HAW HAMBURG

Bachelorarbeit

Martin Willimczik

Hardware Implementierung zur Ortsfrequenzberechnung für Sensorarrays

Fakultät Technik und Informatik
Department Informations- und Elektrotechnik

Faculty of Computer Science and Engineering
Department Information and Electrical Engineering

Martin Willimczik

Hardware Implementierung zur Ortsfrequenzberechnung für Sensorarrays

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang Bachelor of Science Elektro- und Informationstechnik
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

Eingereicht am: 18. März 2019

Martin Willimczik

Thema der Arbeit

Hardware Implementierung zur Ortsfrequenzberechnung für Sensorarrays

Stichworte

2D-DFT, Fouriertransformation, automatische Hardware-Erzeugung, Sensorarray

Kurzzusammenfassung

In dieser Arbeit wurde eine Hardware-Implementierung zur Berechnung individueller Ausgangspunkten einer 2D-DFT entwickelt. Der Ansatz verwendet Koeffizientenmustererkennung zur Reduktion des allgemeinen Logikbedarfs. Es wurde ein Softwareframework zur automatischen Erzeugung der Hardwarearchitektur programmiert. Die Ergebnisse zeigen, dass die erzeugte Hardware die Projektanforderungen von Flächenbedarf und Genauigkeit erfüllt und eine ASIC-Implementierung möglich ist.

Martin Willimczik

Title of Thesis

Hardware implementation calculating spatial frequencies for sensor arrays

Keywords

2D-DFT, Fourier Transform, Hardware-Generator, sensor-array

Abstract

In this thesis a hardware implementation for calculating individual output points of a 2D-DFT was developed. The approach uses coefficient pattern recognition to reduce the general logic requirement. A software framework has been developed to automatically generate the hardware architecture. The results show that the generated hardware meets the project requirements of area and accuracy and that an ASIC implementation is possible.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungen

ASIC Application-specific integrated circuit.

CORDIC Coordinate Rotation Digital Computer.

DFT Diskrete Fourier Transformation.

FFT Fast Fourier Transform.

FPGA field-programmable gate array.

FT Fourier Transformation.

HAW Hochschule für Angewandte Wissenschaften.

ISAR Signalverarbeitung für Integrated Sensor-Arrays basierend auf dem Tunnel-Magnetoresistiven Effekt für den Einsatz in der Automobilelektronik.

RAM Random Access Memory.

ROM Read Only Memory.

SNR Signal-to-noise ratio.

SQNR Signal-to-quantization-noise ratio.

1 Einleitung

Diese Arbeit beschäftigt sich mit der Implementierung einer zweidimensionalen diskreten Fourier Transformation für einen magnetischen Winkelsensor-ASIC. Die Entwicklung findet unter dem Dach des ISAR-Projekts an der Hochschule für Angewandte Wissenschaften (HAW) statt und beschäftigt sich mit der Entwicklung von Signalverarbeitungsalgorithmen für Arrays aus magnetischen Winkelsensoren. Ziel ist die Entwicklung eines kostengünstigen integrierten Sensorchips, der immun gegen externe Störfelder ist und sich für eine Vielzahl von Anwendungen eignet. Beispiele hierfür sind ABS- und ESP-Sensoren, Winkelgeber für bürstenlose Gleichstrommotoren und weitere Anwendungen, bei denen die Orientierung und Drehzahl eines Permanentmagnetgebers präzise, berührungslos und störsicher erfasst werden müssen.

1.1 Motivation und Ziel dieser Arbeit

Im ISAR-Projekt wird für die Berechnung von Winkeln aus Sensordaten eine 2D-DFT verwendet. Da ein Sensorchip mit integrierter Signalverarbeitung das Ziel ist, wird eine Hardware-Implementierung der 2D-DFT benötigt. Im Zuge der Entwicklung der Signalverarbeitungsalgorithmen wurde ein Ansatz gefunden, der den Winkel mit wenigen Punkten der 2D-DFT berechnet. Die bisher im Verlauf des ISAR-Projekts entwickelten 2D-DFT Implementierungen führen immer eine vollständige Transformation durch. Das Ziel dieser Arbeit ist daher die Entwicklung einer Hardware-Implementierung, die nur die benötigten Punkte berechnet. Die Ergebnisse der Entwicklung sollen die Kernfrage beantworten, ob der Hardwareaufwand durch die Beschränkung auf wenige Ausgangspunkte auf ein Niveau reduziert werden kann, das eine Implementierung auf einem kostengünstigen Sensor-Application-specific integrated circuit (ASIC) erlaubt.

1.2 Anforderungskatalog

Die Entwicklung der Signalverarbeitungsalgorithmen für die Sensorarrays ist bei weitem noch nicht abgeschlossen, sodass viele der im Folgenden genannten Werte nur als Anhaltspunkte und nicht als exakte Spezifikationen zu verstehen sind. Die angegebenen Werte entsprechen dem aktuellen Forschungsstand, es wird jedoch erwartet, dass sich im weiteren Projektverlauf Veränderungen ergeben.

1.2.1 Größe der Eingangsmatrix

Die Größen der verwendeten Sensorarrays bewegen sich im Bereich von 3x3 bis 9x9 Sensoren. Die Arrays sind immer quadratisch, folglich sind es die Matrizen der Eingangsdaten auch. In der Signalverarbeitung ist eine Interpolation der Sensordaten vorgesehen, wodurch sich die Größe der Transformation gegenüber der Anzahl der Sensoren verdoppelt. Die zu verarbeitenden Matrixgrößen liegen daher im Bereich von 6x6 bis 18x18 Punkten.

1.2.2 Bitbreiten und Genauigkeit

Da das Ziel der Entwicklung ein ASIC ist, muss der Algorithmus mit einem Festkomma-Datenpfad implementiert werden. Die Auflösung der einlaufenden Sensordaten wird mit je 12 Bit für Real- und Imaginärteil angenommen. Diese Vorgabe basiert auf den verfügbaren Analog-Digitalwandler IP-Cores der Halbleiterhersteller und der nötigen Samplerate. Die Bitbreite der Ausgangsdaten soll der der Eingangsdaten entsprechen. Zum erforderlichen Signal-to-noise ratio (SNR) bzw. der Genauigkeit der Fourier Transformation (FT) gibt es noch keine Analysen, daher wird ein Mittelweg zwischen von Präzision und Hardwareaufwand angestrebt.

1.2.3 Ortsfrequenzen

Da in der weiteren Signalverarbeitung nur bestimmte Ortsfrequenzen, im folgenden auch Ausgangspunkt genannt, benötigt werden, sollen daher nicht alle, sondern ausschließlich die benötigten, Datenpunkte berechnet werden. Dies erfordert eine separierbare Form der Berechnung.

1.2.4 Geschwindigkeit und Latenz

Verarbeitungsgeschwindigkeit ist eine wichtige Anforderung, die in anderen Arbeiten zum Thema noch nicht zufriedenstellend gelöst wurde. In der Arbeit von Lattman wurde in Kapitel 5.3 [3] ein Zeitfenster von 2083 Taktzyklen für die gesamte Signalverarbeitung berechnet. Diese Forderung basiert auf einer maximal zu erfassenden Drehzahl von 8000 U/min in Kombination mit 1° Winkelauflösung. Für die Berechnung der 2D-DFT sind maximal 50% dieses Taktbudgets vorgesehen.

2 Grundlagen

Wie aus dem Anforderungskatalog zu erkennen ist, wird nicht von beliebigen MxN Transformationen, sondern von quadratischen Transformationen der Größe MxM Punkte, ausgegangen. Alle hier im Grundlagenabschnitt gezeigten Formeln beinhalten nicht den Skalierungsfaktor, der normalerweise vorangestellt wird. Skalierung wird in dieser Arbeit als separates Problem von dem der eigentlichen Transformation behandelt. Im Normalfall wird so skaliert, dass eine Folge von DFT und inverser DFT am Ende einen Faktor von 1 hat.

2.1 Entwicklung der 2D-DFT aus der 1D-DFT

Ausgangspunkt ist die eindimensionale Diskrete Fourier-Transformation mit der bekannten Formel:

$$F(n) = \sum_{x=0}^{M-1} f(x) \cdot e^{-i \cdot 2\pi \cdot \frac{nx}{M}} \tag{2.1}$$

n ist der Index des Ausgangspunktes, während x der Index der Eingangspunkte ist.

Ausgehend von der DFT erhält man die 2D-DFT, indem man sie zeilen- und spaltenweise (oder umgekehrt) auf die Eingangsmatrix anwendet.

2D-DFT über MxM Punkte:

$$F(m,n) = \sum_{y=0}^{M-1} \sum_{x=0}^{M-1} f(y,x) \cdot e^{\frac{-i \cdot 2\pi \cdot (ym+xn)}{M}} \tag{2.2}$$

m und n sind die Indizes der Ausgangspunkte, x und y finden hier als Index für die Eingangsdaten Verwendung.

2.2 Abgeleitete Summenform der 2D-DFT

Als Beispiel wird hier die ausgeschriebene Summenformel für Ausgangspunkt (2,2) einer 3x3 DFT dargestellt:

$$\begin{aligned} F(2,2) = {} & f_{(1,1)} \cdot (1 + 0i) + f_{(1,2)} \cdot (-\frac{1}{2} - \frac{\sqrt{3}}{2}i) + f_{(1,3)} \cdot (-\frac{1}{2} + \frac{\sqrt{3}}{2}i) \\ & + f_{(2,1)} \cdot (-\frac{1}{2} - \frac{\sqrt{3}}{2}i) + f_{(2,2)} \cdot (-\frac{1}{2} + \frac{\sqrt{3}}{2}i) + f_{(2,3)} \cdot (1 + 0i) \\ & + f_{(3,1)} \cdot (-\frac{1}{2} + \frac{\sqrt{3}}{2}i) + f_{(3,2)} \cdot (1 + 0i) + f_{(3,3)} \cdot (-\frac{1}{2} - \frac{\sqrt{3}}{2}i) \end{aligned} \tag{2.3}$$

Wie sich hier erkennen lässt, wird bei der 2D-DFT, genau wie bei der 1D-Form, jeweils ein Eingangssample mit einem komplexen Faktor multipliziert. Die komplexen Produkte werden dann aufsummiert. Unabhängig von der reihen- und spaltenweisen Konstruktion der 2D-DFT Formel erhält man das Endergebnis ohne die Berechnung von Zwischenergebnissen.

3 Entwicklung der Hardwarearchitektur

Nun soll für diese 2D-DFT Formel, unter Berücksichtigung der gegebenen Anforderungen, eine Hardwarearchitektur entwickelt werden. Auf dem Gebiet der diskreten Fourier-Transformation wurde über die Jahre sehr viel Forschung betrieben und erhebliche Fortschritte in der Reduktion der, für die DFT erforderlichen, Rechenoperationen erzielt. Viele dieser Rechentricks sind jedoch nicht anwendbar, wenn die Transformationsgröße keine Potenz von 2 ist (klassische FFT). Bei kleinen Transformationsgrößen, wie sie hier benötigt werden, bieten diese Verfahren für eine Hardware-Implementierung kaum nennenswerte Vorteile. Denn viele der komplexeren Verfahren benötigen eine sehr komplizierte und damit flächenmäßig große Kontrolllogik, sowie beliebig adressierbaren Speicher (RAM). Dieser ist zwar im aktuellen Entwicklungssystem vorhanden, jedoch würde für die endgültige Realisierung der Hardware eine Signalverarbeitungs-Pipeline bevorzugt werden. Dazu kommt, dass viele der DFT Optimierungen Zwischenergebnisse mehrerer Ausgangspunkte zusammenfassen um Operationen zu sparen. Das Ziel ist jedoch, beliebige nicht zusammenhängende Ausgangspunkte zu berechnen, daher wird die direkte Implementierung der Formel gewählt. Dies ist auch bekannt als Brute-Force-Ansatz, da die DFT ohne komplizierte Verfahren zur Einsparung von Rechenoperationen gelöst wird.

3.1 Grundlage der Architektur

Bei Betrachtung der zur Verfügung stehenden Taktzyklen und der maximalen Transformationsgröße wird erkennbar, dass eine parallele Berechnung der Ausgangspunkte erforderlich ist. Also muss jeder Ausgangspunkt dedizierte Hardware erhalten. Daraus ergibt sich folglich die Grundlage der Rechenarchitektur, wie in Abbildung 3.1 zu sehen: Ein Hauptmodul, das die Samples aus dem Speicher liest, an parallel arbeitende Rechenmodule leitet und am Ende die Ergebnisse seriell zurückschreibt. Innerhalb der

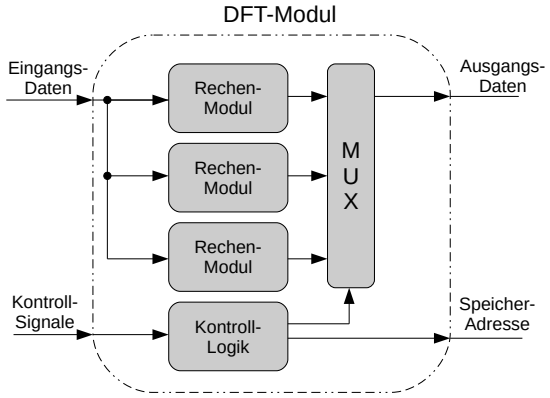

Abbildung 3.1: Entwicklungsansatz 2D-DFT-Modul

Module wird ein Koeffizienten-ROM, ein komplexer Multiplizierer und ein komplexer Akkumulator benötigt. Dieser Ansatz wird im Folgenden noch weiter optimiert.

3.2 Optimierung der Multiplikation mit Konstantenmultiplizierern

Eine Möglichkeit zur Optimierung bei Implementierungen, welche Festkomma-Arithmetik verwenden, ist die Verwendung von Festwertmultiplizieren anstelle echter Multiplizierer. Konstantenmultiplizierer sind Ketten von Additionen und Schiebeoperationen, die die Multiplikation mit einer Konstanten nachbilden. Das folgende Beispiel multipliziert den Eingang x mit der Konstanten 50:

$$50 \cdot x \equiv (((x \underbrace{\ll 1}_{\cdot 2^1} + x) \underbrace{\ll 3}_{\cdot 2^3}) + x) \underbrace{\ll 1}_{\cdot 2^1} \quad x \in \mathbb{Z} \tag{3.1}$$

Hier wurde also eine Multiplikation mit 50 nur mithilfe von Bitschiebeoperationen und Additionen realisiert, welche als Gatter erheblich weniger Platz und Energie benötigen als ein echter Multiplizierer. Diese Konstantenmultiplizierer müssen jedoch für jede einzelne Konstante synthetisiert werden, daher macht dieser Ansatz nur Sinn, wenn es eine begrenzte Anzahl von Konstanten gibt, mit denen multipliziert wird. Diese Voraussetzung ist für die für das Projekt relevanten Transformationsgrößen definitiv gegeben. Für die Berechnung eines Ausgangspunkts einer 15x15 Transformation sind üblicherweise 7

betragsmäßig unterschiedliche Koeffizienten erforderlich. Für die Erzeugung der Konstantenmultiplizierer wird das von Voronenko und Püschel [5] entwickelte Programm „kmult“ genutzt. Dieses erzeugt optimierte Multiplizierer für mehrere Konstanten, die gemeinsame Zwischenwerte der Konstanten mehrfach verwenden um Logikressourcen zu sparen. Im Vergleich zu den von Cadence Genus erzeugten Konstantenmultiplizierern benötigen die von kmult erzeugten Konstantenmultiplizierer etwa 60% weniger Chipfläche. Der große Unterschied kommt durch die stark reduzierte Anzahl von Addierern in dem von kmult erzeugten Design zustande, die Anzahl der Gatter an sich reduzierte sich um nur 6%. Aufgrund dieser Vorteile wurde dieser Ansatz für die DFT-Implementierung verwendet.

3.2.1 CORDIC

Als Alternative zu Konstantenmultiplizierern wurde CORDIC in Erwägung gezogen. CORDIC steht für **Co**ordinate **R**otation **Di**gital **C**omputer. Dieser Algorithmus kann genutzt werden, um trigonometrische Funktionen und Exponentialfunktionen durch Addition und Schiebeoperationen zu approximieren und wäre geeignet, die DFT-Koeffizienten zu erzeugen. Da es sich um einen iterativen Algorithmus handelt, wären für die 4 Stellen Genauigkeit, die für 10Bit Nachkommapräzision benötigt werden, mindestens 10 Iterationen notwendig, wie in der Arbeit von Helck in Kap. 5.4 [1] ermittelt wurde. Der Zeitaufwand für 10 Iterationen je Sample verletzt für größere Matrixgrößen die Latenzanforderung, weswegen dieser Ansatz verworfen wurde.

3.3 Koeffizientenmuster der 2D-DFT

Unabhängig davon, ob Konstantenmultiplizierer oder echte Multiplizierer in Kombination mit einem ROM zum Einsatz kommen, müssen immer noch die Indizes des jeweiligen Koeffizienten ermittelt werden. Hier könnte wieder ein ROM zum Einsatz kommen, jedoch wurde eine Alternative entdeckt. Da die Koeffizienten, mit denen multipliziert wird, Sin- und Cos-Schwingungen entsprechen, ist die Abfolge der Koeffizientenindizes ebenfalls periodisch. Dies bedeutet, dass sich die Indizes der Koeffizienten durch einen Zähler generieren lassen. Ein Zähler hat gegenüber einem ROM den Vorteil eines geringeren Flächenbedarfs. Bei der Betrachtung der Koeffizienten innerhalb des relevanten Größenbereichs lassen sich 4 generelle Muster erkennen. Diese übergeordneten Muster verfügen

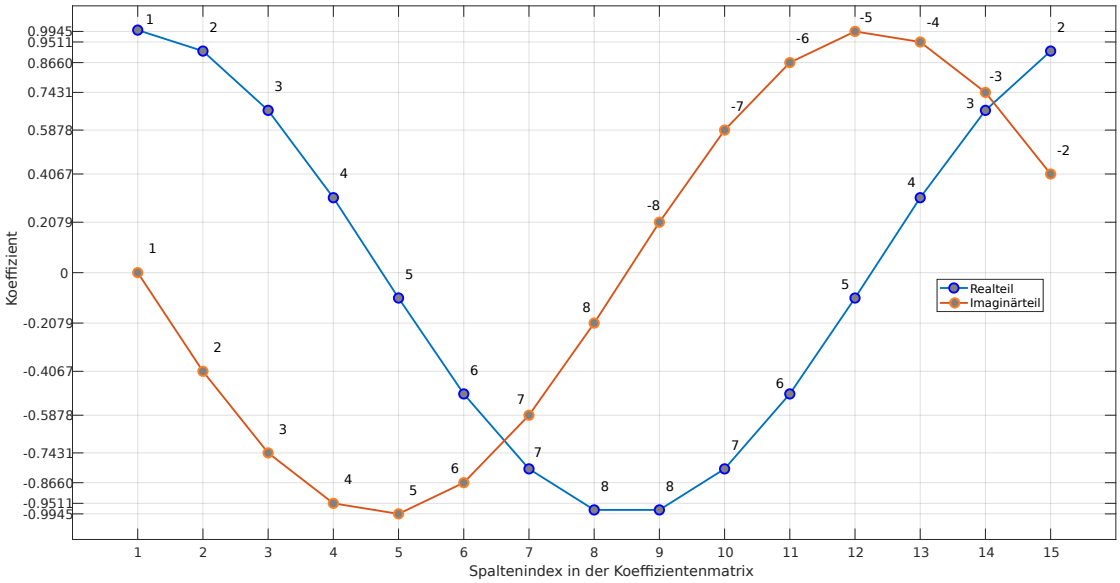

Abbildung 3.2: Koeffizienten für Ausgangspunkt (2, 2) einer 15x15 Punkte 2D-DFT

zudem noch über Permutationen, die berücksichtigt werden müssen. Für jedes Muster und seine jeweiligen Permutationen ist eine individuelle Zählerlogik erforderlich, um die Berechnungen durchzuführen.

3.3.1 Koeffizientenfolge

Bei der Betrachtung der Koeffizientenmuster wird zunächst die Abfolge der Koeffizienten betrachtet. Diese Koeffizientenfolgen sind weitgehend unabhängig von der Art des Musters. Jedem betragsmäßig einzigartigen Koeffizienten wird ein Index zugeordnet. Wenn das Vorzeichen eines Koeffizienten verglichen mit seinem ersten Auftreten unterschiedlich ist, wird dies durch einen negativen Index gekennzeichnet. Dies ist in Abbildung 3.2 zu sehen. Diese Koeffizientenindizes sind die Ziffern an den Markern. Es ist erkennbar, dass betragsmäßig nur 8 Koeffizientenindizes statt 15 existieren und dass sich diese mit einem linearen Zähler abbilden lassen. Negative Koeffizientenindizes symbolisieren einen Koeffizienten mit invertiertem Vorzeichen, im Vergleich zu dessen normalem Vorzeichen.

$$
\begin{array}{rcccccccccccccc}
\mathit{Real}: & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 8 & 7 & 6 & 5 & 4 & 3 & 2 \\
\mathit{Imag}: & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & -8 & -7 & -6 & -5 & -4 & -3 & -2
\end{array}
$$

Koeffizientenindexfolge für Ausgangspunkt (2, 2) einer 15x15 Punkte DFT

In diesem Fall gibt es beim höchsten Index 8 einen Haltepunkt, da zweimal hintereinander derselbe Index vorkommt. Zudem ist das Vorzeichen beim Imaginärteil (Imag) ab dem Haltepunkt invertiert. Das Zählerverhalten ist lediglich an den Extrempunkten (höchster und niedrigster Index) unterschiedlich.

An den Extrempunkten können Haltepunkte und Vorzeichenwechsel vorkommen, oder es wird nur die Zählrichtung geändert. Untere oder oberen Haltepunkte treten für Real- und Imaginärteil stets gemeinsam auf. Vorzeicheninvertierung tritt bei Real- und Imaginärteil unabhängig auf. Dies führt zu 16 möglichen Permutationen für die Koeffizientenindexzähler.

Es existieren jedoch auch Ausgangspunkte, bei denen die Koeffizienten für die zweite Halbwelle nicht dieselben Punkte treffen, daher müssen in diesen Fällen alle Punkte des Sinus oder Cosinus gespeichert werden. Das entstehende Zählmuster ist dann kein Auf- und Ab mehr sondern entspricht einem Zähler, der bei Erreichen des höchsten Koeffizientenindex wieder zurückgesetzt wird. Der Anteil dieser Fälle liegt bei Matrixgrößen, bei denen diese Art Muster vorkommt, bei etwa 10%. Zudem weisen die Ausgangspunkte, bei denen dieses Muster vorkommt, weitere Anomalien auf, welche in Abschnitt 3.3.5 näher erläutert werden.

Analog zum Beispiel oben, hier ein Beispiel für eine Zählfolge mit Rücksetzung:

Real :	1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8
Imag :	1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8

Koeffizientenindexfolge mit Rücksetzung

Nach der Betrachtung der Koeffizientenabfolge folgt nun die Betrachtung der verschiedenen Koeffizientenmuster.

3.3.2 Trivialmuster

Das Trivialmuster weist lediglich einen Koeffizienten auf, zumeist Faktoren von 1 oder 0. Permutationen dieses Musters sind die zeilen- und spaltenweise Invertierung des Vorzeichens des Koeffizienten, diese können auch gemeinsam auftreten. Für dieses Muster ist kein Zähler erforderlich.

Trivialmuster (einfachster Fall)

$$\begin{vmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{vmatrix} \quad \begin{vmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{vmatrix}$$

Real *Imag*

Trivialmuster mit wechselnden Vorzeichen bei Spaltenwechsel

$$\begin{vmatrix} 1 & -1 & 1 & -1 & 1 \\ 1 & -1 & 1 & -1 & 1 \\ 1 & -1 & 1 & -1 & 1 \\ 1 & -1 & 1 & -1 & 1 \\ 1 & -1 & 1 & -1 & 1 \end{vmatrix} \quad \begin{vmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{vmatrix}$$

Real *Imag*

Trivialmuster mit wechselnden Vorzeichen bei Zeilenwechsel

$$\begin{vmatrix} 1 & 1 & 1 & 1 & 1 \\ -1 & -1 & -1 & -1 & -1 \\ 1 & 1 & 1 & 1 & 1 \\ -1 & -1 & -1 & -1 & -1 \\ 1 & 1 & 1 & 1 & 1 \end{vmatrix} \quad \begin{vmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{vmatrix}$$

Real *Imag*

3.3.3 Zeilenmuster

Zeilenmuster bestehen, wie der Name vermuten lässt, aus sich wiederholenden Zeilen. Einfache Zeilenmuster treten grundsätzlich immer in der ersten Zeile der Ausgangspunkte auf. Eine Permutation des einfachen Zeilenmusters ist eine zeilenweise Invertierung der Vorzeichen aller Koeffizienten. Komplexe Zeilenmuster zeichnen sich durch eine Phasenverschiebung beim Zeilenwechsel aus, diese ist relativ zur vorherigen Zeile und konstant.

Einfaches Zeilenmuster

$$\left|\begin{array}{rrrrr} 1 & 2 & 3 & 2 & 1 \\ 1 & 2 & 3 & 2 & 1 \\ 1 & 2 & 3 & 2 & 1 \\ 1 & 2 & 3 & 2 & 1 \\ 1 & 2 & 3 & 2 & 1 \end{array}\right| \quad \left|\begin{array}{rrrrr} 1 & 2 & 3 & -2 & -1 \\ 1 & 2 & 3 & -2 & -1 \\ 1 & 2 & 3 & -2 & -1 \\ 1 & 2 & 3 & -2 & -1 \\ 1 & 2 & 3 & -2 & -1 \end{array}\right|$$

Real *Imag*

Einfaches Zeilenmuster mit Vorzeichenwechsel

$$\left|\begin{array}{rrrrr} 1 & 2 & 3 & 2 & 1 \\ -1 & -2 & -3 & -2 & -1 \\ 1 & 2 & 3 & 2 & 1 \\ -1 & -2 & -3 & -2 & -1 \\ 1 & 2 & 3 & 2 & 1 \end{array}\right| \quad \left|\begin{array}{rrrrr} 1 & 2 & 3 & -2 & -1 \\ -1 & -2 & -3 & 2 & 1 \\ 1 & 2 & 3 & -2 & -1 \\ -1 & -2 & -3 & 2 & 1 \\ 1 & 2 & 3 & -2 & -1 \end{array}\right|$$

Real *Imag*

Zeilenmuster mit Phasenverschiebung um 1

$$\left|\begin{array}{rrrrr} 1 & 2 & 3 & 2 & 1 \\ 2 & 3 & 2 & 1 & 2 \\ 3 & 2 & 1 & 1 & 2 \\ 2 & 1 & 1 & 2 & 3 \\ 1 & 1 & 2 & 3 & 2 \end{array}\right| \quad \left|\begin{array}{rrrrr} 1 & 2 & 3 & -2 & -1 \\ 2 & 3 & -2 & -1 & 1 \\ 3 & -2 & -1 & 1 & 2 \\ -2 & -1 & 1 & 2 & 3 \\ -1 & 1 & 2 & 3 & -2 \end{array}\right|$$

Real *Imag*

3.3.4 Spaltenmuster

Einfache Spaltenmuster sind analog zu den einfachen Zeilenmustern. Sie bestehen aus sich wiederholenden Spalten. Auch diese Muster verfügen über eine Permutation, bei der in jeder neuen Spalte die Vorzeichen invertiert werden. Einfache Spaltenmuster treten grundsätzlich immer in der ersten Spalte der Ausgangspunkte auf. Komplexe Spaltenmuster verhalten sich ebenfalls analog zu den komplexen Zeilenmustern, indem jede neue Spalte eine konstante Phasenverschiebung zu der vorangegangenen Spalte aufweist.

Einfaches Spaltenmuster

$$\left|\begin{matrix} 1 & 1 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 & 2 \\ 3 & 3 & 3 & 3 & 3 \\ 2 & 2 & 2 & 2 & 2 \\ 1 & 1 & 1 & 1 & 1 \end{matrix}\right| \quad \left|\begin{matrix} 1 & 1 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 & 2 \\ 3 & 3 & 3 & 3 & 3 \\ -2 & -2 & -2 & -2 & -2 \\ -1 & -1 & -1 & -1 & -1 \end{matrix}\right|$$

Real *Imag*

Einfaches Spaltenmuster mit Vorzeichenwechsel

$$\left|\begin{matrix} 1 & -1 & 1 & -1 & 1 \\ 2 & -2 & 2 & -2 & 2 \\ 3 & -3 & 3 & -3 & 3 \\ 2 & -2 & 2 & -2 & 2 \\ 1 & -1 & 1 & -1 & 1 \end{matrix}\right| \quad \left|\begin{matrix} 1 & -1 & 1 & -1 & 1 \\ 2 & -2 & 2 & -2 & 2 \\ 3 & -3 & 3 & -3 & 3 \\ -2 & 2 & -2 & 2 & -2 \\ -1 & 1 & -1 & 1 & -1 \end{matrix}\right|$$

Real *Imag*

Spaltenmuster mit Phasenverschiebung um 1

$$\left|\begin{matrix} 1 & 2 & 3 & 2 & 1 \\ 2 & 3 & 2 & 1 & 1 \\ 3 & 2 & 1 & 1 & 2 \\ 2 & 1 & 1 & 2 & 3 \\ 1 & 1 & 2 & 3 & 2 \end{matrix}\right| \quad \left|\begin{matrix} 1 & 2 & 3 & -2 & -1 \\ 2 & 3 & -2 & -1 & 1 \\ 3 & -2 & -1 & 1 & 2 \\ -2 & -1 & 1 & 2 & 3 \\ -1 & 1 & 2 & 3 & -2 \end{matrix}\right|$$

Real *Imag*

3.3.5 Zeilen- und Spaltenmuster

Zeilen- und Spaltenmuster stellen die komplizierteste Form von Koeffizientenmustern dar. Dieses Muster erscheint erstmalig bei einer Matrixgröße von 12x12, tritt aber nicht immer bei allen Matrixgrößen oberhalb von 12x12 auf. Dies ist der einzige Mustertyp, bei dem die Zählfolge mit Überlauf vorkommt. Anders als bei den anderen Mustertypen, kann ein Ausgangspunkt beim Zeilen- und Spaltenmuster mehrere unterschiedliche Zählfolgen haben. Außerdem treten bei diesem Muster Zeilenwiederholungen auf, in manchen Fällen mit invertierten Vorzeichen. Die beste Betrachtungsweise ist daher, das Muster als einen

Satz einzigartiger Zeilen zu behandeln, der sich in regelmäßigem Abstand wiederholt. Das angegebene Beispiel zeigt diesen Mustertyp, wie er bei einer 15x15 Matrix vorkommt, zu Illustrationszwecken auf 8x8 gekürzt. Die drei farbig hinterlegten Zeilen wiederholen sich. Die 3. Zeile (Rot) entspricht der 2. mit umgekehrter Zählrichtung, beim Imaginärteil außerdem mit umgekehrten Vorzeichen. Das zweite Beispiel enthält in der 4. Zeile eine invertierte Zeile (Blau).

1	2	3	4	1	2	3	4
5	6	7	8	5	6	7	8
5	8	7	6	5	8	7	6
1	2	3	4	1	2	3	4
5	6	7	8	5	6	7	8
5	8	7	6	5	8	7	6
1	2	3	4	1	2	3	4
5	6	7	8	5	6	7	8

Real

1	2	3	4	1	2	3	4
5	6	7	8	5	6	7	8
−5	−8	−7	−6	−5	−8	−7	−6
1	2	3	4	1	2	3	4
5	6	7	8	5	6	7	8
−5	−8	−7	−6	−5	−8	−7	−6
1	2	3	4	1	2	3	4
5	6	7	8	5	6	7	8

Imag

1	2	−1	2	1	2	−1	2
3	4	−3	−4	3	4	−3	−4
−3	4	3	−4	−3	4	3	−4
−1	−2	1	−2	−1	−2	1	−2
3	4	−3	−4	3	4	−3	−4
−3	4	3	−4	−3	4	3	−4
1	2	−1	2	1	2	−1	2
3	4	−3	−4	3	4	−3	−4

Real

1	2	1	−2	1	2	1	−2
3	4	−3	−4	3	4	−3	−4
3	−4	−3	4	3	−4	−3	4
−1	−2	−1	2	−1	−2	−1	2
3	4	−3	−4	3	4	−3	−4
3	−4	−3	4	3	−4	−3	4
1	2	1	−2	1	2	1	−2
3	4	−3	−4	3	4	−3	−4

Imag

4 Implementierung der Hardwarearchitektur

Die Hardwarearchitektur wurde als Modul für das Testsystem entwickelt, welches Helck [1] für das ISAR-Projekt entwickelt hat. Eine Übersicht des entwickelten Moduls ist in Abbildung 4.1 zu sehen. Die Architektur ist in zwei Hardwarebeschreibungssprachen geschrieben und besteht aus einem in Verilog und einem in VHDL geschriebenen Teil. Das in VHDL geschriebene Testsystem gibt das äußere Interface des Moduls vor. Der verwendete Generator für die Konstantenmultiplizierer liefert diese als Verilog-Datei, daher wurde Verilog für alle signalverarbeitenden Module verwendet. Dies vermeidet Probleme durch möglicherweise vorhandene Unterschiede in den Implementierungen der Zahlensysteme. Zudem vereinfacht die Zweiteilung die Instanziierung der vielen Verilog-Module innerhalb eines einzigen Verilog-Toplevelmoduls.
Das VHDL-Modul übernimmt die folgenden Funktionen: Bus-Arbitrierung, Speicheradressierung, Erzeugung von Steuersignalen und die Skalierung der Ergebnisse. Das Verilog-Toplevelmodul instanziiert die Koeffizientenmodule, erzeugt Steuersignale für die Koeffizientenmodule und enthält den Ausgangs-Multiplexer.
Die Koeffizientenmodule bestehen aus dem komplexen Multiplizierer, einem Akkumulator, einem ROM für Vorzeichen und der Zählerlogik, welche die Koeffizientenfolge erzeugt. Es wurden zunächst zwei Datenpfade entwickelt. Der erste mit einer mit einer vollparallelen komplexen Multiplikation in einem Takt, der zweite mit einer sequenziellen Variante, die zwei Taktzyklen benötigt. Die sequenzielle Verarbeitung reduziert den Durchsatz um 50%, bringt aber auch eine Reduktion des Flächen-/Logikbedarfs von beinahe 50% mit sich. Da der reduzierte Durchsatz des sequenziellen Verfahrens trotzdem hoch genug ist, um die Latenzanforderung zu erfüllen, wurde die Entwicklung des parallelen Datenpfads eingestellt. Die Implementierung des parallelen Datenpfades ist daher nur zum Teil umgesetzt und wenig getestet, liefert aber für einige Koeffizientenmuster funktionierende Hardware. Alle im Design vorkommenden Reset-Signale sind asynchron und low-aktiv, da die verwendete Standardzellen-Bibliothek dieser Konvention folgt.

Durchsatzformeln für eine MxM Transformation:

$$\text{Parallel:} \quad \text{Taktzyklen} = M + 2 \tag{4.1}$$

$$\text{Sequenziell:} \quad \text{Taktzyklen} = 2M + 3 \tag{4.2}$$

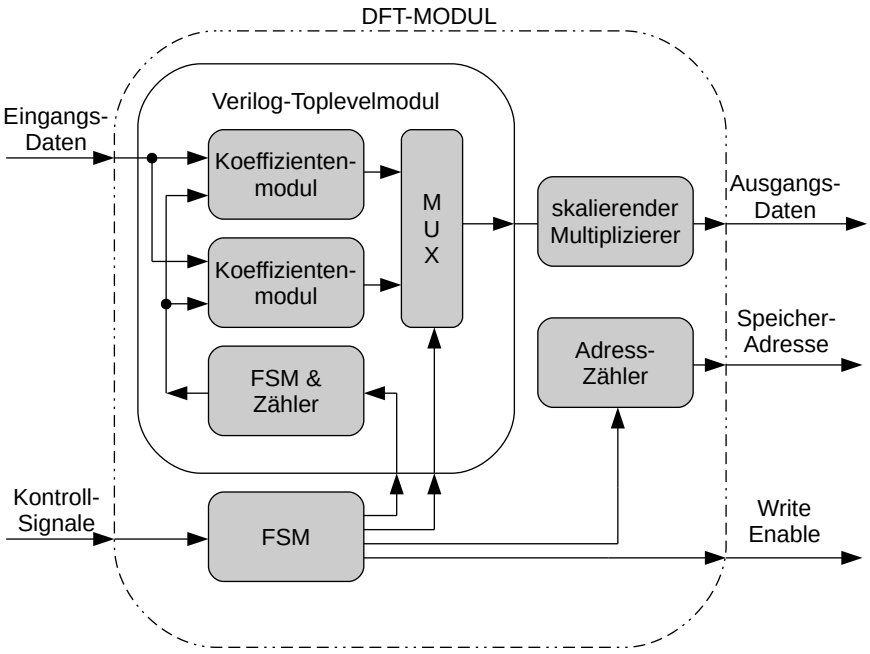

Abbildung 4.1: Übersicht Hardwarearchitektur

4.1 Koeffizientenmodul

Ein Koeffizientenmodul berechnet einen spezifischen Ausgangspunkt. Je nach Koeffizientenmuster des Punkts enthält das Koeffizientenmodul unterschiedliche Logik. Einige Komponenten werden jedoch bei allen Musterarten verwendet.

4.1.1 Gemeinsame Komponenten

Alle Arten von Koeffizientenmodulen haben stets die folgenden Bestandteile:

- Multiplizierermodul (siehe 4.2)

- Akkumulator (siehe 4.3)
- Reset-Logik
- sample_strobe-Logik
- Koeffizientenvorzeichen-ROM

Reset-Logik

Die Reset-Logik dient dazu, die internen Register bei Resets oder beim Start einer neuen Berechnung in einen definierten Ausgangszustand zu bringen. Beim Sequenziellen Verfahren hat die Reset-Logik ein Register, welches den START-Eingang synchronisiert. Ein UND-Gatter verbindet den invertierenden Ausgang des START-Registers und den regulären Reset, wenn einer der beiden auf „low“ gesetzt wird, werden die Register zurückgesetzt. Bei der parallelen Variante wird das START-Signal direkt auf einen invertierenden Eingang des UND-Gatters gelegt. Es wurde nicht für alle Register im Design ein Reset vorgesehen. Auf einem FPGA hat das Weglassen von Resets keinerlei Vorteile, bei einer ASIC-Implementierung jedoch sind Register ohne Reset flächenmäßig kleiner. Bei Pipelining-Registern, die keinerlei Einfluss auf Kontrollsignale haben, wurde kein Reset vorgesehen. Flächenmäßig betrachtet sind Register mit Reset 38% größer, daher spart das Weglassen der Resets aufgrund der Anzahl der Pipelining-Register eine beträchtliche Fläche. Nachteil der Einsparung ist, dass dadurch bei der Simulation einige Signale am Anfang der Berechnung als undefiniert angezeigt werden. Das Reset-Modul hätte in das Verilog-Toplevelmodul integriert werden können, wurde aber aufgrund seiner sehr kleinen Fläche von zwei Gattern lokal im Koeffizientenmodul belassen, um das asynchrone Reset-Signal nicht über viele Module verteilen zu müssen. Wenn man den Verdrahtungsaufwand betrachtet, ist es günstiger, das Reset-Signal lokal zu erzeugen, als es mit einer globalen Reset-Einheit verbinden zu müssen.

sample_strobe-Logik:

Die sample_strobe-Logik erzeugt ein Signal, welches bei jedem Taktzyklus den Zustand wechselt und synchron zum START-Signal ist. Das sample_strobe-Signal wird nur benötigt, wenn für die Multiplikation das sequenzielle Verfahren verwendet wird. Die Logik

besteht aus einem Register, dessen Eingang mit seinem invertierendem Ausgang verbunden ist und über einen Reset-Eingang verfügt. Dieses Signal steuert z.b. die Enable-Eingänge von Registern an, die nur bei jedem zweiten Taktzyklus aktiv sein müssen. Wie das Reset-Modul ist auch dieses Modul redundant angelegt und wird aus denselben Gründen lokal im Koeffizientenmodul belassen.

Koeffizientenvorzeichen-ROM:

Das Vorzeichen-ROM speichert die Vorzeichen der Koeffizienten, da diese den Multiplizierern separat zugeführt werden und nicht inhärent im Konstantenmultiplizierer integriert sind. Je nach Koeffizientenmodul, kann dieses ROM auch nur einen Eintrag haben (Trivialfall). Allgemein enthält es für jeden Koeffizientenindex das entsprechende Vorzeichen. Je nach Koeffizientenmuster wird das Vorzeichen-ROM noch um weitere Einträge erweitert, zum Beispiel um den Koeffizientenindex einer Zeile.

4.1.2 Realisierung von Trivialmustern

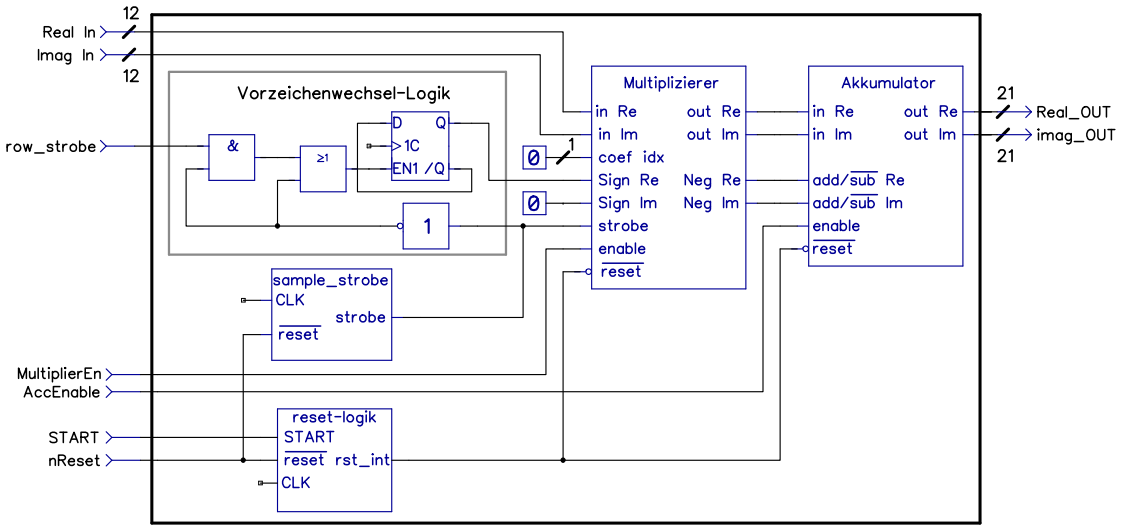

Abbildung 4.2: Koeffizientenmodul für Trivialmuster (vereinfacht)

Für Trivialmuster wird kein Koeffizientenindexzähler benötigt, daher wird der Index fest auf „0“ gesetzt. Wenn die Vorzeichen der Koeffizienten keinen Veränderungen unterliegen, werden diese ebenfalls fest durch eine Konstante definiert. Für Fälle mit wechselnden Vorzeichen wird ein Register erzeugt, welches die Vorzeichen invertiert. Das Register wird

je nach auftretendem Muster bei Zeilenwechsel, Spaltenwechsel oder beidem invertiert (siehe Abb. 4.2).

4.1.3 Realisierung von Zeilenmustern

Die Koeffizientenlogik für Zeilenmuster lässt sich in einfache und komplexe Varianten unterteilen.

Realisierung von einfachen Zeilenmustern

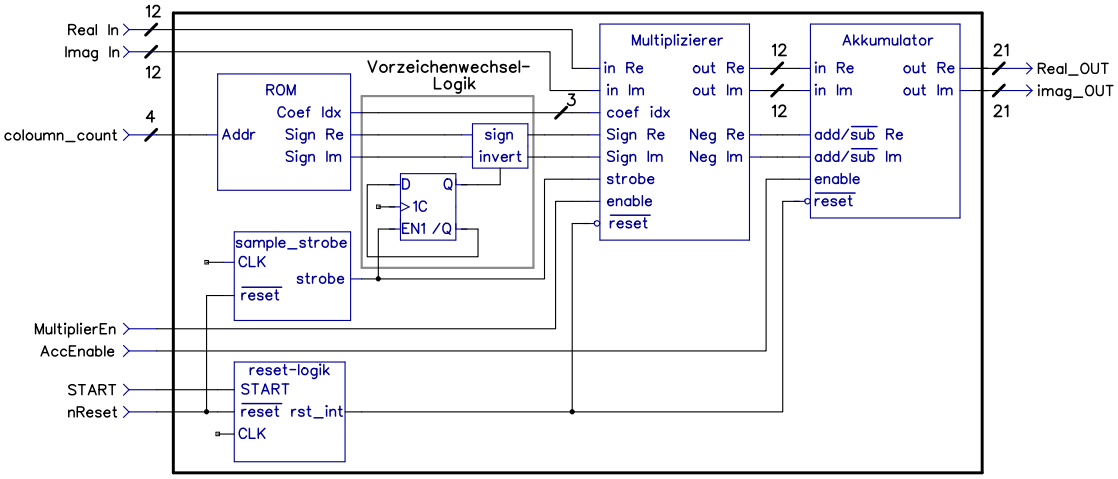

Abbildung 4.3: Koeffizientenmodul für einfache Zeilenmuster (vereinfacht)

Einfache Zeilenmuster besitzen in jeder Zeile dieselbe Abfolge von Koeffizienten, was im Umkehrschluss bedeutet, dass jede Spalte nur einen Koeffizienten besitzt. Um diese Muster nachzubilden (siehe Abb. 4.3), wird das Vorzeichen-ROM um die Koeffizientenindizes der Spalten erweitert. Der Spaltenzähler des Verilog-Toplevelmoduls adressiert direkt das ROM und wählt Koeffizientenindex und Vorzeichen für die aktuelle Spalte aus. Es ist auch eine Zählerimplementierung möglich, doch im Vergleich hat die ROM-basierte Implementierung einen geringeren Flächenbedarf. Es wird erwartet, dass sich dieser Vorteil für größere Transformationsgrößen, die außerhalb des für das Projekt relevanten Bereiches liegen, umkehren wird. Allgemein benötigen die binären Koeffizientenindexzähler $log_2(M)$ Bits zur Darstellung der Koeffizientenindizes. Ein ROM muss jeweils einen Eintrag für jeden Punkt der Zeile beinhalten und wächst daher linear mit der Transformationsgröße M.

Realisierung von komplexen Zeilenmustern

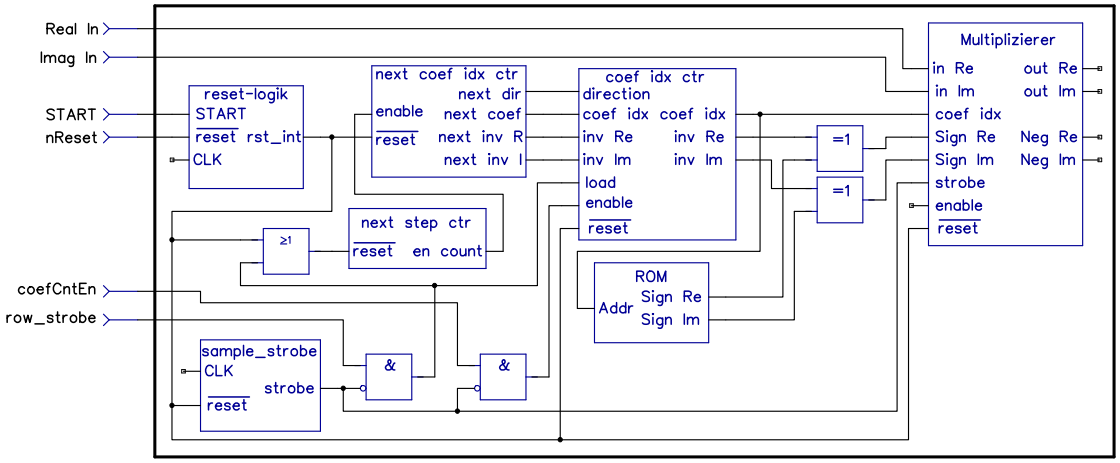

Abbildung 4.4: Koeffizientenlogik für komplexe Zeilenmuster (vereinfacht)

Komplexe Zeilenmuster zeichnen sich dadurch aus, dass die Koeffizientenfolge der nächsten Zeile eine konstante Phasenverschiebung gegenüber der aktuellen Zeile aufweist. Um einen Vergleich mit den ROM-basierten Ansätzen zu ermöglichen, wurde diese Koeffizientenmustervariante mit Zählern umgesetzt, auch wenn diese flächenmäßig minimal größer ausfallen. Eine vereinfachte Darstellung der Logik ist in Abbildung 4.4 zu sehen. Ein Zähler erzeugt den Koeffizientenindex für die aktuelle Zeile und bestimmt, ob Vorzeichen invertiert werden. Da für die direkte Berechnung des Zählerzustands der nächsten Zeile keine Lösung gefunden wurde, bereitet ein weiterer Zähler den Zählerzustand für die nächste Zeile vor. Der Zählerzustand besteht aus dem Invertierungsstatus der Vorzeichen, Zählrichtung und Zählerstand. Der vorbereitende Zähler besitzt dasselbe Verhalten wie der Koeffizientenindexzähler. Der Enable-Eingang dieses vorbereitenden Zählers wird durch einen dritten Zähler gesteuert. Dieser dritte Zähler wird bei Zeilenwechsel (und Reset) mit der Anzahl Stellen, um die die Zeilen phasenverschoben werden, geladen und zählt abwärts. Dadurch führt der vorbereitende Zähler nur die für die Phasenverschiebung nötige Anzahl an Zählschritten durch. Nach Ablauf der Zählschritte entspricht der interne Zustand des vorbereitenden Zählers dem korrekten Zählerzustand für den Beginn der nächsten Zeile. Da die Phasenverschiebung modulo der Zeilenlänge ist, ist die Erzeugung des nächsten Zustands auf jeden Fall vor dem Zeilenende abgeschlossen. Bei Zeilenwechsel übernimmt der Koeffizientenindexzähler den internen Zustand des vorbereitenden Zählers.

4.1.4 Realisierung von Spaltenmustern

Für Spaltenmuster gibt es, wie bei den Zeilenmustern, einfache und komplexe Varianten der Koeffizientenlogik.

Realisierung von einfachen Spaltenmustern

Einfache Spaltenmuster zeichnen sich dadurch aus, dass alle Spalten die gleiche Koeffizientenfolge besitzen. Sie werden analog zu den einfachen Zeilenmustern über ein Koeffizientenindex-ROM erzeugt. Einziger Unterschied ist, dass die Adressierung nicht über den Spaltenzähler, sondern über den Zeilenzähler erfolgt.

Realisierung von komplexen Spaltenmustern

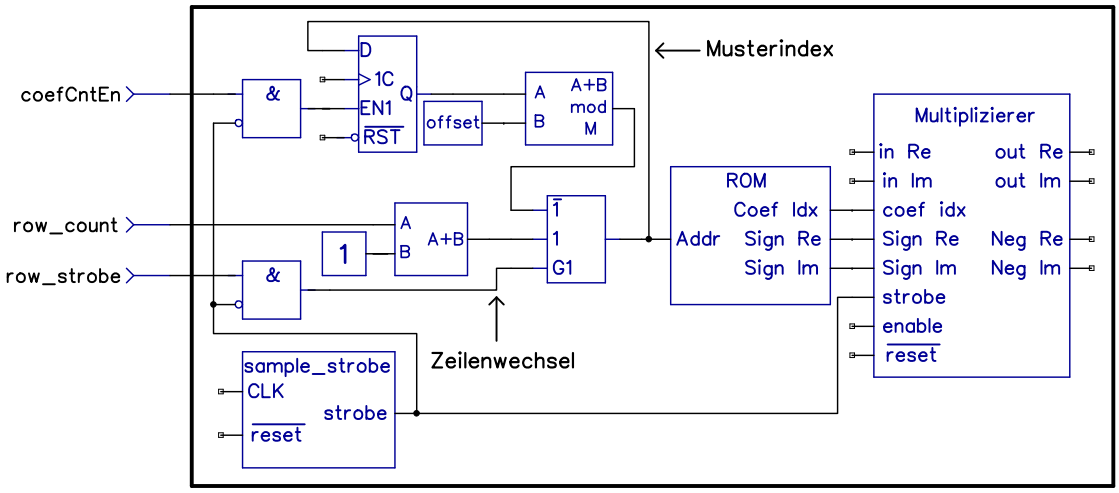

Abbildung 4.5: Koeffizientenlogik für komplexe Spaltenmuster (vereinfacht)

Komplexe Spaltenmuster haben analog zu den komplexen Zeilenmustern eine Phasenverschiebung zwischen den Spalten. Die Koeffizientenfolge innerhalb der Spalten ist jedoch immer gleich.

Wie schon in der Sektion über komplexe Zeilenmuster beschrieben, wurde keine Lösung zur direkten Erzeugung des nächsten Zählerzustands gefunden. Beim komplexen Spaltenmuster muss die Berechnung der Phasenverschiebung bei jeder neuen Spalte und damit

jedem neuen Sample erfolgen, nicht nur beim Zeilenwechsel. Die Zählerlösung der Zeilenmuster kann daher hier nicht zum Einsatz kommen, weil nur ein bis maximal zwei Taktzyklen zur Verfügung stehen.

Für die folgenden Erklärungen ist es wichtig zwischen dem Musterindex und dem Koeffizientenindex zu unterscheiden. Der Koeffizientenindex indiziert die einzigartigen Koeffizienten, während der Musterindex die Position in der Koeffizientenfolge $\{0 \dots M-1\}$ angibt. Die beiden Indizes ineinander zu übersetzen ist das Hauptproblem, verkompliziert durch die Vorzeicheninvertierungen. Bei der Zählerlösung ist diese Übersetzung nicht erforderlich, da die Zähler in Koeffizientenindizes zählen und auch die Invertierung der Vorzeichen richtig behandeln.

Dieses Problem wurde durch die Verwendung eines ROM in Kombination mit einem Modulo-Addierer gelöst (siehe Abb. 4.5). Um den Musterindex der nächsten Spalte zu berechnen, wird die Phasenverschiebung zum aktuellen Musterindex addiert, modulo der Zeilenlänge M. Die Übersetzung wird über ein ROM erledigt, welches den Musterindex in den Koeffizientenindex und die zugehörigen Vorzeichen übersetzt. Beim Zeilenwechsel wird der Musterindex auf den Zeilenzählerstand plus eins gesetzt. Da die erste Spalte als Referenz für die Koeffizientenfolge dient, sind die Musterindizes für die erste Spalte gleich der Zeilennummer.

Beispiel für komplexes Spaltenmuster 9x9 Punkt: (2, 4)

Realteil:

0	3	3	0	3	3	0	3	3
1	4	2	1	4	2	1	4	2
2	4	1	2	4	1	2	4	1
3	3	0	3	3	0	3	3	0
4	2	1	4	2	1	4	2	1
4	1	2	4	1	2	4	1	2
3	0	3	3	0	3	3	0	3
2	1	4	2	1	4	2	1	4
1	2	4	1	2	4	1	2	4

Imaginärteil:

0	3	−3	0	3	−3	0	3	−3
1	4	−2	1	4	−2	1	4	−2
2	−4	−1	2	−4	−1	2	−4	−1
3	−3	0	3	−3	0	3	−3	0
4	−2	1	4	−2	1	4	−2	1
−4	−1	2	−4	−1	2	−4	−1	2
−3	0	3	−3	0	3	−3	0	3
−2	1	4	−2	1	4	−2	1	4
−1	2	−4	−1	2	−4	−1	2	−4

Die Koeffizienten wurden in Koeffizientenindizes $\{0 \dots 4\}$ übersetzt. Die Phasenverschiebung zwischen den Spalten beträgt 3. Farbig hinterlegt sind die Informationen, die im

Zeile	Spalte	Muster-Index	Koeffizienten-Index	Muster index + Offset Mod(M)
1	1	0	0	3
1	2	3	3	6
1	3	6	3	0
1	4	0	0	3
2	1	1	1	4
2	2	4	4	7
2	3	7	2	1
2	4	1	1	4

Tabelle 4.1: Beispiel 9x9 Koeffizientenindex: (2, 4)

ROM gespeichert werden, der Koeffizientenindex und die Vorzeichen von Real- und Imaginärteil. In Tabelle 4.1 wird der Ablauf der Berechnungen exemplarisch für die ersten vier Spalten der ersten beiden Zeilen gezeigt.

4.1.5 Zeilen- und Spaltenmuster

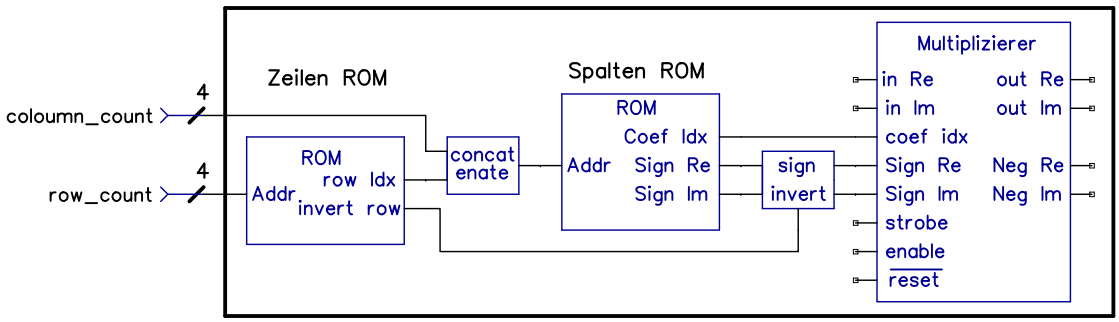

Abbildung 4.6: Verilog-Toplevelmodul (vereinfacht)

Dieser Mustertyp weist das größte Spektrum an Permutationen auf, was die Nachbildung erschwert. Um diese Arbeit innerhalb des Zeitrahmens abschließen zu können, musste auf eine detaillierte Nachbildung dieses Mustertyps verzichtet werden. Daher wurde ein ROM-basierter Ansatz entwickelt, wie in Abbildung 4.6 dargestellt. Dieser besteht aus einem Spalten-ROM, welcher die Koeffizientenindizes und Vorzeichen für jede auftretende einzigartige Zeile enthält. Die Adressierung erfolgt durch den Spaltenzähler und das Zeilen-ROM. Das Zeilen-ROM ordnet der Zeilennummer die entsprechende einzigartige Zeile im Spalten-ROM zu. Außerdem enthält es ein Bit, welches erlaubt die Vorzeichen einer Zeile zu invertieren. Da invertierte Zeilen häufig vorkommen, ist dies eine einfache

Optimierung, um die Anzahl der zu speichernden, einzigartigen Zeilen zu reduzieren.

4.2 Komplexer Multiplizierer

Der von Voronenko und Püschel [5] entwickelte Konstantenmultiplizierer-Generator kmult wird benutzt, um die Konstantenmultiplizierer für die benötigten Koeffizienten zu erzeugen. Detailinformationen dazu sind in Kapitel 5.3.4 der Arbeit zu finden. Die erzeugten Verilog-Module sind vollständig kombinatorisch und haben je einen Ein- und Ausgang für die Daten im Zweierkomplement Format sowie einen zweiten Eingang (control), welcher zwischen den realisierten Koeffizienten umschaltet.

Der Generator wurde so konfiguriert, dass die Konstantenbits der Anzahl der Nachkommabits entspricht (Constantbitwidth=Fractionbitwidth). Das bedeutet, die Faktoren bestehen nur aus Nachkommabits. In diesem Modus liefern die Konstantenmultiplizierer das Ergebnis im selben Zahlenformat, wie die Eingangsdaten und dementsprechend auch mit derselben Bitbreite. Dies ist möglich, weil der Betrag der Koeffizienten einer DFT immer im Bereich $[0 \ldots 1]$ liegt. Die von kmult erzeugten Multiplizierer liefern auch für Konstanten von 1 richtige Ergebnisse, was normalerweise bei einem Faktor ohne Vorkommabits unmöglich ist, da nur Zahlen kleiner 1 darstellbar sind. Abweichend von anderen Arbeiten im ISAR Projekt, wird hier somit nicht mit S1Q10 Konstanten gearbeitet, sondern mit Konstanten im Q10 Format. Das Q-Format wird im Lehrbuch Digitaltechnik [4] Kap. 5.6 erklärt.

Für die parallele Variante des komplexen Multiplizierers (Abb. 4.7) werden jeweils zwei Konstantenmultiplizierer für die Real- und Imaginärteile der Koeffizienten benötigt, bei der sequenziellen Variante (Abb. 4.8) nur einer. Da die Konstantenmultiplizierer den größten Teil der Logikbedarfs eines Koeffizientenmoduls ausmachen, ergibt sich eine beinahe 50%-ige Reduktion der Logik bei Verwendung der sequenziellen Variante. Die Konstantenmultiplizierer führen nur reale Multiplikationen durch, daher muss die komplexe Multiplikation aus reellen Multiplikationen gebildet werden.

4.2.1 Ablauf der Multiplikation

Eine wichtige Optimierung liegt in der Verarbeitung der Koeffizientenvorzeichen. Denn anstatt die Vorzeichen des Samples über die Bildung des Zweierkomplements zu inver-

tieren, wurde hier eine Lösung gefunden, die Vorzeichenwechsel durch ein Umstellen der Formeln der komplexen Multiplikation zu erzielen. Die Koeffizientenvorzeichen werden dem Multiplizierer als separate Eingangssignale zugeführt. Für diese Funktionalität muss auch der Akkumulator mit einbezogen werden. Er muss so ausgelegt werden, dass er auch subtrahieren kann. Dies ermöglicht zusammen mit der Logik im Multiplizierer die externen Koeffizientenvorzeichen Eingänge. Für ein und zwei Takte pro Sample fällt die Multiplizierer-Logik unterschiedlich aus.

Parallele Multiplikation

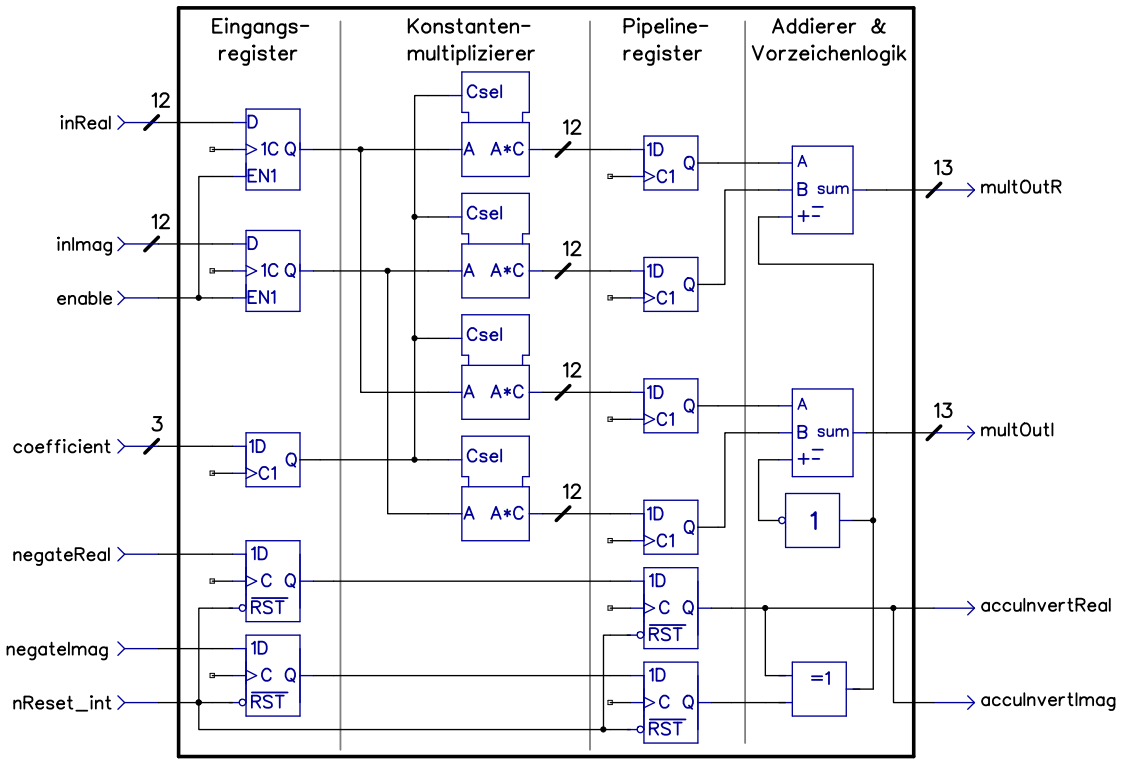

Abbildung 4.7: Paralleler komplexer Multiplizierer (vereinfachte Darstellung)

Für die parallele Multiplizierer-Variante werden die folgenden Formeln implementiert:

$$Re = \begin{cases} a \cdot c + b \cdot d, & SR \oplus SI = 1 \\ a \cdot c - b \cdot d, & SR \oplus SI = 0 \end{cases}$$
$$Im = \begin{cases} a \cdot d - b \cdot c, & SR \oplus SI = 1 \\ a \cdot d + b \cdot c, & SR \oplus SI = 0 \end{cases} \tag{4.3}$$

(Sample: $a + ib$, Koeffizient $c + id$ Vorzeichen des Koeffizienten: SR, SI 1=negativ)
Das Multiplizierermodul signalisiert dem Akkumulator mithilfe der zwei „accuInvert“ Signale, wann addiert oder subtrahiert werden soll (0=Addition, 1=Subtraktion). Diese Signale werden bei negativem Vorzeichen des realen Koeffizienten gesetzt. Die Bitbreite des Ausgangs des Moduls ist um ein Bit breiter als die der Konstantenmultiplizierer, um die Bitbreitenerhöhung durch die Addierer/Subtrahierer am Ausgang zu berücksichtigen.

Sequenzielle Multiplikation

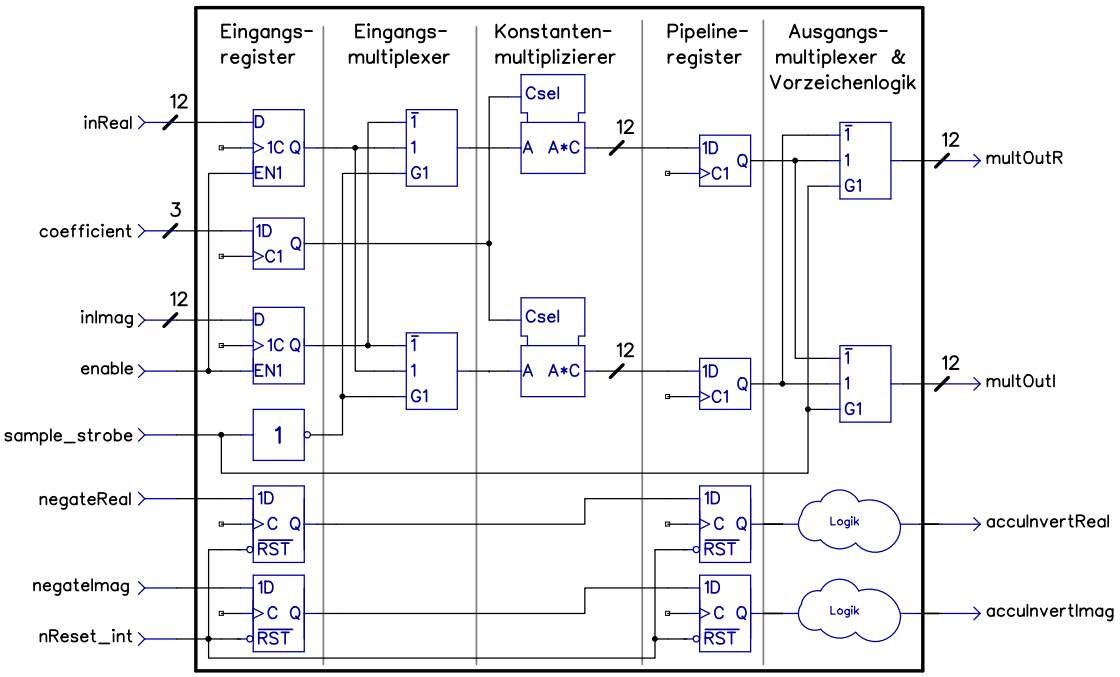

Abbildung 4.8: Sequenzieller komplexer Multiplizierer (vereinfachte Darstellung)

Für das sequenzielle Verfahren werden die vier Produkte der komplexen Multiplikation

in zwei Schritten dem Akkumulator zugeführt. Dies erfordert im Eingangs- und Ausgangspfad der Konstantenmultiplizierer je einen Multiplexer. Die Ausgangsbitbreite beim sequenziellen Verfahren entspricht der der Konstantenmultiplizierer. Hinzu kommt eine Logik, die die externen Vorzeichen-Eingänge verarbeitet. Diese ist bedeutend komplexer als beim Eintaktverfahren und liefert zwei Additions- und Subtraktionsausgänge. Die Abfolge bei allen möglichen Vorzeichenkombinationen ist in Tabelle 4.2 abzulesen.

Zyklus	Vz. Real.	Vz. Imag.	Real. Ausgang	Imag. Ausgang	Akku Real.	Akku Imag.
1	+	+	$a \cdot c$	$a \cdot d$	-	+
2	+	+	$b \cdot d$	$b \cdot c$	+	+
1	-	+	$a \cdot c$	$a \cdot d$	-	-
2	-	+	$b \cdot d$	$b \cdot c$	-	+
1	+	-	$a \cdot c$	$a \cdot d$	+	+
2	+	-	$b \cdot d$	$b \cdot c$	+	-
1	-	-	$a \cdot c$	$a \cdot d$	+	-
2	-	-	$b \cdot d$	$b \cdot c$	-	-

(Sample: $a + ib$, Koeffizient $c + id$)

Tabelle 4.2: Berechnungsschema für sequenzielle komplexe Multiplikation

4.2.2 Pipelining

Um den kritischen Pfad kurz zu halten, wurden am Eingang des Multiplizierermoduls Register vorgesehen. Die langen Addierer- und Multiplexerketten, aus denen die Multiplizierer aufgebaut sind, führen zu einem sehr langen kritischen Pfad, typischerweise der längste im gesamten Design. Die Verzögerung der Konstantenmultiplizierer steigt mit der Anzahl der realisierten Konstanten an. Dies bedeutet, dass **die maximale Taktfrequenz direkt von den Konstantenmultipliziern bestimmt wird**. Deshalb wurden hinter den Konstantenmultipliziern ein Satz Pipeline-Register eingefügt. Die externen Vorzeicheneingänge werden auf eine Verzögerungskette von zwei Registern gelegt, welche diese Signale in Phase mit dem Ausgang der Konstantenmultiplizierer halten.

4.3 Komplexer Akkumulator

Die beiden vom Multiplizierer gelieferten Produkte werden direkt in den Eingang des komplexen Akkumulators geleitet. Dieser besteht, wie in Abbildung 4.9 vereinfacht dargestellt, aus zwei Registern und zwei Addierer/Subtrahierern. Der Ausgang des Registers

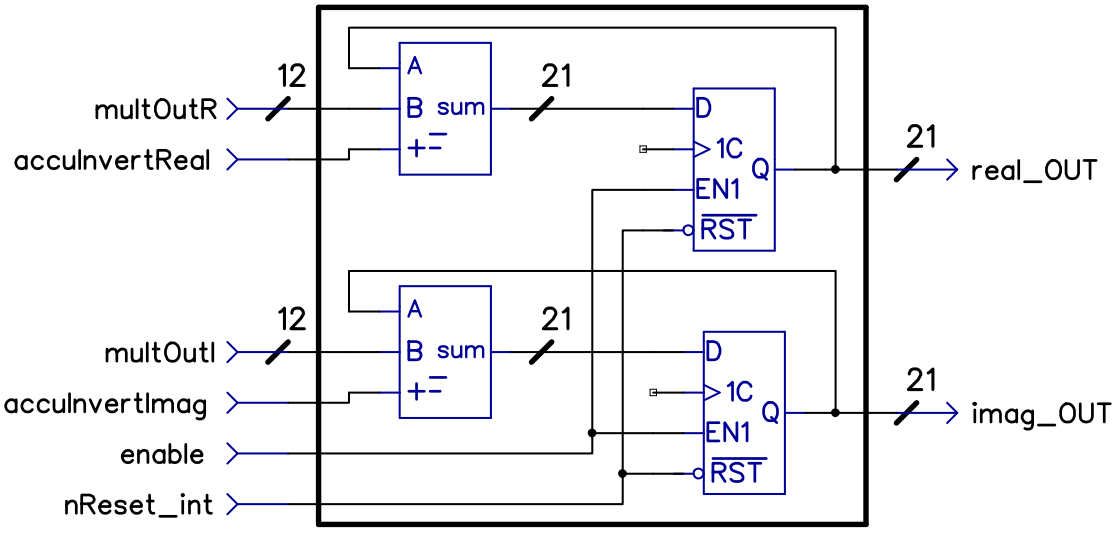

Abbildung 4.9: Komplexer Akkumulator (vereinfachte Darstellung)

wird an einen der Eingänge des Addierer/Subtrahierers zurückgeführt. Der andere Eingang des Addierer/Subtrahierers ist der Eingang des Akkumulators. Es sind separate Addieren/Subtrahieren Eingänge für Real- und Imaginärteil vorgesehen. Diese direkt vom Multiplizierermodul angesteuerten Eingänge bestimmen das Verhalten des Akkumulators: 0 entspricht Addition, 1 Subtraktion. Dazu kommen Kontrollsignale für das Register: Ein Enable Eingang, welcher das Register aktiviert und ein Reset Eingang, der den Akkumulatorwert zurücksetzt. Die Eingangsbitbreite des komplexen Akkumulators entspricht beim sequenziellen Verfahren der Ausgangsbitbreite der Konstantenmultiplizierer. Bei der parallelen Variante ist sie ein Bit breiter, um die Bitbreitenerhöhung durch die Addition im Multiplizierer zu berücksichtigen. Falls die gewählte Bitbreite des Akkumulatorausgangs kleiner ist als die native Bitbreite des Akkumulators, wird automatisch eine Einheit synthetisiert, die den Ausgangswert korrekt auf den darstellbaren Wertebereich der Ausgangsbitbreite limitiert. Diese Begrenzung wird auch als Sättigung bezeichnet. Ohne diese Einheit würde der Ausgangswert bei Überschreitung der größten positiv darstellbaren Zahl zur maximal negativen Zahl umschlagen.

4.4 Verilog-Toplevelmodul

Das Verilog-Topmodul erzeugt alle Kontrollsignale, die die Koeffizientenmodule zum Betrieb benötigen. Dafür enthält es, wie in Abb. 4.10 zu sehen, einen Zustandsautomaten,

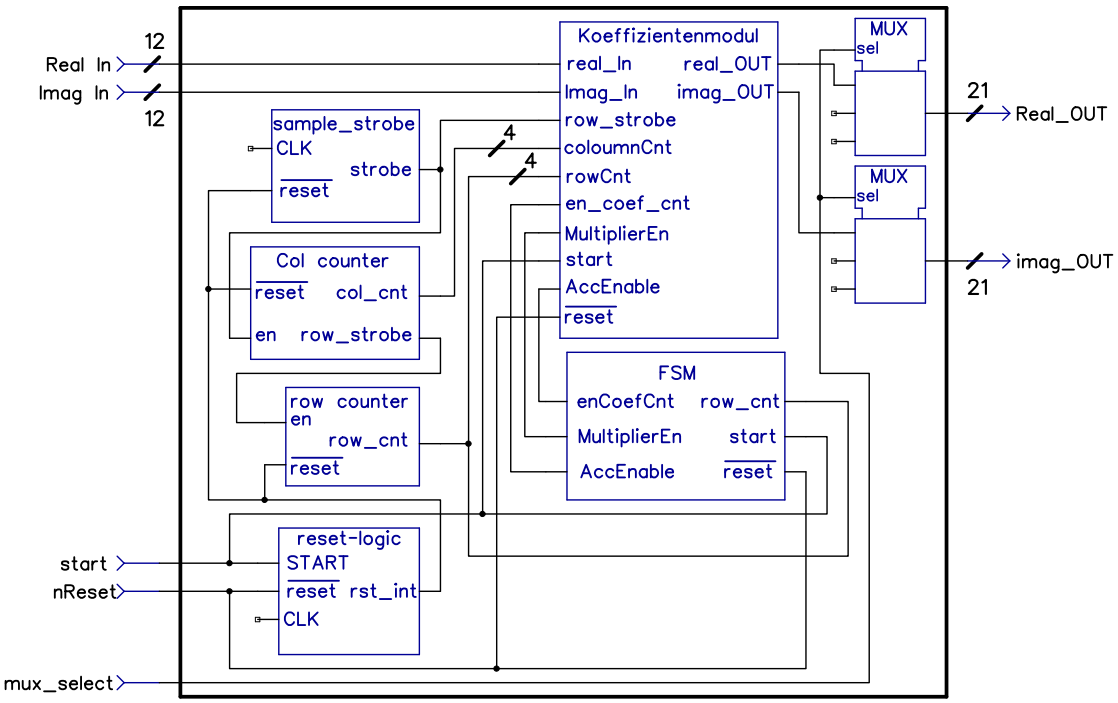

Abbildung 4.10: Koeffizientenlogik für komplexe Spaltenmuster (vereinfacht)

Zeilen- und Spaltenzähler und einen Multiplexer, der die Ausgänge der einzelnen Module anwählen kann.

Zeilen- und Spaltenzähler

Die Zeilen- und Spaltenzähler werden für eine Vielzahl von Zwecken genutzt. Sie werden von dem Zustandsautomaten genutzt, um das Ende der Berechnung feststellen zu können. Zudem werden die Zählerstände den Koeffizientenmodulen zugeführt. Diese Eingänge finden nicht in jedem Koeffizientenmodul Verwendung und in diesen Fällen wird die Optimierungsfunktion der Synthesewerkzeuge die nicht benötigten Verbindungen entfernen. Zudem wird der Spaltenzählerstand genutzt, um das „row-strobe“ Signal zu erzeugen. Dieses signalisiert „High“, wenn der Spaltenzähler die letzte Spalte erreicht. Für das sequenzielle Verfahren hat das Spaltenzählerregister einen Enable-Eingang, welcher an das „sample-strobe“ Signal gekoppelt ist, sodass nur jeden zweiten Taktzyklus gezählt wird.

Zustandsautomat

Der Zustandsautomat ist im „One-Hot“ Format kodiert und besitzt bei paralleler Multiplikation sechs und bei sequenzieller sieben Zustände. Der einzige Unterschied zwischen den beiden Varianten besteht in einem zusätzlichen Wartezustand für die sequenzielle Variante. Der Automat liefert drei Kontrollsignale an die Koeffizientenmodule und erhält das START-Signal und den Zeilenzähler als Eingänge. Die Zustandsfolge ist in den Zustandstabellen 4.3 und 4.4 abzulesen.

Zustand		Eingänge		Ausgänge		
aktuell	nächster	Start	rowCnt =Limit	Acc. en.	Coeff Cnt. en.	Mult. en.
idle	idle	0	X	0	0	0
idle	start 1	1	X	0	0	0
start 1	start 2	X	X	0	1	1
start 2	calc	X	X	0	1	1
calc	calc	X	0	1	1	1
calc	fin 1	X	1	1	1	1
fin 1	fin 2	X	X	1	0	1
fin 2	idle	X	X	1	0	0

Tabelle 4.3: Zustandstabelle für den parallelen Datenpfad

Zustand		Eingänge		Ausgänge		
aktuell	nächster	Start	rowCnt =Limit	Acc. en.	Coeff Cnt. en.	Mult. en.
idle	idle	0	X	0	0	0
idle	start 1	1	X	0	0	0
start 1	start 2	X	X	0	1	1
start 2	calc	X	X	0	1	1
calc	calc	X	0	1	1	1
calc	fin 1	X	1	1	1	1
fin 1	fin 2	X	X	1	0	1
fin 2	fin 3	X	X	1	0	1
fin 3	idle	X	X	1	0	0

Tabelle 4.4: Zustandstabelle für den sequenziellen Datenpfad

Eingang: „START“ startet die Berechnung, X=Don’t-Care
Eingang: „rowCnt=Limit“ hat der Zeilenzähler den Endstand erreicht, X=Don’t-Care
Ausgang: „Mult. en.“ aktiviert die Eingangsregister der Multiplizierermodule
Ausgang: „Coeff Cnt. en.“ aktiviert die Koeffizientenzähler in den Koeffizientenmodulen

Ausgang: „Acc. en.“ aktiviert den Akkumulator, „High“ addieren/subtrahieren, „low“ Akkumulatorstand halten

4.5 DFT-Modul (VHDL)

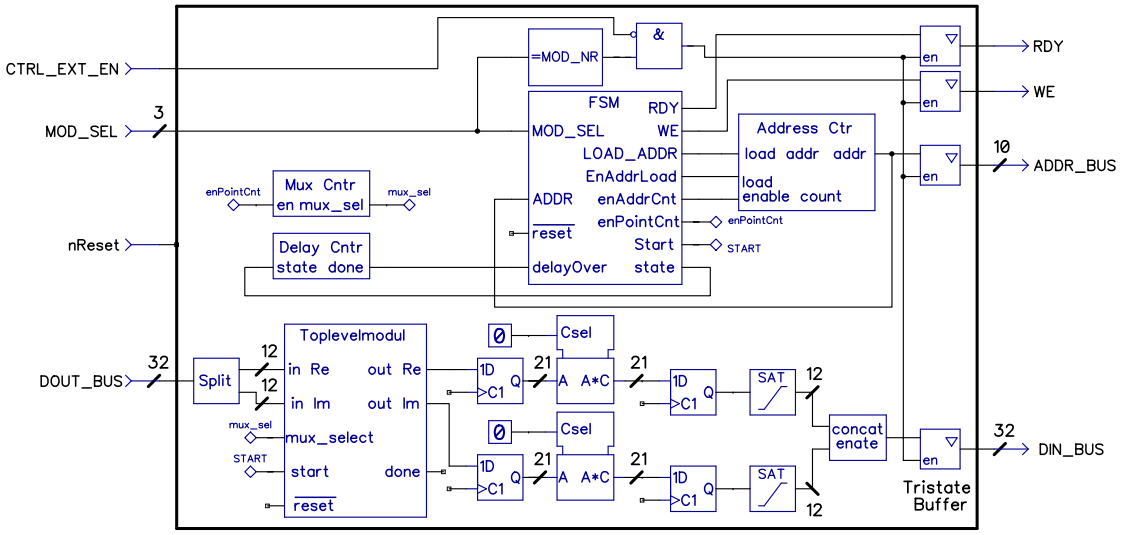

Abbildung 4.11: DFT-Modul (vereinfacht)

Das DFT-Modul ist in der Modulhierarchie der automatisch erzeugten Quellen das höchste, denn es steuert den Zugriff auf den Block-RAM und instanziiert das Verilog-Toplevelmodul. Für den Nutzer ist dieses Modul der beste Ansatzpunkt, wenn die erzeugte Hardware an eine andere Umgebung angepasst werden soll.

Das DFT-Modul, wie in Abb. 4.11 dargestellt, besteht aus einem Zustandsautomaten, drei Zählern und den Multiplizierern zur Skalierung. Wichtigster Zähler ist der ladbare Adresszähler für die RAM Adressen. Ein weiterer Zähler wird von dem Zustandsautomaten für das Abwarten der Pipeline-Verzögerung vor dem Auslesen der Ergebnisse genutzt. Der dritte und letzte Zähler steuert den Ausgangsmultiplexer des Verilog-Toplevelmoduls an und ließt sequenziell die Ergebnisse ein. Der Ausgang des Multiplexers wird mit Registern entkoppelt, um den kombinatorischen Pfad kurz zu halten. Die Ausgänge der Register sind mit den skalierenden Festwertmultiplizieren verbunden. Auf die Multiplizierer folgen weitere Register zur Entkopplung und eine Sättigungseinheit, welche den

Ausgangswert auf den darstellbaren Bereich beschränkt. Real- und Imaginärteil werden nun zu einem 32-Bit Ausgangswort kombiniert, welches auf den Datenbus gelegt wird.

Wenn der MOD_SEL Eingang gleich der Modulnummer ist, wird das DFT-Modul aktiviert und übernimmt den Daten- und Adressbus. Die Modulnummer ist per generic konfigurierbar. Der Adresszähler wird mit der Startadresse geladen und einen Taktzyklus später wird das START-Signal gesetzt, welches die Berechnungen in den Koeffizientenmodulen startet. Einen Takt später wird der Adresszähler aktiviert. Die Adressen werden nun linear hochgezählt, bis die Endadresse erreicht ist. Während auf den Ablauf des Verzögerungszählers gewartet wird, wird die Schreibadresse in den Adresszähler geladen. Ist der Verzögerungszähler abgelaufen, startet der Zähler, der den Multiplexer im Verilog-Toplevelmodul kontrolliert. Dadurch werden nun sequenziell die Ausgänge aller Koeffizientenmodule angewählt. Nach weiteren zwei Taktzyklen wird das Write-Enable-Signal gesetzt, denn jetzt liegt das erste skalierte Ergebnis am Ausgang des DFT-Moduls vor. Außerdem beginnt der Adresszählers die Schreibadressen hochzuzählen. Ist die letzte Schreibadresse erreicht, werden der Write-Enable und alle Zähler deaktiviert. Außerdem wird der RDY-Ausgang gesetzt.

4.6 Numerische Auslegung

Eine Vielzahl der in der Architektur vorhandenen Bitbreiten sind konfigurierbar. Das Skript brute2snr.m hilft bei der Dimensionierung, indem es Überläufe identifiziert und Voraussagen über die zu erwartende Genauigkeit liefert. Es wurden die folgenden Standardwerte festgelegt, welche auch für die Erzeugung der Testwerte in Kapitel 6 genutzt wurden.

- Eingangsbitbreite: 12 Bit
- Akkumulatorbitbreite: 21 Bit
- Ausgangsbitbreite: 12 Bit
- Skalierungsfaktor: $\frac{1}{M \cdot M}$
- Skalierungsmultiplizierer-Nachkommabits: 16 Bit
- Koeffizienten-Nachkommabits: 7 Bit

4.6.1 Eingangsbitbreite

Die Eingangsbitbreite ist frei konfigurierbar, das Design wird darauf automatisch angepasst. Es wird von vorzeichenbehafteten Eingangsdaten im Zweierkomplement ausgegangen.

4.6.2 Akkumulatorbitbreite

Die Akkumulatorbitbreite muss so ausgelegt werden, dass ein Überlauf sicher ausgeschlossen werden kann. Die nötige Bitbreite ist direkt berechenbar, wie in Formel 4.4 beschrieben. Es wurden zwei Optionen getestet, um das Problem eines Akkumulatorüberlaufs zu vermeiden, eine Erhöhung der Bitzahl oder eine Begrenzung des Wertebereichs. Die Erhöhung der Akkumulatorbitbreite verbrauchte weniger Logikressourcen als eine Begrenzungseinheit und wurde deshalb für die Implementierung ausgewählt.

$$\begin{aligned} \text{Akkumulatorbitbreite } &= \lceil log_2(2^{(k-1)} \cdot 2 \cdot M^2) \rceil + 1 \\ &\text{k = Eingangsbitbreite} \end{aligned} \tag{4.4}$$

In dieser Formel zur Errechnung der benötigten Akkumulatorbitbreite wird zunächst der höchste, mit einer gegebenen Eingangsbitbreite erreichbare, Akkumulatorwert berechnet. Dazu wird der höchste darstellbare Samplewert mit zwei multipliziert, da dies dem maximalen Ergebnis der komplexen Multiplikation vor dem Akkumulator entspricht. Dann wird dieser Wert mit der Anzahl der Eingangssamples multipliziert. Dies ist nun der Höchstwert des Akkumulators, von diesem wird dann der Logarithmus-Dualis berechnet, dessen Ergebnis aufgerundet wird. Nun ist die Mindestanzahl von Bits, die für den Akkumulator benötigt werden, bekannt und es muss dann noch ein Bit für das Vorzeichen dazugerechnet werden.

Dieser Worst Case entspricht einer perfekt auf die Ortsfrequenz des Ausgangspunktes eingestellte, Anregung mit Fullscale-Aussteuerung am Eingang. Wenn es möglich ist, gesicherte Annahmen über die Eingangssignale zu treffen, lässt sich gegenüber diesem Worst Case möglicherweise Bitbreite einsparen.

4.6.3 Ausgangsskalierung

Bei der Wahl des Skalierungsfaktors muss die Ausgangsbitbreite bedacht werden, denn die Ausgangswerte nach Skalierung müssen in die Ausgangsbitbreite passen, um Fehler durch Überläufe zu vermeiden. Für diesen Fall wurde im Design eine Begrenzungseinheit nach dem Multiplizierer vorgesehen, welche die Ausgangswerte auf den darstellbaren Wertebereich begrenzt. Unterstützt werden Skalierungsfaktoren von $[0 \ldots 1]$. Wenn die Eingangsbitbreite der Ausgangsbitbreite entspricht, ist der maximale Skalierungsfaktor $1/M^2$. Allgemein gilt: der maximal mögliche Akkumulatorwert, multipliziert mit dem Skalierungsfaktor, ergibt den maximalen Ausgangswert. Der Skalierungsfaktor limitiert den erreichbaren Signal-to-quantization-noise ratio (SQNR) am Ausgang.

4.6.4 Genauigkeit & SQNR

Die Auslegung erfolgt in zwei Schritten. Zunächst wird der maximal erreichbare Ausgangs-SQNR bestimmt. Mithilfe dieses Wertes wird dann die nötige Anzahl Nachkommabits für die Koeffizienten festgelegt.

Der maximal mögliche Ausgangs-SQNR lässt sich mit brute2snr.m bestimmen. Dazu muss der im Akkumulator erreichte SQNR deutlich höher als der Ausgangs-SQNR sein. Dafür wird in der Simulation eine ausreichend hohe Anzahl Nachkommabits für die Koeffizienten festgelegt. Dann werden die Nachkommabits für den skalierenden Multiplizierer so lange erhöht, bis der Ausgangs-SQNR nicht mehr signifikant ansteigt. Der nun erreichte SQNR entspricht der maximal erreichbaren Genauigkeit für die gewählte Ausgangsbitbreite und den daraus resultierenden Skalierungsfaktor.

Ermittlung nötiger Nachkommabits für die Skalierung

Der Hardwarebedarf für den skalierenden Multiplizierer ist im Vergleich zu den Koeffizientenmultiplizierern relativ klein. Daher sollte Anzahl der Nachkommabits so gewählt werden, dass der erreichte SQNR nicht deutlich verschlechtert wird. Für den Standardwert wurden deshalb 16 Nachkommabits gewählt.

Ermittlung nötiger Nachkommabits für die Koeffizienten

Die Anzahl der Koeffizienten-Nachkommabits bestimmt den auf Akkumulatorebene erreichbaren SQNR. Die Erhöhung der Multiplizierer-Nachkommabits hat erhebliche Auswirkungen auf den Logikbedarf, da die Konstantenmultiplizierer den Großteil des Logikbedarfs ausmachen. Die Anzahl der Nachkommabits für die Koeffizienten sollte daher so gewählt werden, dass der SQNR auf Akkumulatorebene knapp oberhalb der maximal erreichbaren Ausgangs-SQNR liegt. Denn ein höherer SQNR auf Akkumulatorebene hat keinen Nutzen, wenn dieser nicht am Ausgang nutzbar ist. Für die Standardwerte wurden daher sieben Nachkommabits gewählt.

4.7 Änderungen am Testsystem

Es wurden Änderungen am Testsystem vorgenommen, um Simulation und Synthese zu vereinfachen und vorhandene Fehlerquellen zu beseitigen.

4.7.1 BRAM-Modul

Der BRAM war in einer für die FPGA-Synthesewerkzeuge inkompatibler Weise beschrieben, was korrigiert wurde. Dabei wurde der nicht genutzte Read-Enable entfernt und eine symbolische Verzögerungszeit für den Ausgang eingefügt. Die Synthesewerkzeuge ignorieren die Verzögerungszeiten und führen daher nicht zu Änderungen an der Logik. Symbolische Verzögerungen reduzieren zwar die Simulationsgeschwindigkeit, erleichtern aber die Entwicklung enorm, da Ursache und Wirkung bei der funktionalen Simulation leichter auseinander gehalten werden können. Außerdem verhindern sie Probleme durch den Gated-Clock. Der BRAM erhält den Takt direkt, während die Module mit einem Gated-Clock versorgt werden. Eventbasierte Simulatoren erzeugen ein neues Clock-Event, wenn das Taktsignal durch das Gate neu zugewiesen wird. Diese Zuweisung erfolgt einen Simulationsschritt nach der Taktflanke des Non-Gated-Clock. Dadurch reagieren die Module einen Taktzyklus zu früh auf eine Änderung der BRAM-Ausgangsdaten.

4.7.2 Dummy-Signalverarbeitungsmodul

Es wurde ein Dummy-Signalverarbeitungsmodul geschrieben, was die freien Modul-Slots im Toplevel des Testsystems einnehmen kann, um Fehler durch fehlende Module zu vermeiden. Die „module_control“ Logik aktiviert ohne Modifikation alle vier Modul-Slots sequenziell hintereinander. Das führt zu undefinierten Signalen auf den Bussen, wenn ein Slot angewählt wird, welcher nicht mit einem Modul besetzt ist. Zudem werden Fehler beim hinzufügen weiterer Module verhindert, da nur die Instanziierung und nicht auch die „module_control“ Logik geändert werden muss. Das Dummy-Signalverarbeitungsmodul ist mit einem generic zur Konfiguration der MOD_NR ausgestattet. Es übernimmt, wenn MOD_SEL der MOD_NR entspricht, die Kontrolle über den Adress- und Datenbus und setzt diese auf „low“ und das RDY-Signal auf „High“.

4.7.3 Interne Tristate-Busse

Interne Tristate-Busse sind ein Designaspekt, der nicht geändert wurde, aber auf den trotzdem eingegangen werden sollte. Das Testsystem verwendet für die Ausgänge der Signalverarbeitungsmodule, den Adressbus und das RDY-Signal interne Tristate-Busse. Da Tristate-Bustreiber für interne Signale in modernen FPGAs nicht vorhanden sind, werden sie durch das Synthesewerkzeug durch Multiplexer ersetzt. Dies führt zu deutlichen Unterschieden in der realisierten Logik, zwischen den für Funktionstests genutzten FPGAs und der ASIC Implementierung.

Das größte Problem liegt jedoch in der Arbitrierung.
Das MOD_SEL Signal überträgt das aktuell aktive Modul als Binärzahl. Dieses Signal wird in den Modulen auf einen Decoder gelegt, der direkt auf die Bustreiber wirkt. Das bedeutet, dass Glitches in einem der Decoder oder Timing-Unterschiede zwischen diesen Decodern zu Kurzschlüssen auf dem Daten- oder Adressbus führen werden. In einen ASIC könnte dies im schlimmsten Fall die internen Bustreiber zerstören oder im besten Fall zu undefiniertem Verhalten führen. Wenn die Tristate-Busse beibehalten werden sollen, muss eine zentrale Bus-Arbitrierung im Kontrollmodul erfolgen, um Kurzschlüsse auszuschließen. Die bevorzugte Lösung wären jedoch Multiplexer-basierte Bussysteme.

4.7.4 Register Initialwerte

In einigen Modulen des Testsystems waren Register mit einem Initialwert beschrieben. Dies ist auf FPGAs möglich, auf einem ASIC jedoch nicht. Initialwerte sind problematisch, weil sie fehlende Resets in der Simulation maskieren können. Oberstes Ziel eines Testsystems für eine ASIC-Implementierung sollte die Minimierung von Unterschieden in Verhalten und Logik sein. Die Initialwerte wurden daher entfernt.

4.7.5 Zweiflankenlogik

Das Testsystem verwendet für die Adressierung des Speichers eine Zweiflankenlogik. Der BRAM reagiert auf die steigende Flanke, während die Adresszähler auf die fallende Flanke reagieren. Alle Logikelemente der im Projekt verwendeten Standardzellen-Bibliothek reagieren aber nur auf die steigende Flanke. Das bedeutet, dass für die von der fallenden Flanke gesteuerte Elemente ein separater Taktbaum oder Inverter eingefügt werden muss. Als Grund für die Verwendung von Zweiflankenlogik wurde die Latenz von einem Taktzyklus, bis das Ergebnis vorliegt, angegeben. Diese Latenz muss jedoch nur für den ersten Zugriff abgewartet werden, weitere Zugriffe, wie z.B. durch einen linearen Adresszähler, erfordern keine weiteren Wartezeiten. Der Latenzvorteil der Zweiflankenlösung ist für lineare oder vorhersehbare Speicherzugriffsmuster nutzlos. Zweiflankenlogik hat den Nachteil, dass zwei Taktbäume synthetisiert werden müssen, was den Flächenbedarf für das Signalrouting erhöht. Außerdem halbiert eine Zweiflankenlogik effektiv die maximal erzielbare Taktfrequenz für ein Design. Alle in dieser Arbeit implementierte Logik reagiert ausschließlich auf die steigende Flanke.

5 brute2dft Framework zur Hardwareerzeugung

Um dem Projekt eine Lösung für die noch nicht exakt bekannten Anforderungen zu liefern, wurde eine Reihe von Matlab-Skripten erstellt. Diese generieren anhand der Benutzervorgaben automatisch eine funktionierende Hardwarearchitektur.

5.1 Überblick über die Software

Das brute2dft Framework erfordert vom Nutzer bestimmte Eingangsparameter, anhand derer die Hardwarebeschreibungen für 2D-DFT Module erzeugt werden.

Benötigte Eingangsparameter:

- Liste der zu erzeugenden Ausgangspunkte
- Transformationsgröße
- Bitbreite der Ein- und Ausgangsdaten
- Akkumulatorbitbreite
- Anzahl der Nachkommabits für die Koeffizientenmultiplizierer
- Paralleles oder sequenzielles Multiplikationsverfahren
- Ausgangs-Skalierungsfaktor
- Anzahl Nachkommabits für den Ausgangsmultiplizierer
- Auswahl der zu erzeugenden Ausgangsdateien

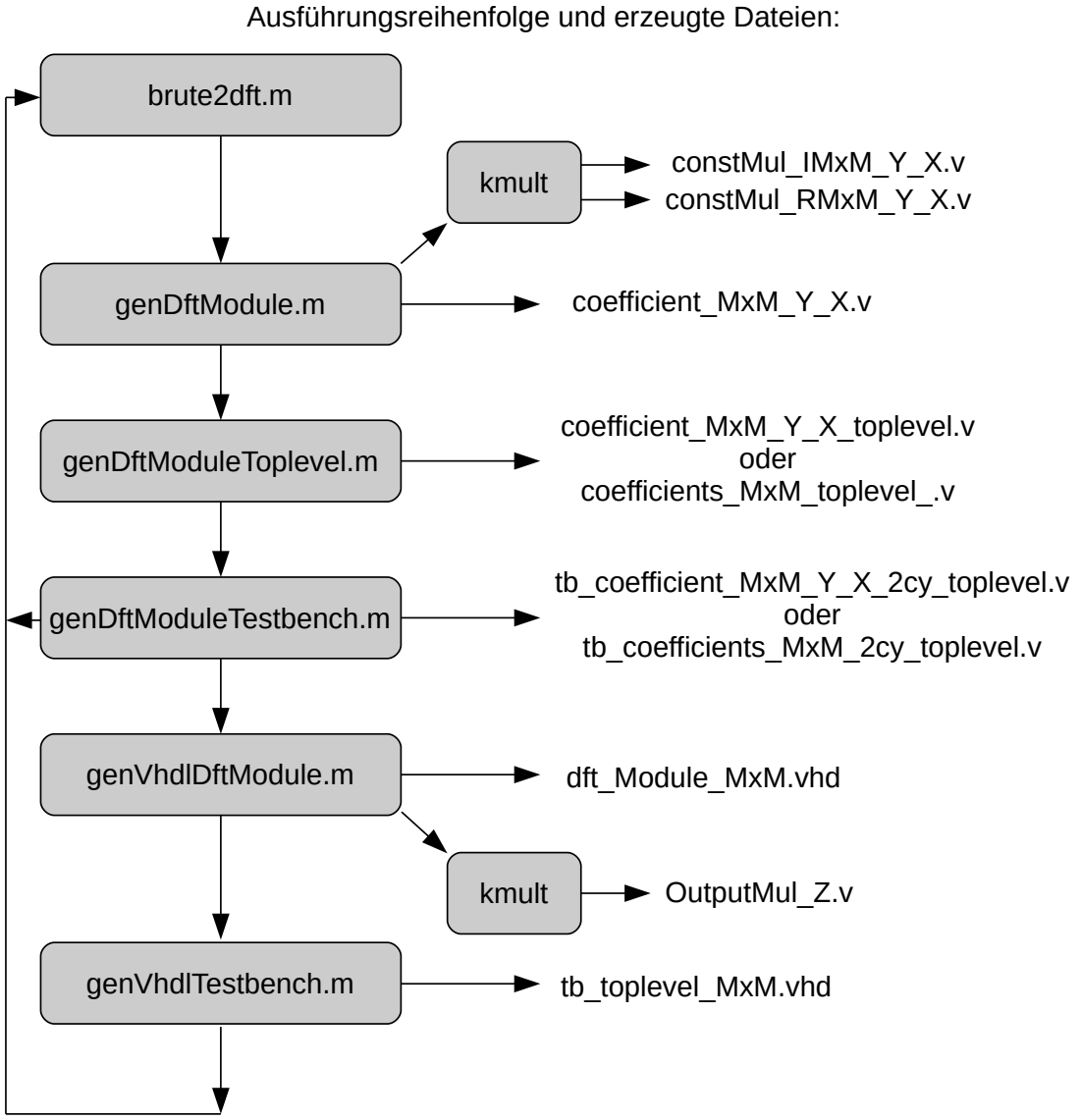

Abbildung 5.1: Ausführungsreihenfolge und erzeugte Dateien

Der Prozess zur Hardwareerzeugung startet, wie in Abbildung 5.1 zu sehen ist, mit der Hauptdatei **brute2dft**. Dieses Skript errechnet die Koeffizienten für die spezifizierte Transformationsgröße und startet alle weiteren Skripte. In brute2dft.m werden alle Parameter definiert. Sollen Parameter geändert werden, kann dies entweder in dieser Datei geschehen oder die Parametervariablen können vorher im Workspace definiert werden.
Das erste von brute2dft gestartete Skript ist **genDftModule**. Es erzeugt die Koeffizientenmodule und liefert den Koeffizientenmustertyp als Returnwert. GenDftModule startet nach Identifikation des Koeffizientenmusters das externe Programm kmult, welches die Konstantenmultiplizierer für das Koeffizientenmodul erzeugt.
Als nächstes wird **genDftModuleToplevel** gestartet. Dieses Skript erzeugt das Verilog-Toplevelmodul. Wenn nur ein oder gar kein Ausgangspunkt spezifiziert wurden, wird für jedes Koeffizientenmodul ein Toplevelmodul erzeugt. Die erzeugte Datei trägt die Transformationsgröße und einen spezifischen Ausgangspunkt im Dateinamen und instanziiert nur dieses individuelle Koeffizientenmodul. Dieser Modus wird bei der vollständigen Verifikation verwendet, um die Koeffizientenmodule einzeln testen zu können. Wenn es mehr als einen Ausgangspunkt gibt, wird für alle Koeffizientenmodule ein gemeinsames Toplevelmodul erzeugt, welches die Transformationsgröße MxM im Dateinamen trägt.
Dann wird **genDftModuleTestbench** gestartet. Dieses Skript erzeugt eine Testbench, mit der das Verilog-Toplevelmodul getestet werden kann. Es erzeugt eine zufällige Eingangsmatrix, deren Werte als Eingangssignale in die Testbench übertragen werden. Gleichzeitig wird diese Matrix einer simulierten Festkomma-2D-DFT unterzogen, deren Ergebnis einen exakten Vergleichswert für die automatische Überprüfung der Ergebnisse in der Testbench liefert. Zudem gibt das Testbench-Skript das Ergebnis einer Gleitkomma-2D-DFT von derselben Eingangsmatrix zum Vergleich aus.
Nach Erzeugung der Verilog-Testbench wird das **genVhdlDftModule** Skript gestartet, welches das mit dem Testsystem kompatible DFT-Modul erzeugt. Dabei wird wieder das kmult Programm genutzt, um den Konstantenmultiplizierer für die Skalierung des Ausgangs zu erzeugen.
Als letztes Skript startet das **genVhdlTestbench** Skript welches eine Testbench für das Toplevel des Testsystems erzeugt. Es funktioniert ähnlich wie das Skript, welches die Verilog-Toplevelmodul-Testbench erzeugt. Unterschiede sind die Ausgabe im VHDL Format, und dass die Skalierung des Ergebnisses mit in Betracht gezogen wird.

5.2 brute2dft.m

Diese Datei ist das Hauptskript und damit der Einstiegspunkt für den Rest des Hardwareerzeugungsframeworks. Die Skriptdatei enthält die vom Nutzer definierbaren Parametervariablen.

5.2.1 Parametervariablen

- nPoints - Matrixgröße, in dieser Arbeit auch M genannt (Default=15)
- outputDftTerm - Vektor von Ausgangspunkten [1 1; 1 2; 1 3 ...] oder leerer Vektor
- inputBitwidth - Eingangsbitbreite (Default=12)
- outputBitwidth - Ausgangsbitbreite des DFT-Moduls nach Skalierung (Default=12)
- coefFracBitwidth - Anzahl der Nachkommabits für die DFT-Koeffizienten (Default=7)
- accumulatorBitwidth - Bitbreite des Akkumulators (Default=nextpow2(2^(inputBitwidth-1)*2*nPoints^2)+1)
- twoCycle - Bestimmt, ob sequenzielles Multiplikationsverfahren verwendet wird (Default=true)
- outputDividerFactor - Skalierungsfaktor des Ergebnisses (Default= 1/nPoints^2)
- outputDividerFracBitwidth - Anzahl Nachkommabits für den skalierenden Multiplizierer (Default=16)
- generateFile - Bestimmt, welche Ausgangsdateien erzeugt werden:
  - generateFile(1) - Koeffizientenmodul(e) (coefficient_MxM_Y_X.v)
  - generateFile(2) - Festwertmultiplizierer für Koeffizienten (constMul_RMxM_Y_X.v, constMul_IMxM_Y_X.v)
  - generateFile(3) - kmult zur Erzeugung der Konstantenmultiplizierer verwenden
  - generateFile(4) - Verilog Toplevel Testbench (tb_coefficient_MxM_X_Y_2cy_toplevel.v | tb_coefficient_MxM_2cy_toplevel.v)

- generateFile(5) - Verilog Toplevel
  (coefficient_MxM_Y_X_toplevel.v | coefficient_MxM_toplevel.v)
- generateFile(6) - DFT-Modul (dft_module_MxM.vhd)
- generateFile(7) - Festwertmultiplizierer für Skalierung
  (outputMul_Z.v, Z = Festkomma Skalierungsfaktor)
- generateFile(8) - Testbench für Testsystem Toplevel (tb_toplevel_MxM.vhd)

outputDftTerm Variable

Diese Variable enthält die Liste der zu erzeugenden Ausgangspunkte. Wenn die Variable leer ist, wird der Massentestmodus aktiviert. Dieser startet den Skriptablauf für alle Punkte einer Matrixgröße. In diesem Modus werden nur die genDftModule, genDftModuleToplevel und genDftModuleTestbench Skripte gestartet. Da dieser Modus für Massentests der Koeffizientenmodule vorgesehen ist, macht hier die Erzeugung der VHDL-Infrastruktur wenig Sinn, weshalb die entsprechenden Skripte nicht gestartet werden. Außerdem wird in diesem Modus die resultMatrix Variable mit den erkannten Mustertypen aller Punkte erzeugt.

generateFile Variable

Diese Variable bestimmt, welche Ausgangsdateien erzeugt werden. Für jede Ausgangsdatei existiert ein Index in diesem Logical-Vektor, „True“ bedeutet, dass die betreffende Datei erzeugt wird. Auch ohne „True“ Signal werden die für eine Ausgangsdatei verantwortlichen Skripte dennoch gestartet, da deren Output trotzdem hilfreich sein kann. Zum Beispiel simulieren die Testbencherzeugenden Skripte auch bei deaktivierter HDL-Erzeugung einen 2D-DFT-Durchlauf.

5.2.2 Skript Ablauf

Nach dem Start prüft brute2dft, ob die Parameter mit dem Framework kompatibel sind. Dies umfasst eine Mindestgröße von 3x3 und einen Skalierungsfaktor größer 0 und kleiner

oder gleich 1. Nach der Überprüfung der Parameter werden mit Formel 5.1 alle Transformationskoeffizienten berechnet und in einem 4D-Array gespeichert.

$$\text{Koeffizient}(y, x, m, n) = e^{\frac{-i \cdot 2\pi \cdot (ym + xn)}{M}}$$
$$y, x \text{ - Indizes des Ausgangspunkts, } m, n \text{ - Sample Indizes} \tag{5.1}$$

Nachdem die Koeffizienten erzeugt wurden, wird nun über den Vektor der Ausgangspunkte iteriert und für jeden angegebenen Punkt einmal genDftModule gestartet. Nach der Erzeugung aller Koeffizientenmodule wird genDftModuleToplevel gestartet, das ein Verilog-Toplevelmodul generiert, welches alle erzeugten Koeffizientenmodule instanziiert. Danach werden die Skripte gestartet, die das DFT-Modul, die Testbench für das Testsystem Toplevel sowie die Verilog-Toplevelmodul Testbench erzeugen.

5.2.3 Workspace Interaktion

Brute2dft löscht beim Start nicht den Workspace. Dazu kommt, dass wenn eine Parametervariable schon existiert, diese nicht überschrieben wird. Dies ist nützlich, um brute2dft ohne Parameterübergabe vom Workspace aus aufrufen zu können. Es erfordert jedoch Aufmerksamkeit vom Nutzer, wenn dieser Änderungen an den Parametern im Skript durchführt. Es muss dann entweder der Workspace gelöscht oder die im Workspace vorhandenen Variablen bearbeitet oder gelöscht werden. Im Workspace verbleiben nach Ausführen des Skripts alle Parametervariablen und das 4D-Array von Koeffizienten. Falls im Massentestmodus ausgeführt, verbleibt außerdem die resultMatrix Variable, welche die identifizierten Koeffizientenmuster-Codes enthält.

5.3 genDftModule.m

Dieses Skript erzeugt nach Analyse des Koeffientenmusters ein Koeffizientenmodul und die dafür benötigten Festwertmultiplizierer.

5.3.1 Parameter

- nPoints - Transformationsgröße

- outputDftTerm - Ausgangspunkt in Form eines Zeilenvektors z.B. [3 5]
- inputBitwidth - Eingangsbitbreite
- coefFracBitwidth - Anzahl der Nachkommabits für die DFT-Koeffizienten
- accumulatorBitwidth - Bitbreite des Akkumulators
- twoCycle - Bestimmt, ob sequenzielles Multiplikationsverfahren verwendet wird
- generateFile - Bestimmt, ob eine Ausgangsdatei geschrieben wird

5.3.2 Musteranalyse

Erster Schritt ist die Extraktion der Koeffizientenmatrix aus dem 4D-Array. Dazu wird der outputDftTerm als Index in das 4D-Array von Koeffizienten genutzt. Die Koeffizienten werden gerundet, um Probleme bei der Mustererkennung durch minimale Unterschiede zwischen den Koeffizienten zu vermeiden. Gerundet wird auf zwei dezimale Nachkommastellen mehr, als die Anzahl Nachkommabits zulässt. Die Auflösung in dezimalen Nachkommastellen lässt sich durch Multiplikation von log(2) mit der Anzahl von Nachkommabits erhalten.

Nach dem Runden beginnt die eigentliche Mustererkennung. Die Erkennung funktioniert per Ausschlussprinzip, es wird nacheinander auf alle bekannten Koeffizientenmusterarten getestet. Wurde der Mustertyp erfolgreich erkannt, wird der Rest der Analyse übersprungen. Dies wird ausgenutzt um Code einzusparen. Die Erkennung der einfachen Mustervarianten wird vor der Erkennung der komplexen Varianten ausgeführt. So muss in der Erkennung des komplexen Typs kein Ausschluss für die einfache Variante eingebaut werden.

Erkennungsschritt 1: Trivialmuster

Als erstes wird nach Trivialmustern gesucht. Dazu wird geprüft, ob die Koeffizientenmatrix mehr als einen betragsmäßig unterschiedlichen Koeffizienten aufweist. Wenn ja, wird abgebrochen und nach anderen Mustern gesucht. Wenn nein, ist von einem Trivialmuster auszugehen. Dann werden die Vorzeichen des ersten Koeffizienten der zweiten Zeile und die Koeffizienten des zweiten Samples der ersten Zeile geprüft, um festzustellen, ob Vorzeichenwechsel stattfinden. Dies geschieht separat für Real- und Imaginärteil. Am Ende

der Erkennung wird der erste Koeffizient als Eingangswert für den Konstantenmultiplizierergenerator extrahiert.

Erkennungsschritt 2: Einfache Spaltenmuster

Weiter geht es mit der Suche nach einfachen Spaltenmustern. Hier wird nun geprüft, ob jede Zeile betragsmäßig nur einen Koeffizienten aufweist. Wenn nein, wird abgebrochen und nach anderen Mustertypen gesucht. Wenn ja, wird geprüft, ob Vorzeichenwechsel innerhalb der Zeilen vorkommen, um Spaltenmuster mit Vorzeichenwechsel erkennen zu können. Als nächstes wird die Anzahl Koeffizienten festgestellt, indem die betragsmäßig einzigartigen Koeffizienten der ersten Spalte gezählt werden. Die Liste dieser Koeffizienten wird als Eingangswert für den Konstantenmultiplizierergenerator gespeichert.

Erkennungsschritt 3: Einfache Zeilenmuster

Nun geht es mit der Suche nach einfachen Zeilenmustern weiter. Die Erkennung funktioniert wie bei einfachen Spaltenmustern mit dem Unterschied, dass hier geprüft wird, ob jede Spalte nur einen Koeffizienten aufweist.

Erkennungsschritt 4: Komplexe Zeilenmuster

Nach den einfachen Mustertypen wird nach komplexen Zeilenmustern gesucht. Dabei wird geprüft, ob alle Zeilen denselben Satz Koeffizienten enthalten. Ein einfacher Vergleich der Zeilen ist aufgrund der Phasenverschiebung nicht möglich. Sind in allen Zeilen die gleichen Koeffizienten vorhanden, liegt ein komplexes Zeilenmuster vor. Dann wird die Koeffizientenfolge analysiert. Dabei wird das Verhalten an den Extrempunkten der Koeffizientenfolge auf Vorzeichenwechsel und Haltepunkte untersucht. Die Analyseergebnisse werden später für die Synthese der Zähler verwendet. Ebenfalls erfasst wird der Offset bzw. die Phasenverschiebung zwischen den Zeilen. Danach wird die Liste der betragsmäßig einzigartigen Koeffizienten der ersten Zeile als Eingangswert für den Konstantenmultiplizierergenerator gespeichert.

Erkennungsschritt 5: Komplexe Spaltenmuster

Das nächste Muster ist das komplexe Spaltenmuster. Die Analyse verläuft analog zu der der komplexen Zeilenmuster, nur dass hier geprüft wird, ob alle Spalten den gleichen Koeffizientensatz beinhalten. Für die komplexen Spaltenmuster wird die Koeffizientenfolge nicht analysiert, da dies nur für die Zählersynthese erforderlich ist, welche für dieses Muster nicht zum Einsatz kommt. Allerdings wird die Phasenverschiebung zwischen den Spalten erfasst. Danach wird die Liste der betragsmäßig einzigartigen Koeffizienten der ersten Spalte als Eingangswert für den Konstantenmultiplizierergenerator gespeichert.

Erkennungsschritt 6: Zeilen- und Spaltenmuster

Der letzter Erkennungsschritt entspricht einem Catch-All-Fall, denn es gibt keine Kriterien für diesen Mustertyp. Dieser Erkennungsschritt ist in der Lage, jede Art von Muster nachzubilden. Die Koeffizientenmatrix wird auf mehrfach auftretende Zeilen analysiert und auf solche Zeilen, die mit invertiertem Vorzeichen auftreten. So entsteht eine Liste der einzigartigen Zeilen. Die betragsmäßig einzigartigen Koeffizienten dieser Zeilen werden für die Verwendung durch den Konstantenmultiplizierergenerator in einer Liste gespeichert. Außerdem werden die Koeffizientenindizes nach Zeilen gruppiert und für die ROM-Erzeugung in einer separaten Liste gespeichert. Damit ist die Koeffizientenmusteranalyse abgeschlossen.

5.3.3 Koeffizientenmodulerzeugung

Nach der Koeffizientenmusteranalyse erfolgt die Modulerzeugung. Ausgeführt wird dieser Teil nur, wenn das entsprechende Bit in generateFile gesetzt ist.
Viele Teile der Codererzeugung sind für alle Mustertypen gleich und werden in dieser Sektion beschrieben. Die erzeugten Module erhalten einen deskriptiven Dateinamen, um das Überschreiben anderer Ausgangsdateien zu verhindern. Der Dateiname des Moduls beginnt immer mit “coefficient_“, gefolgt von der Transformationsgröße. Auf die Transformationsgröße folgen die Indizes des Ausgangspunkts, den das Modul berechnet.
An den Anfang der Ausgangsdatei wird stets ein Kommentarblock geschrieben, der das Erzeugungsdatum, die Transformationsgröße und den Ausgangspunkt beinhaltet. Diesem Abschnitt folgt dann direkt die Definition des Koeffizientenmoduls mit seinen Ein- und Ausgängen. Danach folgen einige Konfigurationsparameter wie Bitbreiten und Limits für

die Zähler und je nach Mustertyp können hier noch zusätzliche Parameter dazukommen. Nach den Parametern folgen die im Koeffizientenmodul benötigten Wire- und Register-Deklarationen, die für alle Mustertypen benötigt werden. Danach folgen die Deklarationen, die nur für das jeweilige Muster benötigt werden. Darauf folgt die Instanziierung des Akkumulators und des Multiplizierermoduls, was für alle Mustertypen gleichermaßen erforderlich ist. Für das Akkumulatormodul wird immer dasselbe Modul instanziiert, während das Multiplizierermodul für jedes Koeffizientenmodul individuell erzeugt wird. Das Multiplizierermodul muss für jedes Koeffizientenmodul unterschiedlich sein, da es die modulspezifischen Konstantenmultiplizierer instanziiert. Die eigentliche Multipliziererlogik wird dadurch nicht verändert. Für die Multipliziererlogik existieren zwei verschiedene Varianten, eine zur parallelen und eine zur sequenziellen Multiplikation. Das individuelle Multiplizierermodul wird als separates Verilog-Modul ans Ende der Ausgangsdatei geschrieben. Den Instanziierungen folgt die ebenfalls in jedem Modul vorhandene Reset-Logik und im Falle der sequenziellen Multiplikation die sample_strobe Logik. Danach folgt die Modullogik, die je nach Muster unterschiedlich ausfällt.

Modullogikerzeugung für Zeilenmuster

Die Logikerzeugung für Zeilenmuster ist in einfache und komplexe Muster unterteilt. Für die einfachen Muster wird lediglich ein ROM erzeugt, das der aktuellen Spalte einen Koeffizientenindex und die Vorzeichen zuordnet. Diese ROMs werden durch eine einfache Schleife erzeugt, welche die in der Analyse isolierten Koeffizientenindizes und Vorzeichen sequenziell in die Ausgangsdatei schreibt. Falls beim Muster Vorzeichenwechsel vorkommen, wird zusätzlich noch die Logik zur Invertierung der Vorzeichenbits bei Zeilenwechsel eingefügt. Für die komplexen Muster wird das Ergebnis der Analyse der Koeffizientenfolge genutzt, um die nötigen Zähler zu erzeugen. Dabei wird das Verhalten an den Extrempunkten, wie in der Analyse erkannt, nachgebildet. Dazu kommt ein Vorzeichen-ROM, welches den Koeffizientenindexzählerstand in Koeffizientenvorzeichen umsetzt.

Modullogikerzeugung für Spaltenmuster

Die Logikerzeugung für Spaltenmuster ist ebenfalls in einfache und komplexe Muster unterteilt. Für die einfachen Muster wird lediglich ein ROM erzeugt, das der aktuellen Zeile einen Koeffientenindex und die Vorzeichen zuordnet. Falls sich beim einfachen Muster die Vorzeichen bei Spaltenwechsel umkehren, wird dafür die benötigte Logik eingefügt.

Für die komplexen Spaltenmuster wird die Modulo-Addiererlogik zur Berechnung des nächsten Musterindex eingefügt. Dabei wird die im Analyseschritt erkannte Phasenverschiebung als Parameter im Modul definiert. Das dazugehörige ROM funktioniert im Prinzip wie beim einfachen Spaltenmuster und wird ebenso erzeugt. Jedoch wird nun nicht der Spaltenzähler zur Adressierung verwendet, sondern das Ergebnis des Modulo-Addierers. Als Ausgang liefert das ROM den Koeffizientenindex und die Vorzeichen der Koeffizienten.

Modullogikerzeugung für Trivialmuster

Die Erzeugung der Modullogik für Trivialmuster ist die Einfachste, denn es wird lediglich eine feste Zuweisung von Koeffizientenindex und Vorzeichen in die Ausgangsdatei geschrieben. Für die möglichen Permutationen mit Vorzeichenwechseln wird die einfache Zuweisung der Vorzeichen durch eine Invertierungslogik erweitert. Die Invertierungslogik wird fallspezifisch erzeugt in Abhängigkeit davon, wo die Vorzeichenwechsel auftreten (bei Zeilenwechsel, bei Spaltenwechsel oder beidem). Diese Logik wird separat für Real- und Imaginärteil erzeugt.

Modullogikerzeugung für Zeilen- und Spaltenmuster

Für die Zeilen- und Spaltenmuster werden im Unterschied zu anderen Mustertypen zwei ROMs benötigt. Aus der Analyse sind die Anzahl und Indizes der einzigartigen Zeilen bekannt und ob diese mit invertierten Vorzeichen auftreten. Diese Informationen werden im ersten ROM gespeichert. Es wird vom Zeilenzähler adressiert und liefert einen Zeilenindex und einen Invertierungsstatus als Ausgang. Außerdem liefert die Analyse eine Liste der einzigartigen Koeffizienten und deren Koeffizientenindizes, diese Informationen werden im zweiten ROM gespeichert. Das erste ROM adressiert in Kombination mit dem Spaltenzähler das zweite ROM. Das zweite ROM enthält die selben Informationen wie die ROMs bei anderen Mustern (Koeffizientenindex und Vorzeichen) mit dem Unterschied, dass die Informationen für mehrere Zeilen gespeichert werden müssen.

5.3.4 Modullogikerzeugung für Konstantenmultiplizierer

Nach Erzeugung der Koeffizientenmoduldatei wird die Liste der betragsmäßig einzigartigen Koeffizienten an die generateFixedMultiplier Funktion übergeben. Ist das zweite

Bit der generateFile-Variablen gesetzt, wird ein Konstantenmultiplizierer erzeugt. Dabei bestimmt das dritte Bit, ob dafür kmult verwendet wird. Die erzeugte Ausgangsdatei erhält ebenso wie das Koeffizientenmodul einen individuellen Namen nach dem gleichen Schema. Der Name beginnt stets mit "constMul_", gefolgt von der Transformationsgröße. Weiter folgt dann ein „R" oder „I" abhängig davon, ob der Multiplizierer den Real- oder Imaginärteil der Koeffizienten beinhaltet. Am Ende des Dateinamens stehen die Indizes des Ausgangspunkts.

Die im Gleitkommaformat übergebene Liste von Koeffizienten wird durch Multiplikation und Runden zu ganzen Zahlen gewandelt. Diese Liste von Ganzzahlkoeffizienten wird in einen String geschrieben, den kmult zusammen mit dem Dateinamen als Startparameter erhält. Danach muss in der erzeugten Datei noch der Modulname angepasst werden, denn das von kmult erzeugte Verilog-Modul trägt immer den Namen "adderchain". Da dies zu Namenskonflikten führen würde, wird die erzeugte Ausgangsdatei geöffnet und der String adderchain durch einen nach dem üblichen Schema individualisierten Modulnamen ersetzt. Kmult liegt als GPL-lizenzierte Open-Source-Software vor und kann für jedes übliche Betriebssystem mit einem C++ Compiler erstellt werden.

Wenn die Verwendung von kmult deaktiviert ist, werden die Konstanten als Faktoren, zwischen denen mit einem Multiplexer umgeschaltet wird, in die Ausgangsdatei geschrieben. Diese Form wird von den Synthesewerkzeugen erkannt und ebenfalls zu Konstantenmultiplizierern synthetisiert, allerdings ohne ressourcensparende Wiederverwendung von Zwischentermen. Die von den Synthesewerkzeugen erzeugte Logik ist in etwa gleich groß, wie die von kmult erzeugte, wenn nur ein Koeffizient benötigt wird (z.B. 8x8). Bei mehreren Koeffizienten ist kmult jedoch weit überlegen.

5.4 genDftModuleToplevel.m

Dieses Skript erzeugt das Verilog-Toplevelmodul. Es instanziiert eines oder mehrere Koeffizientenmodule und enthält die Zeilen- und Spaltenzähler.

5.4.1 Parameter

- nPoints - Transformationsgröße
- outputDftTerms - Liste von Ausgangspunkten oder einzelner Punkt

- inputBitwidth - Bitbreite der Eingangsdaten
- accumulatorBitwidth - Bitbreite des Akkumulatorausgangs
- twoCycle - Bestimmt, ob sequenzielles Multiplikationsverfahren verwendet wird
- generateFile - Bestimmt, ob eine Ausgangsdatei geschrieben wird

5.4.2 Funktion

GenDftModuleToplevel hat, wenn das entsprechende Bit (5) in generateFile nicht gesetzt ist, keinerlei Funktion. Je nachdem, ob das Skript mit einem Ausgangspunkt oder mehreren aufgerufen wurde, besitzt es geringfügig unterschiedliches Verhalten. Wird nur ein Punkt angegeben, werden dessen Indizes nach dem bekannten Schema im Namen des Moduls und dem Namen der Ausgangsdatei genannt. Damit werden bei den Massentests Namenskonflikte verhindert. Für mehrere Ausgangspunkte enthält der Datei- und Modulname nur die Transformationsgröße MxM.

Die Erzeugung der Ausgangsdatei beginnt genau wie beim Koeffizientenmodul damit, dass ein Header in die Ausgangsdatei geschrieben wird. Auf den Header folgt die Moduldefinition mit Ein- und Ausgängen, angepasst auf die konfigurierten Bitbreiten. Danach werden mit einer Schleife die Koeffizientenmodule aller in outputDftTerms angegebenen Ausgangspunkte instanziiert. Darauf folgt die Erzeugung der Zeilen- und Spaltenzähler, diese werden für die Transformationsgröße entsprechend ausgelegt. Dazu kommt der Zustandsautomat, der die Kontrollsignale für die Koeffizientenmodule erzeugt, die Abbruchbedingung wird entsprechend der Transformationsgröße angepasst. Der Zustandsautomat erhält, wenn die sequenzielle Multiplikation verwendet wird, einen zusätzlichen Wartezustand. Außerdem wird für die sequenzielle Variante eine sample_strobe Logik für den Spaltenzähler eingefügt. Der Ausgangsmultiplexer wird automatisch der Anzahl der Module angepasst, wird nur ein Modul instanziiert, wird er weggelassen.

5.5 genDftModuleTestbench.m

Dieses Skript erzeugt die Testbenchdatei für das Verilog-Toplevelmodul.

5.5.1 Parameter

- nPoints - Transformationsgröße
- Transformcoefficients - Transformationskoeffizienten
- outputDftTerms - Liste der Ausgangspunkte in Form eines Spaltenvektors
- inputBitwidth - Eingangsbitbreite
- coefFracBitwidth - Anzahl der Nachkommabits für die DFT-Koeffizienten
- accumulatorBitwidth - Bitbreite des Akkumulators
- twoCycle - Bestimmt, ob sequenzielles Multiplikationsverfahren verwendet wird
- generateFile - Bestimmt, ob eine Ausgangsdatei geschrieben wird

5.5.2 Funktion

Dieses Skript funktioniert ähnlich wie genDftModuleToplevel, da es mit einem oder mehreren Ausgangspunkten aufgerufen werden kann. Der Dateiname der Testbench beginnt entweder mit tb_coefficient_MxM_Y_X im Fall eines einzigen Ausgangspunktes oder mit tb_coefficients_MxM bei mehreren Ausgangspunkten. Der Name endet jedoch immer auf _toplevel.v. Das Skript liefert nur eine Ausgangsdatei, wenn Bit (4) in generateFile gesetzt ist. Zu Beginn wird mit rand() eine zufällige Matrix von komplexen Eingangssamples erzeugt. Diese werden dann auf den Ganzzahl-Wertebereich der Module skaliert, um Vollaussteuerung zu erreichen. Die Samples werden im Int16 Format in Matlab gespeichert. Dann werden die Testdaten in einer Gleitkomma-2D-FFT verarbeitet, um einen Referenzwert für spätere Vergleiche zu erhalten. Außerdem werden die Testdaten als Eingangssignale in die Testbenchdatei geschrieben. Danach wird für jeden Ausgangspunkt eine Festkomma-2D-DFT durchgeführt. Die Berechnung erfolgt auf dieselbe Weise wie in den Modulen. Dazu werden Ganzzahltypen (Int) verwendet, die Matlab Festkomma-Toolbox wurde nicht benutzt. Die Simulation produziert die gleichen numerischen Fehler wie die echte Hardware und damit steht ein exakter Vergleichswert zur Verifikation zur Verfügung. Am Ende der Simulation werden auf der Matlab Konsole die Ergebnisse der Gleit- und Festkomma-DFT sowie die relative Abweichung ausgegeben. Die Simulation wird auch bei deaktivierter Dateierzeugung durchgeführt. Die Festkomma-Referenzwerte

werden als Abfrage in die Testbench eingefügt und diese liefert dann auf der Simulationskonsole „OK“ oder „FAIL“ als Ergebnis. Diese Ausgabe wird bei den Massentests für die Funktionsüberprüfung verwendet.

5.6 genVhdlDftModule.m

Dieses Skript erzeugt das DFT-Modul und den Konstantenmultiplizierer für die Skalierung der Ergebnisse.

5.6.1 Parameter

- nPoints - Transformationsgröße
- Transformcoefficients - Transformationskoeffizienten
- outputDftTerms - Liste der Ausgangspunkte
- inputBitwidth - Eingangsbitbreite
- outputBitwidth - Ausgangsbitbreite
- coefFracBitwidth - Anzahl der Nachkommabits für die DFT-Koeffizienten
- accumulatorBitwidth - Bitbreite des Akkumulators
- twoCycle - Bestimmt, ob sequenzielles Multiplikationsverfahren verwendet wird
- generateFile - Bestimmt, ob eine Ausgangsdatei geschrieben wird

5.6.2 Funktion

Ist das entsprechende Bit (6) in generateFile nicht gesetzt, hat dieses Skript keine Funktion. Die Ausgangsdatei trägt stets den Namen “dft_Module_“, gefolgt von der Transformationsgröße MxM. Außer den Bitbreiten wird an diesem Modul relativ wenig automatisch angepasst. Unterschiede kommen nur durch die Anzahl der Ausgangspunkte zustande. Wenn nur ein Ausgangspunkt angegeben wurde, wird die Kontrolllogik, die den Ausgangsmultiplexer des Verilog-Toplevelmoduls steuert, nicht erzeugt und der instanziierte Modulname wird entsprechend angepasst. Die Anfangsadressen für Lesen und

Schreiben können vom Nutzer per generic in der DFT-Moduldatei angepasst werden. Die Erzeugung des Konstantenmultiplizierers für die Skalierung erfolgt wieder durch kmult. Die Eingangsbitbreite des Multiplizierers entspricht der Ausgangsbitbreite des Akkumulators. Bit (7) in generateFile bestimmt, ob der Konstantenmultiplizierer erzeugt wird, während Bit (3) kontrolliert, ob dabei kmult verwendet wird. Zu Vermeidung von Namenskonflikten enthält der Modul- und Dateiname des Multiplizierers den realisierten Skalierungsfaktor als Ganzzahl.

5.7 genVhdlTestbench.m

Dieses Skript erzeugt eine Testbench für das Toplevel des Testsystems.

5.7.1 Parameter

- nPoints - Transformationsgröße
- Transformcoefficients - Transformationskoeffizienten
- outputDftTerms - Liste der Ausgangspunkte
- inputBitwidth - Eingangsbitbreite
- coefFracBitwidth - Anzahl der Nachkommabits für die DFT-Koeffizienten
- accumulatorBitwidth - Bitbreite des Akkumulators
- generateFile - Bestimmt, ob eine Ausgangsdatei geschrieben wird

5.7.2 Funktion

Ist das entsprechende Bit (8) in generateFile nicht gesetzt, wird keine Ausgangsdatei geschrieben. Die Ausgangsdatei trägt stets den Namen “tb_toplevel_“, gefolgt von der Transformationsgröße MxM. Wieder wird eine Matrix mit zufälligen Testdaten erzeugt. Die Testdaten werden in passender Form für das externe Interface des Testsystems in die Testbench geschrieben. Das bedeutet, sie werden byteweise in den BRAM geschrieben. Nachdem der Testdatensatz vollständig im BRAM liegt, wird die Berechnung gestartet

und am Ende werden die Ergebnisse aus dem BRAM gelesen. Dies entspricht einem Ende-zu-Ende Systemtest. Wie in genDftModuleTestbench wird eine Festkommasimulation für alle angegebenen Ausgangspunkte durchgeführt, mit dem Unterschied, dass auch der skalierende Multiplizierer simuliert wird. Ein automatischer Ergebnisvergleich erfolgt nicht, die Vergleichswerte werden lediglich als Kommentar in der Testbench hinterlegt.

5.8 brute2snr.m

Dieses Skript dient dazu, den Signalrauschabstand einer Implementierung zu berechnen und wird nicht automatisch gestartet. Es verwendet dazu die von brute2dft erzeugten Koeffizienten und Variablen, weshalb dieses Skript erst nach einem brute2dft Durchlauf verwendet werden kann.

5.8.1 Verfahren

Das Verfahren zur Ermittlung des SQNR wurde dem Abschnitt 3.2 des Artikels von Wang, Kuo und Jou [6] entnommen. Mit den diskretisierten Eingangswerten wird eine Gleitkomma-Fast Fourier Transform (FFT) und eine Festkomma-FFT durchgeführt. Es wird das Verhältnis zwischen der Quadratsumme der Ergebnisse der Gleitkomma-FFT und der Quadratsumme der Abweichungen zwischen den Beträgen der Gleit- und Festkomma-DFT gebildet. Um das Verfahren auf den 2D-Fall anzupassen, wurden die Summenformeln auf alle Eingangswerte erweitert. Außerdem wird nun eine 2D-FFT für die Gleitkommareferenz genutzt.

$$\mathrm{SQNR} = 10log_{10}\left(\frac{\sum_{y=0}^{M-1}\sum_{x=0}^{M-1}|F(y,x)|^2}{\sum_{y=0}^{M-1}\sum_{x=0}^{M-1}\left(|F(y,x)| - |\acute{F}(y,x)|\right)^2}\right) \tag{5.2}$$

In Formel 5.2 enthält $F(y,x)$ das Ergebnis der Gleitkomma-2D-DFT, während $\acute{F}(y,x)$ das Ergebnis der Festkomma-Simulation enthält.

5.8.2 Ablauf

Die Ergebnisse werden unter Verwendung des Festkomma-Modells ermittelt, welches auch bei der Erzeugung der Testbenches verwendet wird. Die Parameter und Transformationskoeffizienten für das Modell werden dem aktuellen Workspace entnommen. Für die Ermittlung des SQNR muss eine Anregung mit der größtmöglichen Signalstärke erzeugt werden. Dafür wird, wie bei der Testbencherzeugung, eine zufällige Eingangsmatrix erzeugt, die den gesamten darstellbaren Wertebereich abdeckt. Danach wird die Eingangsmatrix jeweils der Gleit- und der Festkomma-2D-DFT unterzogen. Dann werden die Quadratsummen der Gleitkomma-2D-DFT und der Abweichung zwischen Gleit- und Festkomma-DFT gebildet. Am Ende wird das logarithmische Verhältnis zwischen den Quadratsummen als SQNR ausgegeben.

Der SQNR wird für zwei Stufen der Signalverarbeitungskette getrennt ermittelt. Die erste Stufe ist der SQNR auf Ebene des Akkumulators und wird für die Dimensionierung der Nachkommabits der Koeffizienten genutzt. Die zweite Stufe ist der SQNR nach der Skalierung am Ausgang des Designs. Dieser Wert wird für die Dimensionierung der Nachkommabits der Skalierung genutzt und gibt die Genauigkeit der gesamten Signalverarbeitungskette an.

Der ermittelte SQNR gilt immer für die gesamte 2D-DFT und nicht für einen spezifischen Ausgangspunkt. Da zufällige Eingangsdaten verwendet werden, kommt es zu Abweichungen zwischen verschiedenen Durchläufen. Daher wurde eine Mittelwertbildung in brute2snr integriert, bei der 20 Simulationen durchgeführt und deren Ergebnisse gemittelt werden. Eine höhere Anzahl Iterationen kann von Nutzer festgelegt werden, wenn eine höhere Präzision benötigt wird.

6 Ergebnisse

In diesem Kapitel werden die Ergebnisse der Tests, die mit der erzeugten 2D-DFT-Hardware durchgeführt wurden, vorgestellt. Die Implementierung wurde in Bezug auf Funktionalität, Genauigkeit, Flächenbedarf und Geschwindigkeit überprüft.

6.1 Verifikation der Funktionalität

Die Funktionalität von brute2dft wurde anhand der generierten Testbenches sowie mit dem realen Testsystem überprüft.

6.1.1 Funktionale Simulation

Die Funktionalität der von brute2Dft generierten HDL-Quellcodes wurde im Bereich 3x3 bis 28x28 Punkte vollständig verifiziert. Das bedeutet, dass für alle möglichen Punkte aller Matrixgrößen ein Koeffizientenmodul und Toplevel sowie die dazugehörige Testbench generiert wurde. Die erzeugten Dateien wurden dann einer funktionalen Simulation unterzogen. Dafür wurde der Verilog-Simulator Icarus Verilog in Kombination mit einem Bash-Skript verwendet.

6.1.2 Tests mit dem Testsystem

Für die Anwendungsfälle 8x8 und 15x15 wurden für jeden Mustertyp Timing-Simulationen der FPGA-Implementierung inklusive des Testsystems durchgeführt. Diese Testfälle wurden dann auch auf dem realen Testsystem auf Funktionalität überprüft. Das Testsystem besteht aus einem Zedboard mit einem Xilinx Zynq XC7Z020 FPGA und einem Texas-Instruments Connected LaunchPad mit einem TM4C1294 Mikrocontroller. Alle durchgeführten Tests verliefen erfolgreich. Für eine vollständige Verifikation aller möglichen

Ausgangspunkte auf dem Testsystem war im Rahmen dieser Arbeit nicht genug Zeit vorhanden.

6.2 Ergebnisse der Koeffizientenmusteranalyse für verschiedene Matrixgrößen

In diesem Abschnitt sind für einige Matrixgrößen die Koeffizientenmustertypen dargestellt, wie sie von genDftModule klassifiziert wurden. Tabelle 6.1 ist die Legende für die Muster-Codes, die von genDftModule verwendet werden.

Muster Art	**Muster-Code**
Fehler bei der Erkennung	0
einfaches Spaltenmuster	1
einfaches Zeilenmuster	2
komplexes Zeilenmuster	3
komplexes Spaltenmuster	4
Trivialmuster	5
Zeilen- und Spaltenmuster	6
einfaches Spaltenmuster mit Vorzeichenwechsel	7
einfaches Zeilenmuster mit Vorzeichenwechsel	8

Tabelle 6.1: Legende für Koeffizientenmuster-Codes

3x3:

5	2	2
1	3	3
1	3	3

4x4:

5	2	5	2
1	3	7	3
5	8	5	8
1	3	7	3

5x5:

5	2	2	2	2
1	3	3	3	3
1	3	3	3	3
1	3	3	3	3
1	3	3	3	3

6x6:

5	2	2	5	2	2
1	3	4	7	4	3
1	3	3	7	3	3
5	8	8	5	8	8
1	3	3	7	3	3
1	3	4	7	4	3

Transformationsgröße 7x7:

5	2	2	2	2	2	2
1	3	3	3	3	3	3
1	3	3	3	3	3	3
1	3	3	3	3	3	3
1	3	3	3	3	3	3
1	3	3	3	3	3	3
1	3	3	3	3	3	3

Transformationsgröße 8x8:

5	2	2	2	5	2	2	2
1	3	4	3	7	3	4	3
1	3	3	3	7	3	3	3
1	3	4	3	7	3	4	3
5	8	8	8	5	8	8	8
1	3	4	3	7	3	4	3
1	3	3	3	7	3	3	3
1	3	4	3	7	3	4	3

Transformationsgröße 9x9:

5	2	2	2	2	2	2	2	2
1	3	3	4	3	3	4	3	3
1	3	3	4	3	3	4	3	3
1	3	3	3	3	3	3	3	3
1	3	3	4	3	3	4	3	3
1	3	3	4	3	3	4	3	3
1	3	3	3	3	3	3	3	3
1	3	3	4	3	3	4	3	3
1	3	3	4	3	3	4	3	3

Transformationsgröße 10x10:

5	2	2	2	2	5	2	2	2	2
1	3	4	3	4	7	4	3	4	3
1	3	3	3	3	7	3	3	3	3
1	3	4	3	4	7	4	3	4	3
1	3	3	3	3	7	3	3	3	3
5	8	8	8	8	5	8	8	8	8
1	3	3	3	3	7	3	3	3	3
1	3	4	3	4	7	4	3	4	3
1	3	3	3	3	7	3	3	3	3
1	3	4	3	4	7	4	3	4	3

Transformationsgröße 11x11:

5	2	2	2	2	2	2	2	2	2	2
1	3	3	3	3	3	3	3	3	3	3
1	3	3	3	3	3	3	3	3	3	3
1	3	3	3	3	3	3	3	3	3	3
1	3	3	3	3	3	3	3	3	3	3
1	3	3	3	3	3	3	3	3	3	3
1	3	3	3	3	3	3	3	3	3	3
1	3	3	3	3	3	3	3	3	3	3
1	3	3	3	3	3	3	3	3	3	3
1	3	3	3	3	3	3	3	3	3	3
1	3	3	3	3	3	3	3	3	3	3

Transformationsgröße 12x12:

5	2	2	2	2	2	5	2	2	2	2	2
1	3	4	4	4	3	7	3	4	4	4	3
1	3	3	6	4	3	7	3	4	6	3	3
1	3	6	3	6	3	7	3	6	3	6	3
1	3	3	6	3	3	7	3	3	6	3	3
1	3	4	4	4	3	7	3	4	4	4	3
5	8	8	8	8	8	5	8	8	8	8	8
1	3	4	4	4	3	7	3	4	4	4	3
1	3	3	6	3	3	7	3	3	6	3	3
1	3	6	3	6	3	7	3	6	3	6	3
1	3	3	6	4	3	7	3	4	6	3	3
1	3	4	4	4	3	7	3	4	4	4	3

Transformationsgröße 15x15:

5	2	2	2	2	2	2	2	2	2	2	2	2	2	2
1	3	3	4	3	4	4	3	3	4	4	3	4	3	3
1	3	3	4	3	4	4	3	3	4	4	3	4	3	3
1	3	3	3	3	6	3	3	3	3	6	3	3	3	3
1	3	3	4	3	4	4	3	3	4	4	3	4	3	3
1	3	3	6	3	3	6	3	3	6	3	3	6	3	3
1	3	3	3	3	6	3	3	3	3	6	3	3	3	3
1	3	3	4	3	4	4	3	3	4	4	3	4	3	3
1	3	3	4	3	4	4	3	3	4	4	3	4	3	3
1	3	3	3	3	6	3	3	3	3	6	3	3	3	3
1	3	3	6	3	3	6	3	3	6	3	3	6	3	3
1	3	3	4	3	4	4	3	3	4	4	3	4	3	3
1	3	3	3	3	6	3	3	3	3	6	3	3	3	3
1	3	3	4	3	4	4	3	3	4	4	3	4	3	3
1	3	3	4	3	4	4	3	3	4	4	3	4	3	3

6.3 Numerische Präzision & SQNR

Der SQNR und die Genauigkeit der erzeugten 2D-DFT hängt von den vom Nutzer gewählten Parametern ab. Der Nutzer kann brute2snr.m verwenden, um die optimalen Parameter für seine Anforderungen zu ermitteln.

Skalierungs-faktor	Koeff. bits	Skal. bits	Ausgangs-bitbreite	Akku-SQNR(dB)	Ausgangs-SQNR(dB)	Over-Flow
$1/M$	5	14	12	40,22	17,37	O
$1/M^2$	5	14	12	41,86	40,52	
$1/M^2$	6	16	12	46,31	43,61	
$1/M^2$	7	16	12	52,83	45,54	
$1/M^2$	8	16	12	58,07	45,6	
$1/M^2$	9	16	12	62,79	45,9	
$1/M^2$	10	16	12	65,34	45,93	
$1/M$	10	16	12	65,55	16,63	O
$1/M$	10	16	14	65,51	66,33	

Overflow: O=Ausgangsüberlauf

Tabelle 6.2: Ergebnisse: brute2snr für Transformationsgröße 15x15

Für eine Ausgangsbitbreite von 12 Bit bringt eine Erhöhung der Nachkommabits für die Koeffizienten über 7 Bits keinen weiteren SNR-Gewinn mehr. Wird der Skalierungsfaktor reduziert, resultiert dies sofort in Überlauf am Ausgang. Wird die Ausgangsbitbreite auf 14 Bit erhöht und gleichzeitig der Skalierungsfaktor angepasst, kann eine deutlich höhere SNR am Ausgang erreicht werden.

6.4 Flächenbedarf und Geschwindigkeit

6.4.1 Logikzellenbedarf für verschiedene Transformationsgrößen

Für eine Auswahl von Matrixgrößen wurde von jedem Mustertyp je ein Koeffizientenmodul synthetisiert und die Ergebnisse in einer Tabelle festgehalten. Dies erlaubt in Kombination mit der Koeffizientenmusteryp-Übersicht aus dem vorangegangenen Kapitel eine schnelle Aufwandsabschätzung für mögliche Implementierungen. Die Unterschiede im Logikbedarf zwischen verschiedenen Koeffizientenmodulen desselben Mustertyps innerhalb einer Matrixgröße sind vernachlässigbar. Diese Zahlen beinhalten nicht den Logikbedarf für das DFT-Modul und das Verilog-Toplevelmodul.

Synthesebedingungen:

- Halbleiterprozess: Austria Microsystems (AMS) 350 nm CMOS
- Taktfrequenz: 25 Mhz - höhere Zielfrequenzen lösen möglicherweise Optimierungen aus, die den Flächenbedarf erhöhen.
- Typische Performancewerte des Prozesses für parasitäre Elemente, Verzögerungszeiten etc.
- Optimierungsaufwand wurde auf mittleren Aufwand „medium“ konfiguriert
- Verwendete Software: Cadence Genus Version: 17.11-s014_1

Mustertyp(code)	Anzahl Standardzellen	Fläche (μm^2)	Maximale Taktfrequenz (Mhz)
Trivialmuster(5)	141	36.891	106
Trivial mit VZ Wechsel(5)	162	41.259	92
Einfaches Zeilenmuster(2)	296	58.858	80
Zeilenmuster mit VZ Wechsel(8)	300	59.368	80
Einfaches Spaltenmuster(1)	296	58.858	80
Spaltenmuster mit VZ Wechsel(7)	299	59.386	80
Komplexes Zeilenmuster(3)	346	65.028	82
Komplexes Spaltenmuster(4)	311	61.734	80

Tabelle 6.3: Logikbedarf Koeffizientenmodule 8x8

Mustertyp(code)	Anzahl Standardzellen	Fläche (μm^2)	Maximale Taktfrequenz (Mhz)
Trivialmuster(5)	149	39.293	98
Einfaches Zeilenmuster(2)	700	98.170	70
Einfaches Spaltenmuster(1)	700	98.170	70
Komplexes Zeilenmuster(3)	753	103.867	71
Komplexes Spaltenmuster(4)	762	104.067	72
Zeilen- und Spaltenmuster(6)	811	104.304	63

Tabelle 6.4: Logikbedarf Koeffizientenmodule 15x15

6.4.2 Anwendungsbeispiel 15x15 mit 10 Ausgangspunkten:

Es wurde ein Beispielprojekt für den typischen Anwendungsfall synthetisiert.
Zur Auswahl der Ausgangspunkte:
Die Ergebnisse einer 2D-DFT werden üblicherweise mit dem Gleichanteil (Ausgangspunkt 1, 1) in der Mitte dargestellt. Analog zur Darstellung der 1D-DFT mit dem Gleichanteil in der Mitte. Mit dieser Betrachtungsweise wurden die Ausgangspunkte im Kreis um den Gleichanteil angeordnet. Diese Anordnung entspricht dem antizipierten Verwendungszweck im ISAR-Projekt.
Gewählte Ausgangspunkte: (1, 2), (1, 15), (2, 1) ,(2, 2), (2, 15), (15, 1), (15, 2), (15, 15)

Das Projekt beinhaltet das DFT-Modul, das Verilog-Toplevelmodul und 8 Koeffizientenmodule. Das Testsystem wurde dabei nicht mitsynthetisiert.
Ergebnisse:

- Logikzellen: 5037

- Fläche: 718.627 (μm^2)
- Maximale Taktfrequenz: 66 Mhz

6.4.3 Vollständige 8x8 2D-DFT:

Um die Nachteile des Brute-Force Ansatzes zu zeigen wurde eine vollständige 8x8 2D-DFT synthetisiert. Dies ermöglicht auch einen direkten Vergleich mit der Arbeit von Lattmann [3].

	Lattmann	diese Arbeit	Verhältnis
Logikzellen	15.310	22.456	1,46x
Fläche (μm^2)	1.524.960	3.922.828	2,57x
Latenz	512	195(131)	0,38x

Klar erkennbar ist der enorme Flächen- und Logikbedarf, allerdings ist die Latenz um etwa denselben Faktor geringer. ($0,38^{-1} = 2,63x$) Die echte Latenz ist tatsächlich noch geringer, denn in den Registern liegt das Ergebnis schon nach 131 Takten vor. Das sequenzielle Auslesen fügt dann noch weitere 64 Takte hinzu. In Fällen, wo eine 2D-DFT mit extrem kurzer Latenzzeit benötigt wird, wäre es möglich, die automatisch erzeugte Architektur manuell anzupassen und ein paralleles Auslesen der Ergebnisse zu realisieren.

7 Bewertung der Ergebnisse, Zusammenfassung und Ausblick

In diesem Kapitel werden zunächst die erzielten Ergebnisse bewertet und mit den Anforderungen verglichen. Im zweiten Teil wird die Arbeit zusammengefasst. Zum Schluss werden Ansatzpunkte für die Weiterentwicklung genannt.

7.1 Zusammenfassung

Zunächst wurde die 2D-DFT in eine Form gebracht, die die direkte Berechnung individueller Ortsfrequenzen zulässt. In den entstehenden Koeffizienten der 2D-DFT wurden Muster entdeckt und analysiert. Diese Entdeckung wurde in der Entwicklung der Hardwareimplementierung verwendet, um Ressourcen zu sparen. Für jedes auftretende Muster wurde eine eigene Hardwarearchitektur entwickelt. Da eine allgemeine Implementierung das Ziel war, wurde ein Softwareframework zur automatischen Erzeugung der entwickelten Hardwarearchitekturen geschrieben. Es wurde ein Verfahren zur Dimensionierung der Parameter der Hardwaremodule entwickelt und die dafür notwendige Software erstellt. Das erstellte Framework wurde genutzt, um eine Auswahl von Hardwaremodulen zu erzeugen. Die Module wurden für einen ASIC Prozess synthetisiert, um den Hardwareaufwand abschätzen zu können. Außerdem wurden die nötigen Modultypen für eine Anzahl von Matrixgrößen ermittelt. Das Framework und die damit erzeugten Module bieten eine praktikable Lösung für die im ISAR-Projekt gestellten Anforderungen.

7.2 Bewertung der Ergebnisse

Zunächst wird das Erreichte mit den Anforderungen verglichen, die die Grundlage für diese Arbeit darstellen.

Eine Lösung zur separaten Berechnung einzelner Ortsfrequenzen wurde gefunden und in einem Softwareframework implementiert.

Das Softwareframework brute2dft ist in der Lage, für den relevanten Bereich von Matrixgrößen (3x3 bis 18x18) funktionierende Hardwarebeschreibungen zu generieren. Die Bitbreiten der erzeugten Hardware sind frei konfigurierbar und das Framework bietet Möglichkeiten, die resultierende Genauigkeit abzuschätzen. Zudem können Testbenches zur automatischen Verifikation der generierten Module erzeugt werden. Mit dem Framework wird die Vielfalt der möglichen Konfigurationen optimal abgedeckt.

Die Latenzanforderung von 1000 Takten wird auch für die größte angenommene Matrixgröße (18x18) mit 652 Taktzyklen deutlich unterschritten.

Die Kernfrage war, ob durch eine Beschränkung auf wenige Ausgangspunkte eine ausreichend kompakte und schnelle 2D-DFT realisiert werden kann. Zur Beantwortung dieser Frage wird das in Kap. 6.4.2 mithilfe von brute2dft erzeugte Beispielprojekt mit der Arbeit von Koundoul [2] verglichen. In beiden Fällen wird eine 15x15 2D-DFT durchgeführt. Anders als bei Koundoul, berechnet das Beispielprojekt nur 8 der 225 Ausgangspunkte. Die Ergebnisse der Synthese zeigen, dass beide Lösungen einen vergleichbaren Flächenbedarf haben (diese Arbeit: $0,72\text{mm}^2$, Koundoul: $0,70\text{mm}^2$). Dieser Flächenbedarf wurde vom ISAR-Projekt als akzeptabel eingestuft. Das Design von Koundoul benötigt allerdings 10.350 Taktzyklen zur Berechnung der 2D-DFT, was die Latenzanforderung von 1000 Takten um den Faktor 10 überschreitet. Im brute2dft Beispielprojekt ist die Berechnung nach nur 462 Takten abgeschlossen und liegt damit weit unter der Anforderung. Damit ist das Beispielprojekt um den Faktor 22 schneller, als das in der Arbeit von Koundoul realisierte Design. Unter Annahme der Hypothese, dass 8 Punkte zur Winkelberechnung ausreichen, lässt sich somit sagen, dass hier eine praktikable Lösung gefunden wurde.

7.3 Ausblick

Das Softwareframework bietet noch Raum für Verbesserungen. Die Möglichkeit zur Erzeugung von Modulen mit parallelem Datenpfades ist zwar im Framework vorhanden, aber die Implementierung ist weitestgehend ungetestet und enthält noch Fehler. Ein weiterer Ansatzpunkt für Verbesserungen ist die Realisierung der Zeilen- und Spaltenmuster,

deren Hardwareumsetzung bisher nicht optimiert wurde. Zudem bieten auch die Implementierungen der anderen Koeffizientenmuster vermutlich noch Raum für Optimierung. Beispielsweise eine automatische Umschaltung zwischen Zähler- oder ROM-basierten Implementierungen je nachdem, welche den geringeren Flächenbedarf hat. Das im Projekt verwendete field-programmable gate array (FPGA) basierte Testsystem sollte aufgrund der in Kapitel 4.7 identifizierten Probleme überarbeitet werden. Im Zuge der Überarbeitung sollte auch über ein industriekonformes externes Interface, wie zum Beispiel SPI oder I2C, nachgedacht werden.

Literaturverzeichnis

[1] Helck, Jannes: *Digitale Signalverarbeitungs-Module für einen Chipentwurf für ein Sensor-Array*, HAW Hamburg, Bachelor Thesis, 2018

[2] Koundoul, Ada: *Signalverarbeitung für ein magnetisches Sensor-Array als digitaler Chipentwurf.* Berliner Tor 5, 20099 Hamburg, HAW Hamburg, Diplomarbeit, 2018

[3] Lattmann, Thomas: *Chipimplementation einer zweidimensionalen Fouriertransformation für die Auswertung eines Sensor-Arrays*, HAW Hamburg, Bachelor Thesis, 2018

[4] Reichardt, J.: *Digitaltechnik: Eine Einführung mit VHDL.* De Gruyter, 2016 (De Gruyter Studium). – URL https://books.google.de/books?id=qT17DQAAQBAJ. – ISBN 9783110478341

[5] Voronenko, Yevgen ; Püschel, Markus: Multiplierless Multiple Constant Multiplication. In: *ACM Trans. Algorithms* 3 (2007), Mai, Nr. 2. – URL http://doi.acm.org/10.1145/1240233.1240234. – ISSN 1549-6325

[6] Wang, C. ; Kuo, C. ; Jou, J.: Hybrid Wordlength Optimization Methods of Pipelined FFT Processors. In: *IEEE Transactions on Computers* 56 (2007), Aug, Nr. 8, S. 1105–1118. – ISSN 0018-9340

A Anhang

A.1 Matlab Quellcode

A.1.1 Quellcode: brute2dft.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if ~exist('nPoints', 'var')
    nPoints=15; %Transform size
end
if ~exist('outputDftTerm', 'var')
    outputDftTerm=[]; %Output Points
end
if ~exist('inputBitwidth', 'var')
    inputBitwidth=12;  %width of input data
end
if ~exist('outputBitwidth', 'var')
    outputBitwidth=12; %final output data width (from VHDL module after devider)
end
if ~exist('coefFracBitwidth', 'var')
    coefFracBitwidth=7;  %number of fractional bit for coefficients
end
if ~exist('accumulatorBitwidth', 'var')
    accumulatorBitwidth= nextpow2(2^(inputBitwidth-1)*2*nPoints^2)+1; %sets accu width to be 100% worst case safe
end
if ~exist('twoCycle', 'var')
    twoCycle=true; % sets 2 clk per sample or 1 clk per sample
end
if ~exist('outputDividerFactor', 'var')
    outputDividerFactor=1/nPoints^2; %divide output by this Value, default = 1/nPoints^2
end
if ~exist('outputDividerFracBitwidth', 'var')
```

```
    outputDividerFracBitwidth=16;  %this determines the precision of the
    output divider, default = 14
end
%Generate Level:
if ~exist('generateFile', 'var')
    generateFile=false(1,8);
    generateFile(1)=false;  % 1= coefficient module(s)
    generateFile(2)=false; % 2= Coefficient constant multipliers
    generateFile(3)=false; % 3= use kmult
    generateFile(4)=false;  % 4= verilog Toplevel Testbench
    generateFile(5)=false;  % 5= verilog Toplevel
    generateFile(6)=false; % 6= VHDL module
    generateFile(7)=false; % 7= scaling constant multipliers for VHDL Mod
    generateFile(8)=false; % 8= VHDL Toplevel Testbench
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

if nPoints<3
    error('brute2dft: Transform size too small!')
end
if ~twoCycle
    disp('brute2dft: WARNING single cycle Support ist experimental')
end
if outputDividerFactor<=0 || outputDividerFactor>1
    error('brute2dft: Scaling factor error: must be above zero and below or
     equal to one!')
end


Transformcoefficients=complex(zeros(nPoints,nPoints,nPoints,nPoints),zeros(
    nPoints,nPoints,nPoints,nPoints));
for coefidxY = 0:nPoints-1
    for coefidxX = 0:nPoints-1
        for idxY = 0:nPoints-1
            for idxX = 0:nPoints-1
                Transformcoefficients(coefidxY+1,coefidxX+1,idxY+1,idxX+1)=
    exp((-2i*pi*(idxX*coefidxX+idxY*coefidxY))/nPoints);
            end
        end
    end
end


if isempty(outputDftTerm) %generate all possible points
```

```
67      resultMatrix=zeros(nPoints);
        for idxY = 1:nPoints
69          for idxX = 1:nPoints
                DftTerm=[idxX idxY];
71              resultMatrix(idxX,idxY)=genDftModule(nPoints,
        Transformcoefficients, DftTerm, inputBitwidth, coefFracBitwidth,
        accumulatorBitwidth, twoCycle, generateFile);
                genDftModuleTestbench(nPoints, Transformcoefficients, DftTerm,
        inputBitwidth, coefFracBitwidth, accumulatorBitwidth, twoCycle,
        generateFile);
73              genDftModuleToplevel(nPoints, DftTerm, inputBitwidth,
        accumulatorBitwidth, twoCycle, generateFile);
            end
75          disp(['Progress:' num2str((idxY/nPoints)*100,'%3.0f') '%']);
        end
77      if isempty(find(resultMatrix==0,1))
            disp('All identified OK');
79      end
    else %generate a list of points
81      for idx = 1:size(outputDftTerm,1)
            genDftModule(nPoints, Transformcoefficients, outputDftTerm(idx,:,:)
        , inputBitwidth, coefFracBitwidth, accumulatorBitwidth, twoCycle,
        generateFile);
83      end
        genDftModuleToplevel(nPoints, outputDftTerm, inputBitwidth,
        accumulatorBitwidth, twoCycle, generateFile);
85      genDftModuleTestbench (nPoints, Transformcoefficients, outputDftTerm,
        inputBitwidth, coefFracBitwidth, accumulatorBitwidth, twoCycle,
        generateFile);
        genVhdlDftModule(nPoints, outputDftTerm, inputBitwidth, outputBitwidth,
         accumulatorBitwidth, outputDividerFactor, outputDividerFracBitwidth,
        twoCycle, generateFile);
87      genVhdlTestbench(nPoints, Transformcoefficients, outputDftTerm,
        inputBitwidth, accumulatorBitwidth, coefFracBitwidth,
        outputDividerFactor, outputDividerFracBitwidth, generateFile);
    end
```

Listing A.1: brute2dft.m

A.1.2 Quellcode: genDftModule.m

```
function result=gendftmodule(nPoints, Transformcoefficients, outputDftTerm,
    inputBitwidth, coefFracBitwidth, accumulatorBitwidth, twoCycle,
    generateFile)

disp('genDftModule: starting Verilog Generator');
if exist('Transformcoefficients')==0
    disp('genDftModule: Error: generate/load Transform coefficients first')
    ;
    return;
end
if nPoints<3
    disp('genDftModule: Error: nPoints too small');
    return;
end

accuoutputBitwidth=accumulatorBitwidth; %accumulator output width, if not
    equal to accumulatorBitwidth accu output will be truncated and
    saturated

disp(['genDftModule: Output Term: ', num2str(outputDftTerm)]);
outputFileName=['coefficient_',num2str(nPoints), 'x', num2str(nPoints),'_',
    num2str(outputDftTerm(1)),'_',num2str(outputDftTerm(2)),'.v'];
disp(['genDftModule: Output file: ',outputFileName]);

%extract our set of coefficients from the 4D array of matricies
%Array format: coefficient = OutputTermX, OutputTermY, inputValueX,
    inputValueY
%we reduce the significant digits of the coefficients to two digits above
%the final output prescision so that unique() doesnt give too many results
%with differences way below the final accuracy
decimalResulution=ceil(log10(2)*coefFracBitwidth)+2;
realCoef(:,:)=round(real(Transformcoefficients(outputDftTerm(1),
    outputDftTerm(2),:,:))*10^decimalResulution)/10^decimalResulution;
imagCoef(:,:)=round(imag(Transformcoefficients(outputDftTerm(1),
    outputDftTerm(2),:,:))*10^decimalResulution)/10^decimalResulution;


%#########################################################################
%####################START OF PATTERN DETECTION###########################
%#########################################################################
disp('genDftModule: Starting Pattern detection...');
result=5; %first assume trivial pattern
[uniqueCoefReal,UnIndexRealA,UnIndexRealC]=unique(abs(realCoef),'stable');
[uniqueCoefImag,UnIndexImagA,UnIndexImagC]=unique(abs(imagCoef),'stable');
```

```
if length(uniqueCoefReal) == 1 && length(uniqueCoefImag) == 1
37
    flipRealOnSample=false;
39  flipRealOnRow=false;
    flipImagOnRow=false;
41  flipImagOnSample=false;

43  if realCoef(1,1) ~= realCoef(1,2)
        flipRealOnSample=true;
45  end
    if realCoef(1,1) ~= realCoef(2,1)
47      flipRealOnRow=true;
    end
49
    if imagCoef(1,1) ~= imagCoef(1,2)
51      flipImagOnSample=true;
    end
53  if imagCoef(1,1) ~= imagCoef(2,1)
        flipImagOnRow=true;
55  end

57  realCoefGen=abs(realCoef(1,1));
    imagCoefGen=abs(imagCoef(1,1));
59  if flipRealOnSample|| flipRealOnRow || flipImagOnRow ||
    flipImagOnSample
        disp(['genDftModule: Trivial-pattern with sign changes detected,
    Real: on Sample:' num2str(flipRealOnSample) ' on Row:' num2str(
    flipRealOnRow) ...
61          ' Imag: on Sample:' num2str(flipImagOnSample) ' on Row:'
    num2str(flipImagOnRow)]);
    else
63      disp(['genDftModule: Trivial-pattern without sign changes detected'
    ]);
    end
65

else
67  result=0; %keep looking for other patterns
end
69%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%% Simple Column
71%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if result==0
73  result=1; %assume simple column pattern
    for idxX = 2:nPoints  %check that Rows only have ONE coefficient each
```

```
75          if length(unique(abs(realCoef(idxX,:)),'stable')) ~= 1
                result=0;
77          end
            if length(unique(abs(imagCoef(idxX,:)),'stable')) ~= 1
79              result=0;
            end
81      end
        if result == 1 %ok simple column confirmed analyze
83
            if ~any(sign(realCoef(:,1))+sign(realCoef(:,2))) && ~any(sign(
    imagCoef(:,1))+sign(imagCoef(:,2)))
85              invertOnSample=true;
                disp('genDftModule: Simple Column pattern with inverting signs
    on sample detected');
87              result=7;
            else
89              invertOnSample=false;
                disp('genDftModule: Simple Column pattern detected');
91          end

93          [uniqueCoefReal,UnIndexRealA,UnIndexRealC]=unique(abs(realCoef(:,1)
    ),'stable');
            [uniqueCoefImag,UnIndexImagA,UnIndexImagC]=unique(abs(imagCoef(:,1)
    ),'stable');
95          nextcolumnOffset=0;
            numberOfcoefficients=max(UnIndexRealC); %=1
97          realCoefGen=abs(realCoef(1:numberOfcoefficients,1));
            imagCoefGen=abs(imagCoef(1:numberOfcoefficients,1));
99      end
    end
101 if result==0 %this is no column pattern, check for others
        result=2; %assume a simple row pattern
103     for idxX = 2:nPoints  %check that each column has only one coef
            if length(unique(abs(realCoef(:,idxX)),'stable')) ~= 1
105             result=0;
            end
107         if length(unique(abs(imagCoef(:,idxX)),'stable')) ~= 1
                result=0;
109         end
        end
111     if result == 2 %ok simple row confirmed analyze
            if ~any(sign(realCoef(1,:))+sign(realCoef(2,:))) && ~any(sign(
    imagCoef(1,:))+sign(imagCoef(2,:)))
113             invertOnRow=true;
```

```
            disp('genDftModule: Simple Row pattern detected with inverting
    signs on row detected');
            result=8;
        else
            invertOnRow=false;
            disp('genDftModule: Simple Row pattern detected');
        end

        [uniqueCoefReal,UnIndexRealA,UnIndexRealC]=unique(abs(realCoef(1,:)
    ),'stable');
        [uniqueCoefImag,UnIndexImagA,UnIndexImagC]=unique(abs(imagCoef(1,:)
    ),'stable');
        nextRowOffset=0;
        numberOfcoefficients=max(UnIndexRealC); %=1
        realCoefGen=abs(realCoef(1,1:numberOfcoefficients));
        imagCoefGen=abs(imagCoef(1,1:numberOfcoefficients));
        limitPoints=[];
        signType=zeros(1,2);
    end
end
if result==0 %check next possibilty
    result=3; %assume row repeat
    invertOnRow=false; %currently not supported in complex row pattern

    realCoefRowMatch=ismember(unique(abs(realCoef(1,:)),'stable'), unique(
    abs(realCoef(2,:)),'stable'));
    imagCoefRowMatch=ismember(unique(abs(imagCoef(1,:)),'stable'), unique(
    abs(imagCoef(2,:)),'stable'));
    %find the pattern offset between rows
    %this is the difference between the first coefficient of the pattern
    %and starting coefficient of the second row.
    %this offset is constant between rows!
    %current rows first coefficient number + offset = next rows first
    %coefficient number
    matchingNextRowIndex=intersect(find(realCoef(1,:) == realCoef(2,1)),
    find(imagCoef(1,:) == imagCoef(2,1)));
    nextRowOffsetFound=~isempty(matchingNextRowIndex);
%     nextRowOffsetFound = false;
%     for offset=0:nPoints-1
%         if isequal(circshift(realCoef(1,:),offset),realCoef(2,:)) &&
    isequal(circshift(imagCoef(1,:),offset),imagCoef(2,:))
%             nextRowOffsetFound=true;
%             nextRowOffset=offset;
%             break
```

```
%          end
%      end

    %checks that the number of coefficients are the same in each row
    indicating a row pattern
     if  sum(realCoefRowMatch) == length(realCoefRowMatch) && sum(
    imagCoefRowMatch) == length(imagCoefRowMatch) && sum(realCoefRowMatch)
    ==sum(imagCoefRowMatch) && nextRowOffsetFound
         numberOfcoefficients=length(realCoefRowMatch);

         nextRowOffset=matchingNextRowIndex(1)-1; %only select the first
    full match

         [uniqueCoefReal,UnIndexRealA,UnIndexRealC]=unique(abs(realCoef(1,:)
    ),'stable');
         [uniqueCoefImag,UnIndexImagA,UnIndexImagC]=unique(abs(imagCoef(1,:)
    ),'stable');

         disp('genDftModule: shifted row-repeat-pattern detected');
         if numberOfcoefficients > 1
             [limitPoints,signType]=analyzePattern(nPoints,realCoef(1,:),
    imagCoef(1,:));
             if ~any(signType) || ~any(limitPoints) %not detected sign type
    or limit point type
                 result=0;
             else
                 realCoefGen=abs(realCoef(1,1:numberOfcoefficients));
                 imagCoefGen=abs(imagCoef(1,1:numberOfcoefficients));
             end
         else %number of coef < 1
             result=0;
         end
     else  %not a row pattern!
         result=0;
     end
end %result==0

if result == 0
     result =4; %assume complex column pattern
     realcolumnMatch=ismember(unique(abs(realCoef(:,1)),'stable'), unique(
    abs(realCoef(:,2)),'stable'));
     imagcolumnMatch=ismember(unique(abs(imagCoef(:,1)),'stable'), unique(
    abs(imagCoef(:,2)),'stable'));
     numberOfcoefficients=length(realcolumnMatch);
```

```
185     %find the pattern offset between rows this is the difference between
        the first coefficient of the pattern
        %and starting coefficient of the second column. this offset is constant
         between columns!
187     %current columns first coefficient number + offset = next columns first
         coefficient number
        matchingNextcolumnIndex=intersect(find(realCoef(:,1) == realCoef(1,2)),
         find(imagCoef(:,1) == imagCoef(1,2)));
189     %checks that the number of coefficients are the same in each coloummn
        if  sum(realcolumnMatch) == length(realcolumnMatch) && sum(
        imagcolumnMatch) == length(imagcolumnMatch) && sum(realcolumnMatch)==
        sum(imagcolumnMatch) && ~isempty(matchingNextcolumnIndex)
191         disp('genDftModule: complex column pattern detected, analysing...')
        ;
            invertOnSample=false; %we dont support complex column + inverted
        sample yet
193         nextcolumnOffset=matchingNextcolumnIndex(1)-1; %only select the
        first full match
            [uniqueCoefReal,UnIndexRealA,UnIndexRealC]=unique(abs(realCoef(:,1)
        ),'stable');
195         [uniqueCoefImag,UnIndexImagA,UnIndexImagC]=unique(abs(imagCoef(:,1)
        ),'stable');
            realCoefGen=abs(realCoef(1:numberOfcoefficients,1));
197         imagCoefGen=abs(imagCoef(1:numberOfcoefficients,1));
        else
199         result=0;
        end
201 end %result==0

203 if result == 0
        result =6; %assume row and column pattern the end all be all class of
        pattern
205     %this can deal with literally EVERYTHING you throw at it
        MatchingRows=zeros(1,nPoints);
207     for matchRow=1:nPoints
            for row=1:nPoints
209             if realCoef(matchRow,:)==realCoef(row,:) & imagCoef(matchRow,:)
        ==imagCoef(row,:)
                    MatchingRows(row)=matchRow;
211             elseif realCoef(matchRow,:)== -realCoef(row,:) & imagCoef(
        matchRow,:)== -imagCoef(row,:)
                    MatchingRows(row)=-matchRow;
213             end
            end
```

```
215         if isempty(find(MatchingRows==0))
                break %stop matching if everything has been matched
217         end
    end
219     UniqueRowsCounts=unique(abs(MatchingRows));

221     realCoefCont=[];
    imagCoefCont=[];
223     for row=1:length(UniqueRowsCounts) %assemble a list of unique
    coefficients
            realCoefCont=[realCoefCont realCoef(UniqueRowsCounts(row),:)];
225         imagCoefCont=[imagCoefCont imagCoef(UniqueRowsCounts(row),:)];
    end
227     [uniqueCoefReal,UnIndexRealA,UnIndexRealC]=unique(abs(realCoefCont),'
    stable');
    [uniqueCoefImag,UnIndexImagA,UnIndexImagC]=unique(abs(imagCoefCont),'
    stable');
229     %checks we have some unique rows and we have matched them all
    if  all(UnIndexRealC==UnIndexImagC) && isempty(find(MatchingRows==0,1))
231         disp('genDftModule: complex row and column pattern detected');


233
            realCoefGen=uniqueCoefReal;
235         imagCoefGen=uniqueCoefImag;
            numberOfcoefficients=length(uniqueCoefReal);
237     else
            result=0;
239     end
end %result==0
241
disp(['genDftModule: finished analysis, Pattern ID:' num2str(result)]);
243
if (result==3 || result==2 || result==8) && generateFile(1) %complex row
    pattern
    coefCounterWidth=nextpow2(numberOfcoefficients);
245 %open output file
    fileID = fopen(outputFileName,'w');
247 writeHeader(fileID,nPoints, outputDftTerm,inputBitwidth,
    accuoutputBitwidth, twoCycle);
    fprintf(fileID, 'parameter ctr_width=%d;\n', nextpow2(nPoints));
249 fprintf(fileID, 'parameter coef_width=%d;\n', coefCounterWidth);
    fprintf(fileID, 'parameter coef_limit=%d; //number of coefficients-1\n'
251 , numberOfcoefficients-1);
```

```
    fprintf(fileID, 'parameter Opt_width=%d; //Output data width\n',
    accuoutputBitwidth);
253 fprintf(fileID, 'parameter Inp_width=%d; //Input data width\n',
    inputBitwidth);
    if ~isempty(limitPoints)
255     fprintf(fileID, 'parameter coef_steps_per_row=%d; //change of index
     of starting coefficient per row\n', nextRowOffset);
    end
257
    writeStandardWiresAndRegs(fileID, twoCycle);
259
    writeGenericRowPatternWiresRegs(fileID, twoCycle,limitPoints, signType,
     invertOnRow);
261 instanciateAccu(fileID,accumulatorBitwidth, twoCycle);
    instanciateMult(fileID,nPoints, outputDftTerm, twoCycle);
263 writeResetLogic(fileID, twoCycle);
    if twoCycle
265     printSamplestrobe(fileID);
    end
267 if invertOnRow
        printRowInvert(fileID);
269 end
    %writeGenericCounters(fileID,twoCycle);
271 if isempty(limitPoints) %if this is a simple pattern
        writeSimpleLookupTable(fileID, realCoef(1,:),imagCoef(1,:),
    UnIndexRealC,'columnCnt');
273 else
        writeGenericRowPatternCounters(fileID, twoCycle, limitPoints,
    signType);
275     writeGenericSignTable(fileID, realCoef(1,1:numberOfcoefficients),
    imagCoef(1,1:numberOfcoefficients));
    end
277 %writeGenericStateMachine(fileID, twoCycle);
    fprintf(fileID, 'endmodule\n\n');
279 if twoCycle
        writeMultiplier_2cy(fileID,nPoints,outputDftTerm);
281 else
        writeMultiplier_1cy(fileID,nPoints,outputDftTerm);
283 end
    fclose(fileID); %close output file
285
    generateFixedMultiplier(0,nPoints,inputBitwidth, coefFracBitwidth,
    coefFracBitwidth, outputDftTerm, realCoefGen,generateFile);
```

```
287     generateFixedMultiplier(1,nPoints,inputBitwidth, coefFracBitwidth,
        coefFracBitwidth, outputDftTerm, imagCoefGen,generateFile);


289
end %end of shifted-row-pattern HDL generation
291
if (result==4 || result==1 || result == 7) && generateFile(1) %complex
    column pattern
293     coefCounterWidth=nextpow2(numberOfcoefficients);
        %open output file
295     fileID = fopen(outputFileName,'w');
        %start writing the customized multiplier module
297     writeHeader(fileID,nPoints, outputDftTerm,inputBitwidth,
        accuoutputBitwidth, twoCycle);
        fprintf(fileID, 'parameter coef_width=%d;\n', coefCounterWidth);
299     fprintf(fileID, 'parameter ctr_width=%d;\n', nextpow2(nPoints));
        fprintf(fileID, 'parameter column_limit=%d; //when this is reached
        calculation is done\n', nPoints-1);
301     fprintf(fileID, 'parameter coef_limit=%d; //number of coefficients-1\n'
        , numberOfcoefficients-1);
        fprintf(fileID, 'parameter Opt_width=%d; //Output data width\n',
        accuoutputBitwidth);
303     fprintf(fileID, 'parameter Inp_width=%d; //Input data width\n',
        inputBitwidth);
        if nextcolumnOffset>0
305         fprintf(fileID, 'parameter coef_steps_per_sample=%d; //change of
        coefficient per sample\n', nextcolumnOffset);
            fprintf(fileID, 'parameter modulo=%d; //coefficient modulus\n',
        nPoints);
307     end

309     writeStandardWiresAndRegs(fileID, twoCycle);

311     writeGenericColumnPatternWiresRegs(fileID, twoCycle,nextcolumnOffset,
        invertOnSample);
        instanciateAccu(fileID,accumulatorBitwidth, twoCycle);
313     instanciateMult(fileID,nPoints, outputDftTerm, twoCycle);
        writeResetLogic(fileID, twoCycle);
315     if twoCycle
            printSamplestrobe(fileID);
317     end
        if invertOnSample
319         printSampleInvert(fileID);
        end
```

```
321     %writeGenericCounters(fileID,twoCycle);
        if nextcolumnOffset>0
323         writeColumnPatternCounters(fileID, twoCycle, realCoef(:,1),
        imagCoef(:,1));
            writecolumnPatternLookupTable(fileID, realCoef(:,1), imagCoef(:,1),
         UnIndexRealC);
325     else
            writeSimpleLookupTable(fileID, realCoef(:,1),imagCoef(:,1),
        UnIndexRealC,'rowCnt');
327     end
        %writeGenericStateMachine(fileID, twoCycle);
329     fprintf(fileID, 'endmodule\n\n');
        if twoCycle
331         writeMultiplier_2cy(fileID,nPoints,outputDftTerm);
        else
333         writeMultiplier_1cy(fileID,nPoints,outputDftTerm);
        end
335     fclose(fileID); %close output file
        generateFixedMultiplier(0,nPoints,inputBitwidth, coefFracBitwidth,
        coefFracBitwidth, outputDftTerm, realCoefGen,generateFile);
337     generateFixedMultiplier(1,nPoints,inputBitwidth, coefFracBitwidth,
        coefFracBitwidth, outputDftTerm, imagCoefGen,generateFile);
    end %end of shifted-row-pattern HDL generation
339
    if result==5 && generateFile(1) %trivial pattern
341     %open output file
        fileID = fopen(outputFileName,'w');
343     writeHeader(fileID,nPoints, outputDftTerm,inputBitwidth,
        accuoutputBitwidth, twoCycle);
        fprintf(fileID, 'parameter coef_width=%d;\n', 1);
345     fprintf(fileID, 'parameter ctr_width=%d;\n', nextpow2(nPoints));
        fprintf(fileID, 'parameter column_limit=%d; //when this is reached
        calculation is done\n', nPoints-1);
347     fprintf(fileID, 'parameter Opt_width=%d; //Output data width\n',
        accuoutputBitwidth);
        fprintf(fileID, 'parameter Inp_width=%d; //Input data width\n',
        inputBitwidth);
349     writeStandardWiresAndRegs(fileID, twoCycle);
        if realCoef(1,1)>=0
351         realSign=0;
        else
353         realSign=1;
        end
355     if imagCoef(1,1)>=0
```

```
        imagSign=0;
    else
        imagSign=1;
    end
    writeTrivialPatternWiresRegs(fileID, twoCycle);
    instanciateAccu(fileID,accumulatorBitwidth, twoCycle);
    instanciateMult(fileID,nPoints, outputDftTerm, twoCycle);
    writeResetLogic(fileID, twoCycle);
    if twoCycle
        printSamplestrobe(fileID);
    end
    %writeGenericCounters(fileID,twoCycle);
    writeTrivialPatternCounters(fileID, twoCycle, realSign, imagSign,
    flipRealOnSample ,flipRealOnRow, flipImagOnSample, flipImagOnRow) ;
    %writeGenericStateMachine(fileID, twoCycle);
    fprintf(fileID, 'endmodule\n\n');
    if twoCycle
        writeMultiplier_2cy(fileID,nPoints,outputDftTerm);
    else
        writeMultiplier_1cy(fileID,nPoints,outputDftTerm);
    end
    fclose(fileID); %close output file
    generateFixedMultiplier(0,nPoints,inputBitwidth, coefFracBitwidth,
    coefFracBitwidth, outputDftTerm, realCoefGen,generateFile);
    generateFixedMultiplier(1,nPoints,inputBitwidth, coefFracBitwidth,
    coefFracBitwidth, outputDftTerm, imagCoefGen,generateFile);
end %end of trivial pattern HDL generation

if  result==6 && generateFile(1) %complex row & column pattern
    coefCounterWidth=nextpow2(numberOfcoefficients);
    %open output file
    fileID = fopen(outputFileName,'w');
    writeHeader(fileID,nPoints, outputDftTerm,inputBitwidth,
    accuoutputBitwidth, twoCycle);
    fprintf(fileID, 'parameter coef_width=%d;\n', coefCounterWidth);
    fprintf(fileID, 'parameter ctr_width=%d;\n', nextpow2(nPoints));
    fprintf(fileID, 'parameter column_limit=%d; //when this is reached
    calculation is done\n', nPoints-1);
    fprintf(fileID, 'parameter coef_limit=%d; //number of coefficients-1\n'
    , numberOfcoefficients-1);
    fprintf(fileID, 'parameter Opt_width=%d; //Output data width\n',
    accuoutputBitwidth);
    fprintf(fileID, 'parameter Inp_width=%d; //Input data width\n',
    inputBitwidth);
```

```
    fprintf(fileID, 'parameter idx_width=%d; //row index width\n', nextpow2
    (max(MatchingRows)));
393 writeStandardWiresAndRegs(fileID, twoCycle);
    writeRowAndColumnPatternWiresRegs(fileID, twoCycle, ~all(MatchingRows
    >0));
395 instanciateAccu(fileID,accumulatorBitwidth, twoCycle);
    instanciateMult(fileID,nPoints, outputDftTerm, twoCycle);
397 writeResetLogic(fileID, twoCycle);
    if twoCycle
399     printSamplestrobe(fileID);
    end
401 %writeGenericCounters(fileID,twoCycle);
    writeUniqueRowLookupTable(fileID, MatchingRows, ~all(MatchingRows>0));
403 writeRowAndColumnPatternLookupTable(fileID, nPoints, realCoefCont,
    imagCoefCont, UnIndexRealC, MatchingRows);
    %writeGenericStateMachine(fileID, twoCycle);
405 fprintf(fileID, 'endmodule\n\n');
    if twoCycle
407     writeMultiplier_2cy(fileID,nPoints,outputDftTerm);
    else
409     writeMultiplier_1cy(fileID,nPoints,outputDftTerm);
    end
411 fclose(fileID); %close output file
    generateFixedMultiplier(0,nPoints,inputBitwidth, coefFracBitwidth,
    coefFracBitwidth, outputDftTerm, realCoefGen, generateFile);
413 generateFixedMultiplier(1,nPoints,inputBitwidth, coefFracBitwidth,
    coefFracBitwidth, outputDftTerm, imagCoefGen, generateFile);
end %end of row and column pattern HDL generation
415

disp(['genDftModule: end of run for Output Term: ', num2str(outputDftTerm)
    ]);
417

end %of function gendftmodule()
419


421 % run the external fixed multiplier generator program
function generateFixedMultiplier(realImag, nPoints, inputBitwidth,
    constantWidth, FracWidth, outputDftTerm, coefficients, generateFile)
423

    if realImag == 1 %% 0=real 1=imag
425     multipliername=sprintf('constMul_I%dx%d_%d_%d.v', nPoints, nPoints,
     outputDftTerm(1), outputDftTerm(2));
        modulename=sprintf('adderchainI%dx%d_%d_%d',nPoints,nPoints,
    outputDftTerm(1),outputDftTerm(2));
```

```
427     else
            multipliername=sprintf('constMul_R%dx%d_%d_%d.v', nPoints, nPoints,
        outputDftTerm(1), outputDftTerm(2));
429         modulename=sprintf('adderchainR%dx%d_%d_%d',nPoints,nPoints,
        outputDftTerm(1),outputDftTerm(2));
        end
431     coefficientString='';
        for n=1:length(coefficients)
433         coefficientString = [coefficientString, ' ', num2str(round(
        coefficients(n)*2^constantWidth))];
        end
435     if generateFile(2) && generateFile(3)
            if ispc
437             command=sprintf('multiplierGen\\kmult.exe -o %s -i %d -c %d -f
        %d -O 1 %s', ...
                    multipliername, inputBitwidth, constantWidth, FracWidth,
        coefficientString);
439         else
                command=sprintf('./multiplierGen/kmult -o %s -i %d -c %d -f %d
        -O 1 %s', ...
441                 multipliername, inputBitwidth, constantWidth, FracWidth,
        coefficientString);
            end
443         status=system(command);
            if status == 0
445             % rename the module to the correct name
                fid = fopen(multipliername,'rt') ;
447             X = fread(fid);
                fclose(fid);
449             X = char(X.');
                Y = strrep(X, 'adderchain',modulename);
451             fid = fopen(multipliername,'wt');
                fwrite(fid,Y);
453             fclose(fid);
            end
455     elseif generateFile(2)
            disp(['genDftModule: NOT using kmult, using fallback']);
457         fileID = fopen(multipliername,'w');
            fprintf(fileID, '/************************/\n');
459         fprintf(fileID, '/*** fixed multiplier ***/\n');
            fprintf(fileID, '/************************/\n');
461         fprintf(fileID, 'module %s\n',modulename);
            fprintf(fileID, '(\n\t');
463         fprintf(fileID, 'input signed [%d:0] INA,\n\t',inputBitwidth-1);
```

```
        fprintf(fileID, 'input [%d:0] control,\n\t',nextpow2(length(
    coefficients))-1);
465     fprintf(fileID, 'output signed [%d:0] Q\n);\n\t', inputBitwidth-1);
        fprintf(fileID, 'reg signed [%d:0] mulresult;\n\t',inputBitwidth-1+
    constantWidth+2);
467     fprintf(fileID, 'always @(control or INA)\n\t\t');
        fprintf(fileID, 'case(control)\n\t\t\t');
469
        for count = 0:length(coefficients)-1
471         integerCoef=round(coefficients(count+1)*2^constantWidth);
            if count ~= length(coefficients)-1
473             if integerCoef==2^constantWidth
                    fprintf(fileID, '%d''d%d: mulresult <= {INA[%d],INA[%d
    ],INA,%d''d0};\n\t\t\t',nextpow2(length(coefficients)),count,
    inputBitwidth-1,inputBitwidth-1,constantWidth);
475             elseif integerCoef==0
                    fprintf(fileID, '%d''d%d: mulresult <= %d''sd0;\n\t\t\t
    ',nextpow2(length(coefficients)),count,inputBitwidth+constantWidth+2);
477             else
                    fprintf(fileID, '%d''d%d: ',nextpow2(length(
    coefficients)),count);
479                 fprintf(fileID, 'mulresult <= INA*%d''sd%d;\n\t\t\t',
    constantWidth+2,integerCoef );
                end
481         else
                if integerCoef==2^constantWidth
483                 fprintf(fileID, 'default: mulresult <= {INA[%d],INA[%d
    ],INA,%d''d0};\n\t\t\t',inputBitwidth-1,inputBitwidth-1,constantWidth);
                elseif integerCoef==0
485                 fprintf(fileID, 'default: mulresult <= %d''sd0;\n\t\t\t
    ',inputBitwidth+constantWidth+2);
                else
487                 fprintf(fileID, 'default: ');
                    fprintf(fileID, 'mulresult <= INA*%sd''d%d;\n\t\t\t',
    constantWidth+2,integerCoef );
489             end
            end
491     end
        fprintf(fileID, 'endcase\n\t');
493     fprintf(fileID,  'assign Q = mulresult[%d:%d];\n',inputBitwidth-1+
    constantWidth,constantWidth);
        fprintf(fileID, 'endmodule\n');
495     fclose(fileID);
    end
```

```
497 end

499 function writeHeader(fileID, nPoints, outputDftTerm, inputBitwidth,
        outputBitwidth, twoCycle)

501     fprintf(fileID, '/*****************************************/\n');
        fprintf(fileID, '/***      Coefficient Module (%2d, %2d)      ***/\n',
        outputDftTerm(1),outputDftTerm(2));
503     fprintf(fileID, '/***          Transform Size: %dx%d         ***/\n',
        nPoints,nPoints);
        fprintf(fileID, '/***       Built: %s      ***/\n', datetime('now'));
505     fprintf(fileID, '/***   generated by brute2Dft framework   ***/\n');
        fprintf(fileID, '/***     written by Martin Willimczik     ***/\n');
507     fprintf(fileID, '/*****************************************/\n');
        if twoCycle
509         fprintf(fileID, 'module coefficient_%dx%d_%d_%d_2cy(\n\t',nPoints,
        nPoints,outputDftTerm(1),outputDftTerm(2));
        else
511         fprintf(fileID, 'module coefficient_%dx%d_%d_%d_1cy(\n\t',nPoints,
        nPoints,outputDftTerm(1),outputDftTerm(2));
        end
513     fprintf(fileID, 'input CLK,\n\t');
        fprintf(fileID, 'input nReset,\n\t');
515     fprintf(fileID, 'input start,\n\t');
        fprintf(fileID, 'input MultiplierEn,\n\t');
517     fprintf(fileID, 'input AccEnable,\n\t');
        fprintf(fileID, 'input en_coef_cnt,\n\t');
519     fprintf(fileID, 'input [%d:0] columnCnt,\n\t',nextpow2(nPoints)-1);
        fprintf(fileID, 'input row_strobe,\n\t');
521     fprintf(fileID, 'input [%d:0] rowCnt,\n\t',nextpow2(nPoints)-1);
        fprintf(fileID, 'input [%d:0] real_IN,\n\t', inputBitwidth-1);
523     fprintf(fileID, 'input [%d:0] imag_IN,\n\t', inputBitwidth-1);
        fprintf(fileID, 'output [%d:0] real_OUT,\n\t',outputBitwidth-1);
525     fprintf(fileID, 'output [%d:0] imag_OUT\n);\n',outputBitwidth-1);


527
    end
529
    function writeStandardWiresAndRegs(fileID, twoCycle)
531     fprintf(fileID, '/********************/\n');
        fprintf(fileID, '/*** wires & regs ***/\n');
533     fprintf(fileID, '/********************/\n');
        fprintf(fileID, 'reg [coef_width-1:0] coef_cnt;  //coefficient selector
        lines\n');
```

```
535     fprintf(fileID, 'reg coefSign_real, coefSign_imag; //coefficient sign\n
        ');
        fprintf(fileID, 'reg start_reg; //de-glitches start input for reset\n')
        ;
537     fprintf(fileID, 'wire nRESET_int;\n');
        if twoCycle
539         fprintf(fileID, 'wire [Inp_width-1:0] multOutR, multOutI; //outp
        from mult module\n');
        else
541          fprintf(fileID, 'wire [Inp_width:0] multOutR, multOutI; //outp
        from mult module\n');
        end
543
        fprintf(fileID, 'wire accuInvertReal, accuInvertImag; //add-subtract
        control for accumulator\n');
545     if twoCycle
            fprintf(fileID, 'reg sample_strobe; //signifies next sample');
547     end
        fprintf(fileID, '\n');
549 end

551 function writeGenericRowPatternWiresRegs(fileID, twoCycle, limitPoints,
        signType, invertOnRowNeeded)
        if  isempty(limitPoints)
553         invert_imag_needed=false;
            invert_real_needed=false;
555     else
            if (signType(2)==1)
557             invert_imag_needed=false;
            else
559             invert_imag_needed=true;
            end
561         if (signType(1)==1)
                invert_real_needed=false;
563         else
                invert_real_needed=true;
565         end
            fprintf(fileID, 'reg [coef_width-1:0]  next_coef_cnt;\n');
567         fprintf(fileID, 'reg [ctr_width-1:0]  next_coef_step_counter;\n');
        end
569     if  ~isempty(limitPoints)
            fprintf(fileID, 'reg next_coef_direction, coef_direction; // /up
        down\n');
571     end
```

```
573     if  ~isempty(limitPoints)
            if twoCycle == 1
575             fprintf(fileID, 'wire coefCntrEnable = en_coef_cnt & ~
    sample_strobe;\n');
            else
577             fprintf(fileID, 'wire coefCntrEnable = en_coef_cnt;\n');
            end
579     end
        if invertOnRowNeeded
581          fprintf(fileID, 'reg invert_both_signs;\n');
        end
583     if invert_imag_needed
            fprintf(fileID, 'reg invert_imag_sign, next_invert_imag_sign; //
    invert imaginary sign\n');
585         fprintf(fileID, 'wire negateImag = invert_imag_sign ^ coefSign_imag
    ;\n');
        elseif invertOnRowNeeded
587         fprintf(fileID, 'wire negateImag = invert_both_signs ^
    coefSign_imag;\n');
        else
589         fprintf(fileID, 'wire negateImag = coefSign_imag;\n');
        end
591     if invert_real_needed
            fprintf(fileID, 'reg invert_real_sign, next_invert_real_sign; //
    invert real sign\n');
593         fprintf(fileID, 'wire negateReal = invert_real_sign ^ coefSign_real
    ;\n');
        elseif invertOnRowNeeded
595         fprintf(fileID, 'wire negateReal = invert_both_signs ^
    coefSign_real;\n');
        else
597         fprintf(fileID, 'wire negateReal = coefSign_real;\n');
        end
599     fprintf(fileID, '\n');
    end
601
    function writeGenericColumnPatternWiresRegs(fileID, twoCycle,
        nextcolumnOffset,invertOnSample)
603     if nextcolumnOffset>0
            fprintf(fileID, 'reg [coef_width-1:0]  next_coef_cnt;\n');
605         fprintf(fileID, 'reg next_imag_sign, next_real_sign;\n');
            fprintf(fileID, 'reg [ctr_width-1:0]  full_coef_cnt, modulo_add_out
    ;\n');
```

```
607         fprintf(fileID, 'reg [ctr_width:0] modulo_sum, modulo_sum_minus_mod
    ;\n');
    end
609
    if nextcolumnOffset>0
611         if twoCycle
                fprintf(fileID, 'wire coefCntrEnable = en_coef_cnt & ~
    sample_strobe;\n');
613         else
                fprintf(fileID, 'wire coefCntrEnable = en_coef_cnt;\n');
615         end
            fprintf(fileID, 'wire [ctr_width-1:0] full_next_coef_cnt;\n');
617     else
            fprintf(fileID, 'wire coefCntrEnable = en_coef_cnt & row_strobe & ~
    sample_strobe;\n');
619     end
    if invertOnSample
621         fprintf(fileID, 'reg invert_both_signs;\n');
            fprintf(fileID, 'wire negateReal = invert_both_signs ^
    coefSign_real;\n');
623         fprintf(fileID, 'wire negateImag = invert_both_signs ^
    coefSign_imag;\n');
    else
625         fprintf(fileID, 'wire negateReal = coefSign_real;\n');
            fprintf(fileID, 'wire negateImag = coefSign_imag;\n');
627     end
    fprintf(fileID, '\n');
629 end

631 function writeRowAndColumnPatternWiresRegs(fileID, twoCycle,
    signInvertNeeded)
    fprintf(fileID, 'reg [idx_width-1:0] rowIdx; //which row lookup to
    select\n');
633     if signInvertNeeded
            fprintf(fileID, 'reg invert_both_signs;\n');
635         fprintf(fileID, 'wire negateReal = invert_both_signs ^
    coefSign_real;\n');
            fprintf(fileID, 'wire negateImag = invert_both_signs ^
    coefSign_imag;\n');
637     else
            fprintf(fileID, 'wire negateReal = coefSign_real;\n');
639         fprintf(fileID, 'wire negateImag = coefSign_imag;\n');
        end
641     fprintf(fileID, '\n');
```

```
end

function writeTrivialPatternWiresRegs(fileID, twoCycle)
    fprintf(fileID, 'reg next_coefSign_real;\n');
    fprintf(fileID, 'wire negateReal = coefSign_real;\n');
    fprintf(fileID, 'wire negateImag = coefSign_imag;\n');
    fprintf(fileID, '\n');
end

function writeResetLogic(fileID, twoCycle)
    fprintf(fileID, '/********************/\n');
    fprintf(fileID, '/*** reset logic ***/\n');
    fprintf(fileID, '/********************/\n');
    if twoCycle
        fprintf(fileID, 'always@(posedge CLK)\nbegin\n\tstart_reg <= #1
    start;\nend\n');
    else
        fprintf(fileID, 'always@(start)\nbegin\n\tstart_reg <= #1 start;\
    nend\n');
    end
    fprintf(fileID, 'assign #1 nRESET_int = nReset & ~start_reg; //reset on
     external reset or start\n\n');
end


function instanciateAccu(fileID, accuWidth, twoCycle)
    if twoCycle
        fprintf(fileID, 'dft_accumulator #(.Opt_width(Opt_width), .
    Inp_width(Inp_width), .acc_width(%d)) accu\n', accuWidth);
    else
        fprintf(fileID, 'dft_accumulator #(.Opt_width(Opt_width), .
    Inp_width(Inp_width+1), .acc_width(%d)) accu\n', accuWidth);
    end
    fprintf(fileID, '(\n\tCLK,\n\tnRESET_int,\n\t');
    fprintf(fileID, 'accuInvertReal,\n\taccuInvertImag,\n\t');
    fprintf(fileID, 'AccEnable,\n\tmultOutR,\n\tmultOutI,\n\t');
    fprintf(fileID, 'real_OUT,\n\timag_OUT\n);\n');
end

function instanciateMult(fileID, nPoints, outputDftTerm, twoCycle)
    if twoCycle == 1
        fprintf(fileID, 'constMul_%dx%d_%d_%d_2cy ',nPoints,nPoints,
    outputDftTerm(1),outputDftTerm(2));
    else
```

```
        fprintf(fileID, 'constMul_%dx%d_%d_%d_1cy ',nPoints,nPoints,
    outputDftTerm(1),outputDftTerm(2));
    end
    fprintf(fileID, '#(.Inp_width(Inp_width), .coef_width(coef_width))
    multiplier\n');
    fprintf(fileID, '(\n\tCLK,\n\t');
    fprintf(fileID, 'nRESET_int,\n\t');
    fprintf(fileID, 'MultiplierEn,\n\t');
    fprintf(fileID, 'coef_cnt,\n\t');
    fprintf(fileID, 'negateReal,\n\t');
    fprintf(fileID, 'negateImag,\n\t');
    if twoCycle == 1
        fprintf(fileID, 'sample_strobe,\n\t');
    end
    fprintf(fileID, 'real_IN,\n\t');
    fprintf(fileID, 'imag_IN,\n\t');
    fprintf(fileID, 'multOutR,\n\t');
    fprintf(fileID, 'multOutI,\n\t');
    fprintf(fileID, 'accuInvertReal,\n\t');
    fprintf(fileID, 'accuInvertImag\n);\n');
end

%Prints sample strobe logic
function printSamplestrobe(fileID)
    fprintf(fileID, '/**********************/\n');
    fprintf(fileID, '/*** sample strobe ***/\n');
    fprintf(fileID, '/**********************/\n');
    fprintf(fileID, 'always@(posedge CLK, negedge nRESET_int)\nbegin\n\t');
    fprintf(fileID, 'if(!nRESET_int) //synchronizes sample_strobe\n\t\t');
    fprintf(fileID, 'sample_strobe <= #1 1''b0;\n\t');
    fprintf(fileID, 'else if (MultiplierEn)\n\t\t');
    fprintf(fileID, 'sample_strobe <= #1 ~sample_strobe;\n\t');
    fprintf(fileID, 'else\n\t\tsample_strobe <= #1 sample_strobe;\nend\n');
end

%Prints sample invert logic
function printSampleInvert(fileID)
    fprintf(fileID, '/**********************/\n');
    fprintf(fileID, '/*** sample Invert ***/\n');
    fprintf(fileID, '/**********************/\n');
    fprintf(fileID, 'always@(posedge CLK, negedge nRESET_int)\n');
    fprintf(fileID, 'begin\n');
    fprintf(fileID, ' if(!nRESET_int)\n');
    fprintf(fileID, '   invert_both_signs <= #1 1''b0;\n');
```

```
    fprintf(fileID, ' else if (~sample_strobe)\n');
723     fprintf(fileID, '   invert_both_signs <= ~invert_both_signs;\n');
    fprintf(fileID, 'end\n');
725 end

727 %Prints sample invert logic
    function printRowInvert(fileID)
729     fprintf(fileID, '/******************/\n');
    fprintf(fileID, '/*** row Invert ***/\n');
731     fprintf(fileID, '/******************/\n');
    fprintf(fileID, 'always@(posedge CLK, negedge nRESET_int)\n');
733     fprintf(fileID, 'begin\n');
    fprintf(fileID, ' if(!nRESET_int)\n');
735     fprintf(fileID, '   invert_both_signs <= #1 1''b0;\n');
    fprintf(fileID, ' else if (row_strobe & ~sample_strobe)\n');
737     fprintf(fileID, '   invert_both_signs <= ~invert_both_signs;\n');
    fprintf(fileID, 'end\n');
739 end

741 %creates the specific multiplier module for two clk per sample
    %the only customized parts are the instanciation of the fixed multipliers
743 function writeMultiplier_2cy(fileID, nPoints, outputDftTerm)

745     fprintf(fileID, '/*************************/\n');
    fprintf(fileID, '/*** complex multiplier ***/\n');
747     fprintf(fileID, '/*************************/\n');
    fprintf(fileID, 'module constMul_%dx%d_%d_%d_2cy\n\t',nPoints,nPoints,
    outputDftTerm(1),outputDftTerm(2));
749     fprintf(fileID, '#(parameter Inp_width=12, coef_width=3)\n\t(\n\t');
    fprintf(fileID, 'input CLK,\n\tinput nReset,\n\tinput enable,\n\tinput
    [coef_width-1:0] coefficient,\n\t');
751     fprintf(fileID, 'input negateReal,\n\tinput negateImag,\n\t');
    fprintf(fileID, 'input sample_strobe,\n\t');
753     fprintf(fileID, 'input signed [Inp_width-1:0] inReal,\n\tinput signed [
    Inp_width-1:0] inImag,\n\t');
    fprintf(fileID, 'output signed [Inp_width-1:0] outReal,\n\toutput
    signed [Inp_width-1:0] outImag,\n\t');
755     fprintf(fileID, 'output negateRealOut,\n\toutput negateImagOut\n\t);\n\
    t');
    fprintf(fileID, '//register\n\t');
757     fprintf(fileID, 'reg signed [Inp_width-1:0] realMulOut_reg,
    imagMulOut_reg;\n\t');
    fprintf(fileID, 'reg signed [Inp_width-1:0] realIn_reg, imagIn_reg;\n\t
    ');
```

```
759     fprintf(fileID, 'reg negateReal_reg, negateImag_reg, negateReal_reg2,
        negateImag_reg2;\n\t');
        fprintf(fileID, 'reg [coef_width-1:0] coefficient_reg;\n\t');
761     fprintf(fileID, '//wires\n\t');
        fprintf(fileID, 'wire signed [Inp_width-1:0] realMulOut, imagMulOut;\n\
        t');
763     fprintf(fileID, 'wire signed [Inp_width-1:0] realMulIn, imagMulIn;\n\t\
        n\t');

765     fprintf(fileID, '//fixed multiplier instances\n\t');
        fprintf(fileID, 'adderchainR%dx%d_%d_%d realMultiplier(realMulIn,
        coefficient_reg, realMulOut);\n\t',nPoints,nPoints,outputDftTerm(1),
        outputDftTerm(2));
767     fprintf(fileID, 'adderchainI%dx%d_%d_%d imagMultiplier(imagMulIn,
        coefficient_reg, imagMulOut);\n\t\n\t',nPoints,nPoints,outputDftTerm(1)
        ,outputDftTerm(2));

769     fprintf(fileID, 'assign realMulIn = (!sample_strobe) ? realIn_reg :
        imagIn_reg;\n\t');
        fprintf(fileID, 'assign imagMulIn = (!sample_strobe) ? realIn_reg :
        imagIn_reg;\n\t');
771     fprintf(fileID, 'assign outReal = (sample_strobe) ? realMulOut_reg :
        imagMulOut_reg;\n\t');
        fprintf(fileID, 'assign outImag = (sample_strobe) ? imagMulOut_reg :
        realMulOut_reg;\n\t');
773     fprintf(fileID, 'assign negateRealOut = (negateReal_reg2 | ~
        sample_strobe)&(~negateImag_reg2 | sample_strobe);\n\t');
        fprintf(fileID, 'assign negateImagOut = (negateReal_reg2 |
        sample_strobe)&(negateImag_reg2 | ~sample_strobe);\n\t');
775     fprintf(fileID, '//register logic\n\t');
        fprintf(fileID, 'always @ (posedge CLK, negedge nReset)\n\tbegin\n\t\t'
        );
777     fprintf(fileID, 'if(!nReset)\n\t\tbegin\n\t\t\t');
        fprintf(fileID, 'negateReal_reg <= #1 1''b0;\n\t\t\t');
779     fprintf(fileID, 'negateImag_reg <= #1 1''b0;\n\t\t\t');
        fprintf(fileID, 'negateReal_reg2 <= #1 1''b0;\n\t\t\t');
781     fprintf(fileID, 'negateImag_reg2 <= #1 1''b0;\n\t\tend\n\t\t');
        fprintf(fileID, 'else\n\t\tbegin\n\t\t\t');
783     fprintf(fileID, 'negateReal_reg <= #1 negateReal;\n\t\t\t');
        fprintf(fileID, 'negateImag_reg <= #1 negateImag;\n\t\t\t');
785     fprintf(fileID, 'negateReal_reg2 <= #1 negateReal_reg;\n\t\t\t');
        fprintf(fileID, 'negateImag_reg2 <= #1 negateImag_reg;\n\t\tend\n\tend\
        n\t');
787     fprintf(fileID, 'always@(posedge CLK)\n\tbegin\n\t\t');
```

```
    fprintf(fileID, 'if (enable)\n\t\tbegin\n\t\t\t');
789 fprintf(fileID, 'realIn_reg <= #1 inReal;\n\t\t\t');
    fprintf(fileID, 'imagIn_reg <= #1 inImag;\n\t\t');
791 fprintf(fileID, 'end\n\t\t');
    fprintf(fileID, 'else\n\t\tbegin\n\t\t\t');
793 fprintf(fileID, 'realIn_reg <= realIn_reg;\n\t\t\t');
    fprintf(fileID, 'imagIn_reg <= imagIn_reg;\n\t\t');
795 fprintf(fileID, 'end\n\tend\n\t');
    fprintf(fileID, 'always@(posedge CLK)\n\tbegin\n\t\t');
797 fprintf(fileID, 'realMulOut_reg <= #1 realMulOut;\n\t\t\t');
    fprintf(fileID, 'imagMulOut_reg <= #1 imagMulOut;\n\t\t\t');
799 fprintf(fileID, 'coefficient_reg <= #1 coefficient;\n\t');
    fprintf(fileID, 'end\nendmodule\n\n');
801 end

803 %creates the multiplier module for 1 sample per clk
    function writeMultiplier_1cy(fileID, nPoints, outputDftTerm)
805

    fprintf(fileID, '/*************************/\n');
807 fprintf(fileID, '/*** complex multiplier ***/\n');
    fprintf(fileID, '/*************************/\n');
809 fprintf(fileID, 'module constMul_%dx%d_%d_%d_1cy\n\t',nPoints,nPoints,
    outputDftTerm(1),outputDftTerm(2));
    fprintf(fileID, '#(parameter Inp_width=12, coef_width=3, Inp_width=12)\
    n\t(\n\t');
811 fprintf(fileID, 'input CLK,\n\tinput nReset,\n\tinput enable,\n\tinput
    [coef_width-1:0] coefficient,\n\t');
    fprintf(fileID, 'input negateReal,\n\tinput negateImag,\n\t');
813 fprintf(fileID, 'input signed [Inp_width-1:0] inReal,\n\tinput signed [
    Inp_width-1:0] inImag,\n\t');
    fprintf(fileID, 'output signed [Inp_width:0] outReal,\n\toutput signed
    [Inp_width:0] outImag,\n\t');
815 fprintf(fileID, 'output negateRealOut,\n\toutput negateImagOut\n\t);\n\
    t');
    fprintf(fileID, '//register\n\t');
817 fprintf(fileID, 'reg signed [Inp_width-1:0] realMulOut1_reg,
    imagMulOut1_reg;\n\t');
    fprintf(fileID, 'reg signed [Inp_width-1:0] realMulOut2_reg,
    imagMulOut2_reg;\n\t');
819 fprintf(fileID, 'reg signed [Inp_width-1:0] realIn_reg, imagIn_reg;\n\t
    ');
    fprintf(fileID, 'reg negateReal_reg, negateImag_reg, negateReal_reg2,
    negateImag_reg2;\n\t');
821 fprintf(fileID, 'reg [coef_width-1:0] coefficient_reg;\n\t');
```

```
    fprintf(fileID, '//wires\n\t');
823 fprintf(fileID, 'wire signed [Inp_width-1:0] realMulOut1, realMulOut2;\
    n\t');
    fprintf(fileID, 'wire signed [Inp_width-1:0] imagMulOut1, imagMulOut2;\
    n\t\n\t');
825 fprintf(fileID, 'wire invertAdders = negateReal_reg2 ^ negateImag_reg2
    ;\n\t\n\t');

827 fprintf(fileID, '//fixed multiplier instances\n\t');
    fprintf(fileID, 'adderchainR%dx%d_%d_%d realMultiplier1(realIn_reg,
    coefficient_reg, realMulOut1);\n\t',nPoints,nPoints,outputDftTerm(1),
    outputDftTerm(2));
829 fprintf(fileID, 'adderchainI%dx%d_%d_%d imagMultiplier1(imagIn_reg,
    coefficient_reg, imagMulOut1);\n\t\n\t',nPoints,nPoints,outputDftTerm
    (1),outputDftTerm(2));
    fprintf(fileID, 'adderchainR%dx%d_%d_%d realMultiplier2(imagIn_reg,
    coefficient_reg, realMulOut2);\n\t',nPoints,nPoints,outputDftTerm(1),
    outputDftTerm(2));
831 fprintf(fileID, 'adderchainI%dx%d_%d_%d imagMultiplier2(realIn_reg,
    coefficient_reg, imagMulOut2);\n\t\n\t',nPoints,nPoints,outputDftTerm
    (1),outputDftTerm(2));

833 fprintf(fileID, 'assign outReal = (invertAdders) ? realMulOut1_reg +
    imagMulOut1_reg : realMulOut1_reg - imagMulOut1_reg;\n\t');
    fprintf(fileID, 'assign outImag = (invertAdders) ? realMulOut2_reg -
    imagMulOut2_reg : realMulOut2_reg + imagMulOut2_reg;\n\t');
835 fprintf(fileID, 'assign negateRealOut = negateReal_reg2;\n\t');
    fprintf(fileID, 'assign negateImagOut = negateReal_reg2;\n\t');
837 fprintf(fileID, '//register logic\n\t');
    fprintf(fileID, 'always @ (posedge CLK, negedge nReset)\n\tbegin\n\t\t'
    );
839 fprintf(fileID, 'if(!nReset)\n\t\tbegin\n\t\t\t');
    fprintf(fileID, 'negateReal_reg <= #1 1''b0;\n\t\t\t');
841 fprintf(fileID, 'negateImag_reg <= #1 1''b0;\n\t\t\t');
    fprintf(fileID, 'negateReal_reg2 <= #1 1''b0;\n\t\t\t');
843 fprintf(fileID, 'negateImag_reg2 <= #1 1''b0;\n\t\tend\n\t\t');
    fprintf(fileID, 'else\n\t\tbegin\n\t\t\t');
845 fprintf(fileID, 'negateReal_reg <= #1 negateReal;\n\t\t\t');
    fprintf(fileID, 'negateImag_reg <= #1 negateImag;\n\t\t\t');
847 fprintf(fileID, 'negateReal_reg2 <= #1 negateReal_reg;\n\t\t\t');
    fprintf(fileID, 'negateImag_reg2 <= #1 negateImag_reg;\n\t\tend\n\tend\
    n\t');
849 fprintf(fileID, 'always@(posedge CLK)\n\tbegin\n\t\t');
    fprintf(fileID, 'if (enable)\n\t\tbegin\n\t\t\t');
```

```
851     fprintf(fileID, 'realIn_reg <= #1 inReal;\n\t\t\t');
        fprintf(fileID, 'imagIn_reg <= #1 inImag;\n\t\t');
853     fprintf(fileID, 'end\n\t\t');
        fprintf(fileID, 'else\n\t\tbegin\n\t\t\t');
855     fprintf(fileID, 'realIn_reg <= realIn_reg;\n\t\t\t');
        fprintf(fileID, 'imagIn_reg <= imagIn_reg;\n\t\t');
857     fprintf(fileID, 'end\n\tend\n\t');
        fprintf(fileID, 'always@(posedge CLK)\n\tbegin\n\t\t');
859     fprintf(fileID, 'realMulOut1_reg <= #1 realMulOut1;\n\t\t');
        fprintf(fileID, 'imagMulOut1_reg <= #1 imagMulOut1;\n\t\t');
861     fprintf(fileID, 'realMulOut2_reg <= #1 realMulOut2;\n\t\t');
        fprintf(fileID, 'imagMulOut2_reg <= #1 imagMulOut2;\n\t\t');
863     fprintf(fileID, 'coefficient_reg <= #1 coefficient;\n\t');
        fprintf(fileID, 'end\nendmodule\n\n');
865 end


867
    %writes the coefficient sign look up table to the output file
869 function writeGenericSignTable(fileID, realCoef, imagCoef)
        fprintf(fileID, '/*******************************/\n');
871     fprintf(fileID, '/*** coef sign lookup table ***/\n');
        fprintf(fileID, '/*******************************/\n');
873     fprintf(fileID, 'always@(*)\nbegin\n\tcase(coef_cnt)\n\t\t');
        for count = 0:length(realCoef)-1
875         fprintf(fileID, '%d''d%d:\n\t\tbegin\n\t\t\t',nextpow2(length(
        realCoef)),count);
            if sign(realCoef(count+1)) >=0
877             fprintf(fileID, 'coefSign_real = 1''b0;\n\t\t\t');
            else
879             fprintf(fileID, 'coefSign_real = 1''b1;\n\t\t\t');
            end
881         if sign(imagCoef(count+1)) >=0
                fprintf(fileID, 'coefSign_imag = 1''b0;\n\t\t');
883         else
                fprintf(fileID, 'coefSign_imag = 1''b1;\n\t\t');
885         end
            fprintf(fileID, 'end\n\t\t');
887     end
        fprintf(fileID, 'default:\n\t\tbegin\n\t\t\t');
889     fprintf(fileID, 'coefSign_real = 1''b0; //dontcare\n\t\t\t');
        fprintf(fileID, 'coefSign_imag = 1''b0; //dontcare\n\t\t');
891     fprintf(fileID, 'end\n\tendcase\nend\n');
    end
893
```

```
%writes the coefficient sign look up table to the output file
function writecolumnPatternLookupTable(fileID, realCoef, imagCoef,
    UnIndexRealC)
    fprintf(fileID, '/*****************************************/\n');
    fprintf(fileID, '/*** coef sign & coef index lookup table ***/\n');
    fprintf(fileID, '/*****************************************/\n');
    fprintf(fileID, 'always@(*)\nbegin\n\tcase(full_next_coef_cnt)\n\t\t');
    for count = 0:length(realCoef)-1
        fprintf(fileID, '%d''d%d:\n\t\tbegin\n\t\t\t',nextpow2(length(
    realCoef)),count);
        fprintf(fileID, 'next_coef_cnt = %d''d%d;\n\t\t\t',nextpow2(max(
    UnIndexRealC)),UnIndexRealC(count+1)-1 );

        if sign(realCoef(count+1)) >=0
            fprintf(fileID, 'next_real_sign = 1''b0;\n\t\t\t');
        else
            fprintf(fileID, 'next_real_sign = 1''b1;\n\t\t\t');
        end
        if sign(imagCoef(count+1)) >=0
            fprintf(fileID, 'next_imag_sign = 1''b0;\n\t\t');
        else
            fprintf(fileID, 'next_imag_sign = 1''b1;\n\t\t');
        end
        fprintf(fileID, 'end\n\t\t');
    end
    fprintf(fileID, 'default:  //dontcare\n\t\tbegin\n\t\t\t');
    fprintf(fileID, 'next_coef_cnt = %d''d0;\n\t\t\t',nextpow2(max(
    UnIndexRealC)));
    fprintf(fileID, 'next_real_sign = 1''b0;\n\t\t\t');
    fprintf(fileID, 'next_imag_sign = 1''b0; \n\t\t');
    fprintf(fileID, 'end\n\tendcase\nend\n');
end

%writes the coefficient sign look up table to the output file
function writeSimpleLookupTable(fileID, realCoef, imagCoef, UnIndexRealC,
    caseExpression)
    fprintf(fileID, '/*****************************************/\n');
    fprintf(fileID, '/*** coef sign & coef index lookup table ***/\n');
    fprintf(fileID, '/*****************************************/\n');
    fprintf(fileID, 'always@(*)\nbegin\n\tcase(%s)\n\t\t', caseExpression);
    for count = 0:length(realCoef)-1
        fprintf(fileID, '%d''d%d:\n\t\tbegin\n\t\t\t',nextpow2(length(
    realCoef)),count);
```

```
931         fprintf(fileID, 'coef_cnt = %d''d%d;\n\t\t\t',nextpow2(max(
    UnIndexRealC)),UnIndexRealC(count+1)-1 );

933         if sign(realCoef(count+1)) >=0
                fprintf(fileID, 'coefSign_real = 1''b0;\n\t\t\t');
935         else
                fprintf(fileID, 'coefSign_real = 1''b1;\n\t\t\t');
937         end
            if sign(imagCoef(count+1)) >=0
939             fprintf(fileID, 'coefSign_imag = 1''b0;\n\t\t');
            else
941             fprintf(fileID, 'coefSign_imag = 1''b1;\n\t\t');
            end
943         fprintf(fileID, 'end\n\t\t');
        end
945     fprintf(fileID, 'default:  //dontcare\n\t\tbegin\n\t\t\t');
        fprintf(fileID, 'coef_cnt = %d''d0;\n\t\t\t',nextpow2(max(UnIndexRealC)
    ));
947     fprintf(fileID, 'coefSign_real = 1''b0;\n\t\t\t');
        fprintf(fileID, 'coefSign_imag = 1''b0; \n\t\t');
949     fprintf(fileID, 'end\n\tendcase\nend\n');
end
951
%writes the coefficient sign look up table to the output file
953 function writeUniqueRowLookupTable(fileID, MatchingRows, invertAllNeeded)
        fprintf(fileID, '/*****************************************/\n');
955     fprintf(fileID, '/*** row index lookup table ***/\n');
        fprintf(fileID, '/*****************************************/\n');
957     fprintf(fileID, 'always@(*)\nbegin\n\tcase(rowCnt)\n\t\t');
        lutBitwidth=nextpow2(length(MatchingRows));
959     for count = 0:length(MatchingRows)-1
            fprintf(fileID, '%d''d%d:\n\t\tbegin\n\t\t\t',lutBitwidth,count);
961         fprintf(fileID, 'rowIdx = %d''d%d;',nextpow2(max(MatchingRows)),abs
    (MatchingRows(count+1))-1);
            if invertAllNeeded
963         if sign(MatchingRows(count+1)) >=0
                fprintf(fileID, '\n\t\t\tinvert_both_signs = 1''b0;');
965         else
                fprintf(fileID, '\n\t\t\tinvert_both_signs = 1''b1;');
967         end
            end
969         fprintf(fileID, '\n\t\tend\n\t\t');
        end
971     fprintf(fileID, 'default:  //dontcare\n\t\tbegin\n\t\t\t');
```

```
    fprintf(fileID, 'rowIdx = %d''d0;',nextpow2(max(MatchingRows)));
    if invertAllNeeded
        fprintf(fileID, '\n\t\t\tinvert_both_signs = 1''b0;');
    end
    fprintf(fileID, '\n\t\tend\n\tendcase\nend\n');
end

function writeRowAndColumnPatternLookupTable(fileID, nPoints, realCoef,
    imagCoef, UnIndexRealC, MatchingRows)
    fprintf(fileID, '/*****************************************/\n');
    fprintf(fileID, '/*** coef sign & coef index lookup table ***/\n');
    fprintf(fileID, '/*****************************************/\n');
    fprintf(fileID, 'always@(*)\nbegin\n\tcase({rowIdx, columnCnt})\n\t\t')
    ;
    lutBitwidth=nextpow2(max(MatchingRows))+nextpow2(nPoints);

    for rowCount =0:max(MatchingRows)-1
        romCount=0;
        for count = nPoints*rowCount+1:nPoints*(rowCount+1)

            fprintf(fileID, '%d''d%d:\n\t\tbegin\n\t\t\t',lutBitwidth,
    bitshift(uint16(rowCount),nextpow2(nPoints))+romCount);
            fprintf(fileID, 'coef_cnt = %d''d%d;\n\t\t\t',nextpow2(max(
    UnIndexRealC)),  UnIndexRealC(count)-1);

            if sign(realCoef(count)) >=0
                fprintf(fileID, 'coefSign_real = 1''b0;\n\t\t\t');
            else
                fprintf(fileID, 'coefSign_real = 1''b1;\n\t\t\t');
            end
            if sign(imagCoef(count)) >=0
                fprintf(fileID, 'coefSign_imag = 1''b0;\n\t\t');
            else
                fprintf(fileID, 'coefSign_imag = 1''b1;\n\t\t');
            end
            fprintf(fileID, 'end\n\t\t');
            romCount=romCount+1;
        end
    end
    fprintf(fileID, 'default:  //dontcare\n\t\tbegin\n\t\t\t');
    fprintf(fileID, 'coef_cnt = %d''d0;\n\t\t\t',nextpow2(max(UnIndexRealC)
    ));
    fprintf(fileID, 'coefSign_real = 1''b0;\n\t\t\t');
    fprintf(fileID, 'coefSign_imag = 1''b0; \n\t\t');
```

```
1011        fprintf(fileID, 'end\n\tendcase\nend\n');
    end
1013
1015
1017    function writeGenericRowPatternCounters(fileID, twoCycle, limitPoints,
            signType)
        flip_real_top = signType(1)==2 || signType(1) ==4;
1019    flip_real_bottom = signType(1)==3 || signType(1) ==4;
        flip_imag_top=signType(2)==2 || signType(2) ==4;
1021    flip_imag_bottom=signType(2)==3 || signType(2) ==4;
        hold_top=false;
1023    hold_bottom=false;
        %analyze upper limit of counter
1025    if any(ismember(limitPoints,1))
            %upper hold point + count up (no-hold) is not supported
1027        if ~any(ismember(limitPoints,3))
                hold_top=true;
1029        else
                disp('Error: pattern conflict at upper bound');
1031        end
        end
1033    %analyze lower limit of counter
        if any(ismember(limitPoints,4))
1035        %hold point + count up (no-hold) is not supported
            if ~any(ismember(limitPoints,2))
1037            hold_bottom=true;
            else
1039            disp('Error: pattern conflict at lower bound');
            end
1041    end
        %on long patterns the flip back on the lower limit is outside the scope
1043    %of analysis.
        %this leads to the next_coefficient counter to stay flipped when it
1045    %shouldn't so
        if length(find(limitPoints))==1 %
1047        if flip_real_top
                flip_real_bottom=true;
1049        end
            if flip_imag_top
1051            flip_imag_bottom=true;
            end
1053    end
```

```
1055    fprintf(fileID, '/*******************************/\n');
        fprintf(fileID, '/*** next coefficient counter ***/\n');
1057    fprintf(fileID, '/*******************************/\n');
        fprintf(fileID, 'always@(posedge CLK, negedge nRESET_int)\n');
1059    fprintf(fileID, 'begin\n\tif(!nRESET_int)\n\t\t');
        fprintf(fileID, 'next_coef_step_counter <= #1 coef_steps_per_row;\n\t')
        ;
1061    if twoCycle
            fprintf(fileID, 'else if (row_strobe & ~sample_strobe)\n\t\t');
1063    else
            fprintf(fileID, 'else if (row_strobe)\n\t\t');
1065    end
        fprintf(fileID, 'next_coef_step_counter <= #1 coef_steps_per_row;\n\t')
        ;
1067    fprintf(fileID, 'else if (next_coef_step_counter=={coef_width{1''b0}})\
        n\t\t');
        fprintf(fileID, 'next_coef_step_counter <= #1 next_coef_step_counter;\n
        \t');
1069    fprintf(fileID, 'else\n\t\tnext_coef_step_counter <= #1
        next_coef_step_counter-1''b1;\n');
        fprintf(fileID, 'end\n\n');
1071
        fprintf(fileID, 'always@(posedge CLK, negedge nRESET_int)\n');
1073    fprintf(fileID, 'begin\n\tif(!nRESET_int)\n\t');
        fprintf(fileID, 'begin\n\t\t');
1075    fprintf(fileID, 'next_coef_cnt <= #1 {coef_width{1''b0}};\n\t\t');
        fprintf(fileID, 'next_coef_direction <= #1 1''b0; //default direction
        is up');
1077    if flip_real_top || flip_real_bottom
            fprintf(fileID, '\n\t\tnext_invert_real_sign <= #1 1''b0;');
1079    end
        if flip_imag_top || flip_imag_bottom
1081        fprintf(fileID, '\n\t\tnext_invert_imag_sign <= #1 1''b0;');
        end
1083    fprintf(fileID, '\n\tend\n\t');
        fprintf(fileID, 'else if (next_coef_step_counter!={ctr_width{1''b0}})
        //enable for this counter\n\t\t');
1085    fprintf(fileID, 'begin\n\t\t\t');
        fprintf(fileID, 'if (next_coef_direction==1''b0) //counting up\n\t\t\t\
        t');
1087    fprintf(fileID, 'if (next_coef_cnt!=coef_limit)\n\t\t\t\t\t');
        fprintf(fileID, 'next_coef_cnt <= #1 next_coef_cnt+1''b1;\n\t\t\t\t');
1089    fprintf(fileID, 'else\n\t\t\t\t');
```

```
    fprintf(fileID, 'begin\n\t\t\t\t\t');
1091    fprintf(fileID, 'next_coef_direction <= #1 1''b1;');
    if ~hold_top
1093        fprintf(fileID, '\n\t\t\t\t\tnext_coef_cnt <= #1 next_coef_cnt-1''
    b1;');
    end
1095    if flip_real_top
        fprintf(fileID, '\n\t\t\t\t\tnext_invert_real_sign <= #1 ~
    next_invert_real_sign;');
1097    end
    if flip_imag_top
1099        fprintf(fileID, '\n\t\t\t\t\tnext_invert_imag_sign <= #1 ~
    next_invert_imag_sign;');
    end
1101    fprintf(fileID, '\n\t\t\t\tend\n\t\t\t');
    fprintf(fileID, 'else //counting down\n\t\t\t\t');
1103    fprintf(fileID, 'if (next_coef_cnt!={coef_width{1''b0}})\n\t\t\t\t\t');
    fprintf(fileID, 'next_coef_cnt <= #1 next_coef_cnt-1''b1;\n\t\t\t\t');
1105    fprintf(fileID, 'else\n\t\t\t\t');
    fprintf(fileID, 'begin\n\t\t\t\t\t');
1107    fprintf(fileID, 'next_coef_direction <= #1 1''b0;');
    if ~hold_bottom
1109        fprintf(fileID, '\n\t\t\t\t\tnext_coef_cnt <= #1 next_coef_cnt+1''
    b1;');
    end
1111    if flip_real_bottom
        fprintf(fileID, '\n\t\t\t\t\tnext_invert_real_sign <= #1 ~
    next_invert_real_sign;');
1113    end
    if flip_imag_bottom
1115        fprintf(fileID, '\n\t\t\t\t\tnext_invert_imag_sign <= #1 ~
    next_invert_imag_sign;');
    end
1117    fprintf(fileID, '\n\t\t\t\tend\n\t\t');
    fprintf(fileID, 'end\n');
1119    fprintf(fileID, 'end\n\n');

1121    fprintf(fileID, '/*************************/\n');
    fprintf(fileID, '/*** coefficient counter ***/\n');
1123    fprintf(fileID, '/*************************/\n');
    fprintf(fileID, 'always@(posedge CLK, negedge nRESET_int)\n');
1125    fprintf(fileID, 'begin\n\tif(!nRESET_int)\n\t');
    fprintf(fileID, 'begin\n\t\t');
1127    fprintf(fileID, 'coef_cnt <= #1 {coef_width{1''b0}};\n\t\t');
```

```
    fprintf(fileID, 'coef_direction <= #1 1''b0; //default direction is up'
    );
    if flip_real_top || flip_real_bottom
        fprintf(fileID, '\n\t\tinvert_real_sign <= #1 1''b0;');
    end
    if flip_imag_top || flip_imag_bottom
        fprintf(fileID, '\n\t\tinvert_imag_sign <= #1 1''b0;');
    end
    fprintf(fileID, '\n\tend\n\t');
    if twoCycle == 1
        fprintf(fileID, 'else if (row_strobe & ~sample_strobe) //load
    counter\n\t');
    else
        fprintf(fileID, 'else if (row_strobe) //load counter\n\t\t');
    end
    fprintf(fileID, 'begin\n\t\t');
    fprintf(fileID, 'coef_cnt <= #1 next_coef_cnt;\n\t\t');
    fprintf(fileID, 'coef_direction  <= #1 next_coef_direction;');
    if flip_real_top || flip_real_bottom
        fprintf(fileID, '\n\t\tinvert_real_sign <= #1 next_invert_real_sign
    ;');
    end
    if flip_imag_top || flip_imag_bottom
        fprintf(fileID, '\n\t\tinvert_imag_sign <= #1 next_invert_imag_sign
    ;');
    end
    fprintf(fileID, '\n\tend\n\t');
    fprintf(fileID, 'else if (coefCntrEnable) //enable for this counter\n\t
    \t');
    fprintf(fileID, 'begin\n\t\t\t');
    fprintf(fileID, 'if (coef_direction==1''b0) //counting up\n\t\t\t\t');
    fprintf(fileID, 'if (coef_cnt!=coef_limit)\n\t\t\t\t\t');
    fprintf(fileID, 'coef_cnt <= #1 coef_cnt+1''b1;\n\t\t\t\t');
    fprintf(fileID, 'else\n\t\t\t\t');
    fprintf(fileID, 'begin\n\t\t\t\t\t');
    fprintf(fileID, 'coef_direction <= #1 1''b1;');
    if ~hold_top
        fprintf(fileID, '\n\t\t\t\t\tcoef_cnt <= #1 coef_cnt-1''b1;');
    end
    if flip_real_top
        fprintf(fileID, '\n\t\t\t\t\tinvert_real_sign <= #1 ~
    invert_real_sign;');
    end
    if flip_imag_top
```

```
        fprintf(fileID, '\n\t\t\t\t\tinvert_imag_sign <= #1 ~
    invert_imag_sign;');
    end
    fprintf(fileID, '\n\t\t\t\tend\n\t\t\t');
    fprintf(fileID, 'else //counting down\n\t\t\t\t');
    fprintf(fileID, 'if (coef_cnt!={coef_width{1''b0}})\n\t\t\t\t\t');
    fprintf(fileID, 'coef_cnt <= #1 coef_cnt-1''b1;\n\t\t\t\t');
    fprintf(fileID, 'else\n\t\t\t\t');
    fprintf(fileID, 'begin\n\t\t\t\t\t');
    fprintf(fileID, 'coef_direction <= #1 1''b0;');
    if ~hold_bottom
        fprintf(fileID, '\n\t\t\t\t\tcoef_cnt <= #1 coef_cnt+1''b1;');
    end
    if flip_real_bottom
        fprintf(fileID, '\n\t\t\t\t\tinvert_real_sign <= #1 ~
    invert_real_sign;');
    end
    if flip_imag_bottom
        fprintf(fileID, '\n\t\t\t\t\tinvert_imag_sign <= #1 ~
    invert_imag_sign;');
    end
    fprintf(fileID, '\n\t\t\t\tend\n\t\t');
    fprintf(fileID, 'end\n');
    fprintf(fileID, 'end\n\n');
end




function writeTrivialPatternCounters(fileID, twoCycle, realSign, imagSign,
    flipRealOnSample ,flipRealOnRow, flipImagOnSample,flipImagOnRow)

    realSignChanging = or(flipRealOnSample, flipRealOnRow);
    imagSignChanging = or(flipImagOnSample, flipImagOnRow);
    fprintf(fileID, '/******************/\n');
    fprintf(fileID, '/*** sign logic ***/\n');
    fprintf(fileID, '/******************/\n');
    fprintf(fileID, 'always@(nReset) //we have to trigger this process
    somehow\nbegin\n\t');
    fprintf(fileID, 'coef_cnt = 1''b0;');
    if ~realSignChanging
        fprintf(fileID, '\n\tcoefSign_real = 1''b%d;',realSign);
    end
    if ~imagSignChanging
```

```
1205            fprintf(fileID, '\n\tcoefSign_imag = 1''b%d;',imagSign);
        end
1207        fprintf(fileID, '\nend\n\n');
        if realSignChanging || imagSignChanging   %%static signs -> dont
        generate this
1209        fprintf(fileID, 'always@(posedge CLK, negedge nRESET_int)\n');
        fprintf(fileID, 'begin\n\tif(!nRESET_int)\n\t');
1211        fprintf(fileID, 'begin');
        if realSignChanging
1213            fprintf(fileID, '\n\t\tcoefSign_real <= 1''b%d;',realSign);
            fprintf(fileID, '\n\t\tnext_coefSign_real <= 1''b%d;',~realSign);
1215        end
        if imagSignChanging
1217            fprintf(fileID, '\n\tcoefSign_imag <= 1''b%d;',imagSign);
            fprintf(fileID, '\n\tnext_coefSign_imag <= 1''b%d;',~imagSign);
1219        end
        fprintf(fileID, '\n\tend\n\t');
1221        if flipRealOnRow || flipImagOnRow
            if twoCycle == 1
1223                fprintf(fileID, 'else if (row_strobe & ~sample_strobe) //load
        reg\n\t');
            else
1225                fprintf(fileID, 'else if (row_strobe) //load counter\n\t\t');
            end
1227            fprintf(fileID, 'begin');
            if flipRealOnRow
1229                fprintf(fileID, '\n\t\tcoefSign_real <= #1 next_coefSign_real;'
        );
                fprintf(fileID, '\n\t\tnext_coefSign_real <= #1 ~
        next_coefSign_real;');
1231            end
            if flipImagOnRow
1233                fprintf(fileID, '\n\t\tcoefSign_imag <= #1 next_coefSign_imag;'
        );
                fprintf(fileID, '\n\t\tnext_coefSign_imag <= #1 ~
        next_coefSign_imag;');
1235            end
            fprintf(fileID, '\n\tend');
1237        end
        if flipRealOnSample || flipImagOnSample
1239            fprintf(fileID, '\n\telse if (~sample_strobe) //enable\n\t');
            fprintf(fileID, 'begin');
1241            if flipRealOnSample
                fprintf(fileID, '\n\t\t\tcoefSign_real <= #1 ~coefSign_real;');
```

```
1243            end
                if flipImagOnSample
1245                fprintf(fileID, '\n\t\t\tcoefSign_imag <= #1 ~coefSign_imag;');
                end
1247            fprintf(fileID, '\n\tend');
        end
1249    fprintf(fileID, '\nend\n\n');
        end
1251
    end
1253
    function writeColumnPatternCounters(fileID, twoCycle, realCoef, imagCoef)
1255
        fprintf(fileID, '/************************************/\n');
1257    fprintf(fileID, '/*** coefficient & sign registers ***/\n');
        fprintf(fileID, '/************************************/\n');
1259    fprintf(fileID, 'always@(posedge CLK, negedge nRESET_int)\n');
        fprintf(fileID, 'begin\n\t');
1261    fprintf(fileID, 'if(!nRESET_int)\n\t');
        fprintf(fileID, 'begin\n\t\t');
1263    fprintf(fileID, 'coef_cnt <= #1 {coef_width{1''b0}};\n\t\t');
        if sign(realCoef(1)) >=0 %set the right sign on reset
1265        fprintf(fileID, 'coefSign_real  <= #1 1''b0; \n\t\t');
        else
1267        fprintf(fileID, 'coefSign_real  <= #1 1''b1; \n\t\t');
        end
1269    if sign(imagCoef(1)) >=0 %set the right sign on reset
            fprintf(fileID, 'coefSign_imag  <= #1 1''b0;\n\t');
1271    else
            fprintf(fileID, 'coefSign_imag  <= #1 1''b1;\n\t');
1273    end
        fprintf(fileID, 'end\n\t');
1275    if twoCycle
            fprintf(fileID, 'else if (coefCntrEnable) // reg enable\n\t');
1277    else
            fprintf(fileID, 'else\n\t');
1279    end
        fprintf(fileID, 'begin\n\t\t');
1281    fprintf(fileID, 'coef_cnt     <= #1 next_coef_cnt;\n\t\t');
        fprintf(fileID, 'coefSign_real  <= #1 next_real_sign;\n\t\t');
1283    fprintf(fileID, 'coefSign_imag  <= #1 next_imag_sign;\n\t');
        fprintf(fileID, 'end\n');
1285    fprintf(fileID, 'end\n');
        fprintf(fileID, '/****************************************/\n');
```

```
1287      fprintf(fileID, '/*** next sample coefficient calculation ***/\n');
          fprintf(fileID, '/****************************************/\n');
1289      fprintf(fileID, 'always@(posedge CLK, negedge nRESET_int)\n');
          fprintf(fileID, 'begin\n\t');
1291      fprintf(fileID, 'if(!nRESET_int)\n\t');
          fprintf(fileID, 'begin\n\t\t');
1293      fprintf(fileID, 'full_coef_cnt <= #1 {ctr_width{1''b0}};\n\t');
          fprintf(fileID, 'end\n\t');
1295      fprintf(fileID, 'else if (coefCntrEnable) // reg enable \n\t');
          fprintf(fileID, 'begin\n\t\t');
1297      fprintf(fileID, 'full_coef_cnt <= #1 full_next_coef_cnt;\n\t');
          fprintf(fileID, 'end\n');
1299      fprintf(fileID, 'end\n\n');

1301      fprintf(fileID, '//Modulo adder\n');
          fprintf(fileID, 'always@(*) \n');
1303      fprintf(fileID, 'begin\n\t');
          fprintf(fileID, 'modulo_sum = #1  full_coef_cnt +
          coef_steps_per_sample;\n\t'); %%%%%
1305      fprintf(fileID, 'modulo_sum_minus_mod=modulo_sum-modulo;\n\t');
          fprintf(fileID, 'if (modulo_sum >= modulo)\n\t\t');
1307      fprintf(fileID, 'modulo_add_out = #1 modulo_sum_minus_mod[ctr_width
          -1:0];\n\t');
          fprintf(fileID, 'else\n\t\t');
1309      fprintf(fileID, 'modulo_add_out = #1 modulo_sum[ctr_width-1:0];\n');
          fprintf(fileID, 'end\n\n');
1311
          fprintf(fileID, '//if we have a row transistion take row count + 1 as
          input to the LUT\n');
1313      fprintf(fileID, 'assign full_next_coef_cnt = (row_strobe & ~
          sample_strobe) ? rowCnt+1''b1 : modulo_add_out ;\n\n');

1315  end

1317  function [limitPoints, signType]=analyzePattern(nPoints, realCoef, imagCoef
          )
      [uniqueCoefReal,UnIndexRealA,UnIndexRealC]=unique(abs(realCoef),'stable');
1319  [uniqueCoefImag,UnIndexImagA,UnIndexImagC]=unique(abs(imagCoef),'stable');
      limitPoints=zeros(1,nPoints);
1321  InvertReal=[1 1 1 1];
      RealSignMatch(1,:)=[1 1 1 1]*sign(realCoef(1));
1323  InvertImag=[1 1 1 1];
      ImagSignMatch(1,:)=[1 1 1 1]*sign(imagCoef(1));
1325  for idxX = 2:nPoints
```

```
    RealSignMatch(idxX,:)=[1 1 1 1]*sign(realCoef(UnIndexRealC(idxX))).*
    InvertReal;
1327    ImagSignMatch(idxX,:)=[1 1 1 1]*sign(imagCoef(UnIndexRealC(idxX))).*
    InvertImag;
    if idxX~=nPoints
1329        if UnIndexRealC(idxX) == UnIndexRealC(idxX+1) && UnIndexRealC(idxX)
    ==max(UnIndexRealC) %ok we found a upper hold point
            InvertReal=InvertReal.*[1 -1 1 -1];
1331            InvertImag=InvertImag.*[1 -1 1 -1];
            disp(['genDftModule: upperHold detect at idxX=' num2str(idxX)])
    ;
1333            limitPoints(idxX)=1;
        elseif  UnIndexRealC(idxX)==1 && UnIndexRealC(idxX+1)~=1 &&
    UnIndexRealC(idxX-1)~=1 %count up point
1335            InvertReal=InvertReal.*[1 1 -1 -1];
            InvertImag=InvertImag.*[1 1 -1 -1];
1337            disp(['genDftModule: count Up detect at idxX=' num2str(idxX)]);
            limitPoints(idxX)=2;
1339        elseif  UnIndexRealC(idxX)==max(UnIndexRealC) &&  UnIndexRealC(idxX
    +1)~=max(UnIndexRealC) &&  UnIndexRealC(idxX-1)~=max(UnIndexRealC) %
    count down point
            InvertReal=InvertReal.*[1 -1 1 -1];
1341            InvertImag=InvertImag.*[1 -1 1 -1];
            disp(['genDftModule: count down detect at idxX=' num2str(idxX)
    ]);
1343            limitPoints(idxX)=3;
        elseif UnIndexRealC(idxX+1) == 1 && UnIndexRealC(idxX)==1 %lower
    holdpoint
1345            InvertReal=InvertReal.*[1 1 -1 -1];
            InvertImag=InvertImag.*[1 1 -1 -1];
1347            disp(['genDftModule: lower Hold detect at idxX=' num2str(idxX)
    ]);
            limitPoints(idxX)=4;
1349        end %end of repeat_point classification if chain
    end
1351 end % end of for loop
   %signType: 1= dont invert, 2= invert on Top, 3=invert on
1353 %bottom, 4=invert on both
   signType=zeros(1,2);
1355 realMatched=false;
   imagMatched=false;
1357 for idxX=1:4
    if isequal(sign(realCoef(1,1:nPoints)),RealSignMatch(:,idxX)') && ~
    realMatched
```

```
1359            signType(1)=idxX;
                realMatched=true;
1361        end
            if isequal(sign(imagCoef(1,1:nPoints)),ImagSignMatch(:,idxX)') && ~
            imagMatched
1363            signType(2)=idxX;
                imagMatched=true;
1365        end
    end
1367
    end
```

Listing A.2: genDftModule.m

A.1.3 Quellcode: genDftModuleToplevel.m

```
    function result=genDftModuleToplevel(nPoints, outputDftTerms, inputBitwidth
        , accumulatorBitwidth, twoCycle, generateFile)
2
        if generateFile(5)
4       disp('genDftModuleToplevel: starting Verilog Generator');
        if nPoints<3
6           disp('genDftModuleToplevel: Error: nPoints too small');
            return;
8       end
        accuoutputBitwidth=accumulatorBitwidth;
10      disp(['genDftModuleToplevel: Transform size: ', num2str(nPoints),' X ',
        num2str(nPoints) ' Points']);
        disp(['genDftModuleToplevel: Accumulator width: ', num2str(
        accumulatorBitwidth)]);
12      numberOfOutputTerms=size(outputDftTerms,1);
        if numberOfOutputTerms==1
14          outputFileName=['coefficient_',num2str(nPoints), 'x', num2str(
        nPoints),'_', num2str(outputDftTerms(1)),'_',num2str(outputDftTerms(2))
        ,'_toplevel.v'];
        else
16          outputFileName=['coefficients_',num2str(nPoints), 'x', num2str(
        nPoints),'_toplevel.v'];
        end
18      disp(['genDftModuleToplevel: Output file: ',outputFileName]);
        if numberOfOutputTerms>1
20          selectWidth=nextpow2(numberOfOutputTerms); %
        else
```

```
22          selectWidth=1;
        end
24      fileID = fopen(outputFileName,'w');
        fprintf(fileID, '/*****************************************/\n');
26      fprintf(fileID, '/***        verilog toplevel Module        ***/\n');
        fprintf(fileID, '/***         Transform size: %dx%d         ***/\n',
    nPoints,nPoints);
28      fprintf(fileID, '/***       Built: %s      ***/\n', datetime('now'));
        fprintf(fileID, '/***   generated by brute2Dft framework   ***/\n');
30      fprintf(fileID, '/***     written by Martin Willimczik     ***/\n');
        fprintf(fileID, '/*****************************************/\n');
32      if numberOfOutputTerms==1
        if twoCycle == 1
34          fprintf(fileID, 'module coefficient_%dx%d_%d_%d_2cy_toplevel(\n\t',
    nPoints,nPoints,outputDftTerms(1),outputDftTerms(2));
        else
36          fprintf(fileID, 'module coefficient_%dx%d_%d_%d_toplevel(\n\t',
    nPoints,nPoints,outputDftTerms(1),outputDftTerms(2));
        end
38      else
            if twoCycle == 1
40              fprintf(fileID, 'module coefficient_%dx%d_2cy_toplevel(\n\t',
    nPoints,nPoints);
            else
42              fprintf(fileID, 'module coefficient_%dx%d_toplevel(\n\t',
    nPoints,nPoints);
            end
44      end
        fprintf(fileID, 'input CLK,\n\t');
46      fprintf(fileID, 'input nReset,\n\t');
        fprintf(fileID, 'input start,\n\t');
48      fprintf(fileID, 'input [%d:0] mux_select,\n\t',selectWidth-1);
        fprintf(fileID, 'input [%d:0] real_IN,\n\t',inputBitwidth-1);
50      fprintf(fileID, 'input [%d:0] imag_IN,\n\t',inputBitwidth-1);
        fprintf(fileID, 'output [%d:0] real_OUT,\n\t',accuoutputBitwidth-1);
52      fprintf(fileID, 'output [%d:0] imag_OUT,\n\t',accuoutputBitwidth-1);
        fprintf(fileID, 'output done\n);\n');
54
        fprintf(fileID, 'parameter ctr_width=%d;\n', nextpow2(nPoints));
56      fprintf(fileID, 'parameter column_limit=%d; //when this is reached
    calculation is done\n', nPoints-1);
        fprintf(fileID, 'parameter Opt_width=%d; //Output data width\n',
    accuoutputBitwidth);
```

```
58      fprintf(fileID, 'parameter Inp_width=%d; //Input data width\n',
        inputBitwidth);

60      writeStandardWiresAndRegs(fileID, twoCycle, numberOfOutputTerms);
        if numberOfOutputTerms>1
62          for count = 1:numberOfOutputTerms
                fprintf(fileID, 'wire [Opt_width-1:0] moduleOutR_%d_%d,
        moduleOutI_%d_%d;\n',outputDftTerms(count,1), outputDftTerms(count,2),
        outputDftTerms(count,1), outputDftTerms(count,2));
64          end
            fprintf(fileID, '\n');
66      else
            fprintf(fileID, 'wire [Opt_width-1:0] moduleOutR_%d_%d, moduleOutI_
        %d_%d;\n\n', outputDftTerms(1,1), outputDftTerms(1,2),outputDftTerms
        (1,1), outputDftTerms(1,2));
68      end

70      for term=1:numberOfOutputTerms
            instanciateDftModule(fileID, nPoints, outputDftTerms(term,:),
        twoCycle)
72      end
        fprintf(fileID, '\n');
74      writeResetLogic(fileID);
        if twoCycle
76          printSamplestrobe(fileID);
        end
78      if numberOfOutputTerms>1
            generateOutputMux(fileID, outputDftTerms, selectWidth);
80      else
            fprintf(fileID, '//output assignment\n');
82          fprintf(fileID, 'assign real_OUT = moduleOutR_%d_%d;\n',
        outputDftTerms(1,1), outputDftTerms(1,2));
            fprintf(fileID, 'assign imag_OUT = moduleOutI_%d_%d;\n',
        outputDftTerms(1,1), outputDftTerms(1,2));
84      end
        writeGenericCounters(fileID, twoCycle);
86      writeGenericStateMachine(fileID, twoCycle);
        writeDoneLogic(fileID,twoCycle);
88      fprintf(fileID, 'endmodule\n');
        fclose(fileID);
90      end

92  end
```

```
94
96 function writeStandardWiresAndRegs(fileID, twoCycle, numberOfOutputTerms)
       fprintf(fileID, '/********************/\n');
98     fprintf(fileID, '/*** wires & regs ***/\n');
       fprintf(fileID, '/********************/\n');
100    fprintf(fileID, 'reg [ctr_width-1:0] rowCnt, columnCnt; //counter
       registers\n');
       fprintf(fileID, 'reg row_strobe; //signals row transition\n');
102    fprintf(fileID, 'reg en_coef_cnt;  //enable coefficient counting\n');
       fprintf(fileID, 'reg AccEnable; //accumulator enable\nreg MultiplierEn
       ;\n');
104    fprintf(fileID, 'reg start_reg; //de-glitches start input for reset\n')
       ;
       fprintf(fileID, 'reg done_reg; //saves that we have finished a piece of
        work\n');
106    if numberOfOutputTerms>1
           fprintf(fileID, 'reg [Opt_width-1:0] muxOutReal, muxOutImag; //de-
       glitches start input for reset\n');
108    end
       if twoCycle == 1
110        fprintf(fileID, 'reg [6:0] state, nextstate; // state registers\n')
       ;
           fprintf(fileID, 'reg sample_strobe; //signifies next sample\n');
112    else
            fprintf(fileID, 'reg [5:0] state, nextstate; // state registers\n'
       );
114    end
       fprintf(fileID, 'wire nRESET_int;\n');
116 end

118 function instanciateDftModule(fileID, nPoints, outputDftTerm, twoCycle)
       if twoCycle == 1
120        fprintf(fileID, 'coefficient_%dx%d_%d_%d_2cy ',nPoints,nPoints,
       outputDftTerm(1),outputDftTerm(2));
       else
122        fprintf(fileID, 'coefficient_%dx%d_%d_%d_1cy ',nPoints,nPoints,
       outputDftTerm(1),outputDftTerm(2));
       end
124    fprintf(fileID, 'dftModule_%d_%d\n',outputDftTerm(1),outputDftTerm(2));
       fprintf(fileID, '(\n\tCLK,\n\t');
126    fprintf(fileID, 'nReset,\n\t');
       fprintf(fileID, 'start,\n\t');
128    fprintf(fileID, 'MultiplierEn,\n\t');
```

```
    fprintf(fileID, 'AccEnable,\n\t');
    fprintf(fileID, 'en_coef_cnt,\n\t');
    fprintf(fileID, 'columnCnt,\n\t');
    fprintf(fileID, 'row_strobe,\n\t');
    fprintf(fileID, 'rowCnt,\n\t');
    fprintf(fileID, 'real_IN,\n\t');
    fprintf(fileID, 'imag_IN,\n\t');
    fprintf(fileID, 'moduleOutR_%d_%d,\n\t',outputDftTerm(1),outputDftTerm
    (2));
    fprintf(fileID, 'moduleOutI_%d_%d\n);\n',outputDftTerm(1),outputDftTerm
    (2));
end

function writeGenericStateMachine(fileID, twoCycle)
if twoCycle
    stateWidth=7;
else
    stateWidth=6;
end

%write header and parameters
fprintf(fileID, '/**********************/\n');
fprintf(fileID, '/*** State Machine ***/\n');
fprintf(fileID, '/**********************/\n');
fprintf(fileID, 'parameter [%d:0] S_RDY    = 0,\n\t\t\t\t\t',stateWidth-1);
fprintf(fileID, 'S_DLY1     = 1,\n\t\t\t\tS_DLY2     = 2,\n\t\t\t\t\t');
fprintf(fileID, 'S_CALCULATE    = 3,\n\t\t\t\t\t');
fprintf(fileID, 'S_DLYEND1 = 4,\n\t\t\t\t\t');
onehotbit=5;
if twoCycle == 1
    fprintf(fileID, 'S_DLYEND2 = %d,\n\t\t\t\t\t',onehotbit);
    onehotbit=onehotbit+1;
    fprintf(fileID, 'S_DLYEND3 = %d;\n//state register\n',onehotbit);
else
    fprintf(fileID, 'S_DLYEND2 = %d;',onehotbit);
    fprintf(fileID, '\n//state register\n');
end
%register
fprintf(fileID, 'always@(posedge CLK, negedge nReset)\nbegin\n\t');
fprintf(fileID, 'if(!nReset)\n\tbegin\n\t\t');
fprintf(fileID, 'state <= %d''d0;\n\t\t',stateWidth);
fprintf(fileID, 'state[S_RDY] <= 1''b1;\n\tend\n\telse\n\t\t');
fprintf(fileID, 'state <= #1 nextstate;\nend\n');
%case block
```

```
fprintf(fileID, 'always @(*)\nbegin\n\t');
fprintf(fileID, 'nextstate = %d''d0; // default\n\t',stateWidth);
fprintf(fileID, 'MultiplierEn=1''b0;\n\tAccEnable=1''b0;\n\t');
fprintf(fileID, 'en_coef_cnt=1''b0;\n\t');
fprintf(fileID, 'case(1''b1)// synopsys full_case parallel_case\n\t\t');
%state RDY
fprintf(fileID, 'state[S_RDY]:\n\t\tbegin\n\t\t\t');
fprintf(fileID, 'if (start)\n\t\t\t\tnextstate[S_DLY1]=1''b1;\n\t\t\t');
fprintf(fileID, 'else\n\t\t\t\tnextstate[S_RDY]=1''b1;\n\t\tend\n\t\t');
%state DLY1
fprintf(fileID, 'state[S_DLY1]: //delay0\n\t\tbegin\n\t\t\t');
fprintf(fileID, 'nextstate[S_DLY2] = 1''b1;\n\t\t\t');
fprintf(fileID, 'MultiplierEn=1''b1;\n\t\ten_coef_cnt=1''b1;\n\t\tend\n\t
    \t');
fprintf(fileID, 'state[S_DLY2]: //delay1\n\t\tbegin\n\t\t\t');
fprintf(fileID, 'nextstate[S_CALCULATE] = 1''b1;\n\t\t\t');
fprintf(fileID, 'MultiplierEn=1''b1;\n\t\ten_coef_cnt=1''b1;\n\t\tend\n\t
    \t');

%STATE
fprintf(fileID, 'state[S_CALCULATE]:\n\t\tbegin\n\t\t\t');

fprintf(fileID, 'if (rowCnt == column_limit && row_strobe==1''b1)\n\t\t\t\
    tnextstate[S_DLYEND1]=1''b1;\n\t\t\t' );
fprintf(fileID, 'else\n\t\t\t\tnextstate[S_CALCULATE]=1''b1;\n\t\t\t');
fprintf(fileID, 'MultiplierEn=1''b1;\n\t\ten_coef_cnt=1''b1;\n\t\t\t');
fprintf(fileID, 'AccEnable=1''b1;\n\t\t');
fprintf(fileID, 'end\n\t\t');

fprintf(fileID, 'state[S_DLYEND1]://wait for pipeline to drain\n\t\tbegin\n
    \t\t\t');
fprintf(fileID, 'nextstate[S_DLYEND2]=1''b1;\n\t\t\t');
fprintf(fileID, 'MultiplierEn=1''b1;\n\t\t\t');
fprintf(fileID, 'AccEnable=1''b1;\n\t\tend\n\t\t');
if twoCycle == 1
    fprintf(fileID, 'state[S_DLYEND2]://wait for pipeline to drain\n\t\
    tbegin\n\t\t\t');
    fprintf(fileID, 'nextstate[S_DLYEND3]=1''b1;\n\t\t\t');
    fprintf(fileID, 'MultiplierEn=1''b1;\n\t\t\t');
    fprintf(fileID, 'AccEnable=1''b1;\n\t\tend\n\t\t');

    fprintf(fileID, 'state[S_DLYEND3]://wait for pipeline to drain\n\t\
    tbegin\n\t\t\t');
    fprintf(fileID, 'nextstate[S_RDY]=1''b1;\n\t\t\t');
```

```
    fprintf(fileID, 'AccEnable=1''b1;\n\t\tend\n\t');
else
    fprintf(fileID, 'state[S_DLYEND2]://wait for pipeline to drain\n\t\
    tbegin\n\t\t\t');
    fprintf(fileID, 'nextstate[S_RDY]=1''b1;\n\t\t\t');
    fprintf(fileID, 'AccEnable=1''b1;\n\t\tend\n\t');
end
fprintf(fileID, 'endcase\nend\n');
end

function writeGenericCounters(fileID, twoCycle)
    fprintf(fileID, '/**********************/\n');
    fprintf(fileID, '/*** column counter ***/\n');
    fprintf(fileID, '/**********************/\n');
    fprintf(fileID, 'always@(posedge CLK, negedge nRESET_int)\n');
    fprintf(fileID, 'begin\n\tif(!nRESET_int)\n\t\t');
    if twoCycle == 1
        fprintf(fileID, 'columnCnt <= #1 {ctr_width{1''b0}};\n\t');
        fprintf(fileID, 'else if (sample_strobe)\n\t\tcolumnCnt <=
    columnCnt;\n\t');
    else
        fprintf(fileID, 'columnCnt <= #1 {{ctr_width-1{1''b0}},1''b1};\n\t'
    );
    end
    fprintf(fileID, 'else if (columnCnt==column_limit) //overflow\n\t\t');
    fprintf(fileID, 'columnCnt <= #1 {ctr_width{1''b0}};\n\t');
    fprintf(fileID, 'else\n\t\t');
    fprintf(fileID, 'columnCnt <= #1 columnCnt+1''b1;\n');
    fprintf(fileID, 'end\n');
    fprintf(fileID, '//row strobe generator\n');
    fprintf(fileID, 'always@(*)\nbegin\n\t');
    fprintf(fileID, 'if(columnCnt==column_limit)\n\t\trow_strobe=1''b1;\n\t
    ');
    fprintf(fileID, 'else\n\t\trow_strobe=1''b0;\nend\n');

    fprintf(fileID, '/********************/\n');
    fprintf(fileID, '/*** row counter ***/\n');
    fprintf(fileID, '/********************/\n');
    fprintf(fileID, 'always@(posedge CLK, negedge nRESET_int)\n');
    fprintf(fileID, 'begin\n\tif(!nRESET_int)\n\t\t');
    fprintf(fileID, 'rowCnt <= #1 {ctr_width{1''b0}};\n\t');
    if twoCycle == 1
        fprintf(fileID, 'else if ((row_strobe & ~sample_strobe)==1''b0)\n\t
    \trowCnt <= rowCnt;\n\t');
```

```
248     else
            fprintf(fileID, 'else if ((row_strobe)==1''b0)\n\t\trowCnt <=
        rowCnt;\n\t');
250     end
        fprintf(fileID, 'else\n\t\t');
252     fprintf(fileID, 'rowCnt <= #1 rowCnt+1''b1;\n');
        fprintf(fileID, 'end\n');
254 end


256
% run the external fixed multiplier generator program
258 % mostly a clone of the function in genVerilog
function generateOutputMux(fileID, DftTerms, selectWidth)
260     fprintf(fileID, '/******************/\n');
        fprintf(fileID, '/*** output mux ***/\n');
262     fprintf(fileID, '/******************/\n');
        fprintf(fileID, 'always@(*)\n');
264     fprintf(fileID, 'begin\n\t');
        fprintf(fileID, 'case (mux_select)\n\t\t');
266     for count = 0:size(DftTerms,1)-1
            fprintf(fileID, '%d''d%d:\n\t\t',selectWidth,count);
268         fprintf(fileID, 'begin\n\t\t\t');
            fprintf(fileID, 'muxOutReal = moduleOutR_%d_%d;\n\t\t\t\t',DftTerms(
        count+1,1), DftTerms(count+1,2));
270         fprintf(fileID, 'muxOutImag = moduleOutI_%d_%d;\n\t\t\t',DftTerms(
        count+1,1), DftTerms(count+1,2));
            fprintf(fileID, 'end\n\t\t');
272     end
        fprintf(fileID, 'default:\n\t\t');
274     fprintf(fileID, 'begin\n\t\t\t');
        fprintf(fileID, 'muxOutReal = moduleOutR_%d_%d;\n\t\t\t\t',DftTerms(1,1),
         DftTerms(1,2));
276     fprintf(fileID, 'muxOutImag = moduleOutI_%d_%d;\n\t\t\t',DftTerms(1,1),
        DftTerms(1,2));
        fprintf(fileID, 'end\n\t');
278     fprintf(fileID, 'endcase\n');
        fprintf(fileID, 'end\n');
280     fprintf(fileID, 'assign real_OUT = muxOutReal;\n');
        fprintf(fileID, 'assign imag_OUT = muxOutImag;\n');
282     fprintf(fileID, '\n');
end
284

%Prints sample strobe logic
286 function printSamplestrobe(fileID)
```

```
    fprintf(fileID, '/*********************/\n');
288     fprintf(fileID, '/*** sample strobe ***/\n');
    fprintf(fileID, '/*********************/\n');
290     fprintf(fileID, 'always@(posedge CLK, negedge nRESET_int)\nbegin\n\t');
    fprintf(fileID, 'if(!nRESET_int) //synchronizes sample_strobe\n\t\t');
292     fprintf(fileID, 'sample_strobe <= #1 1''b0;\n\t');
    fprintf(fileID, 'else if (MultiplierEn)\n\t\t');
294     fprintf(fileID, 'sample_strobe <= #1 ~sample_strobe;\n\t');
    fprintf(fileID, 'else\n\t\tsample_strobe <= #1 sample_strobe;\nend\n');
296 end

298 function writeResetLogic(fileID)
    fprintf(fileID, '/*******************/\n');
300     fprintf(fileID, '/*** reset logic ***/\n');
    fprintf(fileID, '/*******************/\n');
302     fprintf(fileID, 'always@(posedge CLK)\nbegin\n\tstart_reg <= #1 start;\
    nend\n');
    fprintf(fileID, 'assign #1 nRESET_int = nReset & ~start_reg; //reset on
     external reset or start\n\n');
304 end

306 function writeDoneLogic(fileID, twoCycle)
    fprintf(fileID, '/****************/\n');
308     fprintf(fileID, '/*** done reg ***/\n');
    fprintf(fileID, '/****************/\n');
310     fprintf(fileID, 'always@(posedge CLK, negedge nRESET_int)\n');
    fprintf(fileID, 'begin\n\t');
312     fprintf(fileID, 'if(~nRESET_int)\n\t\t');
    fprintf(fileID, 'done_reg <= #1 1''b0;\n\t');
314     if twoCycle == 1
        fprintf(fileID, 'else if (state[S_DLYEND3] == 1''b1)\n\t\t');
316     else
        fprintf(fileID, 'else if (state[S_DLYEND2] == 1''b1)\n\t\t');
318     end
    fprintf(fileID, 'done_reg <= #1 1''b1;\n');
320     fprintf(fileID, 'end\n');
    fprintf(fileID, 'assign done = done_reg;\n');
322


324 end
```

Listing A.3: genDftModuleToplevel.m

A.1.4 Quellcode: genDftModuleTestbench.m

```
function genDftModuleTestbench (nPoints, Transformcoefficients,
    outputDftTerms, inputBitwidth, coefFracBitwidth, accumulatorBitwidth,
    twoCycle, generateFile)
disp('genDftModuleTestbench: 2D-DFT Testbench generator');

numberOfOutputTerms=size(outputDftTerms,1);
if numberOfOutputTerms>1
    selectWidth=nextpow2(numberOfOutputTerms); %
else
    selectWidth=1;
end
accuoutputBitwidth=accumulatorBitwidth;
if numberOfOutputTerms==1
    if twoCycle
        outputFileName=['tb_coefficient_',num2str(nPoints), 'x', num2str(
    nPoints),'_', num2str(outputDftTerms(1,1)),'_',num2str(outputDftTerms
    (1,2)),'_2cy_toplevel.v'];
    else
        outputFileName=['tb_coefficient_',num2str(nPoints), 'x', num2str(
    nPoints),'_', num2str(outputDftTerms(1,1)),'_',num2str(outputDftTerms
    (1,2)),'_toplevel.v'];
    end
    disp(['genDftModuleTestbench: Output Term: ', num2str(outputDftTerms
    (1,:))]);
    disp(['genDftModuleTestbench: only one output term specified, enabling
    accumulator intermediate debug output']);
else
    outputFileName=['tb_coefficients_',num2str(nPoints), 'x', num2str(
    nPoints),'_2cy_toplevel.v'];
end

if generateFile(4)
    disp(['genDftModuleTestbench: Output file: ',outputFileName]);
end

TestvalueReal=(rand(nPoints)-0.5)*1.999;
TestvalueImag=(rand(nPoints)-0.5)*2;
integerReal=int16(TestvalueReal*(2^(inputBitwidth-1)-1));
integerImag=int16(TestvalueImag*(2^(inputBitwidth-1)-1));
floatingpointResult=fft2(complex(integerReal,integerImag));

if generateFile(4)
```

```
34      fileID = fopen(outputFileName,'w'); %opens output file:
        %generate header
36      %fprintf(fileID,'`timescale 1ns / 1ps\n');
        if numberOfOutputTerms==1
38          fprintf(fileID, '/*********************************/\n');
            fprintf(fileID, '/***   debug testbench for   ***/\n');
40          fprintf(fileID, '/***   coefficient %d, %d  ***/\n', outputDftTerms
        (1,1), outputDftTerms(1,2));
            fprintf(fileID, '/*********************************/\n');
42          if twoCycle
                fprintf(fileID, 'module tb_coefficient_%dx%d_%d_%d_2cy_toplevel
        ;\n\t',nPoints,nPoints,outputDftTerms(1,1),outputDftTerms(1,2));
44           else
                fprintf(fileID, 'module tb_coefficient_%dx%d_%d_%d_toplevel;\n\
        t',nPoints,nPoints,outputDftTerms(1,1),outputDftTerms(1,2));
46          end
        else
48          fprintf(fileID, '/*********************************/\n');
            fprintf(fileID, '/***   generated testbench    ***/\n');
50          fprintf(fileID, '/***    for toplevel %dX%d    ***/\n',nPoints,
        nPoints);
            fprintf(fileID, '/*********************************/\n');
52          if twoCycle
                fprintf(fileID, 'module tb_coefficients_%dx%d_2cy_toplevel;\n\t
        ',nPoints,nPoints);
54           else
                fprintf(fileID, 'module tb_coefficients_%dx%d_toplevel;\n\t',
        nPoints,nPoints);
56          end
        end
58      fprintf(fileID, '//uut stimuli:\n\t');
        fprintf(fileID, 'reg CLK;\n\t');
60      fprintf(fileID, 'reg nReset;\n\t');
        fprintf(fileID, 'reg start;\n\t');
62      fprintf(fileID, 'reg [%d:0] mux_select;\n\t', selectWidth-1);
        fprintf(fileID, 'reg signed [%d:0] inputReal;\n\t', inputBitwidth-1);
64      fprintf(fileID, 'reg signed [%d:0] inputImag;\n\t', inputBitwidth-1);
        fprintf(fileID, '//uut outputs:\n\t');
66      fprintf(fileID, 'wire signed [%d:0] outputReal;\n\t',
        accuoutputBitwidth-1);
        fprintf(fileID, 'wire signed [%d:0] outputImag;\n\t',
        accuoutputBitwidth-1);
68      fprintf(fileID, 'wire done;\n\n\t');
        if numberOfOutputTerms==1
```

```
70          if twoCycle
                fprintf(fileID, 'coefficient_%dx%d_%d_%d_2cy_toplevel uut (\n\t
    \t', nPoints, nPoints, outputDftTerms(1,1), outputDftTerms(1,2));
72          else
                fprintf(fileID, 'coefficient_%dx%d_%d_%d_toplevel uut (\n\t\t',
     nPoints, nPoints, outputDftTerms(1,1), outputDftTerms(1,2));
74          end
        else
76          if twoCycle
                fprintf(fileID, 'coefficient_%dx%d_2cy_toplevel uut (\n\t\t',
    nPoints, nPoints);
78          else
                fprintf(fileID, 'coefficient_%dx%d_toplevel uut (\n\t\t',
    nPoints, nPoints);
80          end
        end
82      fprintf(fileID, '.CLK(CLK),\n\t\t');
        fprintf(fileID, '.nReset(nReset),\n\t\t');
84      fprintf(fileID, '.start(start),\n\t\t');
        fprintf(fileID, '.mux_select(mux_select),\n\t\t');
86      fprintf(fileID, '.real_IN(inputReal),\n\t\t');
        fprintf(fileID, '.imag_IN(inputImag),\n\t\t');
88      fprintf(fileID, '.real_OUT(outputReal),\n\t\t');
        fprintf(fileID, '.imag_OUT(outputImag),\n\t\t');
90      fprintf(fileID, '.done(done)\n\t);\n');

92      fprintf(fileID, 'always #25 CLK = ~ CLK; //clk generator\n');
        fprintf(fileID, 'initial begin\n\t');
94      fprintf(fileID, 'CLK=0;\n\t');
        fprintf(fileID, 'nReset=0;\n\t');
96      fprintf(fileID, 'mux_select=%d''d0;\n\t', selectWidth);
        fprintf(fileID, 'start=0;\n\t');
98      fprintf(fileID, 'inputReal=0;\n\t');
        fprintf(fileID, 'inputImag=0;\n\t');
100     fprintf(fileID, '#50;\n\t');
        fprintf(fileID, 'nReset=1;\n\t');
102     fprintf(fileID, '#50;\n\t');
        fprintf(fileID, 'start=1;\n\t');
104     fprintf(fileID, '#50;\n\t');
        fprintf(fileID, 'start=0;\n\t');
106     for idxY = 1:nPoints
            for idxX = 1:nPoints
108             fprintf(fileID, "//input X=%d Y=%d\n\t", idxY, idxX);
                if integerReal(idxY,idxX) >=0
```

```
110                 fprintf(fileID, "inputReal=%d'sd%d;\n\t",inputBitwidth,
    integerReal(idxY,idxX));
                else
112                 fprintf(fileID, "inputReal=-%d'sd%d;\n\t",inputBitwidth,abs
    (integerReal(idxY,idxX)));
                end
114             if integerImag(idxY,idxX) >=0
                    fprintf(fileID, "inputImag=%d'sd%d;\n\t",inputBitwidth,
    integerImag(idxY,idxX));
116             else
                    fprintf(fileID, "inputImag=-%d'sd%d;\n\t",inputBitwidth,abs
    (integerImag(idxY,idxX)));
118             end
                if twoCycle ==1
120                 fprintf(fileID, '#100;\n\t');
                else
122                 fprintf(fileID, '#50;\n\t');
                end
124
            end
126     end
        fprintf(fileID, "inputImag=%d'sd0;\n\t", inputBitwidth);
128     fprintf(fileID, "inputImag=%d'sd0;\n\t", inputBitwidth);
        if twoCycle ==1
130         fprintf(fileID, '#100;\n\t');
        else
132         fprintf(fileID, '#50;\n\t');
        end
134     fprintf(fileID, '\n\t//Results:\n\t');
        %accumulator=zeros(nPoints*nPoints);
136 end


138
    for outTerm = 1:numberOfOutputTerms
140     currentTerm=outputDftTerms(outTerm,:);
        decimalResulution=ceil(log10(2)*coefFracBitwidth)+2;
142     integerCoef(:,:)=round(Transformcoefficients(currentTerm(1),currentTerm
        (2),:,:)*10^decimalResulution)/10^decimalResulution;
        integerCoef = round(integerCoef*2^coefFracBitwidth);
144     accumulator=0+0i;
        for idxY = 1:nPoints
146         for idxX = 1:nPoints
                accumulator=accuFunction(complex(integerReal(idxY,idxX),
    integerImag(idxY,idxX)),integerCoef(idxY,idxX),accumulator,
```

```
    coefFracBitwidth);
148             if numberOfOutputTerms == 1 %assume debug build if only one
    coef
                if generateFile(4)
150                 fprintf(fileID, "//output %d %d \n",idxX,idxY);
                    fprintf(fileID, "//Real=%d; Imaginary=%d\n",real(
    accumulator),imag(accumulator));
152                 end
            end
154         end
    end
156
    if generateFile(4)
158         fprintf(fileID, '\n\t');
            fprintf(fileID, 'mux_select = %d''d%d;\n\t',selectWidth, outTerm-1)
    ;
160         fprintf(fileID, '#50;\n\t');
            if real(accumulator) >=0
162             fprintf(fileID, 'if (outputReal!=%d''sd%d | ^outputReal === 1''
    bX)\n\t\t',accuoutputBitwidth,real(accumulator));
                fprintf(fileID, '$display("Real result: FAIL");\n\t');
164             fprintf(fileID, 'else\n\t\t$display("Real result: OK");\n\t');
            else %negative value format to verilog literal format for negative
    numbers
166             fprintf(fileID, 'if (outputReal!=-%d''sd%d | ^outputReal === 1'
    'bX)\n\t\t',accuoutputBitwidth,abs(real(accumulator)));
                fprintf(fileID, '$display("Real result: FAIL");\n\t');
168             fprintf(fileID, 'else\n\t\t$display("Real result: OK");\n\t');
            end
170         if imag(accumulator) >=0
                fprintf(fileID, 'if (outputImag!=%d''sd%d | ^outputImag === 1''
    bX)\n\t\t',accuoutputBitwidth,imag(accumulator));
172             fprintf(fileID, '$display("Imag result: FAIL");\n\t');
                fprintf(fileID, 'else\n\t\t$display("Imag result: OK");\n\t');
174         else %negative value format to verilog literal format for negative
    numbers
                fprintf(fileID, 'if (outputImag!=-%d''sd%d | ^outputImag === 1'
    'bX)\n\t\t',accuoutputBitwidth,abs(imag(accumulator)));
176             fprintf(fileID, '$display("Imag result: FAIL");\n\t');
                fprintf(fileID, 'else\n\t\t$display("Imag result: OK");\n\t');
178         end
            fprintf(fileID, '#50;\n\t');
180     end
```

```
    disp(['genDftModuleTestbench: FFT results Floatingpoint: ' num2str(
    floatingpointResult(currentTerm(1),currentTerm(2))) ' fixedPoint:'
    num2str(accumulator) ...
        ' Difference: Real:' num2str( ((real(accumulator-
    floatingpointResult(currentTerm(1),currentTerm(2))))/real(
    floatingpointResult(currentTerm(1),currentTerm(2))))*100 ) '%' ...
        ' Imag:' num2str( ((imag(accumulator-floatingpointResult(
    currentTerm(1),currentTerm(2))))/imag(floatingpointResult(currentTerm
    (1),currentTerm(2))))*100 ) '%']);
end
if generateFile(4)
    fprintf(fileID, '$finish;\n\t');
    fprintf(fileID, '\nend\nendmodule\n');
    fclose(fileID);
end

end

function result=coefmultiply(sample, coefficient,fractionbits)
    mulresult=int64(sample)*int64(coefficient);
    result=bitshift(mulresult,-fractionbits);
    %disp(['sample= ' num2str(sample) ' * ' num2str(coefficient) ' = '
    num2str(result)]);
end

function result=accuFunction(sample, coefficient, accumulator, fractionbits
    )
    signrealcoef=sign(real(coefficient));
    signimagcoef=sign(imag(coefficient));
    realcoef=abs(real(coefficient));
    imagcoef=abs(imag(coefficient));

    realXreal=coefmultiply(real(sample),realcoef,fractionbits);
    realXimag=coefmultiply(real(sample),imagcoef,fractionbits);
    imagXreal=coefmultiply(imag(sample),realcoef,fractionbits);
    imagXimag=coefmultiply(imag(sample),imagcoef,fractionbits);

    if signrealcoef >= 0 && signimagcoef >= 0
        result=accumulator+complex(double(realXreal-imagXimag),double(
    realXimag+imagXreal));
    elseif signrealcoef == -1 && signimagcoef >=0
        result=accumulator+complex(double(-realXreal-imagXimag),double(
    realXimag-imagXreal));
    elseif signrealcoef >= 0 && signimagcoef == -1
```

```
            result=accumulator+complex(double(realXreal+imagXimag),double(-
    realXimag+imagXreal));
216     elseif  signrealcoef == -1 && signimagcoef == -1
            result=accumulator+complex(double(-realXreal+imagXimag),double(-
    realXimag-imagXreal));
218     else
            result=accumulator+complex(double(realXreal-imagXimag),double(
    realXimag+imagXreal));
220     end
end
```

Listing A.4: genDftModuleTestbench.m

A.1.5 Quellcode: genVhdlDftModule.m

```
1  function genVhdlDftModule(nPoints, outputDftTerms, inputBitwidth,
       outputBitwidth, accumulatorBitwidth, outputDividerFactor,
       outputDividerFracBitwidth, twoCycle, generateFile)

3  if generateFile(6)
       if exist('outputDftTerms')==0
5          disp('Error! input Parameters');
           return;
7      end
       numberOfOutputTerms=size(outputDftTerms,1);
9      if numberOfOutputTerms>1
           selectWidth=nextpow2(numberOfOutputTerms); %
11     else
           selectWidth=1;
13     end

15     START_ADDR_RD=4;

17     outputFileName=['dft_Module_',num2str(nPoints), 'x', num2str(nPoints),'
       .vhd'];
       disp(['gen VHDL DFT Module: Output file: ',outputFileName]);
19     fileID = fopen(outputFileName,'w');
       fprintf(fileID, '------------------------------------------------\n');
21     fprintf(fileID, '---                DFT module                ---\n');
       fprintf(fileID, '---      Built: %s      ---\n', datetime('now'));
23     fprintf(fileID, '---   generated by brute2Dft framework   ---\n');
       fprintf(fileID, '---      written by Martin Willimczik    ---\n');
25     fprintf(fileID, '------------------------------------------------\n');
```

```
    fprintf(fileID, 'library ieee;\nuse ieee.std_logic_1164.all;\n');
27  fprintf(fileID, 'use ieee.numeric_std.all;\n\npackage DFT_MODULE_PKG is
    \n');
    fprintf(fileID, 'component DFT_MODULE\n');
29  fprintf(fileID, 'port(\n\tCLK     : in  std_logic;\n\t');
    fprintf(fileID, 'nRESET          : in  std_logic;\n\t');
31  fprintf(fileID, 'CTRL_EXT_EN     : in  std_logic;\n\t');
    fprintf(fileID, 'MOD_SEL         : in  std_logic_vector(2 downto 0);\n\
    t');
33  fprintf(fileID, 'RDY             : out std_logic;\n\t');
    fprintf(fileID, 'WE              : out std_logic;\n\t');
35  fprintf(fileID, 'RE              : out std_logic;\n\t');
    fprintf(fileID, 'ADDR_BUS        : out std_logic_vector(9 downto 0);\n\
    t');
37  fprintf(fileID, 'DIN_BUS         : out std_logic_vector(31 downto 0);\n
    \t');
    fprintf(fileID, 'DOUT_BUS        : in  std_logic_vector(31 downto 0)\n)
    ;\n');
39  fprintf(fileID, 'end component;\nend DFT_MODULE_PKG;\n');
    fprintf(fileID, '\n');
41
    fprintf(fileID, 'library ieee;\nuse ieee.std_logic_1164.all;\n');
43  fprintf(fileID, 'use ieee.std_logic_unsigned.all;\n');
    fprintf(fileID, 'use ieee.numeric_std.all;\n\n');
45  fprintf(fileID, 'entity DFT_MODULE is\n');
    fprintf(fileID, 'generic(\n');
47  fprintf(fileID, '    MOD_NR          : std_logic_vector := "001";    --
     module number\n');
    fprintf(fileID, '    COUNTER_WIDTH   : natural := 3;                 --
     counter witdth\n');
49  fprintf(fileID, '    QEND            : unsigned := "011";            --
     pipeline delay counter\n');
    fprintf(fileID, '    START_ADDR_RD   : unsigned := "%s";     -- start
    address read\n',dec2bin(START_ADDR_RD,10));
51  fprintf(fileID, '    START_ADDR_WR   : unsigned := "%s"      -- start
    address write\n',dec2bin(START_ADDR_RD+nPoints^2,10));
    fprintf(fileID, ');\n');
53
    fprintf(fileID, 'port(\n');
55  fprintf(fileID, '    CLK             : in  std_logic;\n');
    fprintf(fileID, '    nRESET          : in  std_logic;\n');
57  fprintf(fileID, '    CTRL_EXT_EN     : in  std_logic;\n');
    fprintf(fileID, '    MOD_SEL         : in  std_logic_vector(2 downto 0)
    ; \n');
```

```
59      fprintf(fileID, '    RDY              : out std_logic;\n');
        fprintf(fileID, '    WE               : out std_logic;\n');
61      fprintf(fileID, '    RE               : out std_logic;\n');
        fprintf(fileID, '    ADDR_BUS         : out std_logic_vector(9 downto 0)
        ;\n');
63      fprintf(fileID, '    DIN_BUS          : out std_logic_vector(31 downto
        0);\n');
        fprintf(fileID, '    DOUT_BUS         : in  std_logic_vector(31 downto
        0)\n');
65      fprintf(fileID, ');\n');
        fprintf(fileID, 'end DFT_MODULE;\n');
67
        fprintf(fileID, 'architecture ARCH of DFT_MODULE is\n');
69      fprintf(fileID, 'signal WE_INT    : std_logic;\n');
        fprintf(fileID, 'signal ADDR, LOAD_ADDR  : std_logic_vector(9 downto 0)
        ;\n');
71      fprintf(fileID, 'signal mux_out_Real, outputMultR_OUTPUT :
        std_logic_vector(%d downto 0);\n', accumulatorBitwidth-1);
        fprintf(fileID, 'signal mux_out_Imag, outputMultI_OUTPUT :
        std_logic_vector(%d downto 0);\n', accumulatorBitwidth-1);
73      fprintf(fileID, 'signal outputMultR_INPUT_REG, outputMultI_INPUT_REG :
        std_logic_vector(%d downto 0);\n', accumulatorBitwidth-1);
        fprintf(fileID, 'signal REAL_SAMPLE, IMAG_SAMPLE : std_logic_vector(%d
        downto 0);\n', inputBitwidth-1);
75      fprintf(fileID, 'signal MULT_OUT_REG_R, MULT_OUT_REG_I :
        std_logic_vector(%d downto 0);\n', accumulatorBitwidth-1);
        fprintf(fileID, 'signal REAL_OUT_SAT, IMAG_OUT_SAT : std_logic_vector(%
        d downto 0);\n', outputBitwidth-1);
77      fprintf(fileID, 'signal DIN       : std_logic_vector(31 downto 0);\n');
        fprintf(fileID, 'signal RDY_INT   : std_logic;\n');
79      fprintf(fileID, 'signal SAMPLE_START : std_logic;\n');
        fprintf(fileID, 'signal DONE : std_logic;\n');
81      fprintf(fileID, 'signal MUX_SELECT : std_logic_vector(%d downto 0);\n',
        selectWidth-1);
        fprintf(fileID, 'signal QPLUS     : unsigned(COUNTER_WIDTH-1 downto 0);\
        n');
83      fprintf(fileID, 'signal Q         : unsigned(COUNTER_WIDTH-1 downto 0);\
        n');
        fprintf(fileID, 'signal CTR_RDY, EN_ADDR_CNT  : std_logic;\n');
85      fprintf(fileID, 'signal ENABLE_POINT_COUNT, EN_ADDR_LOAD  : std_logic;\
        n');
        fprintf(fileID, 'constant OUT_PADDING: std_logic_vector(%d downto 0):=(
        others => ''0''); --fills gaps in output vector\n', ((32-outputBitwidth
        *2)/2)-1);
```

```
87      fprintf(fileID, 'constant UNSAT_POS: std_logic_vector(%d downto 0):=(
        others => ''0''); --comparism value for no overflow\n',
        accumulatorBitwidth-outputBitwidth);
        fprintf(fileID, 'constant UNSAT_NEG: std_logic_vector(%d downto 0):=(
        others => ''1''); --comparism value for no underflow\n',
        accumulatorBitwidth-outputBitwidth);
89      fprintf(fileID, 'constant SAT_POS: std_logic_vector(%d downto 0):= B"%s
        "; --max pos out value\n', outputBitwidth-1, ['0' dec2bin(2^(
        outputBitwidth-1)-1,outputBitwidth-1)] );
        fprintf(fileID, 'constant SAT_NEG: std_logic_vector(%d downto 0):= B"%s
        "; --max neg value\n', outputBitwidth-1,['1' dec2bin(0,outputBitwidth
        -1)]);
91      fprintf(fileID, 'constant END_ADDR_RD : std_logic_vector(9 downto 0) :=
          std_logic_vector(START_ADDR_RD+ B"%s"); -- end address read\n',
        dec2bin(nPoints^2-1));
        fprintf(fileID, 'constant END_ADDR_WR : std_logic_vector(9 downto 0) :=
          std_logic_vector(START_ADDR_WR+ B"%s");  -- end address write\n',
        dec2bin(numberOfOutputTerms-1));
93
        fprintf(fileID, 'type STATES is (INIT, RD0, RD1, RD2, WT0, WR0, WR1,
        WR2, WT1);\n');
95      fprintf(fileID, 'signal STATE, NEXT_STATE: STATES;\n');

97      fprintf(fileID, '--declarations for Verilog Components:\n');

99      declareVHDLcoefficient(fileID, nPoints, twoCycle, outputDftTerms,
        inputBitwidth, accumulatorBitwidth, selectWidth);
        declareVHDLOutputMultiplier(fileID, accumulatorBitwidth, round(
        outputDividerFactor/(2^-outputDividerFracBitwidth)));
101     fprintf(fileID, '--end of verilog declarations\n');
        %end of declarative part
103
        fprintf(fileID, 'begin\n');
105
        fprintf(fileID, '-- delay Counter\n');
107     fprintf(fileID, 'COUNTER_STATE_MEM: process(nRESET, CLK, STATE)\n');
        fprintf(fileID, 'begin\n');
109     fprintf(fileID, '    if nRESET = ''0'' or STATE /= WT0 then\n');
        fprintf(fileID, '        Q <= (others => ''0'');\n');
111     fprintf(fileID, '    elsif CLK  = ''1'' and CLK''event then\n');
        fprintf(fileID, '        Q <= QPLUS;\n');
113     fprintf(fileID, '    end if;\n');
        fprintf(fileID, 'end process;\n');
115
```

```
    fprintf(fileID, 'COUNTER_TRANSITION_LOGIC: process(Q)\n');
117 fprintf(fileID, 'begin\n');
    fprintf(fileID, '    if Q = QEND then\n');
119 fprintf(fileID, '        QPLUS <= (others => ''0'');\n');
    fprintf(fileID, '        CTR_RDY <= ''1'';\n');
121 fprintf(fileID, '    else\n');
    fprintf(fileID, '        QPLUS <= Q + 1;\n');
123 fprintf(fileID, '        CTR_RDY <= ''0'';\n');
    fprintf(fileID, '    end if;\n');
125 fprintf(fileID, 'end process;\n');

127 fprintf(fileID, '------------------\n');
    fprintf(fileID, '--ADDR COUNTER--\n');
129 fprintf(fileID, '------------------\n');
    fprintf(fileID, 'ADDR_COUNTER_REG: process(nRESET, CLK)\n');
131 fprintf(fileID, 'begin\n\t');
    fprintf(fileID, 'if nRESET = ''0'' then\n\t\t');
133 fprintf(fileID, 'ADDR <= (others => ''0'') after 1ns;\n\t');
    fprintf(fileID, 'elsif CLK  = ''1'' and CLK''event then\n\t\t');
135 fprintf(fileID, 'if (EN_ADDR_LOAD = ''1'') then\n\t\t\t');
    fprintf(fileID, 'ADDR <= LOAD_ADDR after 1ns;\n\t\t');
137 fprintf(fileID, 'elsif (EN_ADDR_CNT = ''1'') then\n\t\t\t');
    fprintf(fileID, 'ADDR <= ADDR + 1 after 1ns;\n\t\t');
139 fprintf(fileID, 'end if;\n\t');
    fprintf(fileID, 'end if;\n');
141 fprintf(fileID, 'end process;\n');

143 fprintf(fileID, '-- Module state machine\n');
    fprintf(fileID, 'STATE_MEM: process(CLK, nRESET, MOD_SEL)\n');
145 fprintf(fileID, 'begin\n');
    fprintf(fileID, '    if nRESET = ''0'' or MOD_SEL /= MOD_NR then\n');
147 fprintf(fileID, '        STATE <= INIT;\n');
    fprintf(fileID, '    elsif CLK  = ''1'' and CLK''event then\n');
149 fprintf(fileID, '        STATE <= NEXT_STATE;\n');
    fprintf(fileID, '    end if;\n');
151 fprintf(fileID, 'end process;\n\n');

153 fprintf(fileID, 'TRANSITION_LOGIC: process(STATE, MOD_SEL, CTR_RDY,
    ADDR)\n');
    fprintf(fileID, 'begin\n\t');
155 fprintf(fileID, '--default assignments for all outputs to prevent
    latches;\n\t');
    fprintf(fileID, 'NEXT_STATE <= STATE;\n\t');
157 fprintf(fileID, 'RDY_INT <= ''0'';\n\t');
```

```
    fprintf(fileID, 'WE_INT <= ''0'';\n\t');
159 fprintf(fileID, 'EN_ADDR_LOAD <= ''0'';\n\t');
    fprintf(fileID, 'SAMPLE_START <= ''0'';\n\t');
161 fprintf(fileID, 'EN_ADDR_CNT  <= ''0'';\n\t');
    fprintf(fileID, 'ENABLE_POINT_COUNT <= ''0'';\n\t');
163 fprintf(fileID, 'LOAD_ADDR <= std_logic_vector(START_ADDR_RD);\n\t');
    fprintf(fileID, 'case STATE is\n\t\t');
165 fprintf(fileID, 'when INIT =>\n\t\t\t');
    fprintf(fileID, 'if MOD_SEL = MOD_NR then\n\t\t\t\t');
167 fprintf(fileID, 'NEXT_STATE <= RD0;\n\t\t\t');
    fprintf(fileID, 'end if;\n\t\t\t');
169 fprintf(fileID, 'EN_ADDR_LOAD <= ''1'';\n\t\t');
    fprintf(fileID, 'when RD0  =>\n\t\t\tNEXT_STATE <= RD2;\n\t\t\t');
171 fprintf(fileID, 'SAMPLE_START <= ''1'';\n\t\t');
    fprintf(fileID, 'when RD1  =>\n\t\t\tif ADDR >= END_ADDR_RD then\n\t\t\
    t\t');
173 fprintf(fileID, 'NEXT_STATE <= WT0;\n\t\t\t');
    fprintf(fileID, 'else\n\t\t\t');
175 fprintf(fileID, 'NEXT_STATE <= RD2;\n\t\t\t');
    fprintf(fileID, 'end if;\n\t\t');
177
    fprintf(fileID, 'when RD2  =>\n\t\t\t');
179 fprintf(fileID, 'if ADDR >= END_ADDR_RD then\n\t\t\t\t');
    fprintf(fileID, 'NEXT_STATE <= WT0;\n\t\t\t');
181 fprintf(fileID, 'else\n\t\t\t\t');
    fprintf(fileID, 'NEXT_STATE <= RD1;\n\t\t\t');
183 fprintf(fileID, 'end if;\n\t\t\t');
    fprintf(fileID, 'EN_ADDR_CNT  <= ''1'';\n\t\t');
185
    fprintf(fileID, 'when WT0  => --waits for pipelining delay\n\t\t\t');
187 fprintf(fileID, 'if CTR_RDY = ''1'' then\n\t\t\t\t');
    fprintf(fileID, 'NEXT_STATE <= WR0;\n\t\t\t');
189 fprintf(fileID, 'end if;\n\t\t\t');
    fprintf(fileID, 'EN_ADDR_LOAD <= ''1'';\n\t\t\t');
191 fprintf(fileID, 'LOAD_ADDR <= std_logic_vector(START_ADDR_WR);\n\t\t');


193 fprintf(fileID, 'when WR0  => --start mux counting\n\t\t\t');
    fprintf(fileID, 'NEXT_STATE <= WR1;\n\t\t\t');
195 fprintf(fileID, 'ENABLE_POINT_COUNT <= ''1'';\n\t\t\t');

197 fprintf(fileID, 'when WR1  => --pipeline delay\n\t\t\t');
    fprintf(fileID, 'NEXT_STATE <= WR2;\n\t\t\t');
199 fprintf(fileID, 'ENABLE_POINT_COUNT <= ''1'';\n\t\t');
```

```
201     fprintf(fileID, 'when WR2  => --first valid output data applied to ram\
        n\t\t\t');
        fprintf(fileID, 'if ADDR >= END_ADDR_WR then\n\t\t\t\t');
203     fprintf(fileID, 'NEXT_STATE <= WT1;\n\t\t\t');
        fprintf(fileID, 'end if;\n\t\t\t');
205     fprintf(fileID, 'ENABLE_POINT_COUNT <= ''1'';\n\t\t\t');
        fprintf(fileID, 'WE_INT <= ''1'';\n\t\t\t');
207     fprintf(fileID, 'EN_ADDR_CNT  <= ''1'';\n\t\t');

209     fprintf(fileID, 'when WT1  => --signal processing is done & wait for
        deselect\n\t\t\t');
        fprintf(fileID, 'if MOD_SEL /= MOD_NR then\n\t\t\t\t');
211     fprintf(fileID, 'NEXT_STATE <= INIT;\n\t\t\t');
        fprintf(fileID, 'end if;\n\t\t\t');
213     fprintf(fileID, 'RDY_INT <= ''1'';\n\t');
        fprintf(fileID, 'end case;\n');
215     fprintf(fileID, 'end process;\n\n');

217     if numberOfOutputTerms==1
            fprintf(fileID, 'MUX_SELECT <= ''0'';\n');
219     else
            generatMuxControl(fileID, outputDftTerms);
221     end

223     fprintf(fileID, 'MUX_OUT_REG: process(CLK)\n');
        fprintf(fileID, 'begin\n\t');
225     fprintf(fileID, 'if CLK = ''1'' and CLK''event then\n\t\t');
        fprintf(fileID, 'outputMultR_INPUT_REG <= mux_out_Real;\n\t\t');
227     fprintf(fileID, 'outputMultI_INPUT_REG <= mux_out_Imag;\n\t');
        fprintf(fileID, 'end if;\n');
229     fprintf(fileID, 'end process;\n\n');

231     fprintf(fileID, 'MULT_OUT_REG: process(CLK)\n');
        fprintf(fileID, 'begin\n\t');
233     fprintf(fileID, 'if CLK = ''1'' and CLK''event then\n\t\t');
        fprintf(fileID, 'MULT_OUT_REG_R <= outputMultR_OUTPUT;\n\t\t');
235     fprintf(fileID, 'MULT_OUT_REG_I <= outputMultI_OUTPUT;\n\t');
        fprintf(fileID, 'end if;\n');
237     fprintf(fileID, 'end process;\n');

239     fprintf(fileID, 'SATURATION_LOGIC_REAL: process(MULT_OUT_REG_R)\n');
        fprintf(fileID, 'begin\n\t');
```

```
241     fprintf(fileID, 'if MULT_OUT_REG_R(%d downto %d) = UNSAT_POS or
        MULT_OUT_REG_R(%d downto %d) = UNSAT_NEG then\n\t\t',
        accumulatorBitwidth-1, outputBitwidth-1, accumulatorBitwidth-1,
        outputBitwidth-1);
        fprintf(fileID, 'REAL_OUT_SAT <= MULT_OUT_REG_R(%d downto 0);\n\t',
        outputBitwidth-1);
243     fprintf(fileID, 'elsif MULT_OUT_REG_R(%d) = ''1'' then\n\t\t',
        accumulatorBitwidth-1);
        fprintf(fileID, 'REAL_OUT_SAT <= SAT_NEG;\n\t');
245     fprintf(fileID, 'else \n\t\t');
        fprintf(fileID, 'REAL_OUT_SAT <= SAT_POS;\n\t');
247     fprintf(fileID, 'end if;\n');
        fprintf(fileID, 'end process;\n');
249
        fprintf(fileID, 'SATURATION_LOGIC_IMAG: process(MULT_OUT_REG_I)\n');
251     fprintf(fileID, 'begin\n\t');
        fprintf(fileID, 'if MULT_OUT_REG_I(%d downto %d) = UNSAT_POS or
        MULT_OUT_REG_I(%d downto %d) = UNSAT_NEG then\n\t\t',
        accumulatorBitwidth-1, outputBitwidth-1, accumulatorBitwidth-1,
        outputBitwidth-1);
253     fprintf(fileID, 'IMAG_OUT_SAT <= MULT_OUT_REG_I(%d downto 0);\n\t',
        outputBitwidth-1);
        fprintf(fileID, 'elsif MULT_OUT_REG_I(%d) = ''1'' then\n\t\t',
        accumulatorBitwidth-1);
255     fprintf(fileID, 'IMAG_OUT_SAT <= SAT_NEG;\n\t');
        fprintf(fileID, 'else \n\t\t');
257     fprintf(fileID, 'IMAG_OUT_SAT <= SAT_POS;\n\t');
        fprintf(fileID, 'end if;\n');
259     fprintf(fileID, 'end process;\n');

261     fprintf(fileID, 'REAL_SAMPLE <= DOUT_BUS(31 downto %d);\n', 31-(
        inputBitwidth-1));
        fprintf(fileID, 'IMAG_SAMPLE <= DOUT_BUS(15 downto %d);\n', 15-(
        inputBitwidth-1));
263     fprintf(fileID, 'DIN <= REAL_OUT_SAT & OUT_PADDING & IMAG_OUT_SAT &
        OUT_PADDING;\n');
        fprintf(fileID, '-- Three state bus drivers\n');
265     fprintf(fileID, 'RE <= ''1'' when MOD_SEL = MOD_NR and CTRL_EXT_EN = ''
        0'' else ''0'';  \n');
        fprintf(fileID, 'WE <= WE_INT when MOD_SEL = MOD_NR and CTRL_EXT_EN = '
        '0'' else ''0''; \n');
267     fprintf(fileID, 'ADDR_BUS <= ADDR when MOD_SEL = MOD_NR and CTRL_EXT_EN
        = ''0'' else (others=>''Z'');\n');
```

```
    fprintf(fileID, 'DIN_BUS <= DIN when MOD_SEL = MOD_NR and CTRL_EXT_EN =
    ''0'' else (others=>''Z'');\n');
    fprintf(fileID, 'RDY <= RDY_INT when MOD_SEL = MOD_NR else ''Z'';\n');
    fprintf(fileID, '--coefficient component instanciations:\n');

    instanciateVerilogToplevel(fileID, nPoints, twoCycle, outputDftTerms);

    fprintf(fileID, '--Output scaling multiplier instanciations:\n');
    instanciateVHDLOutputMultiplier(fileID, round(outputDividerFactor/(2^-
    outputDividerFracBitwidth)), 'outputMultR') ;
    instanciateVHDLOutputMultiplier(fileID, round(outputDividerFactor/(2^-
    outputDividerFracBitwidth)), 'outputMultI') ;

    fprintf(fileID, 'end ARCH;\n\n');

    fclose(fileID); %close output file
end %if generateLevel 6
generateOutputMultiplier(accumulatorBitwidth, outputDividerFracBitwidth,
    outputDividerFracBitwidth, round(outputDividerFactor/(2^-
    outputDividerFracBitwidth)), generateFile);

end %of function

%instanciates one coefficient Module
function instanciateVerilogToplevel(fileID, nPoints,twoCycle,
    outputDftTerms)
    if size(outputDftTerms,1)==1
        if twoCycle
            fprintf(fileID, 'verilogToplevel: coefficient_%dx%d_%d_%
    d_2cy_toplevel port map (\n\t' ...
            , nPoints,nPoints,outputDftTerms(1,1),outputDftTerms(1,2));
        else
            fprintf(fileID, 'verilogToplevel: coefficient_%dx%d_%d_%
    d_toplevel port map (\n\t' ...
            , nPoints,nPoints,outputDftTerms(1,1),outputDftTerms(1,2));
        end
    else
        if twoCycle
            fprintf(fileID, 'verilogToplevel: coefficient_%dx%
    d_2cy_toplevel port map (\n\t', nPoints,nPoints);
        else
            fprintf(fileID, 'verilogToplevel: coefficient_%dx%d_toplevel
    port map (\n\t', nPoints,nPoints);
        end
```

```
    end
303     fprintf(fileID, 'CLK  => CLK,\n\t');
    fprintf(fileID, 'nReset  => nRESET,\n\t');
305     fprintf(fileID, 'start  => SAMPLE_START,\n\t');
    fprintf(fileID, 'mux_select => MUX_SELECT,\n\t');
307     fprintf(fileID, 'real_IN  => REAL_SAMPLE,\n\t');
    fprintf(fileID, 'imag_IN  => IMAG_SAMPLE,\n\t');
309     fprintf(fileID, 'real_OUT  => mux_out_Real,\n\t');
    fprintf(fileID, 'imag_OUT  => mux_out_Imag,\n\t');
311     fprintf(fileID, 'done => DONE\n');
    fprintf(fileID, ');\n');
313 end

315 function declareVHDLcoefficient(fileID, nPoints, twoCycle, outputDftTerms,
    inputBitwidth, outputBitwidth, selectWidth)
    if size(outputDftTerms,1)==1
317         if twoCycle
                fprintf(fileID, 'COMPONENT coefficient_%dx%d_%d_%d_2cy_toplevel
     port (\n\t' ...
319             , nPoints,nPoints,outputDftTerms(1,1),outputDftTerms(1,2));
            else
321             fprintf(fileID, 'COMPONENT coefficient_%dx%d_%d_%d_toplevel
    port (\n\t' ...
                , nPoints,nPoints,outputDftTerms(1,1),outputDftTerms(1,2));
323         end
    else
325         if twoCycle
                fprintf(fileID, 'COMPONENT coefficient_%dx%d_2cy_toplevel port
    (\n\t', nPoints,nPoints);
327         else
                fprintf(fileID, 'COMPONENT coefficient_%dx%d_toplevel port (\n\
    t', nPoints,nPoints);
329         end
    end
331     fprintf(fileID, 'CLK  : in std_logic;\n\t');
    fprintf(fileID, 'nReset : in std_logic;\n\t');
333     fprintf(fileID, 'start : in std_logic;\n\t');
    fprintf(fileID, 'mux_select : in std_logic_vector (%d downto 0);\n\t',
    selectWidth-1);
335     fprintf(fileID, 'real_IN  : in std_logic_vector(%d downto 0);\n\t',
    inputBitwidth-1);
    fprintf(fileID, 'imag_IN : in std_logic_vector(%d downto 0);\n\t',
    inputBitwidth-1);
```

```
337     fprintf(fileID, 'real_OUT : out std_logic_vector(%d downto 0);\n\t',
        outputBitwidth-1);
        fprintf(fileID, 'imag_OUT : out std_logic_vector(%d downto 0);\n\t',
        outputBitwidth-1);
339     fprintf(fileID, 'done : out std_logic\n');
        fprintf(fileID, ');\nEND COMPONENT;\n');
341 end

343 function declareVHDLOutputMultiplier(fileID, inputBitwidth, coefficient)
        fprintf(fileID, 'COMPONENT outputMul_%s port (\n\t' ...
345         , num2str(coefficient));
        fprintf(fileID, 'INA : in std_logic_vector(%d downto 0);\n\t',
        inputBitwidth-1);
347     fprintf(fileID, 'control : in std_logic_vector;\n\t');
        fprintf(fileID, 'Q : out std_logic_vector(%d downto 0)\n',inputBitwidth
        -1);
349     fprintf(fileID, ');\nEND COMPONENT;\n');
    end
351
    function instanciateVHDLOutputMultiplier(fileID, coefficient, moduleName)
353     fprintf(fileID, '%s: outputMul_%s port map (\n\t' ...
        , moduleName, num2str(coefficient));
355     fprintf(fileID, 'INA => %s_INPUT_REG,\n\t', moduleName);
        fprintf(fileID, 'control => "0",\n\t');
357     fprintf(fileID, 'Q => %s_OUTPUT\n', moduleName);
        fprintf(fileID, ');\n');
359 end
    % run the external fixed multiplier generator program
361 % mostly a clone of the function in genVerilog
    function generateOutputMultiplier(inputBitwidth, constantWidth, FracWidth,
        coefficient, generateFile)
363     coefficientString = num2str(coefficient);
        multipliername=sprintf('outputMul_%s.v', coefficientString);
365     modulename=sprintf('outputMul_%s', coefficientString);
        if generateFile(7) && generateFile(3)
367         if ispc
                command=sprintf('multiplierGen\\kmult.exe -o %s -i %d -c %d -f
        %d -O 1 %s', ...
369                     multipliername, inputBitwidth, constantWidth, FracWidth,
        coefficientString);
            else
371             command=sprintf('./multiplierGen/kmult -o %s -i %d -c %d -f %d
        -O 1 %s', ...
```

```
                multipliername, inputBitwidth, constantWidth, FracWidth,
    coefficientString);
        end
        status=system(command);
        if status == 0
            % rename the module to the correct name
            fid = fopen(multipliername,'rt') ;
            X = fread(fid);
            fclose(fid);
            X = char(X.');
            Y = strrep(X, 'adderchain',modulename);
            fid = fopen(multipliername,'wt');
            fwrite(fid,Y);
            fclose(fid);
        end
    elseif generateFile(7)
        disp(['genVhdlDftModule: NOT using kmult']);
    end
end

function generatMuxControl(fileID, DftTerms)
    fprintf(fileID, '--controls the output multiplexer\n');
    fprintf(fileID, 'MUX_CONTROL: process(CLK, nRESET)\n');
    fprintf(fileID, 'begin\n\t');
    fprintf(fileID, 'if nRESET = ''0'' then\n\t\tMUX_SELECT <=(others => ''
    0'');\n\t');
    fprintf(fileID, 'elsif CLK = ''1'' and CLK''event then\n\t\t');
    fprintf(fileID, 'if ENABLE_POINT_COUNT =''1'' then\n\t\t\t');
    fprintf(fileID, 'MUX_SELECT <= MUX_SELECT+''1'';\n\t\t');
    fprintf(fileID, 'else\n\t\t\t');
    fprintf(fileID, 'MUX_SELECT <= MUX_SELECT;\n\t\t');
    fprintf(fileID, 'end if;\n\t');
    fprintf(fileID, 'end if;\n');
    fprintf(fileID, 'end process;\n\n');
end
```

Listing A.5: genVhdlDftModule.m

A.1.6 Quellcode: genVhdlTestbench.m

```
function genVhdlTestbench(nPoints, Transformcoefficients, outputDftTerms,
    inputBitwidth, accumulatorBitwidth, coefFracBitwidth,
    outputDividerFactor, outputDividerFracBitwidth, generateFile)
```

```
2
disp('genVhdlTestbench: 2D-DFT VHDL Toplevel Testbench generator');
4 if exist('Transformcoefficients')==0
    disp('genVhdlTestbench: Error! generate/load Transform coefficients
    first');
6     return;
end
8 numberOfOutputTerms=size(outputDftTerms,1);

10 printAccuOutput=1; %set to one to see all Accu intermediate values printed
START_ADDR_RD=4;
12 %outputBitwidth=12; %bitwidth of input
TestvalueReal=(rand(nPoints)-0.5)*1.999; %prevent it from actually reaching
     2048
14 TestvalueImag=(rand(nPoints)-0.5)*1.999;
integerReal=int16(TestvalueReal*(2^(inputBitwidth-1)-1));
16 integerImag=int16(TestvalueImag*(2^(inputBitwidth-1)-1));
floatingpointResult=fft2(complex(integerReal,integerImag));
18 if generateFile(8)
    outputFileName=['tb_toplevel_',num2str(nPoints), 'x', num2str(nPoints),
     '.vhd'];
20     disp(['genVhdlTestbench: Output file: ',outputFileName]);
    fileID = fopen(outputFileName,'w'); %opens output file:
22     %generate header
    fprintf(fileID, '-------------------------------------------------\n');
24     fprintf(fileID, '---          VHDL toplevel testbench        ---\n');
    fprintf(fileID, '---       Built: %s      ---\n', datetime('now'));
26     fprintf(fileID, '---    generated by brute2Dft framework   ---\n');
    fprintf(fileID, '---       written by Martin Willimczik     ---\n');
28     fprintf(fileID, '-------------------------------------------------\n');
    fprintf(fileID, 'LIBRARY ieee;\nUSE ieee.std_logic_1164.ALL;\n');
30     fprintf(fileID, 'ENTITY tb_toplevel IS\nEND tb_toplevel;\n');
    fprintf(fileID, 'ARCHITECTURE behavior OF tb_toplevel IS\n');
32     fprintf(fileID, 'COMPONENT TOPLEVEL\nPORT(\n\t');
    fprintf(fileID, 'CLK : IN  std_logic;\n\t');
34     fprintf(fileID, 'nRESET : IN  std_logic;\n\t');
    fprintf(fileID, 'CTRL_EXT_EN : IN  std_logic;\n\t');
36     fprintf(fileID, 'MOD_SEL_EXT_EN : IN  std_logic;\n\t');
    fprintf(fileID, 'MOD_OUT : OUT  std_logic_vector(2 downto 0);\n\t');
38     fprintf(fileID, 'MOD_RDY : OUT  std_logic;\n\t');
    fprintf(fileID, 'MOD_CLK_EN : IN  std_logic;\n\t');
40     fprintf(fileID, 'DSEL : IN  std_logic_vector(2 downto 0);\n\t');
    fprintf(fileID, 'WE : IN  std_logic;\n\t');
42     fprintf(fileID, 'DIN : IN  std_logic_vector(9 downto 0);\n\t');
```

```
    fprintf(fileID, 'DOUT : OUT  std_logic_vector(7 downto 0)\n\t');
44  fprintf(fileID, ');\nEND COMPONENT;\n\n');
    fprintf(fileID, '--DUT - Inputs\n');
46  fprintf(fileID, 'signal CLK : std_logic := ''0'';\n');
    fprintf(fileID, 'signal nRESET : std_logic := ''0'';\n');
48  fprintf(fileID, 'signal CTRL_EXT_EN : std_logic := ''0'';\n');
    fprintf(fileID, 'signal MOD_SEL_EXT_EN : std_logic := ''0'';\n');
50  fprintf(fileID, 'signal MOD_CLK_EN : std_logic := ''0'';\n');
    fprintf(fileID, 'signal DSEL : std_logic_vector(2 downto 0) := (others
    => ''0'');\n');
52  fprintf(fileID, 'signal WE : std_logic := ''0'';\n');
    fprintf(fileID, 'signal DIN : std_logic_vector(9 downto 0) := (others
    => ''0'');\n');
54  fprintf(fileID, '--DUT - Outputs\n');
    fprintf(fileID, 'signal MOD_OUT : std_logic_vector(2 downto 0);\n');
56  fprintf(fileID, 'signal MOD_RDY : std_logic;\n');
    fprintf(fileID, 'signal DOUT : std_logic_vector(7 downto 0);\n');
58  fprintf(fileID, '-- Simulation clock period:\n');
    fprintf(fileID, 'constant CLK_period : time := 25 ns;\n\n');
60  fprintf(fileID, 'BEGIN\n');
    fprintf(fileID, '-- Instantiate the Unit Under Test (UUT)\n');
62  fprintf(fileID, 'DUT: TOPLEVEL PORT MAP (\n');
    fprintf(fileID, 'CLK => CLK,\n\tnRESET => nRESET,\n\t');
64  fprintf(fileID, 'CTRL_EXT_EN => CTRL_EXT_EN,\n\t');
    fprintf(fileID, 'MOD_SEL_EXT_EN => MOD_SEL_EXT_EN,\n\t');
66  fprintf(fileID, 'MOD_OUT => MOD_OUT,\n\t');
    fprintf(fileID, 'MOD_RDY => MOD_RDY,\n\t');
68  fprintf(fileID, 'MOD_CLK_EN => MOD_CLK_EN,\n\t');
    fprintf(fileID, 'DSEL => DSEL,\n\t');
70  fprintf(fileID, 'WE => WE,\n\t');
    fprintf(fileID, 'DIN => DIN,\n\t');
72  fprintf(fileID, 'DOUT => DOUT\n);\n');
    fprintf(fileID, '-- Clock process definitions\n');
74  fprintf(fileID, 'CLK_process :process\n');
    fprintf(fileID, 'begin\n\tCLK <= ''0'';\n\t');
76  fprintf(fileID, 'wait for CLK_period/2;\n\tCLK <= ''1'';\n\t');
    fprintf(fileID, 'wait for CLK_period/2;\nend process;\n');
78  fprintf(fileID, '-- Stimulus process\n');
    fprintf(fileID, 'stim_proc: process\nbegin\n\t');
80  fprintf(fileID, 'nRESET <= ''0'';\n\t');
    fprintf(fileID, 'CTRL_EXT_EN <= ''0'';\n\t');
82  fprintf(fileID, 'MOD_CLK_EN <= ''0'';\n\t');
    fprintf(fileID, 'DSEL <= (others => ''0'');\n\t');
84  fprintf(fileID, 'WE <= ''0'';\n\t');
```

```
    fprintf(fileID, 'DIN <= (others => ''0'');\n\t');
86  fprintf(fileID, '-- hold reset state for 20 ns.\n\t');
    fprintf(fileID, 'wait for CLK_period*1.5;\n\t');
88  fprintf(fileID, 'nRESET <= ''1'';\n\t');
    fprintf(fileID, 'CTRL_EXT_EN <= ''0'';\n\t');
90  fprintf(fileID, 'DIN <= (others => ''0'');\n\t');
    fprintf(fileID, 'WE <= ''0'';\n\t');
92  fprintf(fileID, 'DSEL <= "000";\n\t');
    fprintf(fileID, '-- select write addr.\n\t');
94  fprintf(fileID, 'wait for CLK_period*2;\n\t');
    fprintf(fileID, 'CTRL_EXT_EN <= ''1'';\n\t');
96  fprintf(fileID, 'DIN <= (others => ''0'');\n\t');
    fprintf(fileID, 'WE <= ''1'';\n\t\n');
98  fprintf(fileID, "--start of sample input:\n\t");

100 for idxY = 1:nPoints
        for idxX = 1:nPoints
102         %typecast(int16(real(integerValues(idxY,idxX))), 'uint8');
            realbits=dec2bin(typecast(integerReal(idxY,idxX),'uint16'),16);
104         imagbits=dec2bin(typecast(integerImag(idxY,idxX),'uint16'),16);
            %imagbytes=typecast(int16(imag(integerValues(idxY,idxX))), '
    uint8');
106         fprintf(fileID, "--input X=%d Y=%d RE=0x%d IM=0x%d\n\t", idxY,
    idxX, integerReal(idxY,idxX), integerImag(idxY,idxX) );
            fprintf(fileID, "wait for CLK_period;\n\t");
108         fprintf(fileID, 'DSEL <= "001";\n\t');
            fprintf(fileID, 'DIN <= "%s";\n\t',dec2bin(START_ADDR_RD+((idxX
    -1)+(idxY-1)*15),10));
110         fprintf(fileID, 'wait for CLK_period;\n\t');
            fprintf(fileID, 'DIN <= "%s";\n\t', ['00' realbits(
    inputBitwidth-7:inputBitwidth)]);
112         fprintf(fileID, 'DSEL <= "011";\n\t');
            fprintf(fileID, 'wait for CLK_period;\n\t');
114         fprintf(fileID, 'DIN <= "%s";\n\t', ['00' realbits(
    inputBitwidth+1:16) dec2bin(0,inputBitwidth-8)]);
            fprintf(fileID, 'DSEL <= "010";\n\t');
116         fprintf(fileID, 'wait for CLK_period; \n\t');
            fprintf(fileID, 'DIN <= "%s";\n\t',['00' imagbits(inputBitwidth
    -7:inputBitwidth)]);
118         fprintf(fileID, 'DSEL <= "110";\n\t');
            fprintf(fileID, 'wait for CLK_period;\n\t');
120         fprintf(fileID, 'DIN <= "%s";\n\t',['00' imagbits(inputBitwidth
    +1:16) dec2bin(0,inputBitwidth-8)]);
            fprintf(fileID, 'DSEL <= "111";\n\t');
```

```
122             fprintf(fileID, 'wait for CLK_period;\n\t');
                fprintf(fileID, 'DSEL <= "101";\n\t');
124         end
    end
126     fprintf(fileID, "-- wait for one more cycle before disengaging (to let
    the ram save the data)\n\t");
    fprintf(fileID, "wait for CLK_period*2;\n\t");
128     fprintf(fileID, "CTRL_EXT_EN <= '0';\n\t");
    fprintf(fileID, "WE <= '0';\n\t");
130     fprintf(fileID, "--enable module clock\n\t");
    fprintf(fileID, "MOD_CLK_EN <= '1';\n\t");
132     fprintf(fileID, "--wait for calculation to finish\n\t");
    fprintf(fileID, "wait for CLK_period*%d;\n\t",(nPoints^2)*2+length(
    outputDftTerms)+10); %net processing time + pipeline delay + safety
    margin of 3 cycles
134     fprintf(fileID, "---start reading result(s)\n\t");
    fprintf(fileID, "CTRL_EXT_EN <= '1';\n\t");
136     fprintf(fileID, 'MOD_CLK_EN <= ''0'';\n\t');
    for outTerm = 1:numberOfOutputTerms
138         fprintf(fileID, 'DSEL <= "000";\n\t');
        fprintf(fileID, 'wait for CLK_period;\n\t');
140         fprintf(fileID, 'DSEL <= "001";\n\t');
        fprintf(fileID, 'DIN <= "%s"; -- write read address\n\t',dec2bin(
    START_ADDR_RD+(nPoints^2)+(outTerm-1),10));
142         fprintf(fileID, 'wait for CLK_period;\n\t');
        fprintf(fileID, 'DSEL <= "011";\n\t');
144         fprintf(fileID, 'wait for CLK_period; \n\t');
        fprintf(fileID, 'DSEL <= "010";\n\t');
146         fprintf(fileID, 'wait for CLK_period;\n\t');
        fprintf(fileID, 'DSEL <= "110";\n\t');
148         fprintf(fileID, 'wait for CLK_period;\n\t');
        fprintf(fileID, 'DSEL <= "111";\n\t');
150         fprintf(fileID, 'wait for CLK_period;\n\t');
    end
152     fprintf(fileID, "CTRL_EXT_EN <= '0';\n\t");
    fprintf(fileID, 'MOD_CLK_EN <= ''0'';\n\t');
154     fprintf(fileID, 'DSEL <= "000";\n\t');
    fprintf(fileID, 'DIN <= "0000000000";\n\t');
156     fprintf(fileID, 'wait for CLK_period*2;\n\t');


158

    fprintf(fileID, '--End simulation\n\t');
160     fprintf(fileID, 'wait;\n\t');
    fprintf(fileID, '\n\tend process;\nEND;\n');
```

```
    fprintf(fileID, '--results expected:\n');
end
for outTerm = 1:numberOfOutputTerms
    currentTerm=outputDftTerms(outTerm,:,:);
    decimalResulution=ceil(log10(2)*coefFracBitwidth)+2;
    integerCoef(:,:)=round(Transformcoefficients(currentTerm(1),currentTerm(2),:,:)*10^decimalResulution)/10^decimalResulution;
    integerCoef = round(integerCoef*2^coefFracBitwidth);
    accumulator=0+0i;
    for idxY = 1:nPoints
        for idxX = 1:nPoints
            accumulator=accuFunction(complex(integerReal(idxY,idxX),integerImag(idxY,idxX)),integerCoef(idxY,idxX),accumulator,coefFracBitwidth,accumulatorBitwidth);
        end
    end
    accuDividedReal=int32(real(accumulator)*round((outputDividerFactor)/(2^-outputDividerFracBitwidth)));
    accuDividedImag=int32(imag(accumulator)*round((outputDividerFactor)/(2^-outputDividerFracBitwidth)));
    accuDividedReal=bitshift(accuDividedReal,-outputDividerFracBitwidth);
    accuDividedImag=bitshift(accuDividedImag,-outputDividerFracBitwidth);
    if generateFile(8)
        fprintf(fileID, '--Coef: %d_%d result = Real: %d Imag: %d RAM CONTENT:0x%4.4x%4.4x\n',currentTerm(1),currentTerm(2),accuDividedReal, accuDividedImag, typecast(bitshift(int16(accuDividedReal),4),'uint16'), typecast(bitshift(int16(accuDividedImag),4),'uint16') );
    end
    disp(['genVhdlTestbench: Simulation Results for: ' num2str(currentTerm) ' Fixedpoint: ' num2str(accuDividedReal) ' ' num2str(accuDividedImag) 'i Floatingpoint: ' num2str(floatingpointResult(currentTerm(1),currentTerm(2))*outputDividerFactor)]);
end
if generateFile(8)
    fclose(fileID);
end
end % of function


function result=coefmultiply(sample, coefficient,fractionbits)
    mulresult=int64(sample)*int64(coefficient);
    result=bitshift(mulresult,-fractionbits);
    %disp(['sample= ' num2str(sample) ' * ' num2str(coefficient) ' = ' num2str(result)]);
```

```
194 end

196 function result=accuFunction(sample, coefficient, accumulator, fractionbits
    , accumulatorBitwidth)
        signrealcoef=sign(real(coefficient));
198     signimagcoef=sign(imag(coefficient));
        realcoef=abs(real(coefficient));
200     imagcoef=abs(imag(coefficient));

202     realXreal=coefmultiply(real(sample),realcoef,fractionbits);
        realXimag=coefmultiply(real(sample),imagcoef,fractionbits);
204     imagXreal=coefmultiply(imag(sample),realcoef,fractionbits);
        imagXimag=coefmultiply(imag(sample),imagcoef,fractionbits);
206
        if signrealcoef >= 0 && signimagcoef >= 0
208         result=accumulator+complex(double(realXreal-imagXimag),double(
    realXimag+imagXreal));
        elseif signrealcoef == -1 && signimagcoef >=0
210         result=accumulator+complex(double(-realXreal-imagXimag),double
    (realXimag-imagXreal));
        elseif signrealcoef >= 0 && signimagcoef == -1
212         result=accumulator+complex(double(realXreal+imagXimag),double
    (-realXimag+imagXreal));
        elseif  signrealcoef == -1 && signimagcoef == -1
214         result=accumulator+complex(double(-realXreal+imagXimag),double
    (-realXimag-imagXreal));
        else
216         result=accumulator+complex(double(realXreal-imagXimag),double(
    realXimag+imagXreal));
        end
218     if real(result)>2^(accumulatorBitwidth-1)-1 || real(result)< -2^(
    accumulatorBitwidth-1)
            disp('genVhdlTestbench: real accumulator overflow detected!
    CHECK YOUR BITWIDTH SETTINGS!');
220     end
        if imag(result)>2^(accumulatorBitwidth-1)-1 || imag(result)< -2^(
    accumulatorBitwidth-1)
222         disp('genVhdlTestbench: imag accumulator overflow detected!
    CHECK YOUR BITWIDTH SETTINGS!');
        end
224 end
```

Listing A.6: genVhdlTestbench.m

A.1.7 Quellcode: brute2snr.m

```
if ~exist('Transformcoefficients','var')
    disp('Error! run brute2dft first to generate transform coefficients');
    return;
end
rng('shuffle', 'twister'); %more randomness
disp(['brute2snr: SQNR estimator for 2D-DFT, Transform Size:' num2str(
    nPoints)]);
disp(['brute2snr: Input bitwidth: ', num2str(inputBitwidth) ', Fractional
    Bitwidth: ' num2str(coefFracBitwidth)]);
disp(['brute2snr: Accumulator bitwidth: ', num2str(accumulatorBitwidth) ',
    Divider fractional bitwidth: ' num2str(outputDividerFracBitwidth)]);
nSamples=20;
accuSQNR=zeros(nSamples,1);
theoreticalAccuSNR=zeros(nSamples,1);
SQNR=zeros(nSamples,1);
theoreticalSNR=zeros(nSamples,1);
for sample=1:nSamples
    integerReal=int16(randi(2^inputBitwidth-1,nPoints)-2^(inputBitwidth-1))
    ;
    integerImag=int16(randi(2^inputBitwidth-1,nPoints)-2^(inputBitwidth-1))
    ;
    fixedpointresult=zeros(nPoints);
    fixedpointaccu=zeros(nPoints);
    for coefY=1:nPoints
        for coefX=1:nPoints
            currentTerm=[coefY,coefX];
            decimalResulution=ceil(log10(2)*coefFracBitwidth)+2;
            integerCoef(:,:)=round(Transformcoefficients(currentTerm(1),
    currentTerm(2),:,:)*10^decimalResulution)/10^decimalResulution;
            integerCoef = round(integerCoef*2^coefFracBitwidth);
            accumulator=0+0i;
            for idxY = 1:nPoints
                for idxX = 1:nPoints
                    accumulator=accuFunction(complex(integerReal(idxY,idxX)
    ,integerImag(idxY,idxX)),integerCoef(idxY,idxX),accumulator,
    coefFracBitwidth,accumulatorBitwidth);
                end
            end
            accuDividedReal=int32(real(accumulator)*round((
    outputDividerFactor)/(2^-outputDividerFracBitwidth)));
            accuDividedImag=int32(imag(accumulator)*round((
    outputDividerFactor)/(2^-outputDividerFracBitwidth)));
```

```
            accuDividedReal=bitshift(accuDividedReal,-
    outputDividerFracBitwidth);
34          accuDividedImag=bitshift(accuDividedImag,-
    outputDividerFracBitwidth);
            fixedpointaccu(coefY, coefX) = accumulator;
36          if accuDividedReal>2^(outputBitwidth-1)-1
                disp(['brute2snr: real Output overflow for point: ' num2str
    (currentTerm) ' Output Value: ' num2str(accuDividedReal)]);
38              accuDividedReal=2^(outputBitwidth-1)-1;
            end
40          if accuDividedReal< -2^(outputBitwidth-1)
                disp(['brute2snr: real Output underflow for point: '
    num2str(currentTerm) ' Output Value: ' num2str(accuDividedReal)]);
42              accuDividedReal=-2^(outputBitwidth-1);
            end
44          if accuDividedImag>2^(outputBitwidth-1)-1
                disp(['brute2snr: imag Output overflow for point: ' num2str
    (currentTerm) ' Output Value: ' num2str(accuDividedImag)]);
46              accuDividedImag=2^(outputBitwidth-1)-1;
            end
48          if accuDividedImag< -2^(outputBitwidth-1)
                disp(['brute2snr: imag Output underflow for point: '
    num2str(currentTerm) ' Output Value: ' num2str(accuDividedImag)]);
50              accuDividedImag=-2^(outputBitwidth-1);
            end
52          fixedpointresult(coefY, coefX)=complex(accuDividedReal,
    accuDividedImag);

54      end
    end
56  floatingpointAccuResult=fft2(complex(integerReal,integerImag));
    floatingpointResult=fft2(complex(integerReal,integerImag))*
    outputDividerFactor;
58  errorAccuSum=0;
    floatAccuSum=0;
60  theoreticalAccuErrorsum=0;
    errorSum=0;
62  floatSum=0;
    theoreticalErrorsum=0;
64  for coefY=1:nPoints
        for coefX=1:nPoints
66          floatAccuSum=floatAccuSum+abs(floatingpointAccuResult(coefY,
    coefX))^2;
```

```
            errorAccuSum=errorAccuSum+(abs(floatingpointAccuResult(coefY,
    coefX))-abs(fixedpointaccu(coefY,coefX)))^2;
68          theoreticalAccuErrorsum=theoreticalAccuErrorsum+(abs(
    floatingpointAccuResult(coefY,coefX))-abs(round(floatingpointAccuResult
    (coefY,coefX))))^2;
            floatSum=floatSum+abs(floatingpointResult(coefY,coefX))^2;
70          errorSum=errorSum+(abs(floatingpointResult(coefY,coefX))-abs(
    fixedpointresult(coefY,coefX)))^2;
            theoreticalErrorsum=theoreticalErrorsum+(abs(
    floatingpointResult(coefY,coefX))-abs(round(floatingpointResult(coefY,
    coefX))))^2;
72        end
    end
74  accuSQNR(sample)=10*log10(floatAccuSum/errorAccuSum);
    theoreticalAccuSNR(sample)=10*log10(floatAccuSum/
    theoreticalAccuErrorsum);
76  SQNR(sample)=10*log10(floatSum/errorSum);
    theoreticalSNR(sample)=10*log10(floatSum/theoreticalErrorsum);
78 end
meanAccuSQNR=mean(accuSQNR);
80 meanTheoreticalAccuSNR=mean(theoreticalAccuSNR);
meanSQNR=mean(SQNR);
82 meanTheoreticalSNR=mean(theoreticalSNR);
disp(['brute2snr: Simulated  Accumulator SQNR: ' num2str(meanAccuSQNR,4) '
    dB, Floatingpoint SQNR: ' num2str(meanTheoreticalAccuSNR,4) 'dB']);
84 disp(['brute2snr: Simulated Output SQNR: ' num2str(meanSQNR,4) 'dB,
    Floatingpoint SQNR: ' num2str(meanTheoreticalSNR,4) 'dB']);

86 function result=coefmultiply(sample, coefficient,fractionbits)
    mulresult=int64(sample)*int64(coefficient);
88  result=bitshift(mulresult,-fractionbits);
    %disp(['sample= ' num2str(sample) ' * ' num2str(coefficient) ' = '
    num2str(result)]);
90 end

92 function result=accuFunction(sample, coefficient, accumulator, fractionbits
    , accumulatorBitwidth)
        signrealcoef=sign(real(coefficient));
94      signimagcoef=sign(imag(coefficient));
        realcoef=abs(real(coefficient));
96      imagcoef=abs(imag(coefficient));

98      realXreal=coefmultiply(real(sample),realcoef,fractionbits);
        realXimag=coefmultiply(real(sample),imagcoef,fractionbits);
```

```
100         imagXreal=coefmultiply(imag(sample),realcoef,fractionbits);
            imagXimag=coefmultiply(imag(sample),imagcoef,fractionbits);
102
            if signrealcoef >= 0 && signimagcoef >= 0
104             result=accumulator+complex(double(realXreal-imagXimag),double(
    realXimag+imagXreal));
            elseif signrealcoef == -1 && signimagcoef >=0
106             result=accumulator+complex(double(-realXreal-imagXimag),double
    (realXimag-imagXreal));
            elseif signrealcoef >= 0 && signimagcoef == -1
108             result=accumulator+complex(double(realXreal+imagXimag),double
    (-realXimag+imagXreal));
            elseif  signrealcoef == -1 && signimagcoef == -1
110             result=accumulator+complex(double(-realXreal+imagXimag),double
    (-realXimag-imagXreal));
            else
112             result=accumulator+complex(double(realXreal-imagXimag),double(
    realXimag+imagXreal));
            end
114         if real(result)>2^(accumulatorBitwidth-1)-1 || real(result)< -2^(
    accumulatorBitwidth-1)
                disp('brute2snr: real accumulator overflow detected! adjust
    Accumulator Bitwidth');
116         end
            if imag(result)>2^(accumulatorBitwidth-1)-1 || imag(result)< -2^(
    accumulatorBitwidth-1)
118             disp('brute2snr: imag accumulator overflow detected! adjust
    Accumulator Bitwidth');
            end
120 end
```

Listing A.7: brute2snr.m

A.2 Verilog Quellcode

A.2.1 Quellcode: accumulators.v

```
1 /********************/
  /*** Accumulators ***/
3 /********************/
  module dft_accumulator
5 #(parameter Opt_width=12,
```

```
      Inp_width=12,
      acc_width=14)
(
  input CLK, //clk input
  input nReset, //resets accumulator registers asynchronously
  input addSubReal, //subtract instead of add
  input addSubImag, //subtract instead of add
  input enable, //accumulate or hold value
  input signed [Inp_width-1:0] inReal,
  input signed [Inp_width-1:0] inImag,
  output signed [Opt_width-1:0] outReal,
  output signed [Opt_width-1:0] outImag
  );

//registers
reg signed [acc_width-1:0] accReal_reg;
reg signed [acc_width-1:0] accImag_reg;

//adder-subtractors
wire signed [acc_width-1:0] RealAdder = (addSubReal) ? accReal_reg-inReal :
    accReal_reg+inReal;
wire signed [acc_width-1:0] ImagAdder = (addSubImag) ? accImag_reg-inImag :
    accImag_reg+inImag;

generate
  if(Opt_width==acc_width) //if output = acumulator width > DONT SATURATE
  begin
    //output assignment
    assign outReal = accReal_reg;
    assign outImag = accImag_reg;
  end
  else
  begin
    reg [Opt_width-1:0] accRealSat, accImagSat;
    //output assignment
    assign outReal = accRealSat;
    assign outImag = accImagSat;
    always @* //Saturation unit
    begin
       if ((accReal_reg[acc_width-1:Opt_width-1]=={acc_width-Opt_width+1{1'
    b0}}) ||
         (accReal_reg[acc_width-1:Opt_width-1]=={acc_width-Opt_width+1{1'b1
    }}))
        accRealSat = accReal_reg[Opt_width-1:0];
```

```
        else if (accReal_reg[acc_width-1])  //underflow
47       accRealSat = {1'b1,{Opt_width-1{1'b0}}};
        else //overflow
49       accRealSat = {1'b0,{Opt_width-1{1'b1}}};
      if ((accImag_reg[acc_width-1:Opt_width-1]=={acc_width-Opt_width+1{1'
    b0}}) ||
51        (accImag_reg[acc_width-1:Opt_width-1]=={acc_width-Opt_width+1{1'b1
    }}))
        accImagSat = accImag_reg[Opt_width-1:0];
53      else if (accImag_reg[acc_width-1])  //underflow
        accImagSat = {1'b1,{Opt_width-1{1'b0}}};
55      else //overflow
        accImagSat = {1'b0,{Opt_width-1{1'b1}}};
57    end
  end
59 endgenerate

61 always@(posedge CLK, negedge nReset) //D-FF async active low reset, active
    high enable
 begin
63   if(!nReset)
   begin
65     accReal_reg <= #1 {acc_width{1'b0}};
     accImag_reg <= #1 {acc_width{1'b0}};
67   end
   else
69   begin
     if (enable)
71     begin
       accReal_reg <= #1 RealAdder;
73       accImag_reg <= #1 ImagAdder;
     end
75     else //dont accumulate just hold
     begin
77       accReal_reg <= accReal_reg;
       accImag_reg <= accImag_reg;
79     end
   end
81 end
 endmodule
```

Listing A.8: accumulators.v

A.3 VHDL Quellcode

A.3.1 Quellcode: toplevel.vhd

```
--------------------------------------------------------------------------------

-- Entitiy : TOPLEVEL
--------------------------------------------------------------------------------

-- Copyright 2018
-- Filename          : toplevel.vhd
-- Creation date     : 2018-04-26
-- Authors(s)        : Jannes Helck, Martin Willimczik
-- Version           : 1.10
-- Description       : Top level
--
--------------------------------------------------------------------------------

-- File History :
-- Date          Version      Author       Comment
-- 2018-04-26    1.00         J.Helck      Creation of file
-- 2019-31-01    1.10      M.Willimczik    Added Dummy Modules, Added symb.
    delay,
--                      deleted Read Enable
--
--------------------------------------------------------------------------------


library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

use work.BRAM_PKG.all;
use work.MEMORY_CONTROL_PKG.all;
use work.MODULE_CONTROL_PKG.all;
use work.DFT_MODULE_PKG.all;
use work.DUMMY_MODULE_PKG.all;

entity TOPLEVEL is
port (
    CLK             : in  std_logic;
```

```
    nRESET          : in  std_logic;
    CTRL_EXT_EN     : in  std_logic;
    MOD_SEL_EXT_EN  : in  std_logic;
    MOD_OUT         : out std_logic_vector(2 downto 0);
    MOD_RDY         : out std_logic;
    MOD_CLK_EN      : in  std_logic;
    DSEL            : in  std_logic_vector(2 downto 0);
    WE              : in  std_logic;
    DIN             : in  std_logic_vector(9 downto 0);
    DOUT            : out std_logic_vector(7 downto 0)

);
end TOPLEVEL;

architecture ARCH of TOPLEVEL is

signal RAM_EN       : std_logic;
signal MOD_SEL      : std_logic_vector(2 downto 0);
signal MOD_CLK      : std_logic;
signal RDY          : std_logic;
signal ADDR_BUS     : std_logic_vector(9 downto 0);
signal DIN_BUS      : std_logic_vector(31 downto 0);
signal DOUT_BUS     : std_logic_vector(31 downto 0);

signal WE_RAM       : std_logic;
signal RE_RAM       : std_logic;

signal WE_C1        : std_logic;
signal WE_C4        : std_logic;
signal WE_C5        : std_logic;
signal WE_C6        : std_logic;
signal WE_C7        : std_logic;


begin

RAM_EN  <= '1';
MOD_OUT <= MOD_SEL;
MOD_RDY <= RDY;
MOD_CLK <= CLK when MOD_CLK_EN = '1' else '0';
--MOD_CLK <= CLK ;
--MOD_CLK <= CLK after 500ps when MOD_CLK_EN = '1' else '0' after 500ps;

WE_RAM <= WE_C1 or WE_C4 or WE_C5 or WE_C6 or WE_C7;
```

```
RE_RAM <= '1';

-- Instantiation of Component : MEMORY_CONTROL
C1: MEMORY_CONTROL port map (
    CLK             => CLK,
    nRESET          => nRESET,
    CTRL_EXT_EN     => CTRL_EXT_EN,
    MOD_SEL_EXT_EN  => MOD_SEL_EXT_EN,
    MOD_SEL         => MOD_SEL,
    DIN             => DIN,
    DOUT            => DOUT,
    DSEL            => DSEL,
    WE_IN           => WE,
    WE              => WE_C1,
    RE              => open,
    ADDR_BUS        => ADDR_BUS,
    DIN_BUS         => DIN_BUS,
    DOUT_BUS        => DOUT_BUS
);

-- Instantiation of Component: BRAM
C2: BRAM port map (
    CLK             => CLK,
    nRESET          => nRESET,
    EN              => RAM_EN,
    WE              => WE_RAM,
    RE              => RE_RAM,
    ADDR            => ADDR_BUS,
    DIN             => DIN_BUS,
    DOUT            => DOUT_BUS
);

-- Instantiation of Component: MODULE_CONTROL
C3: MODULE_CONTROL port map (
    CLK             => MOD_CLK,
    nRESET          => nRESET,
    MOD_SEL_EXT_EN  => MOD_SEL_EXT_EN,
    MOD_SEL         => MOD_SEL,
    RDY             => RDY
);

-- Instantiation of Component: DFT_MODULE
C4: DFT_MODULE port map (
    CLK             => MOD_CLK,
```

```
        nRESET              => nRESET,
120     CTRL_EXT_EN         => CTRL_EXT_EN,
        MOD_SEL             => MOD_SEL,
122     RDY                 => RDY,
        WE                  => WE_C4,
124     RE                  => open,
        ADDR_BUS            => ADDR_BUS,
126     DIN_BUS             => DIN_BUS,
        DOUT_BUS            => DOUT_BUS
128 );

130 -- Instantiation of Component: DUMMY_MODULE
    C5: DUMMY_MODULE
132 generic map (MOD_NR    => "010")
    port map (
134     CLK                 => MOD_CLK,
        nRESET              => nRESET,
136     CTRL_EXT_EN         => CTRL_EXT_EN,
        MOD_SEL             => MOD_SEL,
138     RDY                 => RDY,
        WE                  => WE_C5,
140     RE                  => open,
        ADDR_BUS            => ADDR_BUS,
142     DIN_BUS             => DIN_BUS,
        DOUT_BUS            => DOUT_BUS
144 );
    -- Instantiation of Component: DUMMY_MODULE
146 C6: DUMMY_MODULE
    generic map (MOD_NR    => "011")
148 port map (
        CLK                 => MOD_CLK,
150     nRESET              => nRESET,
        CTRL_EXT_EN         => CTRL_EXT_EN,
152     MOD_SEL             => MOD_SEL,
        RDY                 => RDY,
154     WE                  => WE_C6,
        RE                  => open,
156     ADDR_BUS            => ADDR_BUS,
        DIN_BUS             => DIN_BUS,
158     DOUT_BUS            => DOUT_BUS
    );
160 -- Instantiation of Component: DUMMY_MODULE
    C7: DUMMY_MODULE
162 generic map (MOD_NR    => "100")
```

```
port map (
    CLK             => MOD_CLK,
    nRESET          => nRESET,
    CTRL_EXT_EN     => CTRL_EXT_EN,
    MOD_SEL         => MOD_SEL,
    RDY             => RDY,
    WE              => WE_C7,
    RE              => open,
    ADDR_BUS        => ADDR_BUS,
    DIN_BUS         => DIN_BUS,
    DOUT_BUS        => DOUT_BUS
);


end ARCH;
```

Listing A.9: toplevel.vhd

A.3.2 Quellcode: bram.vhd

```
--
-------------------------------------------------------------------------------

-- Entitiy : BRAM
--
-------------------------------------------------------------------------------

-- Copyright 2018
-- Filename          : bram.vhd
-- Creation date     : 2018-05-03
-- Authors(s)        : Jannes Helck, Martin Willimczik
-- Version           : 1.10
-- Description       : Block RAM for transferring data between the
--                     signal processing modules on FPGA
--
-------------------------------------------------------------------------------

-- File History :
-- Date          Version      Author       Comment
-- 2018-05-03    1.00         J.Helck      Creation of file
-- 2019-31-01    1.10      M.Willimczik    fixed inferrence, added symb delay
--
-------------------------------------------------------------------------------
```

```
17
-------------------------------------- Package
    ---------------------------------------
19
library ieee;
21 use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
23
package BRAM_PKG is
25
component BRAM
27 port (
    CLK    : in  std_logic;
29  nRESET: in  std_logic;
    EN     : in  std_logic;
31  WE     : in  std_logic;
    RE     : in  std_logic;
33  ADDR   : in  std_logic_vector(9 downto 0);
    DIN    : in  std_logic_vector(31 downto 0);
35  DOUT   : out std_logic_vector(31 downto 0)
);
37 end component;

39 end BRAM_PKG;

41 ---------------------------------- end Package
    ---------------------------------------

43 library ieee;
use ieee.std_logic_1164.all;
45 use ieee.numeric_std.all;

47 entity BRAM is
port (
49  CLK    : in  std_logic;
    nRESET: in  std_logic;
51  EN     : in  std_logic;
    WE     : in  std_logic;
53  RE     : in  std_logic;
    ADDR   : in  std_logic_vector(9 downto 0);
55  DIN    : in  std_logic_vector(31 downto 0);
    DOUT   : out std_logic_vector(31 downto 0)
57 );
```

```
end BRAM;

architecture ARCH of BRAM is

 type RAM_TYPE is array (1023 downto 0) of std_logic_vector(31 downto 0);
 signal BRAM : RAM_TYPE:= (others => "00000000000000000000000000000000");

 attribute RAM_STYLE : string;
 attribute RAM_STYLE of BRAM : signal is "block";

begin

RAM_P: process(CLK)
begin
if CLK'event and CLK = '1' then
    if EN = '1' then
        if WE = '1' then
            BRAM(to_integer(unsigned(ADDR))) <= DIN;
      end if;
      DOUT <= BRAM(to_integer(unsigned(ADDR))) after 500ps;
    end if;
end if;
end process;

end ARCH;
```

Listing A.10: bram.vhd

A.3.3 Quellcode: module_control.vhd

```
--
    ------------------------------------------------------------------------------

-- Entitiy : MODULE_CONTROL
--
    ------------------------------------------------------------------------------

-- Copyright 2018
-- Filename          : module_control.vhd
-- Creation date     : 2018-05-12
-- Authors(s)        : Jannes Helck
-- Version           : 1.00
-- Description       : Control Unit for sequential activation of the
```

```vhdl
--                          signal processing modules
-- ------------------------------------------------------------------------------------------

-- File History :
-- Date          Version      Author       Comment
-- 2018-05-12    1.00         J.Helck      Creation of file
-- ------------------------------------------------------------------------------------------

---------------------------------------- Package ----------------------------------------

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

package MODULE_CONTROL_PKG is

component MODULE_CONTROL
port (
    CLK             : in  std_logic;
    nRESET          : in  std_logic;
    MOD_SEL_EXT_EN  : in  std_logic;
    MOD_SEL         : out std_logic_vector(2 downto 0);
    RDY             : in  std_logic
);
end component;

end MODULE_CONTROL_PKG;

------------------------------------- end Package -------------------------------------

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity MODULE_CONTROL is
port (
    CLK             : in  std_logic;
    nRESET          : in  std_logic;
    MOD_SEL_EXT_EN  : in  std_logic;
```

```
48      MOD_SEL          : out std_logic_vector(2 downto 0);
        RDY              : in  std_logic
50  );
    end MODULE_CONTROL;
52
    architecture ARCH of MODULE_CONTROL is
54
    type STATES is (M1, M2, M3, M4);
56  signal STATE, NEXT_STATE: STATES;
    signal MOD_SEL_INT : std_logic_vector(2 downto 0);
58
    begin
60
    -- State machine
62  STATE_MEM: process(CLK, nRESET)
    begin
64      if nRESET = '0' then STATE <= M1;
        elsif CLK  = '1' and CLK'event then
66              STATE <= NEXT_STATE;
        end if;
68  end process;

70  TRANSITION_LOGIC: process(RDY, STATE)
    begin
72      NEXT_STATE <= STATE;
        case STATE is
74
            when M1 => MOD_SEL_INT <= "001";
76                     if RDY = '1' then NEXT_STATE <= M2;
                       end if;
78                     NEXT_STATE <= M1;

80          when M2 => MOD_SEL_INT <= "010";
                       if RDY = '1' then NEXT_STATE <= M3;
82                     end if;
                       NEXT_STATE <= M2;
84
            when M3 => MOD_SEL_INT <= "011";
86                     if RDY = '1' then NEXT_STATE <= M4;
                       end if;
88                     NEXT_STATE <= M3;

90          when M4 => MOD_SEL_INT <= "100";
                       if RDY = '1' then NEXT_STATE <= M1;
```

```
92                      end if;
                        NEXT_STATE <= M4;
94
    end case;
96 end process;

98 -- Three state bus drivers
MOD_SEL <= MOD_SEL_INT when MOD_SEL_EXT_EN = '0' else (others=>'Z');
100
end ARCH;
```

Listing A.11: module_control.vhd

A.3.4 Quellcode: dummy_module.vhd

```
1 --
    ------------------------------------------------------------------------------

-- Entitiy : DUMMY_MODULE
3 --
    ------------------------------------------------------------------------------

-- Copyright 2019
5 -- Filename          : dummy_module.vhd
-- Creation date      : 31.01.2019
7 -- Authors(s)        : Martin Willimczik
-- Version            : 1.00
9 -- Description       : Dummy module for the unused module slots in the
    system,
--              makes sure the address and data-in line stay defined when
    unused module slots are selected
11 --
    ------------------------------------------------------------------------------

library ieee;
13 use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
15
package DUMMY_MODULE_PKG is
17
component DUMMY_MODULE
19 generic(    MOD_NR          : unsigned := "010"     -- module number
);
```

```
21 port(
       CLK               : in  std_logic;
23     nRESET            : in  std_logic;
       CTRL_EXT_EN       : in  std_logic;
25     MOD_SEL           : in  std_logic_vector(2 downto 0);
       RDY               : out std_logic;
27     WE                : out std_logic;
       RE                : out std_logic;
29     ADDR_BUS          : out std_logic_vector(9 downto 0);
       DIN_BUS           : out std_logic_vector(31 downto 0);
31     DOUT_BUS          : in  std_logic_vector(31 downto 0)
);
33 end component;

35 end DUMMY_MODULE_PKG;

37 ---------------------------------- end Package
       ------------------------------------

39 library ieee;
use ieee.std_logic_1164.all;
41 use ieee.numeric_std.all;

43 entity DUMMY_MODULE is
generic(
45     MOD_NR            : unsigned := "010"    -- module number
);
47
port(
49     CLK               : in  std_logic;
       nRESET            : in  std_logic;
51     CTRL_EXT_EN       : in  std_logic;
       MOD_SEL           : in  std_logic_vector(2 downto 0);
53     RDY               : out std_logic;
       WE                : out std_logic;
55     RE                : out std_logic;
       ADDR_BUS          : out std_logic_vector(9 downto 0);
57     DIN_BUS           : out std_logic_vector(31 downto 0);
       DOUT_BUS          : in  std_logic_vector(31 downto 0)
59 );
end DUMMY_MODULE;
61
architecture ARCH of DUMMY_MODULE is
63
```

```
signal ADDR     : std_logic_vector(9 downto 0);
signal DIN      : std_logic_vector(31 downto 0);
signal RDY_INT  : std_logic;
begin

RE <= '0';
WE <= '0';
RDY_INT  <= '1';
ADDR <= "0000000000";
DIN  <= "00000000000000000000000000000000";

ADDR_BUS <= ADDR when MOD_SEL = std_logic_vector(MOD_NR) and CTRL_EXT_EN =
    '0' else (others=>'Z');
DIN_BUS <= DIN when MOD_SEL = std_logic_vector(MOD_NR) and CTRL_EXT_EN =
    '0' else (others=>'Z');
RDY <= RDY_INT when MOD_SEL = std_logic_vector(MOD_NR) else 'Z';

end ARCH;
```

Listing A.12: dummy_module.vhd

Erklärung zur selbstständigen Bearbeitung einer Abschlussarbeit

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Ort	Datum	Unterschrift im Original