*. Bradford Books, 2004. – ISBN 0262693151

[15] PAPULA, Lothar: *Mathematik für Ingenieure und Naturwissenschaftler - Band 3*. Vieweg + Teubner, 2008. – ISBN 978–3–8348–0225–5

[16] ROMANO, Joao M. ; ATTUX, Romis R. de F. ; CAVALCANTE, Charles C. ; SUYAMA, Ricardo: *Unsupervised Signal Processing - Channel Equalization and Source Separation*. CRC Press, 2011. – ISBN 978–1–4200–1946–9

[17] CLIFFORD, G.D.: *Vorlesungskript Biomedical Signal and Image Processing - Chapter 15*. http://www.mit.edu/~gari/teaching/6.555/LECTURE_NOTES/ch15_bss.pdf. Version: Frühling 2008. – [Online; Aufruf am 14.03.2014]

[18] JUTTEN, Christian ; TALEB, Anisse: Source Separation: From Dusk Till Dawn. (2000)

[19] HÄNSLER, Eberhard: *Statistische Signale*. Springer-Verlag, 2004. – ISBN 0262693151

[20] CICHOCKI, A ; UNBEHAUEN, R.: Robust neural networks with on-line learning for blind identification and blind separation of sources. In: *Circuits and Systems I: Fundamental Theory and Applications, IEEE Transactions on* 43 (1996), Nov, Nr. 11, S. 894–906. http://dx.doi.org/10.1109/81.542280. – DOI 10.1109/81.542280. – ISSN 1057–7122

[21] COHEN, M.H. ; ANDREOU, AG.: Current-mode subthreshold MOS implementation of the Herault-Jutten autoadaptive network. In: *Solid-State Circuits, IEEE Journal of* 27 (1992), May, Nr. 5, S. 714–727. http://dx.doi.org/10.1109/4.133158. – DOI 10.1109/4.133158. – ISSN 0018–9200

[22] AMARI, S. ; CICHOCKI, A. ; YANG, H. H.: A New Learning Algorithm for Blind Signal Separation. In: *Advances in Neural Information Processing Systems*, MIT Press, 1996, S. 757–763

[23] CARDOSO, J.-F. ; LAHELD, B.H.: Equivariant adaptive source separation. In: *Signal Processing, IEEE Transactions on* 44 (1996), Dec, Nr. 12, S. 3017–3030. http://dx.doi.org/10.1109/78.553476. – DOI 10.1109/78.553476. – ISSN 1053–587X

[24] JEAVONS, P. ; COHEN, D. A. ; SHAWE-TAYLOR, J.: Generating binary sequences for stochastic computing. In: *Information Theory, IEEE Transactions on* 40 (1994), May, Nr. 3, S. 716–720

[25] BROWN, B.D. ; CARD, H.C.: Stochastic neural computation. I. Computational elements. In: *Computers, IEEE Transactions on* 50 (2001), Sep, Nr. 9, S. 891–905. http://dx.doi.org/10.1109/12.954505. – DOI 10.1109/12.954505. – ISSN 0018–9340

[26] LI, Peng ; QIAN, Weikang ; RIEDEL, M.D. ; BAZARGAN, K. ; LILJA, D.J.: The synthesis of linear Finite State Machine-based Stochastic Computational Elements. In: *Design Automation Conference (ASP-DAC), 2012 17th Asia and South Pacific*, 2012. – ISSN 2153–6961, S. 757–762

[27] LI, Peng: *Analysis, Design, and Logic Synthesis of Finite-state Machine-based Stochastic Computing*, University of Minnesota, Dissertation, June 2013

[28] ANDREWS, Kenneth ; HEEGARD, Chris ; KOZEN, Dexter: *A Theory of Interleavers*. http://www.ecommons.cornell.edu/bitstream/1813/7289/1/97-1634.pdf. Version: 1997. – [Online; Aufruf am 24.05.2014]

[29] ALSPECTOR, J. ; GANNETT, J.W. ; HABER, S. ; PARKER, M.B. ; CHU, R.: A VLSI-efficient technique for generating multiple uncorrelated noise sources and its application to stochastic neural networks. In: *Circuits and Systems, IEEE Transactions on* 38 (1991), Jan, Nr. 1, S. 109–123. http://dx.doi.org/10.1109/31.101308. – DOI 10.1109/31.101308. – ISSN 0098–4094

[30] GUPTA, P. K. ; KUMARESAN, R.: Binary multiplication with PN sequences. In: *Acoustics, Speech and Signal Processing, IEEE Transactions on* 36 (1988), Apr, Nr. 4, S. 603–606. http://dx.doi.org/10.1109/29.1564. – DOI 10.1109/29.1564. – ISSN 0096–3518

[31] PLESSMANN, Ralf: Persönliche Kommunikation zur Codierung von Stochastic Computing. (Hamburg 2014)

[32] MATHWORKS: *Introducing MEX-Files*. http://www.mathworks.de/de/help/matlab/matlab_external/introducing-mex-files.html. Version: Oktober 2014. – [Online; Aufruf am 03.10.2014]

[33] MIAO, Lifeng ; CHAKRABARTI, C.: A parallel stochastic computing system with improved accuracy. In: *Signal Processing Systems (SiPS), 2013 IEEE Workshop on*, 2013. – ISSN 2162–3562, S. 195–200

[34] XILINX: *Spartan-6 Family Overview*. http://www.xilinx.com/support/documentation/data_sheets/ds160.pdf. Version: Oktober 2011. – [Online; Aufruf am 07.09.2014]

[35] DIGILENT: *Nexys3 Board Reference Manual*. http://www.digilentinc.com/Data/Products/NEXYS3/Nexys3_rm.pdf. Version: April 2013. – [Online; Aufruf am 27.09.2014]

[36] ASHENDEN, Peter J.: *The Designer's Guide to VHDL*. Elsevier Ltd, Oxford, 2001. – ISBN 978–1558606746

[37] MICRON: *Async/Page/Burst CellularRAM TM 1.5 MT45W8MW16BGX*. http://www.micron.com/-/media/documents/products/data%20sheet/dram/mobile%20dram/psram/128mb_burst_cr1_5_p26z.pdf

[38] XILINX: *System Generator for DSP User Guide (UG640)*. http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_4/sysgen_user.pdf. Version: Oktober 2012. – [Online; Aufruf am 02.10.2014]

[39] XILINX: *Spartan-6 Libraries Guide for Schematic Designs (UG616)*. http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_4/spartan6_scm.pdf. Version: September 2010. – [Online; Aufruf am 06.06.2014]

[40] SARTORI, J. ; SLOAN, J. ; KUMAR, R.: Stochastic computing: Embracing errors in architecture and design of processors and applications. In: *Compilers, Architectures and Synthesis for Embedded Systems (CASES), 2011 Proceedings of the 14th International Conference on*, 2011, S. 135–144

[41] GROSS, W.J. ; GAUDET, V.C. ; MILNER, A.: Stochastic Implementation of LDPC Decoders. In: *Signals, Systems and Computers, 2005. Conference Record of the Thirty-Ninth Asilomar Conference on*, 2005. – ISSN 1058–6393, S. 713–717

[42] TEHRANI, S.S. ; MANNOR, S. ; GROSS, W.J.: Fully Parallel Stochastic LDPC Decoders. In: *Signal Processing, IEEE Transactions on* 56 (2008), Nov, Nr. 11, S. 5692–5703. http://dx.doi.org/10.1109/TSP.2008.929671. – DOI 10.1109/TSP.2008.929671. – ISSN 1053–587X

Versicherung über die Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §16(5) APSO-TI-BM ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen habe ich unter Angabe der Quellen kenntlich gemacht.

Hamburg, 16. Oktober 2014	
Ort, Datum	Unterschrift