Brandenburgische
Technische Universität
Cottbus - Senftenberg

BRANDENBURGISCHE TECHNISCHE UNIVERSITÄT COTTBUS - SENFTENBERG

Fakultät 3 für Maschinenbau, Elektrotechnik und Wirtschaftsingenieurwesen

Lehrstuhl Kommunikationstechnik

Prof. Dr.-Ing. habil. Matthias Wolff

Bachelorarbeit

LTI Vorlesungsexperiment: Faltungshall für verschiedene Impulsantworten

– LTI Lecture experiment: echo convolution with different impulse responses –

vorgelegt von:

Student:	Felix Hahn
Studiengang:	Elektrotechnik
Matrikelnummer:	2822172
Geburtsdatum:	22.03.1989
Adresse:	Sielower Landstraße 35, 03044 Cottbus
E-Mail:	Felix.Hahn@b-tu.de
Betreuer:	Dr.-Ing. Ronald Römer
	Dipl. Ing. Christian Richter

Cottbus, den 29.03.2016

Aufgabenstellung

Für die Vorlesung „Grundzüge der Kommunikationstechnik“ wird ein Demonstrator für lineare zeitinvariante Systeme (LTI) in Form eines Hardware-Moduls verwendet. Der Demonstrator soll angelegte Audiosignale in Echtzeit verarbeiten und am Ausgang als virtuellen Raumklangeffekt ausgeben. Hauptgegenstand der Arbeit ist die Implementation der Impulsantworten für die akustischen Umgebungen Wohnzimmer, Theater und Kirche.

Kurzfassung

Die folgende Arbeit beschäftigt sich mit der Implementation der Faltungsoperation auf einem DSP-Kit. Das Kit soll in der Lage sein, ein Eingangssignal einzulesen und die Faltungssumme zu berechnen, um sie anschließend auszugeben.

Dazu muss zuerst ein Überblick der theoretischen Grundlagen erarbeitet werden. Um eine Auswahl zu treffen, welche Methode zur Realisierung der Faltung benutzt werden soll, werden die Methoden genauer beschrieben, deren Vor- und Nachteile aufgezeigt, miteinander verglichen und die effizienteste Methode ausgewählt.

Daraufhin wird die verwendete Hardware beschrieben. Anschließend wird der ausgewählte Algorithmus und seine Implementation näher erläutert. Durch die Eigenschaften des Algorithmus kommt es zu unvermeidbaren Problemen. Eine Lösung dieser Probleme wird in Abschnitt 5 dargestellt.

Im Anhang sind die Programmablaufpläne dazu aufgeführt.

Der Verfasser/Die Verfasserin erklärt, dass er/sie die vorliegende Arbeit selbständig, ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt hat. Die aus fremden Quellen (einschließlich elektronischer Quellen) direkt oder indirekt übernommenen Gedanken sind ausnahmslos als solche kenntlich gemacht. Die Arbeit ist in gleicher oder ähnlicher Form oder auszugsweise im Rahmen einer anderen Prüfung noch nicht vorgelegt worden.

Ort, Datum Unterschrift

Inhaltsverzeichnis

ABBILDUNGSVERZEICHNIS 3

ABKÜRZUNGSVERZEICHNIS 4

1 AUSFÜHRLICHE DARSTELLUNG DES PROBLEMS. 5

2 GRUNDLAGEN 6

2.1 DISKRETE SIGNALE 6

2.2 DAS LTI-SYSTEM 7

2.3 METHODEN ZUR IMPLEMENTATION 8

2.3.1 ZEITBEREICHSFALTUNG: FIR-FILTER 9

2.3.2 ZEITBEREICHSFALTUNG: IIR-FILTER 10

2.3.3 FREQUENZBEREICHSFALTUNG: SCHNELLE FALTUNG 11

3 HARDWAREBESCHREIBUNG 16

3.1 C6713 16

3.2 TLV320AIC23 STEREOCODEC 17

4 PROGRAMMABLAUF UND IMPLEMENTATION 19

4.1 DIE ZERLEGUNG DER IMPULSANTWORT 19

4.2 ZERLEGUNG UND VERSCHACHTELUNG DER EINGANGSFOLGE 20

4.2.1 IMPLEMENTIERUNG IN MATLAB 20

4.2.2 IMPLEMENTIERUNG IN C 21

4.3 DIE INTERRUPT-SERVICE-ROUTINE AIC_RX() 22

4.4 (INVERSE) FAST-FOURIER-TRANSFORMATION 23

4.5 ELEMENTWEISE MULTIPLIKATION IM SPEKTRALBEREICH 24

4.6 BESTIMMUNG UND NEUKOMPOSITION DER TEILFALTUNG 24

5 BEWERTUNG DES ALGORITHMUS, PROBLEME, VERBESSERUNGSMÖGLICHKEITEN 27

5.1 „ZERO-DELAY CONVOLUTION ALGORITHMUS“ 27

6 ZUSAMMENFASSUNG, MOMENTANER STAND UND AUSSICHTEN 29

7 LITERATURVERZEICHNIS 29

A ANHANG 31

ABBILDUNGSVERZEICHNIS

Abbildung 2.1-1 Blockschaltbild der Nyquist-Abtastung 7

Abbildung 2.3-1 Signalflussdiagramm des FIR-Filters 9

Abbildung 2.3-2 Signalflussdiagramm eines IIR-Filters 10

Abbildung 2.3-3Zusammenhang zwischen Zeit- und Frequenzbereich 11

Abbildung 2.3-4 Darstellung des Overlap Add Verfahrens 12

Abbildung 2.3-5 Ablaufschema einer schnellen Faltung mit P=4 15

Abbildung 3.1-1 schematische Darstellung des inneren Aufbaus des Evaluationsboards 17

Abbildung 3.2-1 Schematische Darstellung des Stereocodecs 18

Abbildung 4.1-1 Aufteilung der Impulsantworten in P Blöcke 20

Abbildung 4.1-2 while-schleife zur Partitionierung der Impulsantwort 20

Abbildung 4.2-1 Schema für die Organisation des Inputbuffers 22

Abbildung 4.6-1 Overlapp Add Verfahren für die Teilfaltung des ersten Eingangsblocks 25

Abbildung 4.6-2 Overlapp Add der Teilfaltung 25

Abbildung 4.6-3 Overlap Add der Zwischenlösung 26

Abbildung 5.1-1 Zerlegung der Impulsantwort nach Gardner 28

ABKÜRZUNGSVERZEICHNIS

(I)FFT	(inverse) Fast Fourier-Transformation
(I)DFT	(inverse) Diskrete Fourier-Transformation
(I)DTFT	(inverse) Discrete Time Fourier-Transformation
DSP	Digitaler Signalprozessor
z. B.	zum Beispiel
IDE	integrierte Entwicklungsumgebung
PC	Personal Computer
ISR	Interrupt Service Routine
LTI	linear time-invariant
LZI	linear zeitinvariant
USB	Universal Serial Bus
McBSP	Multi channel Buffered Serial Port
MAC	Multiply Accumulate
FIR	finite impulse response
IIR	infinite impulse response
TI	Texas Instrument™
EMIF	enhanced memory interface
SDRAM	synchronous dynamic random access memory
VLIW	very long instruction word

1 AUSFÜHRLICHE DARSTELLUNG DES PROBLEMS.

Der Lehrstuhl „Kommunikationstechnik“ besitzt ein Vorführungsexperiment, um StudentInnen anhand eines Beispiels die praktische Realisierung der vermittelten Vorlesungsinhalte zu demonstrieren. Die sogenannte „Black-Box“ wird von außen mit einem Eingangssignal gespeist. Mit den Kippschaltern wird die gewünschte Arbeitsweise eingestellt. Anschließend wird am Ausgang mit einem Oszilloskop oder einem Lautsprecher das berechnete Signal dem Auditorium präsentiert.

Eine der grundlegenden Rechenarten, die für die Signal- und Informationsverarbeitung eine wichtige Rolle spielen, ist die Faltungsoperation. Ihr Ergebnis ist eine Verknüpfung des Eingangssignals mit dem System. Wird z. B. ein Sprachsignal mit einer Tiefpassfunktion (Impulsantwort) gefaltet, wird das Sprachsignal so modifiziert, dass nur Frequenzen bis zur Grenzfrequenz das System passieren. Frequenzen oberhalb der Grenzfrequenz dagegen verschwinden. Das Ausgangssignal wird Systemantwort genannt.

Eine weitere Anwendungsmöglichkeit der Faltung ist die Erzeugung eines Echos. Dafür wird ein trockenes Eingangssignal mit den akustischen Eigenschaften eines Raums gefaltet. Die Systemantwort vermittelt dem Zuhörer das Gefühl, er würde in einem anderen Raum stehen und die Schallquelle befände sich auch in diesem Raum.

Das Ziel dieser Arbeit ist, den Demonstrator so zu modifizieren, dass die Faltung mit langen Impulsantworten in Echtzeit berechnet und ausgegeben wird. Um dies zu erreichen, wird im Folgenden der mathematische und technische Hintergrund thematisiert. Dann werden die Methoden zur technischen Realisierung und deren Vor- und Nachteile aufgezeigt. Anschließend wird der verwendete Algorithmus beschrieben und seine Grenzen dargestellt.

Verbaut wurde ein TMS320C6713 DSK Board, welches im Besitz eines C6713 Signalprozessors ist. Dieser Hochleistungs-DSP ist für die schnelle Verarbeitung von Daten ausgelegt. Ein weiterer Vorteil des C6713 ist die Fähigkeit sowohl mit Fix- als auch Gleitkommaarithmetik zu rechnen.

2 GRUNDLAGEN

Bei der menschlichen Kommunikation wird mit Hilfe der Sprachorgane (Sender) eine Information in Form eines analogen Signals (Sprachlaute) auf das Trägermedium Luft moduliert. Am Ende des Übertragungskanals nimmt der Empfänger mit den Organen zur auditiven Wahrnehmung das Signal auf. Werden nun originales und empfangenes Signal verglichen, kann festgestellt werden, dass das Originalsignal modifiziert wurde. Wird die Messung nun mit unveränderten Parametern wiederholt, zeigt sich bei der Auswertung der Signale, dass der Empfänger die gleichen Signale wie bei den voran gegangenen Messungen unabhängig vom Messzeitpunkt misst. Diese Eigenschaft des Systems wird Zeitinvarianz genannt. Die Art der Modifikation ist ausschließlich von der Umgebung des Messplatzes abhängig (die Übertragungsfunktion des Raumes).

Die zweite wichtige Eigenschaft der Informationsübertragung durch Sprache ist die Linearität. Variiert der Sender die Magnitude des Signals, wird auch beim Empfänger eine proportionale Variation der Magnitude wahrgenommen. Erhöht der Sprecher also die Lautstärke, nimmt der Hörer gleich die Sprache lauter war. Des Weiteren kann das *Superpositionsprinzip* angewandt werden. Treten weitere Schallereignisse im Raum auf, wird die Überlagerung dieser wahrgenommen. Wenn das System diese Eigenschaft aufweist, ist es *linear* [Kam89].

2.1 Diskrete Signale

Die Sprache des Menschen ist ein gutes Beispiel für ein analoges Signal x(t). Wird dieses *kontinuierliche* Signal an äquidistanten Zeitpunkten abgetastet, wird von einem *zeitdiskreten* Signal $x(k) := x(kT_A)$ gesprochen, wobei gilt: $T_A=1/f_A$. T_A wird als Abtastintervall und f_A als Abtastrate bezeichnet. Um das analoge Originalsignal aus den Abtastwerten zu rekonstruieren, muss das Abtasttheorem von Shannon eingehalten werden [Kam89].

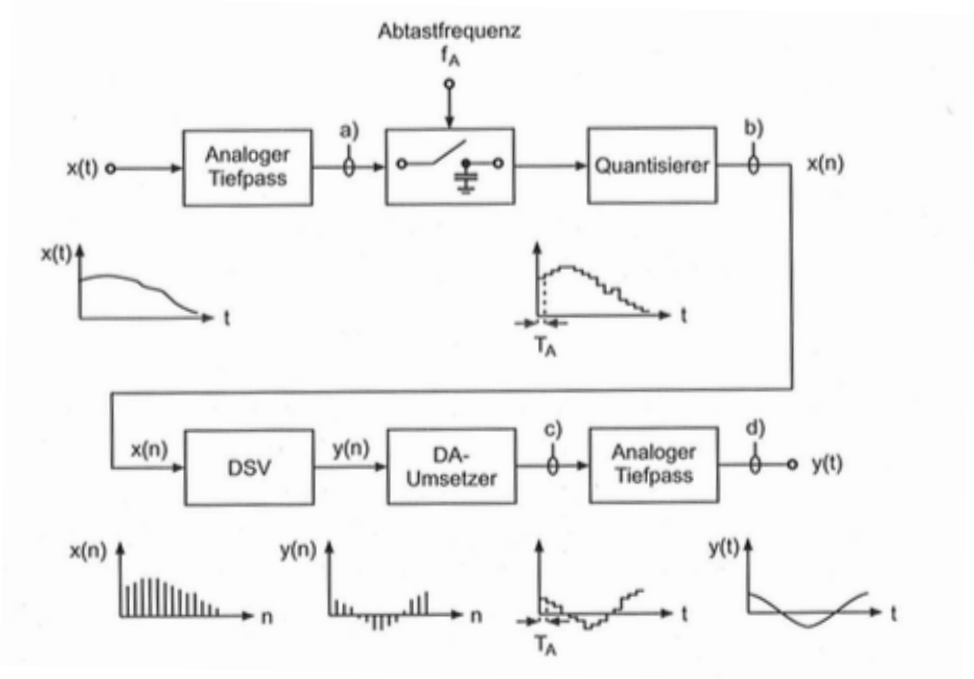


Abbildung 2.1-1 Blockschaltbild der Nyquist-Abtastung
Bild:[Zöl]

Eine Rücktransformation des werte- und zeitdiskreten Signals, in ein analoges Signal, geschieht durch das in Abbildung 2.1-1 abgebildete Schema. Nach der digitalen Signalverarbeitung wird das diskrete Signal mittels einer *Sample&Hold*-Schaltung zu einem zeitkontinuierlichen Signal transformiert. Damit die Bandbreite des analogen Signals beschränkt wird und die Stufen der *Sample&Hold*-Funktion geglättet werden können, muss anschließend ein analoger Tiefpassfilter geschaltet werden.

2.2 Das LTI-System

Ein LTI-System (linear time-invariant system) oder auch LZI-System erfüllt zu jeder Zeit die Bedingungen der Linearität und der Zeitinvarianz. *Linear* ist das System, wenn das *Superpositionsprinzip* gilt. Wenn man das System mit einer beliebigen Eingangsfolge $x_i(k)$ erregt und dann die Systemantwort $y_i(k)$ erhält, so muss die Linearkombination der Eingangssignale $x_i(k)$ stets die Linearkombination der Systemantworten liefern.

Wenn also gilt: $y_i(k) = H\{x_i(k)\}$, gilt bei linearen Systemen ebenso:

$$y(k) = H\left\{\sum_{i=1}^{n} a_i x_i(k)\right\} = \sum_{i=1}^{n} a_i H\{x_i(k)\} = \sum_{i=1}^{n} a_i y_i$$

Bei *Zeitinvarianz* hängt die Systemantwort nicht vom Zeitpunkt der Erregung ab [Kam]. Wird das Eingangssignal um k_0 verschoben, liefert das LTI-System eine auch um k_0 verschobene Systemantwort. Es gilt:

$$\mathrm{y(k - k_0)} = H\{x_i(k - k_0)\}$$

Diese Systeme haben die Eigenschaft, dass durch eine Anregung des Systems mit einer beliebigen Eingangsfolge x(k) die Ausgangsfolge y(k) ausgegeben wird. Es gilt die Beziehung:

$$y(k) := x(k) * h(k) = \sum_{i=-\infty}^{+\infty} x(i)h(k-i) = \sum_{i=-\infty}^{+\infty} x(k-i)h(i)$$

Wobei h(k) die Impulsantwort des Systems wiederspiegelt. Das Symbol „$*$“ wird dabei als Faltungsoperator bezeichnet.

Durch diese Eigenschaften des LTI-Systems können die Funktionen im Zeitbereich mit Hilfe der Fourier-Transformation auch im Frequenzbereich betrachtet werden. Es gilt die Beziehung

$$y(n) = x(n) * h(n) \circ\!\!-\!\!\bullet\; Y(k) = X(k) \cdot H(k)$$

Eine genaue Herleitung dieser Zusammenhänge wird in [Kam] aufgezeigt.

2.3 Methoden zur Implementation

Um die Faltungsoperation zu realisieren, gibt es, wie im Abschnitt 2.2 beschrieben, mehrere Herangehensweisen. Im Folgenden wird auf die Methoden eingegangen und deren Vor- und Nachteile betrachtet. Nach einer Gegenüberstellung erfolgt anschließend die Auswahl der verwendeten Methode.

2.3.1 Zeitbereichsfaltung: FIR-Filter

Ein Filter mit endlicher Impulsantwort (*finite impulse response)* wird auch als FIR-Filter bezeichnet. In Abbildung 2.3-1 ist die Struktur eines FIR-Filters abgebildet. Die Koeffizienten *h(n)* symbolisieren die einzelnen Werte der Impulsantwort des Filters. Pro Takt wird jeder Eingangswert um z^{-1} verzögert und mit dem nächsten Element der Impulsantwort multipliziert. Anschließend erfolgt die Addition aller so entstandenen Elemente. Damit entspricht das Ergebnis des FIR-Filters dem der diskreten Faltung.

$$y(k) = \sum_{i=0}^{m} x(i) \cdot h(k-i)$$

Vorteil dieser Methode ist, dass die Rechenwerke moderner Signalprozessoren auf genau diese Operationen angepasst sind. Solche Bausteine bestehen meist aus einem Multiplizierer in Verbindung mit einem Akkumulator und können in einem Befehlszyklus beide Rechenoperationen hintereinander ausführen (MAC-Befehl) [Kam].

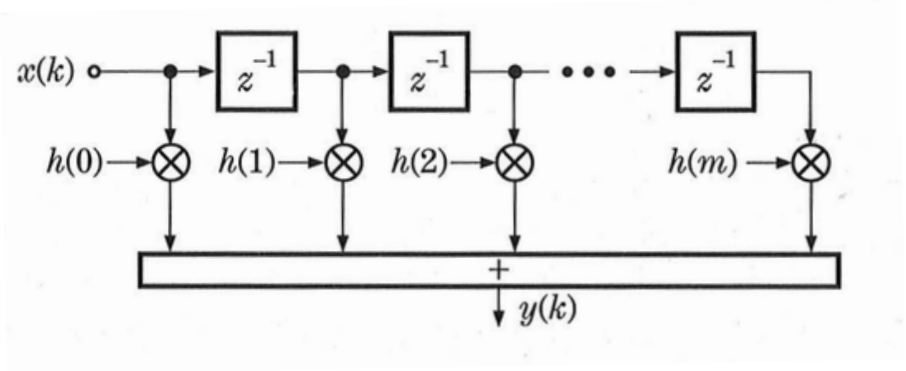


Abbildung 2.3-1 Signalflussdiagramm des FIR-Filters
Bild:[Kam]

Bei einer Abtastfrequenz von $f_A = 16\,kHz$ besitzt eine Impulsantwort mit einer Länge von $T = 2s$ insgesamt 32000 Abtastwerte und somit auch Koeffizienten *h(i)*. Damit muss der DSP für jeden Eingangswert *x(k)* 32000 reelle Multiplikationen und Additionen ausführen. Hochgerechnet sind das 512 Millionen reelle Multiplikationen und Additionen pro Sekunde. Dies übersteigt die Belastungsgrenze des Signalprozessors nicht, aber wenn die Faltung mit wesentlich längeren Impulsantworten erwünscht ist, werden die Rechengrenzen schnell erreicht.

Abhilfe für dieses Problem könnte man leisten, in dem die Impulsantwort durch ein Dezimator heruntergetastet wird. Dadurch könnte die Menge der Operationen auf einen

Bruchteil reduziert werden. Nachteil dieses Verfahrens ist, dass alle Frequenzen des Eingangssignals, die oberhalb der neuen Grenzfrequenz der Impulsantwort liegen, herausgefiltert werden müssen, was wiederum einen unerwünschten Qualitätsverlust mit sich bringt.

Ein weiterer Vorteil des FIR-Filters ist, dass es für jeden Eingangswert nach der Berechnung, ein Ausgangswert zur Verfügung steht, welches dann ausgegeben werden kann. Somit gibt es keinen hörbaren Versatz zwischen Ein- und Ausgabe, was ein weiteres Qualitätskriterium darstellt.

2.3.2 Zeitbereichsfaltung: IIR-Filter

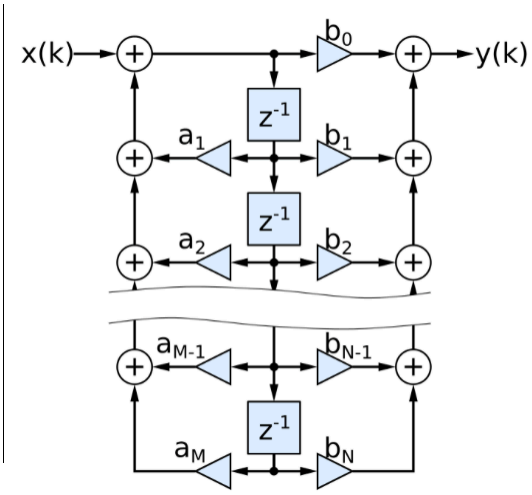


Abbildung 2.3-2 Signalflussdiagramm eines IIR-Filters
Bild: Wikipedia

Eine weitere Form der digitalen Filter ist der auf Abbildung 2.3.2-1 gezeigte IIR-Filter (*infinite impulse response*). Realisiert wird der IIR-Filter durch einen FIR-Filter mit Rückkopplungszweig, welcher die vorherigen Ausgangswerte mit einberechnet. Seine Übertragungsfunktion $H(z) = \frac{Y(z)}{X(z)} = \frac{\sum_{i=0}^{m} b_i z^{-i}}{\sum_{l=0}^{n} a_l z^{-l}}$ ist demnach unendlich lang [Kam].

Vorteil dieser Filterstruktur ist, die Länge der Impulsantwort, denn sie wird durch die Rückkopplung zu einem Bruchteil dezimiert. Dadurch wird ebenfalls der Rechenaufwand verringert.

Erfüllt der entworfene IIR-Filter nicht das Stabilitätskriterium ist die richtige Funktionsweise nicht gegeben.

2.3.3 Frequenzbereichsfaltung: Schnelle Faltung

Grundlage einer effizienten Realisierung von nichtrekursiven Filterstrukturen ist die schnelle Faltung, wobei die Faltungsoperation im Zeitbereich durch eine Transformation des Signals und der Impulsantwort in den Frequenzbereich, anschließende Multiplikation der Fourier-Transformierten und Rücktransformation in den Zeitbereich erfolgt. Es gilt:

$$y(n) = x(n) * h(n) = IDFT\{DFT\{x(n)\} \cdot DFT\{h(n)\}\} .$$

Diese Beziehung kann nur gelten, wenn das transformierte Signal stets periodisch ist, ferner würde die *zeitdiskrete Fourier-Transformation* (DTFT) benutzt werden. Angenommen es wird nur ein kleiner Teil der Eingangsfolge betrachtet und außerhalb der Betrachtungsgrenzen setzt sich das Signal periodisch fort, kann die DFT zur Berechnung der Spektren verwendet werden. Eine schnelle Berechnung der DFT eines Signals, kann mit Hilfe der schnellen Fourier-Transformation (im englischen Fast-Fourier-Transformation oder kurz FFT genannt) ausgeführt werden. Diese Algorithmen sparen bis zu 99% der Operationen gegenüber der DFT ein und sind somit das Werkzeug der Wahl [Zöl]. Zusätzlich gibt die DFT die Möglichkeit, mit Hilfe einer ausgeführten DFT, gleich zwei Teilfolgen vom Zeitbereich in den Frequenzbereich zu transformieren. Die mathematische Herleitung dazu, kann in [Kam] sowie [Zöl] nachvollzogen werden.

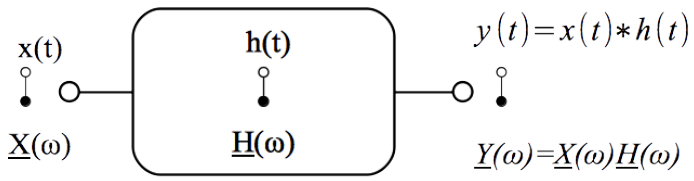


Abbildung 2.3-3Zusammenhang zwischen Zeit- und Frequenzbereich
Bild: eigene Darstellung

Da die Eingangsfolge nicht endlich ist, muss diese partitioniert und anschließend Transformiert werden. Nach der Multiplikation im Frequenzbereich und der anschließenden Rücktransformation, sind lediglich die Teilfaltungsergebnisse vorhanden. Um das endgültige Faltungsergebnis zu bekommen, müssen die Teilfaltungen sich überlappend überlagern (*Overlap Add*). So wird jeder Block des zu verarbeitenden Eingangssignals im Endergebnis berücksichtigt [Zöl].

Falls die Länge der Impulsantwort sehr groß ist, lässt sich ebenfalls eine Partitionierung der Impulsantwort in P Blöcken mit einer Länge von M vornehmen [Zöl].

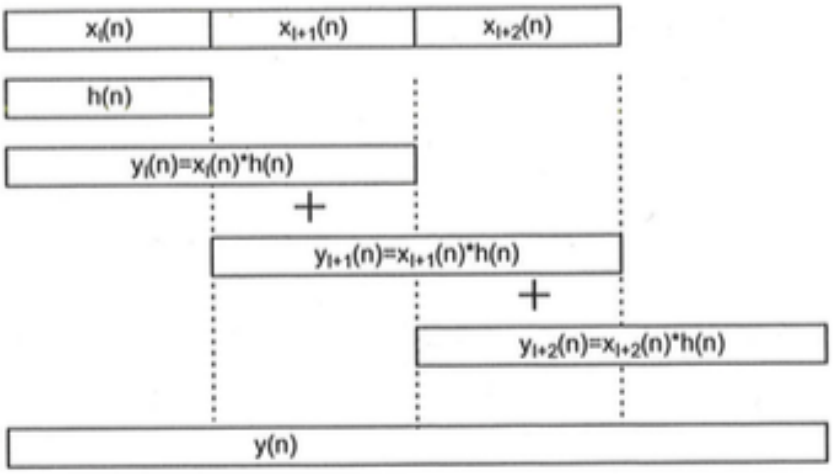


Abbildung 2.3-4 Darstellung des Overlap Add Verfahrens
Bild:[Zöl]

Das Problem hierbei, ist der Versatz zwischen dem Ein- und Ausgangssignal, denn um die DFT der Länge M eines Signals zu berechnen, müssen zuerst M Abtastwerte zwischengespeichert werden. Die DFT kann vorher nicht vom Signalprozessor berechnet werden. Die Ausgabe der Teilfaltung kann erst dann erfolgen. Der dabei entstehende Versatz zwischen Ein- und Ausgabe kann aber auch als Vorverzögerung, welche in modernen Hallgeräten als Parameter einstellbar ist, gewertet werden. Dieser Verzögerung ist aber generell nicht wünschenswert und kann mit dem in Abschnitt 5 aufgeführtem hybriden Algorithmus von Gardner, entgegengetreten werden.

In Abbildung 2.3.3-3 wird das Ablaufschema der schnellen Faltung mit partitionierter Impulsantwort (P = 4) dargestellt. Dieses Schema ist anwendbar, da die Linearität ihre Anwendung findet. Die Eingangsfolge *x(n)* kann durch Teilfolgen $x_m(n)$ in Folgender Form zerlegt werden: $x_m(n) = \begin{cases} x(n), & (m-1)M \le n \le mM-1 \\ 0, & sonst \end{cases}$ (1)

Die Eingangsfolge wird durch Überlagerung dargestellt.

$$x(n) = \sum_{m=1}^{\infty} x_m(n)$$

Das Faltungsergebnis liefert dann

$$y(n) = \sum_{k=0}^{M-1} h(k)x(n-k) \tag{2}$$

$$= \sum_{k=0}^{M-1} h(k) \sum_{m=1}^{\infty} x_m(n-k) \tag{3}$$

$$= \sum_{m=1}^{\infty} \left[\sum_{k=0}^{M-1} h(k)x_m(n-k) \right] \tag{4}$$

Der Term in den eckigen Klammern in Formel (4) entspricht dabei die Faltung der vollständigen Impulsantwort mit einem Eingangsblock der Länge M. Wird nun, wie oben erwähnt, die Impulsantwort *h(n)* mit folgender Regel partitioniert

$$h_p\left(n-(p-1)\frac{M}{P}\right) = \begin{cases} h(n), & (p-1)\frac{M}{P} \le n \le p\frac{M}{P}-1 \\ 0, & sonst \end{cases} \tag{5}$$

folgt

$$h(n) = \sum_{p=1}^{P} h_p\left(n-(p-1)\frac{M}{P}\right). \tag{6}$$

Setzt man nun (6) in (4) ein und formt es um ergibt sich mit $y(n) = \sum_{m=1}^{\infty} y_m(n)$ und $M_P = pM/P$

$$y(n) = \sum_{m=1}^{\infty} \underbrace{\sum_{k=0}^{M-1} h(k)x_m(n-k)}_{y_m(n)}$$

$$= \sum_{m=1}^{\infty} \left[\sum_{k=0}^{M_1-1} h(k)x_m(n-k) + \sum_{k=M_1}^{M_2-1} h(k)x_m(n-k) + \cdots + \sum_{k=M_{P-1}}^{M-1} h(k)x_m(n-k) \right]$$

$$= \sum_{m=1}^{\infty} \left[\underbrace{\sum_{k=0}^{M_1-1} h_1(k)x_m(n-k)}_{y_{m1}} + \underbrace{\sum_{k=0}^{M_1-1} h_2(k)x_m(n-M_1-k)}_{y_{m2}} \right.$$

$$\left. + \underbrace{\sum_{k=0}^{M_1-1} h_3(k)x_m(n-2M_1k)}_{y_{m3}} + \cdots + \underbrace{\sum_{k=0}^{M_1-1} h_P(k)x_m(n-(P-1)M_1-k)}_{y_{m1}} \right]$$

$$= \sum_{m=1}^{\infty} \underbrace{[y_{m1}(n) + y_{m2}(n-M_1) + \cdots + y_{mP}(n-(P-1)M_1)]}_{y_m(n)}$$

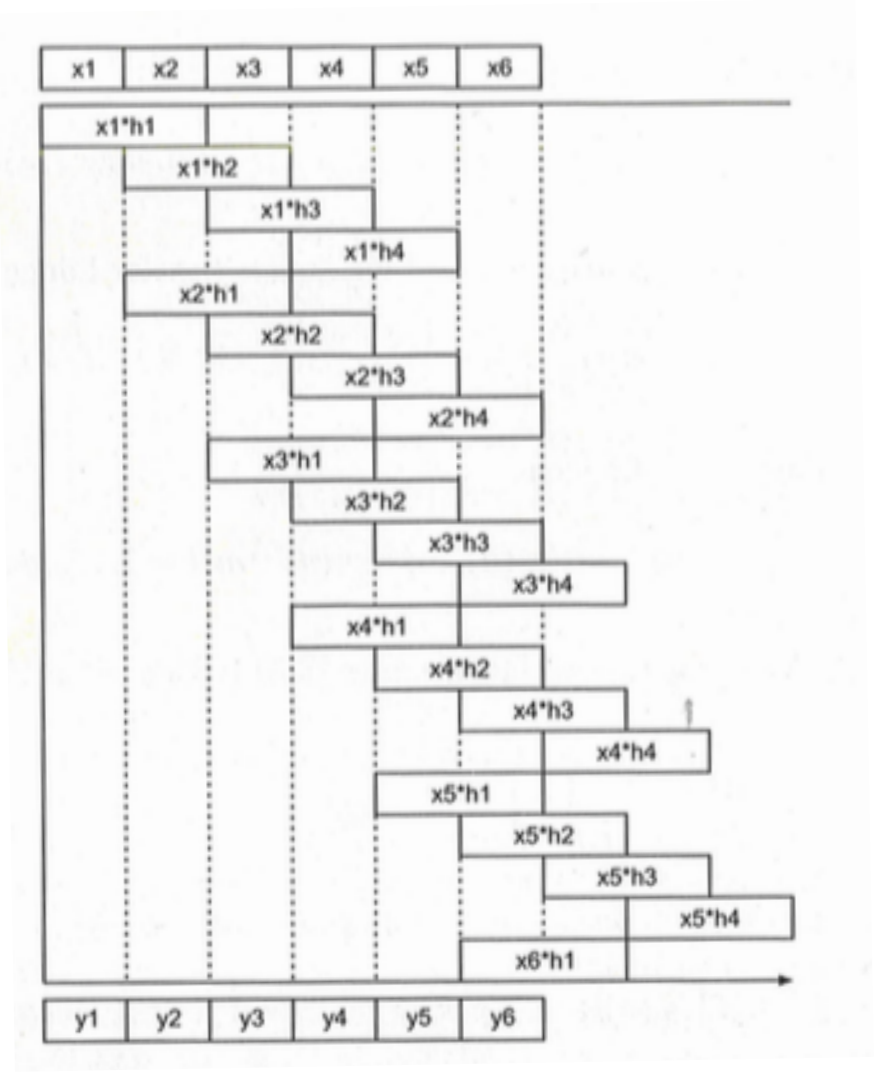


Abbildung 2.3-5 Ablaufschema einer schnellen Faltung mit P=4
Bild:[Zöl]

Weiterführend wird der Rechenaufwand der schnellen Faltung mit dem *Overlap Add* Verfahren in [Kam] mit dem Rechenaufwand eines FIR-Filters verglichen. Es zeigt sich, dass die schnelle Faltung schon bei Filter der Ordnung N > 30 effektiver ist, als die direkte Form der Implementierung. Darum bietet sich die *schnelle Faltung* zur Realisierung des Faltungshalls an und wird zur verwendeten Methode gewählt.

3 HARDWAREBESCHREIBUNG

In der Black-Box wurde das *Evaluationsboard* TMS320C6713 DSK von der Firma Spectrum Digital™ verbaut, welche eine rechenstarke und kostengünstige Entwicklungsplattform für Anwendungen des C6713 DSP ist. Es wird durch ein 5V Netzteil mit Strom versorgt. Durch den USB-Anschluss an der Rückseite der Box kann die Verbindung zum Labor-PC hergestellt werden. Auf der Oberseite der Box sind vier Leuchtdioden, vier Kippschalter, zwei 3,5mm Klinkenbuchsen und vier 4mm Bananenbuchsen angebracht. Diese dienen als Schnittstelle zur Mensch-Maschinen-Interaktion. Die Kippschalter werden zur Auswahl der Arbeitsweise benutzt. Ist ein Kippschalter aktiviert, wird die dazu gehörige Leuchtdiode angeschaltet. An den linken Buchsen können diverse Signalquellen wie z. B. Funktionsgeneratoren oder ein Mikrofon angeschlossen werden. Auf der rechten Seite, dem Ausgang, wird zur Kontrolle ein Oszilloskop oder Lautsprecher angeschlossen. Lautsprecher bzw. Mikrofone ohne eigene Stromversorgung können nicht verwendet werden.

Zuerst muss das Mikrofon durch einen Vorverstärker mit Phantomspeisung auf die richtige Lautstärke angepasst werden, damit kein *Clipping* des Eingangssignals auftritt.

Neben dem C6713 DSP hat ein weiterer wichtiger Baustein auf dem Board Platz gefunden. Der Audiocodec vom Typ TLV320AIC23 tastet das Eingangssignal ab und wandelt es mit Hilfe eines Delta-Sigma-Umsetzers in ein Zeit- und Wertediskretes Signal um. Nach der digitalen Signalverarbeitung werden die Ausgabewerte -auch durch den Audiocodec- in ein analoges Signal gewandelt. Damit ist der Audiocodec die Schnittstelle zwischen analogen und digitalen Signal.

3.1 C6713

Der verbaute digitale Signalprozessor ist aus TI's C67xx Generation der Prozessoren entsprungen, welche fähig sind sowohl mit Festkomma- sowie Gleitkommaarithmetik zu rechnen. Seine Taktfrequenz beträgt laut Datenblatt 225MHz. Da der Prozessor auf eine VLIW (*very large instruction word*) Architektur basiert, ist er in der Lage alle 4,44ns bis zu acht Befehle auszuführen.

Er ist mit insgesamt 512k-Byte Flashspeicher und 16M-Byte SDRAM ausgestattet, wobei die Zugriffsgeschwindigkeiten stark zwischen den Typen variiert. Das EMIF (*enhanced memory interface*) steuert dabei den Zugriff auf externen Speicher.

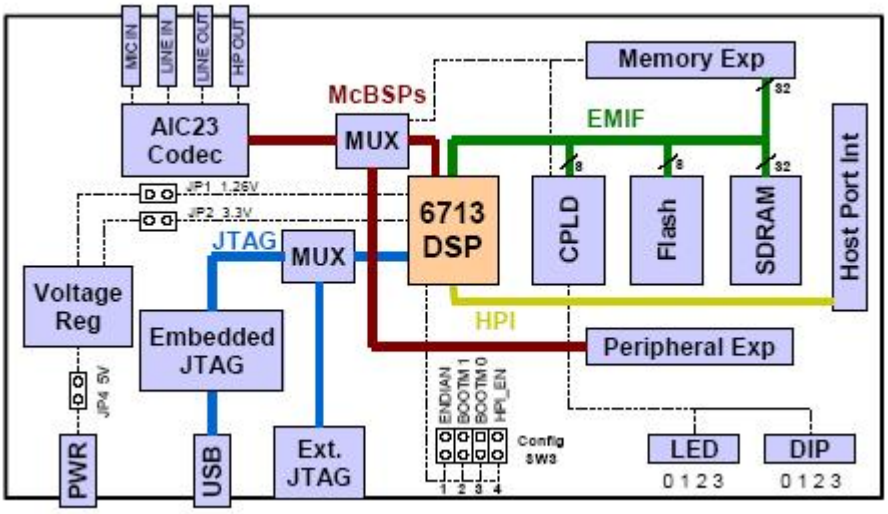


Abbildung 3.1-1 schematische Darstellung des inneren Aufbaus des Evaluationsboards
Bild:[ccsl]

In Abbildung 3.1-1 ist das Schema des Evaluationsboards aufgezeigt. Wie zu erkennen ist, besitzt der DSP mehrere Schnittstellen zur Kommunikation nach außen. Mit dem *McBSP* (*multi channel buffered serial port*) ist es für den DSP möglich, mit dem AIC23 Codec zu kommunizieren. Dafür gibt es zwei Wege. Zum einen wird der Codec über das *McBSP0*-Register konfiguriert und zum anderen geschieht der Datenaustausch zwischen Codec und DSP über das *McBSP1*-Register. [AJR]

3.2 TLV320AIC23 Stereocodec

Wie oben bereits erwähnt, sitzt auf dem Evaluationsboard neben dem Signalprozessor ein weiterer wichtiger Baustein. Der vom Hause Texas Instruments™ hergestellte Audiocodec besitzt Analog/Digital-Umsetzer und Digital/Analog-Umsetzer[TLV]. Damit ist dieser Baustein, für Eingang und Ausgang, die Schnittstelle nach außen. Er tastet das analoge Signal, mit der manuell einstellbaren Abtastfrequenz ab. Dafür benutzt er eine multibit Sigma-Delta Technologie. Nach der Abtastung werden die Daten über das *McBSP1* Register an den DSP gesendet. Nach der Berechnung des Hallalgorithmus gibt der DSP den passenden Ausgangswert mittels *McBSP1*-Register an den Stereocodec.

Alle Einstellungen des Codecs werden durch das *McBSP0*-Register in den Codec geladen[SPRU580g]. Das geschieht in der Initialisierungsphase des Hauptprogramms. Es stehen dabei mehrere Möglichkeiten zur Einstellung zu Verfügung, wie z. B. die Regelung der Lautstärke des Ausgangs oder die Einstellung der Abtastrate.

Das Schema des Codecs ist in Abbildung 3.2-1 dargestellt.

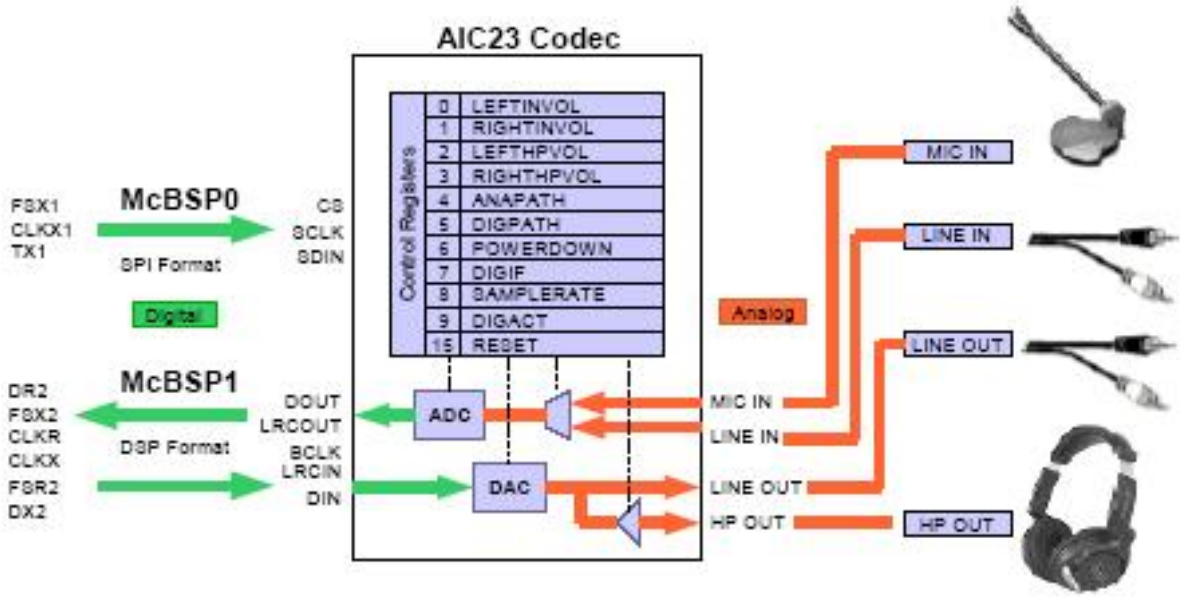


Abbildung 3.2-1 Schematische Darstellung des Stereocodecs
Bild:[ccsl]

4 PROGRAMMABLAUF UND IMPLEMENTATION

Um den Faltungshall langer Impulsantworten zu realisieren, wird der in [Zöl] beschriebene Algorithmus ausgewählt. Zuerst erstellt man ein Referenzprogramm mittels MATLAB, um zu prüfen, ob der Algorithmus sachgemäß funktioniert und die gleichen Ergebnisse wie in einer direkten Implementierung berechnet werden. Nachdem die Richtigkeit der Ergebnisse festgestellt wurde, kann die Implementierung auf Basis des DSP mit Hilfe der IDE *Code Composer Studio v3.1* beginnen. Die Initialisierungssequenz des Starterkits wird dabei von der Vorgängerarbeit übernommen.

Um die bereits implementierten Funktionen des DSPs zu nutzen, müssen zuerst die Randbedingungen für den Einsatz jener Funktionen aufgestellt werden. In [SPRU657c] sind alle für den Algorithmus nötigen Funktionen ausführlich erklärt und deren Quellcode gegeben. Anschließend wird ausführlich die Implementierung des Algorithmus und seiner Funktionen erläutert.

Die Bezeichnungen bestimmter Parameter weichen in folgenden Kapiteln von denen im MATLAB oder C Quellcode ab. Der Bezeichner HOPSIZE wird in der theoretischen Betrachtung einfachheitshalber M genannt. P bezeichnet die Anzahl der Blöcke, in der die Impulsantwort zerlegt wird. Die Beschreibung der Bezeichner sind aus dem Quellcode zu entnehmen.

4.1 Die Zerlegung der Impulsantwort

Es ist eine Impulsantwort eines Theaters, Wohnzimmers und einer Kathedrale gegeben. Diese haben eine Abtastrate von 44,1 kHz (CD-Qualität). Damit diese Impulsantworten verarbeitet werden können, muss zunächst die Abtastrate konvertiert werden, da der DSP mit einer Abtastrate von f_A= 16 kHz das Eingangssignal abtastet. Dafür wurde die *resample*- Funktion von MATLAB verwendet.

Zuerst wurde eine Matrix der Größe [2·M x P] erstellt und mit Nullen gefüllt. Die Zerlegung der Impulsantwort erfolgte anschließend mit der in Abbildung 4.1-2 aufgezeigten verschachtelten while- Schleifen. Es resultiert die Matrix in Abbildung 4.1-1 gezeigt wird. Neben der Zerlegung in P Teilimpulsantworten, wird jede Teilimpulsantwort gleich bis zur Länge 2M mit Nullen aufgefüllt.

$$g(k,l) = \begin{bmatrix} h(1) & h(M+1) & \cdots & h((P-1)M) \\ \vdots & \vdots & & \vdots \\ h(M) & h(2M) & \cdots & h(PM) \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & 0 \end{bmatrix}$$

Abbildung 4.1-1 Aufteilung der Impulsantworten in P Blöcke

Bild: eigene Darstellung

```
l=1;                        %Spaltenzähler init.
k=1;                        %Zeilenzähler init.
t=1;
while t<=M
    while k<=hopsize
    if t<=M
       g(k,l)=hl(t);
    end
       k=k+1;

           t=t+1;

    end
```

Abbildung 4.1-2 while-schleife zur Partitionierung der Impulsantwort

Anschließend wurde jede Spalte der Matrix einzeln mit Hilfe der FFT in den Frequenzbereich transformiert. Die Speicherung der Koeffizienten erfolgte in einer Textdatei, die später zu einer Header-Datei im richtigen Format umgeschrieben wurde.

4.2 Zerlegung und Verschachtelung der Eingangsfolge

Der nächste Schritt der schnellen Faltung ist die Zerlegung und Verschachtelung der Eingangsfolge. Die Implementation mit der Hochsprache C weicht ein wenig von der in MATLAB ab, da C keine komplexen Zahlenformate kennt, MATLAB hingegen schon. Der DSP ließt sukzessiv die 32-Bit Werte aus seinem Speicher, wobei jeder geraden Adresse der Realteil und jeder ungeraden Adresse, den Imaginärteil, eines komplexen Zahlenpaares zugeordnet ist. Deshalb muss bei der Speicherallokation darauf geachtet werden, dass ein Array oder Vektor mit einer Adresse beginnt, bei der die letzten zwei Hex-Zahlen Nullen sind [SPRU657c].

4.2.1 Implementierung in MATLAB

Als erstes wird eine .wav-Datei, welche die Eingabe des Mikrofons simuliert, eingelesen und analog zur Vorgehensweise, welche in Abschnitt 4.1 beschrieben steht,

in mehrere Blöcke unterteilt. Anschließend müssen zwei aufeinander folgende Teilfolgen zu komplexen Teilfolgen verschachtelt werden. Dafür ist der in Abbildung 4.2.1-1 aufgezeigte Quellcode auszuführen.

```
while 2*k<=numel(z)/(2*hopsize)
    z_complex(:,k)=[z(:,2*k-1)+1i*z(:,2*k)];
    k=k+1;
end
```

Abbildung 4.2.1-1 Quellcode: Verschachtelung der Eingangsblöcke zu komplexen Eingangsblöcken

Nun sind die Eingangsabtastwerte für die komplexe FFT vorbereitet, welche mit der in MATLAB implementierten fft()-Funktion für jede Spalte des komplexen Eingangsarrays ausgeführt wird.

4.2.2 Implementierung in C

In [SPRU657c] wird beschrieben, wie die Eingangsdaten formatiert sein müssen, damit der Signalprozessor die schnelle Fourier-Transformation anwenden kann. Zunächst wird als Eingabe ein Pointer vom Datentyp *float* verlangt. Des Weiteren muss der Eingabevektor so formatiert sein, dass jedes gerade Eingabeelement (angefangen beim „nullten Element“) den Realteil und jedes ungerade Eingangselement den Imaginärteil, eines komplexen Zahlenpaares wiederspiegelt. Um die Teilblöcke so zu verschachteln, wurde zunächst mit der *memset()*-Funktion die Größe des Eingabevektors definiert und alle Speicherplätze mit einer Null beschrieben. Dabei ist zu beachten, dass der Vektor viermal so lang wie eine Eingangsteilfolge sein muss, da zum einen mit dieser Funktion auch das *Zeropadding* auf die Länge 2M erfolgt und zum anderen für jeden komplexen Wert zwei Werte nacheinander in den Speicher abgelegt werden müssen.

Die Abtastung des Eingangssignals sowie die Ausgabe des berechneten Ausgabewertes, findet in der Interrupt-Service-Routine (ISR) statt. Sie wird aufgerufen sobald $\Delta t = {}^{1}/_{f_A} = \frac{1}{16000Hz} = 62{,}5\mu s$ erreicht ist, dann wird der momentan anliegende Wert vom Codec abgetastet und auf die richtige Adresse im Speicher abgelegt. Anschließend wird die Position des Pointers, der auf den Eingangsbuffer (T_BUFFER) zeigt, je nach aktueller Lage geändert. Ist das Einlesen der momentanen Eingangsfolge nicht abgeschlossen, springt der Zeiger um zwei Plätze weiter. Ist die momentane Eingangsfolge abgeschlossen, wird die Position des Zeigers auf das erste Element der nächsten Folge gelegt. Wenn der Pointer auf die Stelle 2M-1 im Buffer zeigt, ist die

Eingabe der ersten beiden Teilfolgen abgeschlossen und zwischengespeichert. Anschließend ist er bereit wieder neu beschrieben zu werden, deshalb wird der Pointer auf die Anfangsposition des Eingangsbuffers gelegt. Das Schema dazu ist auf Abbildung 4.2.2-1 zu sehen. Die erste Teilfolge ist blau markiert, die zweite Teilfolge dagegen mit rot.

Damit das Hauptprogramm solange wartet, bis der Buffer voll ist und nicht mit einem falschen Buffer arbeiten muss, ist am Anfang der Berechnung eine Warteschleife eingebaut. Diese wartet auf das Signal, welches in der ISR bei vollem Buffer, gesetzt wird. Empfängt die Warteschleife dieses Zeichen (BUFFER_FULL), ist die Austrittsbedingung erfüllt und die Berechnung nach [Zöl] kann beginnen.

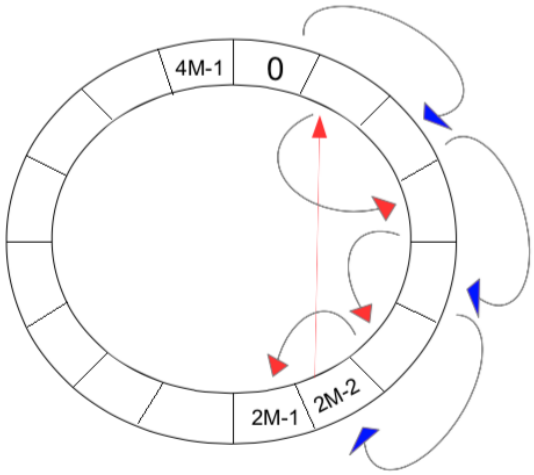


Abbildung 4.2-1 Schema für die Organisation des Inputbuffers
Bild: eigene Darstellung

4.3 Die Interrupt-service-routine AIC_RX()

Am Anfang der *main*-Funktion, nachdem der Speicher für diverse Vektoren und Matrizen bereitgestellt wurde, muss das DSK initialisiert und der Codec gestartet werden. Danach wird die Abtastrate des Codecs auf 16kHz eingestellt und die Interrupts aktiviert. Empfängt der DSP ein Interrupt vom *TLV320AIC23* ausgehend, wird die Interrupt-Service-Routine aufgerufen, der vom Codec abgetastete Wert vom rechten Kanal eingelesen und in den T_BUFFER abgespeichert. Dann wird der Ausgabewert aus dem Ausgangsbuffer (OUT_BUFFER) in die Ausgabevariabel y_n geladen und auf den Codec geschrieben, welcher mittels D/A-Wandler eine Ausgangsspannung erzeugt.

Anschließend werden die Zeiger der Eingangs- und Ausgangsbuffer aktualisiert, damit sind die ISR für die Ein- und Ausgabe verantwortlich.

4.4 (Inverse) Fast-Fourier-Transformation

Wie im obigen Abschnitt beschrieben, verlangt die im DSP integrierte FFT-Funktion *DSPF_sp_cfftr2_dit()* einen Eingabevektor der Länge 4M, welcher nacheinander mit Realteil und Imaginärteil gefüllt sein muss. Sie führt den sogenannten *Radix-2-Algorithmus* von Cooley und Turkey in der Variante „Reduktion im Zeitbereich" aus[Kam]. Der nächste Übergabeparameter ist der „twiddle-factor", ein Vektor, der den *Drehoperator* W_N, welcher vom Radix-2 Algorithmus benötigt wird, beinhaltet. Dieser Vektor wird am Anfang der *main()*-Funktion durch die Funktion gen_*twiddle ()* erzeugt und in w gespeichert, um anschließend die Bitreihenfolge zu vertauschen, ansonsten kann die FFT-Funktion nicht benutzt werden. Der letzte Übergabeparameter ist die Größe der FFT. Dieser muss 4M und eine Potenz von 2 sein, damit die FFT der zwei Teilfolgen auch die richtige Länge besitzt. Die Ausgabe der FFT-Funktion wird im Eingangsvektor mit verkehrter Bitreihenfolge gespeichert. Um die Bitreihenfolge wieder zu drehen, wird die von TI gegebene Funktion *bit_rev()* verwendet. [SPRU657c]

Das Ergebnis der komplexen FFT beinhaltet, laut [Zöl], zwei unabhängige Spektren für die zwei verarbeiteten Teilfolgen. Die dafür nötigen Zusammenhänge werden in [Kam] sowie [Zöl] erläutert.

Nach der im nächsten Abschnitt erklärten Multiplikation der Spektren, müssen alle Teilfaltungsergebnisse, die im Frequenzbereich berechnet wurden, in den Zeitbereich zurück transformiert werden. Die dafür vorgegebene Funktion *DSPF_sp_cifftr2_dif()*, verwendet ebenfalls einen *Radix-2-Algorithmus*, aber dieses mal mit der „Reduktion im Frequenzbereich". Die Übergabeparameter sind ebenfalls analog zur FFT-Funktion ein Eingangsvektor der Länge 4M in verkehrter Bitreihenfolge, der komplexe Drehoperator, und die Länge der IFFT, welche ebenfalls 4M und eine Potenz von 2 sein muss. Anschließend muss das Ergebnis auf die Länge der IFFT normiert werden. Das vertauschen der Bitreihenfolge ist nicht nötig, da die Funktion die Ergebnisse mit richtiger Bitreihenfolge ausgibt.

4.5 Elementweise Multiplikation im Spektralbereich

Der nächste Schritt beinhaltet die Multiplikation im Spektralbereich. Das Spektrum des Eingangssignals wird mit jedem Block der Impulsantwort separat Multipliziert und abgespeichert. Für eine komplexe Multiplikation wird je Real- und Imaginärteil zwei Multiplikationen und eine Addition reell wertiger Zahlen benötigt. Die Ergebnisse spiegeln jede einzelne Teilfaltung im Frequenzbereich wieder. Es muss zunächst darauf geachtet werden, dass auf jedem geraden Speicherplatz der Realteil und auf jedem ungeraden Speicherplatz der Imaginärteil der Spektren gespeichert sind. Das gleiche Format wird auch für die IFFT benötigt. Es müssen nun zwei Fälle unterschieden werden. Zum einen Fall wird ausschließlich der Realteil und im anderen ausschließlich der Imaginärteil berechnet.

$$[Z_R(k) + jZ_I(k)][H_R(k) + jH(k)]$$
$$= Z_R(k)H_R(k) - Z_I(k)H_I(k) + j[Z_R(k)H_I(k) + Z_I(k)H_R(k)]$$

Es entsteht eine Matrix, in der jede Spalte eine Teilfaltung in Frequenzbereich darstellt, die anschließend zurück in den Zeitbereich transformiert wird und damit eine Teilfaltung im Zeitbereich wiederspiegelt.

4.6 Bestimmung und Neukomposition der Teilfaltung

Um die ineinander verschachtelten Teilfaltungen zu sortieren, müssen Real- und Imaginärteil jedes komplexen Zahlenpaares voneinander separiert werden. So entstehen zwei unabhängige Matrizen, in denen die Teilfaltungen der geraden und ungeraden Eingangsteilfolgen stehen. Nun werden die Spalten der Matrizen überlappend aufaddiert, so dass aus der Matrix ein Vektor der Länge $L = (P + 1) \cdot M$ wird, welcher die Faltung des Eingangsblocks mit der kompletten Impulsantwort darstellt (siehe Abbildung 4.6-1). Anschließend werden auch die beiden so entstandenen Vektoren zu einer Zwischenlösung der Länge L+M aufaddiert (siehe Abbildung 4.6-2).

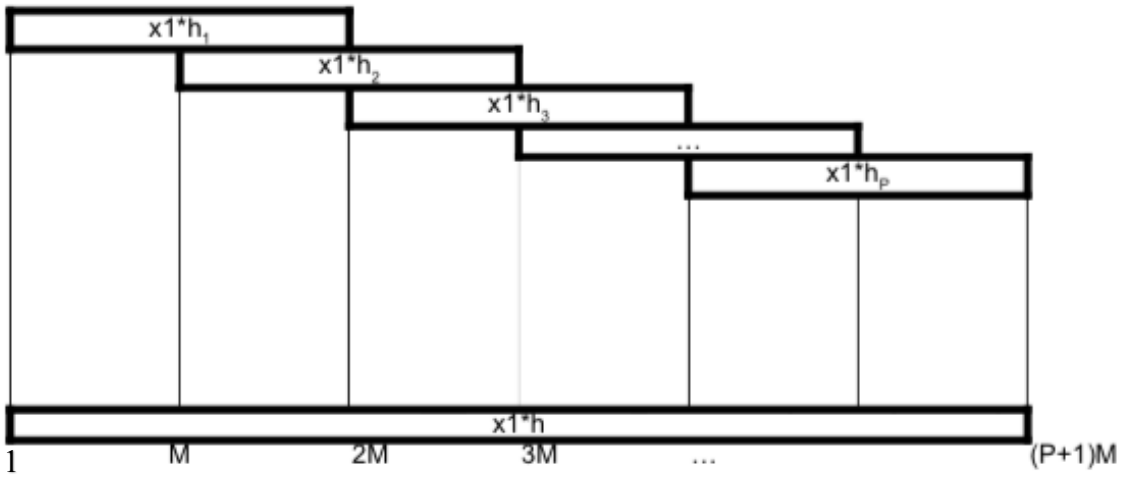


Abbildung 4.6-1 Overlapp Add Verfahren für die Teilfaltung des ersten Eingangsblocks
Bild: eigene Darstellung in Anlehnung an [Zöl]

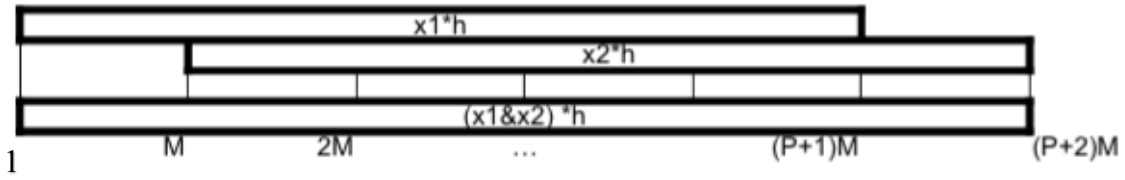


Abbildung 4.6-2 Overlapp Add der Teilfaltung
Bild: eigene Darstellung

Die so entstandene Zwischenlösung kann aber erst auf den Ausgangsbuffer geschrieben werden, wenn die Ausgabe des letzten Wertes der vorherigen Teilfaltung erfolgte, in der keine Überlagerung zwischen der vorherigen und eben berechneten Teilfaltung vorliegt. Um diesen Moment abzupassen, wird ein Signal (BUFFER_FINISH) in der ISR gesetzt, welches die Abarbeitung der Ausgangswerte im OUT_BUFFER symbolisiert. Immer wenn M Werte ausgegeben wurden, erhöht sich das Signal BUFFER_FINISH um 1. Erreicht der Zeiger auf OUT_BUFFER die letzte Stelle, in der die vorherige und momentane Teilfaltung keine Überlappung besitzt, erhöht sich BUFFER_FINISH auf 2 und die Austrittsbedingung der Warteschleife ist erfüllt.

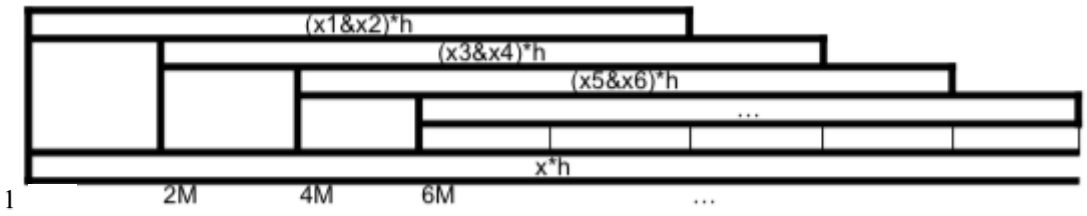


Abbildung 4.6-3 Overlap Add der Zwischenlösung
Bild: eigene Darstellung

Würde es diese Warteschleife nicht geben, kann nicht garantiert werden, dass die Überlappung auch an der richtigen Stelle mitberücksichtigt wird. Hinterher kann der abgearbeitete Teil des OUT_BUFFER gelöscht, BUFFER_FINISH auf Null gesetzt, der Rest der bisher nicht abgearbeiteten Ergebnisse nach vorne verschoben und darauf die neuen Werte der momentan berechneten Teilfaltung aufaddiert werden. Hier endet die Hauptarbeitsschleife und der nächste Block von Eingangswerten kann verarbeitet werden.

Anzumerken ist, dass beim ersten Programmdurchlauf die Zwischenlösung sofort nach der Berechnung in den OUT_BUFFER zu schreiben und die Position des Zeigers auf OUT_BUFFER, auf den Anfang des Buffers, zu setzen ist, ansonsten müsste erst gewartet werden, bis 2M Werte, welche nur mit Nullen gefüllt ist, ausgegeben wurden.

5 BEWERTUNG DES ALGORITHMUS, PROBLEME, VERBESSERUNGSMÖGLICHKEITEN

Der angewandte Algorithmus nach [Zöl] braucht, in der Simulation auf einem *Macbook Pro* von Ende 2011 mit einem 2,8GHz Intel Core I7, zwischen 7ms und 9ms pro Schleifendurchlauf in MATLAB. Das bedeutet nach 71ms steht das Ergebnis zur Ausgabe zur Verfügung. Leider konnte dieser Wert nicht mit der Implementierung des Programms auf dem DSP verglichen werden, da es zu Problemen beim *Debugging*-Vorgang kam, die auf Grund Zeitmangels nicht behoben werden konnten. Deshalb konnte das Programm auch nicht fristgerecht zum laufen gebracht werden.

Die Messungen ergaben, dass der verwendete Algorithmus rechentechnisch zu komplex für den DSP ist. Um einen Eingangsblock der Länge 128 abzuarbeiten, werden momentan 890.000 Prozessortakte benötigt, obwohl nur ca. 540.000 Taktzyklen zur Verfügung stehen. Hier kann die Länge eines Blockes variiert werden, wobei sich gleichzeitig die Anzahl der Blöcke P verändert. Nun kann die Komplexität der unterschiedlichen Verarbeitungsschritte (Transformation und OLA) verglichen werden. Es stellte sich heraus, dass eine Blocklänge von 4096 Abtastwerten benötigt wird, um genug Zeit zur Verfügung zu Stellen, damit die Transformation sowie die Berechnung der einzelnen Überlappungen ausgeführt werden kann. Wird die Anzahl der Blöcke auf eins beschränkt und die Faltung wird durch eine Multiplikation mit eins ersetzt, ist das Ergebnis gleich dem Eingangssignal, da eine Faltung des Eingangssignals mit einem Dirac-Stoß berechnet wird. Dieses Verhalten gilt bis zu einem P von 4. Danach wird die Ausgabe verzerrt, was ein Anzeichen dafür ist, dass die Anzahl an Speicherzugriffen und Berechnungen den DSP an seine Grenzen bringt.

Wie im Abschnitt 2.3.3. angedeutet kann auch mit Hilfe einiger Tricks gegen diese lange Latenzzeit sowie der hohen Prozessorlast vorgegangen werden. Im folgenden Abschnitt wird die von William G. Gardner, in seinem Paper „Efficient Convolution without Input-Output Delay“ vorgeschlagene Herangehensweise an die schnelle Faltung, erläutert.

5.1 „Zero-Delay Convolution Algorithmus“

Gardner schlägt vor, die Impulsantwort in immer größer werdende Blöcke einzuteilen. Dabei fängt er an eine bestimmte Größe für N festzulegen. N muss dabei eine Potenz von

2 sein. Darauf hin wird die Impulsantwort, nach dem Schema in Abbildung 5.1-1, zerstückelt.

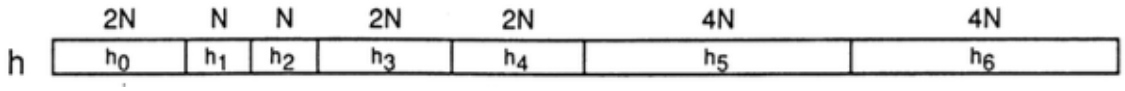


Abbildung 5.1-1 Zerlegung der Impulsantwort nach Gardner
Bild:[GA95]

Die Faltung der ersten Impulsantwort h_0 mit dem Eingangssignal wird in direkter Form als FIR-Filter implementiert. Dadurch wird der erste Teil der Faltung ohne Latenz berechnet und ausgegeben. Alle weiteren Teilfaltungen müssen nun als schnelle Faltung implementiert werden. Dieser hybride Aufbau des Algorithmus gibt dem DSP die Zeit um genug Abtastwerte zwischen zu speichern und die Blocktransformation auszuführen. Es wird vorgeschlagen, dass ein Prozess-Scheduler die Aufgaben, welche nach einer gewissen Priorisierung vom DSP abgearbeitet werden müssen, steuert. Dieser Prozess-Scheduler könnte so aufgebaut sein, dass priorisierte Interrupts immer dann ausgelöst werden, wenn eine wichtigere Teilfaltung zuerst berechnet werden muss. Anschließend springt das Programm an die Stelle zurück, an der er unterbrochen wurde. Nach [Ga95] werden kleinere Teilfaltung höher priorisiert, als längere. Dadurch wird die Belastung des DSP zeitlich verteilt. Ansonsten müsste das Ergebnis immer fertig berechnet sein, wenn ein neuer Eingangsblock abgespeichert wurde [Ga95].

6 ZUSAMMENFASSUNG, MOMENTANER STAND UND AUSSICHTEN

Das Augenmerk dieser Arbeit lag auf der Implementierung eines Programmes zur Berechnung der Faltungssumme von langen Impulsantworten. Zuerst wurden die notwendigen theoretischen Grundlagen dazu erarbeitet und eine Auswahl der Methode aufgrund der vorhandenen Fakten getroffen. Anschließend musste ein Referenzprogramm mit MATLAB erstellt werden. Nach der Validierung des Referenzprogrammes konnte die Implementierung des gleichen Algorithmus mit der Hochsprache C, auf die DSP Plattform, begonnen werden. Als Erstes wird der Algorithmus mit nur einer vorbereiteten Impulsantwort (Theater) implementiert und validiert, um anschließend auch die anderen Impulsantworten einzufügen.

Dazu war es nötig die verwendete Hardware des Demonstrators genauer zu verstehen und sich in die IDE *Code Composer Studio* einzuarbeiten. Das hat mir klargemacht, dass digitale Signalprozessoren für die Verarbeitungen von Daten(-streams) konzipiert wurden und auf ihre Verwendung nur hingewiesen, aber in keiner Lehrveranstaltung weiter thematisiert wurden. Obwohl sie definitiv ihre Daseinsberechtigung haben und in immer mehr Anwendungen ihren Platz finden.

Der C6713 Signalprozessor besitzt dagegen aber nicht ausreichend rechentechnische Kapazität sowie Speicherressourcen, um diese Aufgabe zu bewältigen. Das Problem dabei ist, dass nicht genügend interner RAM zur Verfügung steht. Dadurch ist der Prozessor gezwungen die Daten in den SDRAM zu speichern. Der Zugriff auf den SDRAM ist allerdings zeitaufwendiger, da der Prozessor zuerst die Daten vom SDRAM in den L2-Cache lädt. Dabei entstehen sogenannte „Miss-Cycles“, in dem der DSP angehalten wird, damit er auf den nächsten Takt wartet. Vom L2-Cache aus können die Daten in die einzelnen Funktionsblöcke verarbeitet werden.

Anhand der Validierung des Referenzprogramms wurde gezeigt, dass die Berechnung der schnellen Faltung sich durchaus für die Realisierung der Faltungsoperation in Echtzeit geeignet ist. Wenn man bedenkt, dass auch eine gewisse Belastung im Hintergrund eines jeden PCs für Prozessorauslastung sorgt und angenommen die Rechenleistung des DSP ist gleich der des PCs, ist eine Zeit von 9ms pro Schleifendurchlauf und damit 71ms zwischen Ein- und Ausgabe ausreichend um das System als Echtzeitfähig zu betiteln.

Auf Grund der Probleme mit *Code Composer Studio* sowie der begrenzten Bearbeitungszeit und der Ressourcen, konnte die Validierung der Software allerdings nicht beendet werden. Es wird der Erwerb eine neuere Entwicklungsplattform empfohlen, da zum einen der C6713 obsolet ist und zum anderen bietet der Markt rechenstärkere und kostengünstigere Plattformen, welche momentan weit verbreitet sind, an.

Ziel war es, zuerst die Faltung mit einer Impulsantwort zu realisieren. Danach sollten auch die Faltungen mit den anderen Impulsantworten in jeweils einzelnen Programmen implementiert werden. So kann zwischen den einzelnen Präsentationen der PC vom LTI-System getrennt werden, um das nächste Programm in die IDE zu laden. Allerdings muss nach der Präsentation eines Halls, eine Übertragung des Programmes für den nächsten Hall an den DSP geschehen. Hier existiert noch viel Potential für Erweiterungen. Zum Beispiel könnte mit Hilfe der Kippschalter die Impulsantwort ausgewählt werden. Dadurch wäre es möglich das unnötige Laden der unterschiedlichen Programme zu vermeiden und die Präsentation kann fließender gestaltet werden.

7 LITERATURVERZEICHNIS

BÜCHER UND SKRIPTE

[Dobl] Doblinger, Gerhard. *MATLAB - Programmierung in der digitalen Signalverarbeitung.* Wien: J. Schlembach Fachverlag, 2001.

[Ga95] Gardner, William G. „Efficient Convolution without Input - Output Delay.“ *Journal of Audio Engineering Society.* San Francisco: Audio Engineering Society, 1995. 127-135.

[Kam] Kristian Kroschel, Karl Dirk Kammeyer. *Digitale Signalverarbeitung. Filterung und Spektralanalyse mit MATLAB-Übungen.* 5. Auflage. B.G. Teubner Verlag, 2002.

[Bäs] Meyer-Bäse, U. *Schnelle digitale Signalverarbeitung Algorithmen, Architekturen, Anwendungen.* Springer-Verlag, 2000.

[Hoff] Rüdiger Hoffmann, Matthias Wolff. *Intelligente Signalverarbeitung 1 - Signalanalyse.* Dreden/Cottbus: Springer Vieweg, 2014.

[Zöl] Zölzer, Udo. *Digitale Audiosignalverarbeitung.* 3. Auflage. Vieweg+Teubner Verlag, 2005.

[AJR] Richter, Andre Jan. *Bachelorarbeit Entwicklung eines Vorlesungsexperiments „LTI-System“.* BTU-Cottbus, 2013.

INTERNETSEITEN

18.03.2016

[ccsl] **ECE Illinois: Techno Dancing Robot.** University of Illinois 2004, Zugriff 17:23 Uhr

http://coecsl.ece.illinois.edu/ge423/spring04/group1/design.html

[wikg] **Wikipedia: Gleitkommazahl.** Zugriff: 18:23 Uhr

https://de.wikipedia.org/wiki/Gleitkommazahl

21.03.2016

[wikd] **Wikipedia: Datentyp**. Zugriff: 13:21 Uhr

https://de.wikipedia.org/wiki/Datentyp

[wikf] **Wikipedia: Digitales Filter**. Zugriff: 14:42 Uhr

https://de.wikipedia.org/wiki/Digitales_Filter

[wikc] **Wikipedia: Faltung (Mathematik)**: Zugriff: 15:31 Uhr

https://de.wikipedia.org/wiki/Faltung_(Mathematik)

TECHNISCHE DOKUMENTE

[SPRU657c] Texas Instruments: TMS320C67x DSP Library Programmer's Reference Guide. Januar 2010.

[SPRU580g] Texas Instrumens: TMS320C6000 DSP Multichannel Buffered Serial Port Referenz Guide. Dezember 2006.

[SPRU189g] Texas Instruments: TMS320C6000 CPU and Instruction Set Referenz Guide. Juli 2006.

[SPRU733a] Texas Instruments: TMS320C67x/C67x+ DSP CPU and Instruction Set Referenz Guide. November 2006.

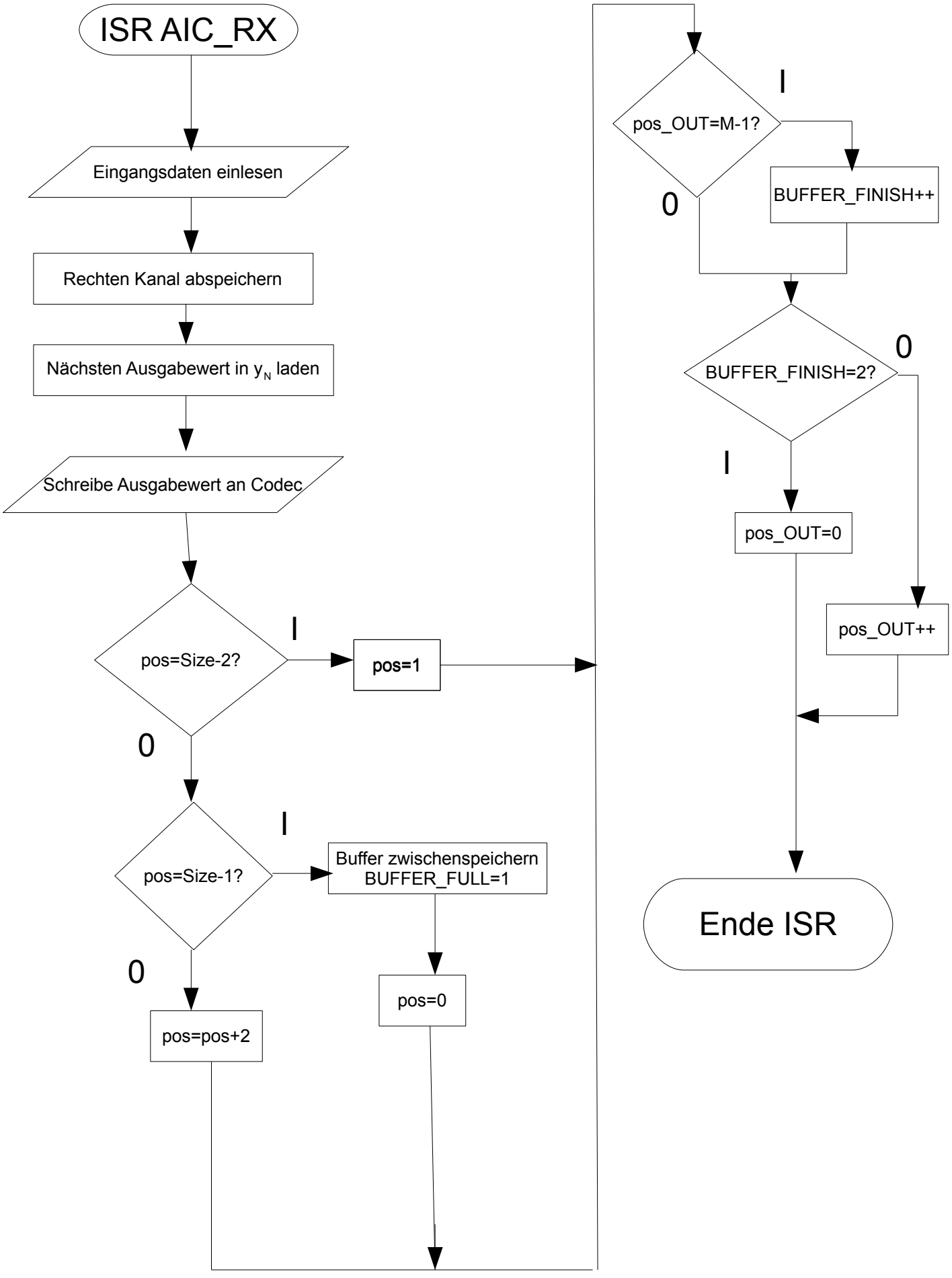
[TLV] Texas Instruments: TLV320AIC23 Stereo Audio CODEC Data Manual. Mai 2002.

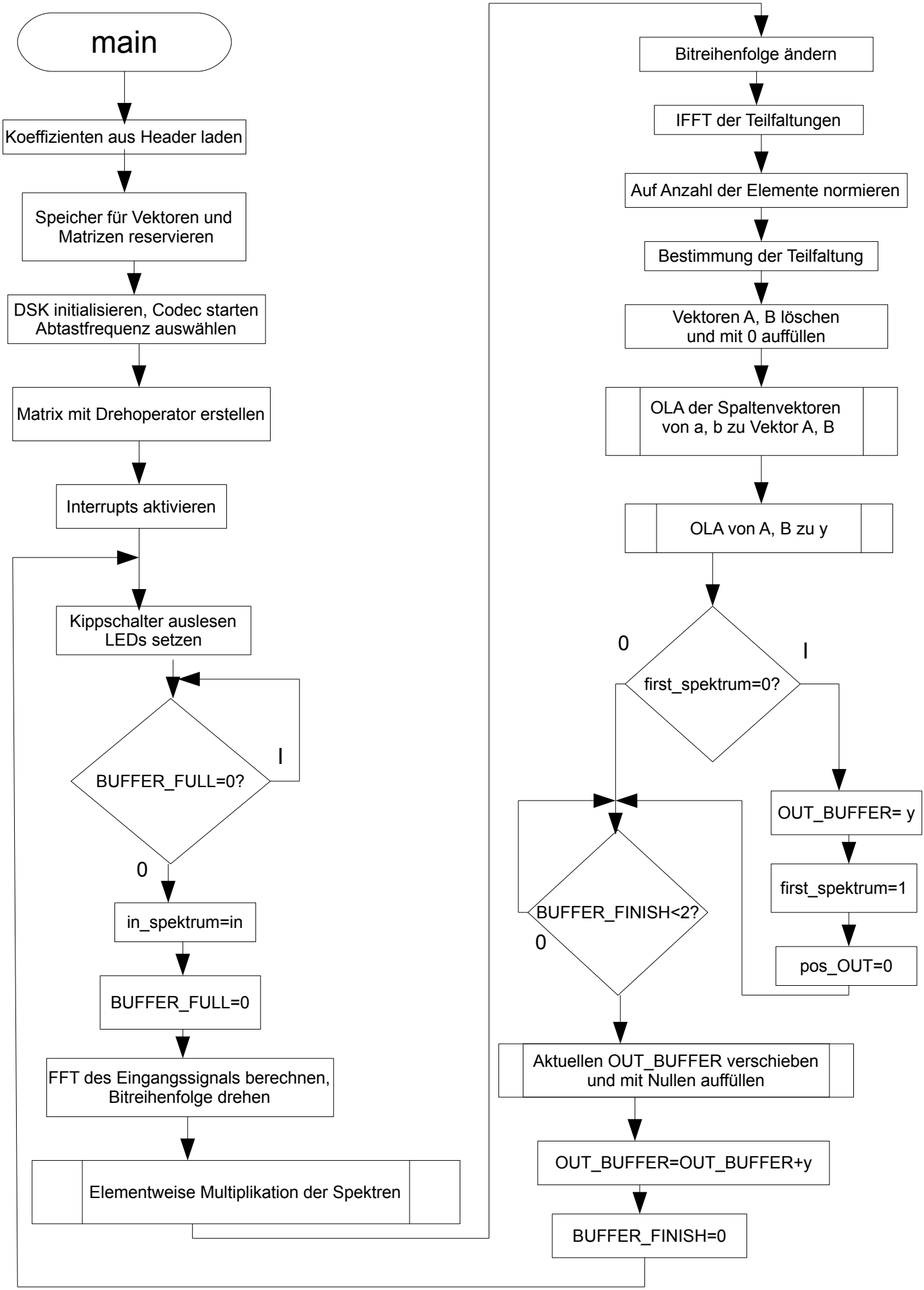
[SPRU197d] Texas Instruments: TMS320C6000 Technical Brief. Februar 1999.

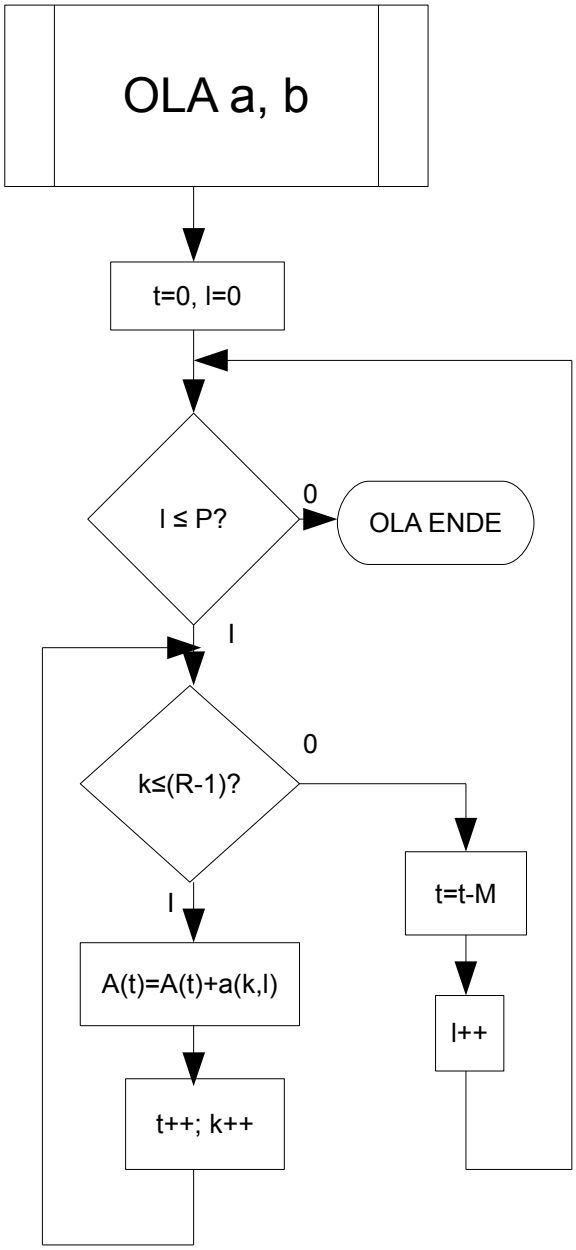
[SPRU198k] Texas Instruments: TMS320C6000 Programmer's Guide Juli 2011

[TMS] Texas Instruments: TMS320C6713 FLOATING-POINT DIGITAL SIGNAL PROCESSOR. November 2005

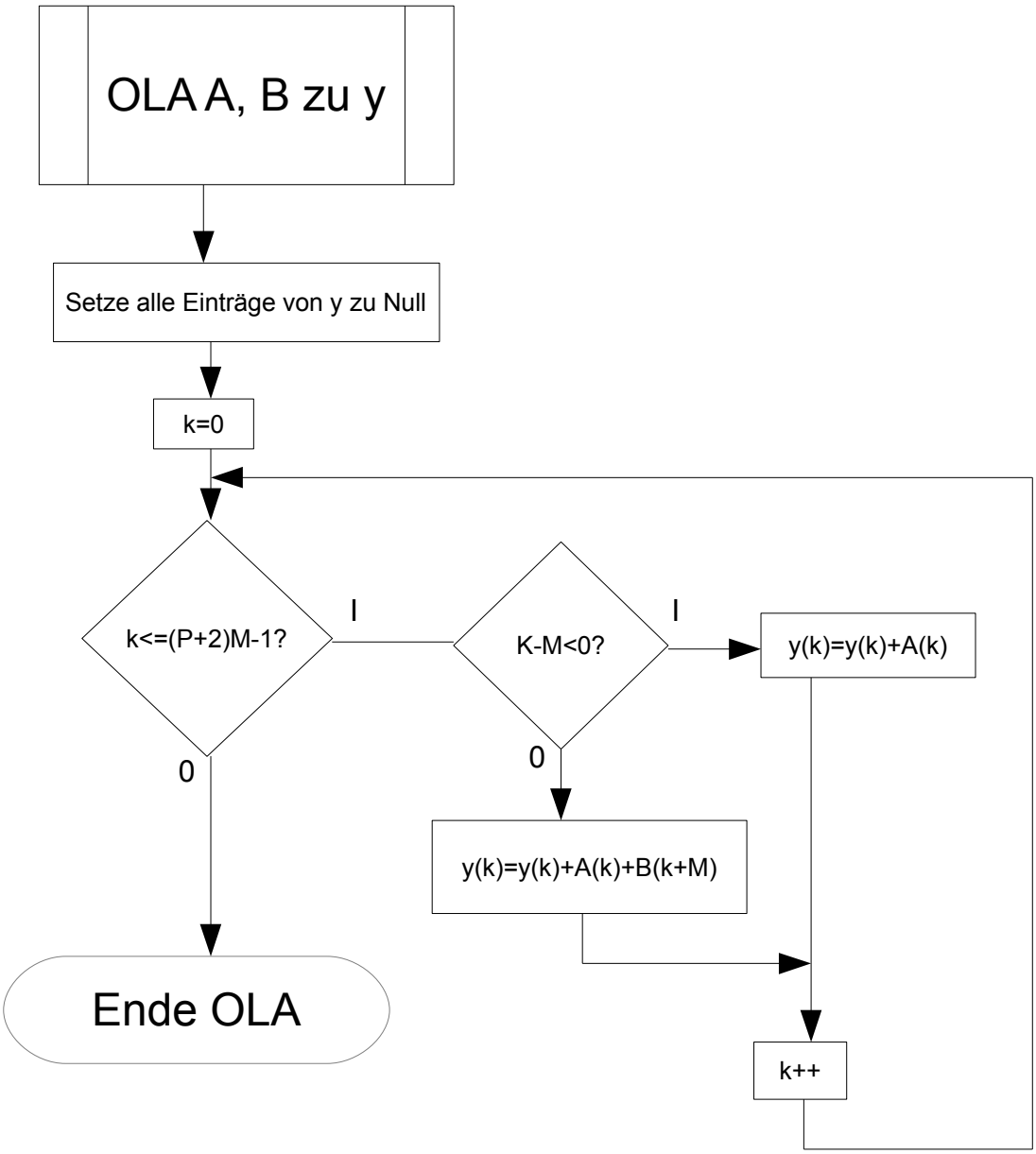
Anhang

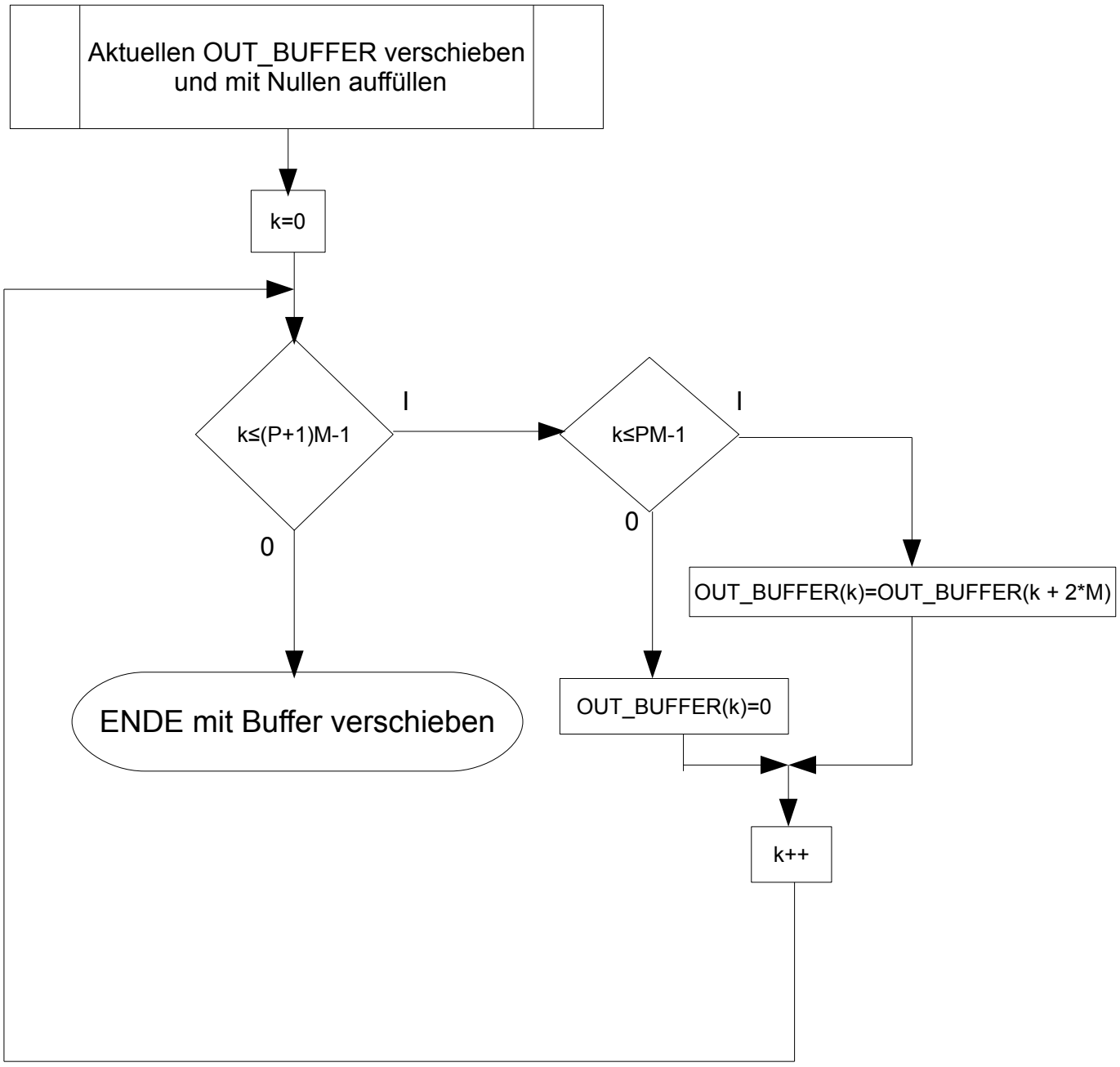

ISR AIC_RX
Eingangsdaten einlesen
Rechten Kanal abspeichern
Nächsten Ausgabewert in y_N laden
Schreibe Ausgabewert an Codec
pos=Size-2?
1
pos=1
0
pos=Size-1?
1
Buffer zwischenspeichern
BUFFER_FULL=1
0
pos=pos+2
pos=0
pos_OUT=M-1?
1
BUFFER_FINISH++
0
BUFFER_FINISH=2?
0
1
pos_OUT=0
pos_OUT++
Ende ISR


main
Koeffizienten aus Header laden
Speicher für Vektoren und Matrizen reservieren
DSK initialisieren, Codec starten Abtastfrequenz auswählen
Matrix mit Drehoperator erstellen
Interrupts aktivieren
Kippschalter auslesen LEDs setzen
BUFFER_FULL=0?
1
0
in_spektrum=in
BUFFER_FULL=0
FFT des Eingangssignals berechnen, Bitreihenfolge drehen
Elementweise Multiplikation der Spektren
Bitreihenfolge ändern
IFFT der Teilfaltungen
Auf Anzahl der Elemente normieren
Bestimmung der Teilfaltung
Vektoren A, B löschen und mit 0 auffüllen
OLA der Spaltenvektoren von a, b zu Vektor A, B
OLA von A, B zu y
first_spektrum=0?
0
1
OUT_BUFFER= y
first_spektrum=1
pos_OUT=0
BUFFER_FINISH<2?
0
Aktuellen OUT_BUFFER verschieben und mit Nullen auffüllen
OUT_BUFFER=OUT_BUFFER+y
BUFFER_FINISH=0



A, a wird im nächsten Durchlauf mit B, b ersetzt


OLA A, B zu y
Setze alle Einträge von y zu Null
k=0
k<=(P+2)M-1?
1
K-M<0?
1
y(k)=y(k)+A(k)
0
0
y(k)=y(k)+A(k)+B(k+M)
Ende OLA
k++


Aktuellen OUT_BUFFER verschieben
und mit Nullen auffüllen
k=0
k≤(P+1)M-1
1
0
k≤PM-1
1
0
OUT_BUFFER(k)=OUT_BUFFER(k + 2*M)
OUT_BUFFER(k)=0
ENDE mit Buffer verschieben
k++