5 Lösungs- bzw. Realisierungsansätze für die digitale Nachbildung der "Ti-TP Kette"

5.1 In der Diplomarbeit verwendeter Controller *(DSP "C6713")*

Wie im vorangegangenen Kapitel bereits vorgestellt, wird zur Lösung ein Controller benötigt, der die Berechnungen mit ca. 250MMACs durchführt. Zusätzlich sollte eine bereits an der HAW vorhandene Hardware bevorzugt werden, da so zunächst keine Kosten entstehen. Die dafür zur Verfügung stehenden Programmierbords sind das "Virtex2-Board" *(FPGA)* und das "TMS320C6713 DSK Board" *(DSP)*.

Der FPGA wird mittels VHDL programmiert. Hierdurch wird eine Implementierung der Arithmetik im Vergleich zu einem in C geschriebenen Code komplizierter *(speziell bei IIR-Filtern)*. Der Grund dafür ist, dass bei der mitgelieferten Softwareversion von "Integrated Software Environment" *(ISE)* der "VHDL Rev. 93"-Standard fossiert wird. Dieser Standard sieht keine "Floating-Point Rechnung" vor *(standardmäßig nur integer Rechnung mit Addition, Subtraktion und Multiplikation sowie Division mit $x/2$ bzw. $2/x$ möglich)*. Daher wäre eine Float-Rechnung nur über einen zusätzlichen Prozessor wie z.B. den "Power PC" *(PPC)* möglich *(der PPC wird auf dem "Virtex2-Board" bereitgestellt)*. Da es sich beim PPC um einen RISK-Prozessor handelt, der nicht für Signalverarbeitungszwecke entwickelt wurde und dieser zusätzlich nur über ein relativ langsames Speicherinterface verfügt, ist die Performance geringer als beim DSP. D.h. es können nur Filter mit kleinen Ordnungszahlen realisiert werden. Beim DSP-Bord hingegen ist der dazugehörige Prozessor *(C6713)* für Signalverarbeitungszwecke ausgelegt. Der Quelltext des DSP-Bords wird zudem in "C" programmiert. Daher wurde eine Lösung mit diesem Bord angestrebt. Aus dem Datenblatt zum TMS320C6713 DSK-Bord geht hervor, dass der C6713 bei einer Taktfrequenz von 225MHz mit den zwei vorhandenen Hardwaremultiplizierern *(Floating- und Fixed-Point)* 450MMACs [11] schafft. Für die Berechnung des IIR-Filters, zur Nachbildung des Kanalmodells, *(max. 262,4 MMACs)* ist dieses Bord also ausreichend

Abb. 5.1:
Das komplette Starter-Kit (DSK) für den TMS320C6713 DSP

dimensioniert. Des weiteren könnte der "C6713" mit 1350 Millionen Fließ-Komma Operationen oder 1800 Millionen Anweisungen. Mit diesen Informationen kann die Laufzeit eines Programms im voraus ermittelt werden.

5.1.1 Code Composer Studio *(CCS)*

Das Code Composer Studio ist die Programmieroberfläche für die C6000 Serie von Texas Instruments *(TI)*, wozu auch der C6713 zählt. Es wird in dem Starterkit *(Abb. 5.1)* mitgeliefert. CCS enthält Tools zur Codeerzeugung *(Compiler, Assembler und Linker)* und unterstützt real-time debugging. Zudem ermöglicht ein einfach zu bedienendes Software-tool das Erstellen *(build)* und Debuggen von Programmen.
Der C-Compiler kompiliert ein C-Quellprogramm *(*.c)*, um eine "Assembly-Quelldatei" *(*.asm)* zu erstellen. Der Assembler assembliert eine Quelldatei *(*.asm)*, um eine "Maschinensprachen-Objekt-Datei" *(*.obj)* zu erstellen. Der Linker kombiniert Objekt-Dateien und Objekt-Bibliotheken *(as input)* um eine ausführbare Datei mit der Endung *(extension)* *.out zu erzeugen *(ähnlich Common Objekt File Format oder auch "COFF")*. Diese ausführbare Datei kann geladen und direkt auf dem C6713 ausgeführt werden.
Auch kann eine Mischung *(Kreuzung)* aus C und Assembly-Code verwendet werden *(*.sa)*. Ein "Linear-Optimierer" optimiert diese Quelldatei, um eine Assembly-Datei zu erstellen *(genau wie der C-Compiler)* s.a. [6].

5.1.2 Berechnungszeit für ein IIR-Filter mit dem "TMS320C6713 DSK-Board" und Programmoptimierungen

Zunächst wurde das bereits in Kapitel 4.5 verwendete Filterprogramm auf das DSK-Bord übertragen *(s.a. Anhang E.1)*. Die ermittelte Berechnungszeit für das Filter 2. Ordnung *(9MAC)* beträgt 0,55µs *(1,82MHz)*. Für die geforderte Anwendung reicht dies, in Bezug auf einen Oversamplingfaktor von 100, allerdings nicht aus. Benötigt werden 156,25ns *(6,4MHz)*. Das heißt es werden gerade einmal $9MAC \cdot 1{,}82MHz = 16{,}364MMACs$ erreicht, laut Datenblatt sollen es allerdings 450MMACs sein. Nun gibt es diverse Möglichkeiten zur Programmbesserung die im folgenden Teil vorgestellt werden, da der C-Code nicht optimal umgesetzt wird.

Mit den CCS eigenen Optimierungsstufen

Das CodeComposerStudio *(CCS)* hält einige Optimierungsoptionen bereit. Eine Kurzanleitung findet sich in [6].

Wendet man diese Optimierungen auf das Testprogramm an, gilt für die –o3 und –o2 Stufe, dass das Programm nicht mehr funktioniert. Mit der –o1-Stufe wird die Berechnungszeit allerdings schon um über 150% verbessert. Für ein Filter 8. Ordnung *(35MAC)* wurde eine Durchlaufzeit so eine Durchlaufzeit von ca. 390ns ermittelt. Die ist für die geforderte Anwendung ausreichend, das so ein Oversampling-Faktor von 40 für die 8. Ordnung erreicht wird.

Weitere Möglichkeiten zur Programmoptimierung bzw. Verkleinerung der Berechnungszeit für das Filter:

Mit Hilfe fertiger Programme:

In der Hilfe von CCS findet sich folgendes Programm unter dem Thema IIR:

```
void iir(short x[],short y[],short c1, short c2, short c3)
{
  int i;  for (i=0; i<100; i++) {
       y[i+1] = (c1*x[i] + c2*x[i+1] + c3*y[i]) >> 15;
       }
}
```

Auf den ersten Blick fällt die Bitverschiebung nach rechts auf *(>>15)*. Diese wird benötigt, da bei der Multiplikation von zwei Short-Variablen der resultierende Integer-Wert $(2^{15} \cdot 2^{15} = 2^{30})$ "gecastet" werden muss, um anschließend wieder als Short-Variable verarbeitet werden zu können $(2^{30} >> 15 = 2^{15})$. Daher wird durch die Verschiebung nur das obere "Wort" genutzt und ein Short-Wert entsteht. Erstaunlich ist zunächst, dass 2^{15} statt 2^{16} geschrieben wird, da es sich jedoch um vorzeichenbehaftete Werte handelt, ist dies korrekt.

Fixpoint-Rechnung statt Floating-Point

Bei einem IIR-Filter in Festkommarechnung müssen Quantisierungseffekte *(bzw. Grenzzyklen)* beachtet und Gegenmaßnahmen getroffen werden *(Kaskadenstruktur, Abrundung, begrenzende Artithmetik statt Überlauf, ...)*. Ganz besonders gilt dies bei IIR-Filtern 8.

Ordnung, da hier durch die relativ kleinen Filterkoeffizienten eine große Abweichung im Berechnungsergebnis entsteht. Es macht nur Sinn, mit so hohen Ordnungszahlen zu rechnen, wenn mit extrem hoher Genauigkeit gerechnet wird, ansonsten bringen die höheren Ordnungen keinen Vorteil. Dies ist darauf zurückzuführen, dass die Pol/Null-stellen nach der Quantisierung außerhalb des Einheitskreises liegen können. Speziell die Pole, die bereits vor der Quantisierung dicht am Einheitskreis liegen.

Schnell entseht dabei der Gedanke, die Koeffizienten *(z.B. 0,0053)* einfach um den Faktor 10.000 zu erhöhen und das Berechnungsergebnis wieder zu dämpfen, indem mit 0.0001 multipliziert wird.

Im Prinzip funktioniert es auch so. Die Koeffizienten werden so skaliert, dass sie in den gewünschten Wertebereich passen. Das Ergebnis der Rechenoperation wird dann auf die gewünschte Bitbreite abgeschnitten bzw. gerundet. Wie diese Skalierung bei IIR-Filtern "genau" durchführt wird, ist sehr unterschiedlich und teilw. sehr komplex. Es gibt kein Patentrezept, daher gilt es, das Verhalten durch Simulationen und Versuchen am realen Filter anzupassen. Dabei sind unterschiedliche Variablentypen zu untersuchen *(Tab. 5.1)*.

Short	-32 768 *(-Umax-1)* bis 32 767
unsigned short	65 535
unsigned integer	4 294 967 295
unsigned long	1 099 511 627 775
unsigned long long	18 446 744 073 709 551 615
float	1.17549435e-38 bis 3.40282347e+38

Tab. 5.1:
Variablentypen und der entsprechende Darstellungsbereich

Im ersten Schritt werden die Koeffizienten betrachtet. Es wird untersucht, wie sich das Filter verhält, wenn anstatt mit den vom Filterdesigner berechneten Koeffizienten *(in quasi unbeschränkter 64 Bit Float-Genauigkeit)* mit den quantisierten Koeffizienten gerechnet wird. Bei 16 Bit und einem IIR-Filter 8. Ordnung wird das nicht mehr viel mit der gewünschten Übertragungsfunktion zu tun haben.

Im zweiten Schritt werden Verfahren untersucht, die die Quantisierungsfehler beim Rechnen kontrollierbar machen *(Bitbreite, Struktur, Grenzzyklen usw.)*. Analytisch werden meist keine funktionsfähigen Lösungen erzielt, es gilt, wie bereits erwähnt, zu simulieren um herauszufinden ob die Fehler akzeptabel sind. Für högere Ordnungszahlen empfiehlt sich auch die Zerlegung in verkette Systeme 2. Ordnung

Fractional rechnen

Fractional rechnen bedeutet, dass eine im 16Bit-Bereich darstellbare Zahl *("Ganze Zahl" dient einer bequemeren Interpretation)* in den Zahlenraum von -1..0..1 "transformiert" wird *("..1" ist nicht ganz richtig. Da die 0 auch dargestellt werden muss, gibt es im positiven Bereich ein Bit zu wenig, um auf die 1 zu kommen)*.

Das heißt, dass $2^{15} = 32767 \,\mathbf{A}\, 1$ *(genauer: $1 - 2^{-15}$)* ist und $-2^{15} = -32768 \,\mathbf{A}\, -1$. Zu bedenken ist, dass das MSB das Vorzeichen ist. Der Sinn dahinter ist u.a., dass sich ohne Überlaufe multiplizieren lässt *(z.B.: $< 1 \cdot < 1$ bleibt < 1)*. Diese 'fractionals' gibt es auch vorzeichenlos und mit mehr Bits; je nachdem muss der "Arithmetic Logical Unit" *(ALU)*, welche die Recheneinheit in einem Controller darstellt, vorgegeben werden, wie das Bitmuster interpretiert werden soll. Näheres in Literatur zum DSP *(Progamming Reference)* [11]. Beim realen Modell wurden so eine um 50% verbesserte Durchlaufzeit ermittelt, allerdings wurde die Quantisierung nur für die 1. Ordnungszahl erfolgreich durchgeführt.

Assembler-Dateien einbinden

Ein einbinden von Assemblerquelltext bringt keinen Zeitgewinn bei der Berechnung des Filteralgorithmus, da der in C geschriebene Programmcode vom Compiler in den meisten Fällen besser umgesetzt wird als ein selbst entwickelter Assemblercode *(zumindest mit dem CCS Optimierungslevel –o3)*.

Um die Ausführung der Dauerschleife im Hauptprogramm oder die für den Oversampling faktor zuständige While-Schleife zu beschleunigen stehen grundsätzlich zwei Verbesserungsmöglichkeiten zur Auswahl.

1. Hardwareloops

Hardwareloops sind Schleifen, die nicht durch logische Abfragen im Programm beendet bzw. wiederholt werden, sondern per direktem Registervergleich in der Hardware abgehandelt werden. D.h. ein Codesegment im Speicher wird solange wiederholt, bis der Wert im Loop-Counter-Register erreicht ist. Dazu wird der Schleife Anfangs- und Endadresse bekannt geben.

2. Circular Buffering

Hierfür wird ein ähnliches Verhalten wie es bei "Hardwareloops" entsteht erzeugt. "Circular Buffer" beschreibt eine Datenstruktur, die einen einzelnen Puffer fester Größe benutzt, als wäre dieser end to end verbunden.

Der Address-Arithmetic-Unit wird Anfangsadresse *(Basis)* und Länge eines Speicherbereiches bekannt gegeben. Zu diesen Angaben gehört ein Indexregister *(aktueller Pointer in diesen Bereich)*. Nun kann dieser Pointer incrementiert *(o. decrementiert)* werden. Hardware-mäßig wird geprüft, ob der angegebene Bereich verlassen wird, falls dies der Fall ist, wird ein Reset des Zeigers ausgelöst und dieser zeigt wieder auf die Basis.

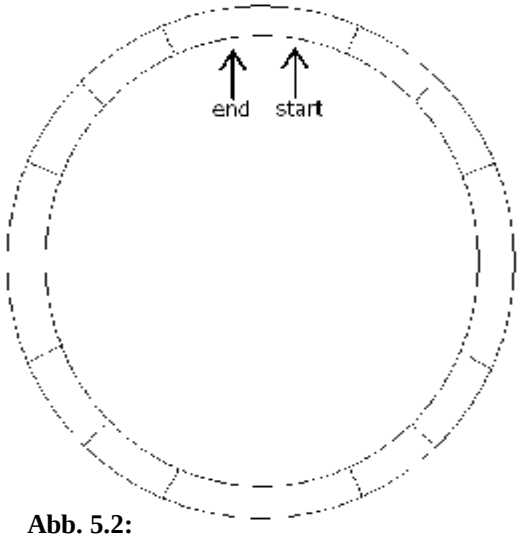

Abb. 5.2:
"Circular Buffer"

Da "Hardwareloops" und "Circular Buffering" hardwaremäßig geschehen und parallel laufen *(können)*, sind sie extrem schnell und effizient.

5.2 Realisierung von IIR-TP-Filtern mit dem "AIC23" on-board-Codec

Um ein analoges Eingangssignal für die Verarbeitung mit dem C6713 zu digitalisieren, wird ein Analog-Digital Wandler (ADC) benötigt. Zur Ausgabe der digital bearbeiteten Werte wird im Allgemeinen ein Digital-Analog-Wandler (DAC) verwendet. Schaltungen, bei denen eine gemeinsame Taktleitung die Abtast- bzw. Updaterate dieser beiden Wandler synchron steuert, werden als Codecs bezeichnet. Da das TMS30C6713 DSK-Bord einen 96KHz "on-board" Codec *(AIC23)* zur Verfügung stellt, wurden erste Tests mit diesem Codec durchgeführt.

Dazu wurde eine theoretische "Ti-TP Kette" mit der Zeitkonstanten *T=12,7324µs* entworfen *(anstelle von T=1,59155µs beim bestehenden Kanalmodell)*. Die entsprechenden analogen Filter der Kette beeinflussen ein 8Kbit/s Eingangssignal exakt wie das eigentliche Kanalmodell ein 64Kbit/s Eingangssignal *(s.a. Anhang E.2)*.

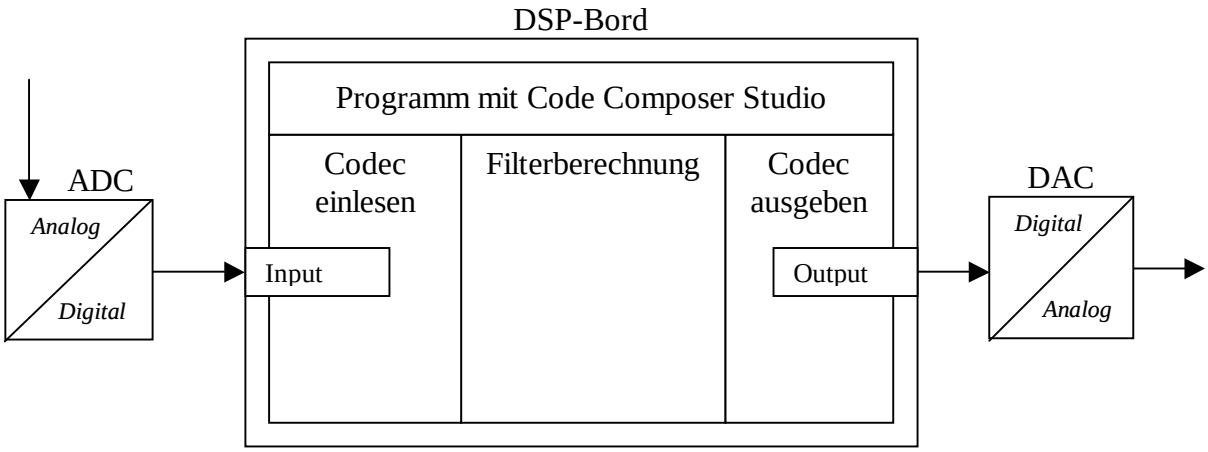

Abb. 5.3:
Prinzipschaltbild des Testaufbaus mit Codec

Da der AIC23 maximal 96000 Werte pro Sekunde ausgeben kann, wird ein Oversampling-Faktor von 12 erreicht. Mit diesem Oversamplingfaktor wird nach Kap. 4.2 der gewünschte Signalverlauf für erste quantitative Vergleiche der analogen mit den digitalen Filtern ausreichend approximiert *(s.a. Abb 5.4)*.

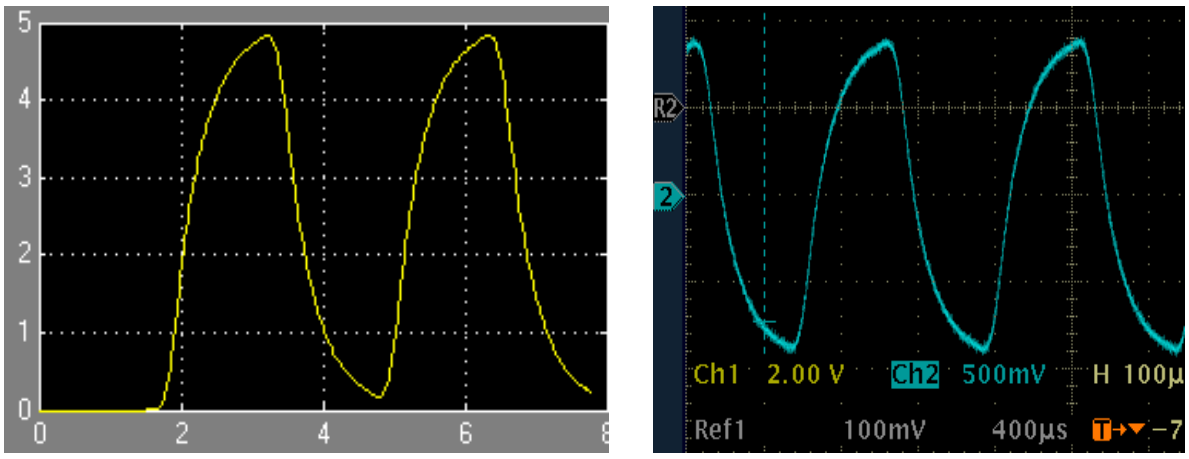

Abb. 5.4:
Einfluss des digitalen Filters *(OZ2, bilinear)* auf ein 8Kbit/s Eingangssignal mit 12-fach Oversampling; Im linken Bild mit Matlab simuliert und rechts die Oszilloskopmessung am realisierten System *(mit DSP und AIC23-Codec)*.

Ausführliche Informationen zu diesen Messungen, sowie ein Vergleich der Amplitudengänge und Signalspektren befindet sich im Anhang E. Anhand der Messungen konnte neben den Anforderungen für einen Codec ein Eindruck über die zusätzlich benötigte Hardware wie z.B. Antialias- und Interpolationstiefpass gewonnen werden.

5.3 Realisierung von IIR-TP-Filtern mit dem "ADDA16" *(500KHz-Codec)*

Da die analogen Filter mit dem AIC23 durch den relativ geringen Oversampling-Faktor nur schlecht nachgebildet werden, wurde zur Ermittlung der später benötigten Oversampling-rate *(evtl. abweichend von der Matlab-Simulation)* der schnellste vorhandene Codec *(ADDA16 Aufsteckmodul)* benutzt. Dazu wurde das gleiche Programm wie für den AIC23 mit ein paar Anpassungen verwendet. D.h. ein Eingangssignal von 8Kbit/s wird mit 50-fach-Oversampling abgetastet. Das Ausgangssignal wird dann mit 400KWerten/s ausgegeben. Die entsprechenden Filterkoeffizienten für die digitale "Ti-TP Kette" befinden sich in Anhang E.3.

Bei den Messungen wurde durch eine quantitative Bewertung der Augenöffnung[1] fest-gestellt, dass ein im Zeitbereich einwandfrei nachgebildetes Signal bei einem Over-samplingfaktor von ca. 25 erreicht wird. Diese Erkenntnisse gilt es mit in die digitale Nachbildung der analogen "Ti-TP Kette" des bestehenden Kanalmodells zu übernehmen.

[1] *Anhand der Augenöffnung wird die Übertragungsqualität ermittelt [1]*

5.4 Synchronisierung von Bitgenerator und Bord

Die in Kap. 5.2 und Kap. 5.3 vorgestellten digitalen Systeme können eine exakte Nachbildung des analogen Kanalmodells auf die vorgestellte Weise nicht realisieren, da die Abtastfrequenz des Codecs *(8KHz)* nicht exakt synchron mit der Frequenz des digitalen Eingangssignals *(8Kbit/s)* übereinstimmt. Dadurch werden je nach Abweichung manche Bits überhaupt nicht abgetastet bzw. zweimal abgetastet.

Um dieses Problem zu lösen wurden zwei mögliche Lösungswege betrachtet. Die erste Möglichkeit, um die Abtastfrequenz des Codecs mit der vom Generator erzeugten Eingangssignalfrequenz zu synchronisieren wäre ein Frequenzteiler. Hierbei wird der Generator des digitalen Eingangssignals von einer externen Quelle, in diesem Fall dem Frequenzteiler, getaktet. Der Frequenzteiler selbst wird von einem auf dem DSK-Bord verwendeten oder bereitgestellten schnelleren Takt getaktet und teilt diesen durch einen Faktor. Dieser geteilte Takt wird in den externen Takteingang des Generators geführt. Das DSK-Bord selbst nutzt zur "on-board" Takterzeugung einen Masterclock, der heruntergeteilt wird. Dadurch wird der vom Frequenzteiler getaktete Generator synchron zur Abtastfrequenz des Codecs erzeugen. Allerdings sind Teiler ohne großen Hardwareaufwand sehr unflexibel bezüglich ihrer Teilerfaktoren, wodurch die gewünschte Taktfrequenz für den Generator nicht exakt erreicht wird. Da diese Aufgabe jedoch den hohen Hardwareaufwand, den ein Teiler für die exakte Erzeugung der gewünschten Frequenz, wie z.B. ein Johnson Counter *(s.a. Anhang E.4)*, mit sich bringen würde, nicht rechtfertigt, wurde der zweite Lösungsansatz über Interrupts bevorzugt und später auch realisiert.

Hierbei löst der vom Generator erzeugte Ausgangstakt einen Interrupt im Programm aus, der das Einlesen des Eingangswertes beinhaltet. Im Hauptprogramm wird dann z.B. die Filterberechnung ausgeführt. Die Synchronisierung des Eingangssignal vom Generator mit der Abtastfrequenz des Codecs stellt sich daher auf die Weise für die geforderte Anwendung als am besten geeignet heraus. Jedoch ist zu beachten, dass ein Signal mindestens 17,8ns am externen Interrupt-Pin anliegen muss, um den entsprechenden Interrupt zuverlässig auszulösen.

Die Betrachtung des vom Generator erzeugten Taktsignals findet sich in Kapitel 3.1. Bei der Messung wurde festgestellt, dass das Taktsignal 800ns lang den High Pegel hält, was zur zuverlässigen Auslösung des Interrupts absolut ausreichend ist. Allerdings ist zu beachten, dass der Interrupt bei fallender Flanke auslöst, da das Taktsignal 100ns vor dem

Datensignal anliegt. Würde der Interrupt bei steigender Flanke auslösen, wäre es nicht sicher, dass der aktuelle Eingangswert des digitalen Datensignals erfasst wird, da vom Anliegen des Interruptsignals bis zum Einlesen des aktuellen Datenwertes voraussichtlich keine 100ns vergehen werden.
Ein System, welches auf diese Weise mit dem Generator synchronisiert wird, findet ab Kapitel 6.1.2 zur Nachbildung des bestehenden Kanalmodells Verwendung und wird im Anhang F.2 vorgestellt.

5.5 Synchronisierung mit Interrupt-Programm

Da beim verwendeten DSP Bord nur die Interrupts 4 bis 7 zur externen Verwendung an der Steckerleiste J3 bereit gestellt werden, wurden nur diese für eine Lösung der Aufgabe verwendet. Als Interrupt Vektortabelle wurde zunächst die in [6] auf der CD zur Verfügung gestellte Assembler Datei "Vectors_poll.asm" benutzt und später mit den Erkenntnissen aus [11] ergänzt. Es folgt die entsprechende, allgemeingültige Initialisierung für die Verwendung eines Interrupts [11]. Voraussetzung für die Initialisierung ist das Einbinden der Chip-Select Libraries *(CSL)* "csl.h" und "csl_irq.h", die Programmierschnittstellen *(APIs)* für das Interruptsystem und eine Reihe von Echtzeitfunktionen bieten.

- Das NMI-Flag muss über die CSL-Routine *IRQ_nmiEnable* gesetzt worden sein.
- Das GIE-Flag (Global Interrupt Enable) muss über die Routine *IRQ_globalEnable* gesetzt worden sein.
- Der/die individuellen Interrupt(s) müssen über die Routine *IRQ_enable* freigegeben worden sein, die das entsprechende Bit im „Interrupt Enable Register“ *IER* setzt.
- Über *IRQ_disable* können Interrupts auch zeitweise gesperrt werden.
- An der zugehörigen Vektortabellen-Adresse muss ein Branch zur ISR stehen.

Für die Realisierung des Kanalmodells wurde die Interrupt Initialisierung aus [11] in das bisherige Programm für das digitale Kanalmodell übertragen und angepasst. Bei der Anpassung der Initialisierung an die geforderte Funktionsweise ist, wie bereits in Kapitel 5.4 erwähnt, zu beachten, dass der Interrupt bei fallender Flanke auslöst. Dies wird im "GPPOL"-Register festgelegt. Das entsprechende Register wird mit allen anderen benötigten Registern in Kapitel 6.1.2.2 vorgestellt.

5.6 Anforderungen an eine fertige DSP-Bord Lösung mit Codec und Angebote

Da an der HAW kein Codec mit ausreichender Updaterate zur Verfügung steht wurden zunächst Angebote für eine Lösung, bestehend aus Programmierbord und Codec-Aufsteckmodul, mit passenden Leistungsmerkmalen eingeholt. Das Bord sollte nicht nur das beschriebene analoge Kanalmodell ersetzen können, sondern auch für andere Aufgaben verwendbar sein und daher folgende Bedingungen erfüllen:

- Gleicher Prozessor wie der im DSP-Labor eingesetzte *(C6713)*
- Hinreichend schnelle Abtastung *(10MHz pro Kanal)* mit flexibler Einstellung der Abtastraten
- Mehrere analoge Multiplex-Eingänge/Ausgänge *(Kanäle)*
- Codec mit mindesten 12Bit Auflösung
- PC externes Programmierbord mit USB- Anschluss zur Anbindung an einen PC

Die Firma GBM bietet eine relativ breite Produktpalette mit entsprechenden Geräten an, woraufhin eine Anfrage bei der Firma erfolgte. Die Angebote für Programmierbords mit entsprechenden Leistungsmerkmalen befinden sich im Anhang E.5.
Daraus geht hervor, dass mit einer komplett vorbereiten Hardware, der in der Aufgabenstellung vorgegebene preisliche Rahmen nicht eingehalten werden kann. Auch bei anderen Anbietern war keine Annäherung an die preisliche Vorgabe von ca. 300€ zu erreichen.
Deshalb wurde dazu übergegangen, eine Lösung mit dem vorhandenen TMS320C6713-DSK-Board und selbst entwickelten Codec zu realisieren. Dabei bietet es sich an, aufgrund des digitalen Eingangssignals den AD-Wandler des Codecs komplett entfallen zu lassen, indem das Eingangssignal direkt auf einen Eingangspin gelegt wird, dessen Zustand abgefragt wird *(s.a. Kap. 6.1)*. Somit wird nur noch ein DA-Wandler zur Ausgabe der Signale benötigt. Für andere Anwendungen kann dann bei Bedarf noch ein AD-Wandler entwickelt werden.
Um einen DAC mit ausreichender Updaterate zu finden, wurden die angebotenen Wandlermodule näher betrachtet. Diese verwenden drei unterschiedliche DAC ICs *(AD9764, AD9765 und AD9774; s.a. Anhang E.6)* der Firma Analog Devices, die sich auch für die geforderte Aufgabe eignen. Daher wurde die entsprechende Produktpalette bei der Suche nach einem geeigneten Wandler mit einbezogen.