

### **DSP-Labor – Durchführung und Auswertung**

In der Laborübung zur Digitalen Signalverarbeitung (LV Methoden der Signalverarbeitung im BEL-EK und LV Signalverarbeitung III im BEL-ES) sollen Sie Kenntnisse der digitalen Signalverarbeitung aus dem seminaristischen Unterricht durch praktische Übungen vertiefen sowie den Aufbau und die Programmierung eines modernen digitalen Signalprozessors (DSP) kennen lernen. Hierzu werden Sie in Gruppen mit maximal drei Studierenden an acht (BEL-EK) bzw. zwölf (BEL-ES) Terminen drei bis fünf Versuche zu den Themen

1. Analog-Digital-Umsetzung und digitale Signale
2. FIR-Filter
3. IIR-Filter
4. DTMF-Detektor mit FFT (optional für BEL-EK, Pflicht für BEL-ES)
5. DTMF-Detektor mit Goertzel-Algorithmus für FFT (Pflicht für BEL-ES)

bearbeiten.

Im Folgenden werden die Aufgaben beschrieben, die Sie zu den jeweiligen Themen im Labor bearbeiten sollen.

Die zur Durchführung benötigten vorbereiteten Quellcodes (Templates) finden Sie in unter [public.beuth-hochschule.de/~purat](http://public.beuth-hochschule.de/~purat) oder in moodle.

#### Auswertung:

Für das Protokoll sind alle während des Labortermins gemäß den Aufgabenstellungen erstellten oder modifizierten Programme zu kommentieren und kurz zu beschreiben. Die vollständigen und kommentierten Programme können Sie im Anhang des Protokolls auflisten. Wenn Sie Änderungen an den Programmen vornehmen, speichern Sie diese jeweils unter einem neuen Namen ab, so dass sie alle modifizierten Programme für das Protokoll zur Verfügung haben. Die von Ihnen hinzugefügten oder modifizierten Programmzeilen sollten Sie im Hauptteil des Protokolls mit den Erläuterungen (in notwendiger Ausführlichkeit und hinreichender Kürze) erläutern.

Darüber hinaus sind die während der Durchführung erworbenen Messergebnisse zu protokollieren und sinnvoll zu kommentieren, d.h. insbesondere mit erwarteten Ergebnissen zu vergleichen bzw. zu begründen. Achten Sie bei den Messungen darauf, dass Sie die Wertebereiche (z.B. für Zeit, Frequenz, Amplitude etc.) so einstellen, dass die zu messenden bzw. verifizierenden Darstellungen (Frequenzgang, Spektrum, Zeitsignal deutlich zu erkennen sind. Die für die Auswertung relevanten Punkte der Durchführung sind im Umdruck hervorgehoben. Zu jedem der 3-5 Versuche ist von jeder Gruppe ein Protokoll anzufertigen. Neben den Messergebnissen und Programmen sind die gesondert angegebenen Aufgaben zu bearbeiten und ins Protokoll mit aufzunehmen.

## 1. Übung (Analog-Digital-Umsetzung und digitale Signale)

In der ersten Laborübung soll zunächst das Verständnis für die Umsetzung zwischen analogen und digitalen Signalen vertieft werden. Es soll deutlich werden, wie die analogen Signale auf dem DSP digital repräsentiert werden. Darüber hinaus sollen erste Programmiererfahrungen mit dem DSP erworben werden.

### 1.1 Einlesen von Signalen:

Die Aufgabe besteht darin, die Signalwerte, die der CODEC an den DSP liefert darzustellen und mit dem analogen Eingangssignal zu vergleichen.

Schließen Sie zunächst die Versorgungsspannung an das Evaluationsboard an und verbinden Sie dann das Board per USB-Kabel mit dem PC. Warten Sie bis die „USB\_MONITOR“ LED („Monitor“, „ZLED03“) auf dem Board leuchtet und starten dann die VisualDSP++ Entwicklungsumgebung (*Integrated Development Environment*, IDE) aus dem Windows-Start-Menü oder mit dem entsprechenden Icon auf dem Desktop. Achten Sie darauf, dass als Session Target das „ADSP-BF561EZ-KIT Lite via Debug Agent“ eingestellt ist. Wenn das nicht der Fall ist, können Sie über den Menüpunkt „Session“/„Select Session“ das Target „ADSP-BF561EZ-KIT Lite via Debug Agent“ auswählen. Der PC kann nun mit dem Board über die USB-Schnittstelle kommunizieren.

Erstellen Sie ein neues Projekt<sup>1</sup> in Ihrem Gruppenverzeichnis mit einer Standard-Applikation für den ADSP-BF561 (File→New→Project)<sup>2</sup>. Die IDE erstellt in Ihrem Gruppenverzeichnis automatisch einen neuen Projektordner. Laden Sie die Templates für die Übung 1.1 von Moodle herunter und legen Sie sie in diesem Projektordner ab<sup>3</sup>:

main.c	Initialisierung des Codecs und Endlosschleife
isr.c	Interrupt-Service-Routine, die vom seriellen Port ausgelöst wird, nachdem ein vollständiger Datenrahmen empfangen wurde
isr.h	Definitionsdatei für isr.c
copydata.c	Modul mit Kopierfunktion
copydata.h	Definitionsdatei für copydata.h
codeclib.dlb	Library mit Initialisierungsroutine für den Codec
codeclib.h	Definitionsdatei für codeclib.dlb
adsp-BF561-codec.ldf	Linker Deskription File mit den notwendigen Informationen für den Linker (Speicherbereiche, Objekte, Prozessoren, ...)

Fügen Sie alle Dateien zu Ihrem Projekt hinzu (Project→Add to Project→File(s)). Fügen Sie in dem Modul copydata.c den in der Vorbereitung erstellten Quellcode (die Funktion copyData(...)) ein. Die Funktion copyData(...) wird bei jedem Interrupt aus der ISR aufgerufen; damit werden die Daten aus dem DMA-Buffer in den zyklischen Speicher geschrieben. Die Größe des zyklischen Speichers beträgt 32.

Erzeugen Sie dann das lauffähige Programm (Project→Build Project). Nach dem Build-Prozess kann das Programm automatisch über die USB-Schnittstelle auf das Evaluationboard geladen und ausgeführt werden. Die Entwicklungsumgebung stoppt die Ausführung des Programms automatisch nach dem Aufruf der Funktion main(). Neben dem C-Quellcode wird nun auch ein Disassembly-Fenster angezeigt, in dem der von der Entwicklungsumgebung erzeugte Befehlscode in disassemblierter (also für den Anwender wieder lesbarer) Form zu sehen ist.

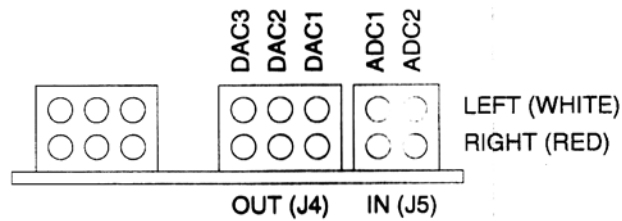
Die folgende Abbildung stellt die Anschlussbelegung der analogen Ein- und Ausgänge des EVB dar.

---

<sup>1</sup> Wählen Sie als Projektnamen einen Namen mit weniger als 9 Zeichen und ohne Umlaute, Sonderzeichen und Leerzeichen! Sonst kann es beim Übersetzen des Programmes zu Fehlermeldungen kommen.

<sup>2</sup> Wählen Sie hierfür folgende Optionen: Single-Core, kein C-Template, keinen eigenen Startupcode und kein eigenes LDF-File.

<sup>3</sup> Wenn sich die Source-Code Files nicht im Projektordner befinden, kann es beim Übersetzen zu Fehlermeldungen kommen.



Legen Sie ein Sinussignal mit einer Amplitude zwischen 1V und 2V und einer Frequenz zwischen 4 und 6 kHz an den rechten Kanal des ADC1 am EVB. Der ADC1-Kanal des EVB entspricht dem internen ADC 0 des Codecs, dessen Werte Sie gemäß der vorbereiteten Funktion `copyData()` einlesen.

Nehmen Sie das Signal mit dem Oszilloskop auf. Als Oszilloskop können Sie das *Virtual Instrument* (VI) „FFT & FRF & Scope“ von Labview benutzen, das Sie auf dem Desktop Ihres PCs finden. Dieses VI analysiert die Signale an der Anschlussbox von National Instruments an Ihrem Arbeitsplatz. Verwenden Sie die Eingänge „ai0“ bzw. „ai1“ dieser Anschlussbox. Kopieren Sie die Darstellung in die Zwischenablage und dann in eine Anwendung (z.B. Word oder IrfanView), so dass sie die Darstellung für Ihr Protokoll benutzen können. Eine Anleitung zum VI finden Sie auf Moodle (Handbücher). Alternativ können Sie natürlich auch das Speicheroszilloskop auf Ihrem Arbeitsplatz benutzen.

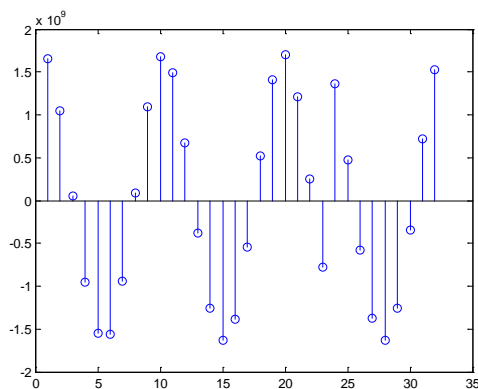
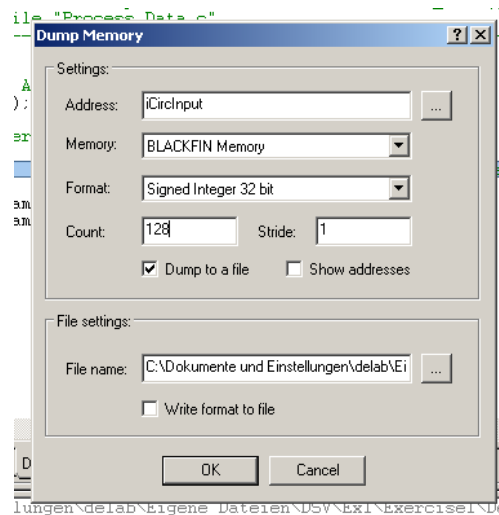
Starten Sie das Programm (Debug→Run). Der Befehlscode wird nun auf dem DSP ausgeführt, d.h. es werden fortwährend Signalwerte vom Codec eingelesen und sukzessive in den zyklischen Speicher geschrieben.

Stoppen Sie das Programm (Debug→Halt). Es müssten sich nun unter der von Ihnen gewählten Adresse des zyklischen Speichers die letzten 32 eingelesenen Signalwerte im DSP-Speicher befinden. Zur besseren Visualisierung dieser Signalwerte sollen diese in MATLAB exportiert werden. Dazu können Sie die Werte aus dem Speicher des DSP in eine Datei schreiben (Memory→Dump). Wählen Sie die Parameter entsprechend der unten dargestellten Abbildung und der von Ihnen gewählten Speicheradresse (in der Abb. iCircInput).

Starten Sie dann MATLAB und wechseln in Ihr Projektverzeichnis. Führen Sie dann die folgenden Befehle aus:

```
fid=fopen('input.dat','r');
x = fscanf(fid,'%d');
fclose(fid);
stem(x);
```

Es sollte sich dann in etwa ein Plot wie unten dargestellt ergeben. Kopieren Sie die Abbildung aus MATLAB in die Zwischenablage (edit → copy figure) und dann in eine von Ihnen gewählte Anwendung (z.B. Word oder IrfanView) für die Auswertung zu Hause.



### Auswertung:

- Das Programm (bzw. die notwendigen Änderungen) und die Messergebnisse sind in der Auswertung entsprechend zu kommentieren.
- Geben Sie eine mathematische Beschreibung für das zeitdiskrete Signal an, indem Sie näherungsweise die Parameter Amplitude  $A$  und normierte Kreisfrequenz  $\Omega_0$  für das Sinussignal aus dem gemessenen Plot bestimmen.
- Ermitteln Sie unter Verwendung der CODEC-Parameter aus dem Datenblatt Frequenz (in Hz) und Amplitude (in V) des analogen Signals aus den entsprechenden Werten des zeitdiskreten Signals und vergleichen Sie die Werte mit dem analogen Eingangssignal, das Sie oszilloskopiert haben.

### 1.2 Ausgeben von Signalen:

In dieser Aufgabe soll ein einfacher Sinusgenerator implementiert werden. Per Taster (SW6) soll die Frequenz in 200-Hertz-Schritten vergrößert und per Taster (SW7) entsprechend wieder verringert werden. Eine LED (LED5) soll anzeigen, dass der Generator die Frequenz 200 Hz ausgibt, eine zweite LED (LED6) soll anzeigen, wenn der Generator die Frequenz 400 Hz ausgibt, und eine dritte LED (LED7) soll anzeigen, dass die Frequenz (4 kHz) ausgegeben wird.

Erstellen Sie ein neues Projekt in Ihrem Gruppenverzeichnis mit einer Standard-Applikation für den ADSP-BF561 (File→New→Project). Die IDE erstellt in Ihrem Gruppenverzeichnis automatisch einen neuen Projektordner. Laden Sie die Templates für die Übung 1.2 aus dem Netz herunter und legen Sie sie in diesem Projektordner ab:

main.c	Initialisierung des Codecs und Endlosschleife
isr.c	Interrupt-Service-Routine, die vom seriellen Port ausgelöst wird, nachdem ein vollständiger Datenrahmen empfangen wurde.
isr.h	Definitionsdatei für isr.c

codeclib.dlb	Library mit Initialisierungsroutine für den Codec
codeclib.h	Definitionsdatei für codeclib.dlb
adsp-BF561-codec.ldf	Linker Deskription File mit den notwendigen Informationen für den Linker (Speicherbereiche, Objekte, Prozessoren, ...)

Fügen Sie alle Dateien zu Ihrem Projekt hinzu.

Die Tastersteuerung und LED-Anzeige ist im Prinzip bereits im Modul isr.c implementiert. Allerdings müssen die entsprechenden Initialisierungen der Kontrollregister der PFs noch hinzugefügt werden. Die Adressen der MMR sind in der Headerdatei „cdefBF561.h“ definiert. Die relevanten Registerinhalte lassen sich nun über die Pointer

```
short *pFIO0_DIR, *pFIO0_INEN, *pFIO2_DIR, *pFIO0_FLAG_D, *pFIO2_FLAG_D
```

auslesen bzw. beschreiben. Modifizieren Sie die Module main.c und isr.c entsprechend der Aufgabenstellung, so dass die Taster korrekt abgefragt werden und die richtigen LEDs leuchten.

Die sechzehn frei verfügbaren LEDs (LED5 bis LED20) des Boards werden durch programmierbare IO Pins (programmable flags PF32 bis PF47) des DSPs angesteuert (Mapping: LED5:PF40 ... LED12:PF47 und LED13:PF32 ... LED20:PF39). SW-technisch wird auf die LEDs über die FIO2 Register zugegriffen, die zu den MMR (memory mapped register) gehören. Durch Beschreiben der entsprechenden Speicheradressen werden also die 16bit-Register angesprochen. Zunächst muss die Richtung des Pins (input/output) konfiguriert werden, indem die Bits des Registers FIO2\_DIR auf 1 gesetzt werden. Dann können die Pins durch die Register FIO2\_FLAG\_D(ata)/C(lear)/S(set)/T(oggle) modifiziert werden.

Die vier Taster (SW6 bis SW9) sind mit den Pins PF5 bis PF8 verbunden (Mapping: SW6:PF5 ... SW9:PF8). Jeder als Input konfigurierte Pin muss zunächst über das Register FIO0\_INEN enabled werden. Der Status eines Tasters kann über das FIO0\_FLAG\_D(ata) register gelesen werden. Wenn das Bit gesetzt ist (1), ist der Taster gedrückt; wenn das Bit nicht gesetzt ist (0), ist der Taster nicht gedrückt.

Erstellen Sie ein weiteres C-Modul (z.B. gensinus.c) und eine zugehörige Definitionsdatei (z.B. gensinus.h), indem Sie Ihren in der Vorbereitung erstellten Quellcode (die Funktion genSinus()) samt notwendiger Parameterdefinitionen und Deklarationen einfügen. Ändern Sie das Modul isr.c so, dass die Funktion genSinus() aus der ISR aufgerufen wird und somit einmal pro Abtasttakt ein neuer Sinuswert erzeugt wird. Der Sinuswert soll über den rechten DAC1-Kanal des EVB (Internal DAC 0 des Codec) ausgegeben werden. Die normierte Amplitude  $A=1$  soll dabei der maximalen Aussteuerung des Codec-Ausgangs entsprechen.

Testen Sie den Funktionsgenerator für  $A = 0.5$  und nehmen entsprechende Signal-Oszillogramme für 200Hz, 400Hz, und 4kHz auf. Nehmen Sie auch die Spektren der Signale mit der FFT-Funktion des VI's auf.

### **Auswertung:**

- Das Programm (bzw. die notwendigen Änderungen) und die Messergebnisse sind in der Auswertung entsprechend zu kommentieren.

### **1.3 Verarbeiten von Signalen:**

In dieser Aufgabe sollen die eingelesenen Signale verarbeitet und gleich wieder ausgegeben werden. Die Verarbeitung beschränkt sich dabei auf einfache statische Signalmanipulationen wie Verstärkung, und Dämpfung. Die Verarbeitung wird zu Übungszwecken in Assembler programmiert.

Erstellen Sie ein neues Projekt in Ihrem Gruppenverzeichnis mit einer Standard-Applikation für den ADSP-BF561 (File→New→Project). Die IDE erstellt in Ihrem Gruppenverzeichnis automatisch einen neuen Projektordner. Laden Sie die Templates für die Übung 1.3 aus dem Netz herunter und legen Sie sie in diesem Projektordner ab:

main.c	Initialisierung des Codecs und Endlosschleife
--------	---

isr.c	Interrupt-Service-Routine, die vom seriellen Port ausgelöst wird, nachdem ein vollständiger Datenrahmen empfangen wurde.
isr.h	Definitionsdatei für isr.c
process_data.c	Modul mit der Verarbeitungsroutine
process_data.h	Definitionsdatei für process_data.c
codeclib.dlb	Library mit Initialisierungsroutine für den Codec
codeclib.h	Definitionsdatei für codeclib.dlb
adsp-BF561-codec.ldf	Linker Deskription File mit den notwendigen Informationen für den Linker (Speicherbereiche, Objekte, Prozessoren, ...)
isr.asm	Interrupt-Service-Routine in Assemblercode
process_data_s.c	Modul mit der Verarbeitungsroutine in 16 Bit
process_data_s.asm	Modul mit der Verarbeitungsroutine in Assemblercode

Fügen Sie alle Dateien (ohne die C-Code-Datei „process\_data\_s.c“ und die Assemblercode-Dateien „isr.asm“, „process\_data\_s.asm“) zu Ihrem Projekt hinzu.

Die ISR kopiert die Werte des Codec-internen ADC0 (linker und rechter Kanal) vom DMA-Empfangsbuffer in die Variablen iADC1L bzw. iADC1R und die Variablen iDAC1L bzw. iDAC1R in den DMA-Sendebuffer für den Codec-internen DAC0 (linker und rechter Kanal). Dann wird ein Flag cNewSample gesetzt, welches dem Hauptprogramm anzeigt, dass neue Werte eingelesen wurden. In der Schleife im Hauptprogramm wird dann die Funktion process\_data() für die Verarbeitung der Werte aufgerufen. In der Funktion process\_data() im Modul process\_data.c werden die Variablen iADC1L/R in die Variablen iDAC1L/R kopiert, so dass beim nächsten Aufruf der ISR der im letzten Aufruf eingelesene Wert nun ausgegeben wird. Hierdurch entsteht eine kleine zusätzliche, aber gegenüber anderen Einflüssen vernachlässigbare Verzögerung von einem Abtasttakt.

Testen sie das Programm, indem Sie zunächst Eingangs- und Ausgangssignal mit dem Oszilloskop (VI) darstellen und vergleichen. Verwenden Sie hierzu Sinussignale.

Messen Sie mit Hilfe des VI's den Frequenzgang des Gesamtsystems (Erläuterungen hierzu finden Sie in der Anleitung zum VI). Wählen Sie hierbei eine lineare Frequenzachse und die Einstellung „unwrapped phase“ für die Darstellung des Phasengangs, um dessen linearen Verlauf besser zu erkennen.

Wenn Sie das Programm stoppen, können Sie sich im Speicherfenster (Memory→BLACKFIN Memory) die aktuellen Speicherinhalte in verschiedenen Formaten anzeigen lassen. Geben Sie als Adresse den Variablennamen von iADC1L bzw. iADC1R an und stellen Sie sich den Wert im Hex-Format bzw. als Signed Integer 32 bit dar. Notieren Sie verschiedene Werte und überprüfen die Formatumwandlung vom Hex-Format in den Integer-Wert. Überprüfen Sie, in welchen drei Bytes des 32-Bit-Wortes (long Integer) die 24 vom Codec gelieferten Bits des Signalwertes stehen.

Die Signalein- und -ausgabe soll nun in Assemblercode durchgeführt werden. Ersetzen Sie dazu zunächst im Projekt die C-Datei „isr.c“ durch die Assemblerdatei „isr.asm“. Der Assemblercode führt genau die gleichen Kopieraktionen durch wie der C-Code. Testen Sie das Programm erneut.

Der DSP ist für die Verarbeitung von 16-Bit-Daten optimiert. Erstellen Sie hierzu auf Basis des Assemblerprogramms „isr.asm“ ein neues Assemblerprogramm „isr\_s.asm“ so, dass

- in der ISR nur die höherwertigen 16 Bit der vom Codec gelieferten 32 Bit aus dem DMA-Empfangsspeicher in die 16-Bit-Variablen sADC1L/R (short Integer) kopiert werden
- in der ISR nur die 16-Bit-Variablen sDAC1L/R (short Integer) in die höherwertigen 16 Bit der zum Codec geschickten 32 Bit in dem DMA-Empfangsspeicher kopiert werden und

Erstellen Sie ebenfalls eine entsprechende Definitionsdatei „isr\_s.h“ für die 16-Bit-Variante der ISR.

Ersetzen Sie im Projekt das Assemblerprogramm „isr.asm“ durch die von Ihnen erstellte 16-Bit-Variante „isr\_s.asm“, und ersetzen Sie ebenfalls das Modul „process\_data.c“ durch die 16-Bit-Variante „process\_data\_s.c“, die bereits vorliegt. Testen Sie die 16-Bit-Version des Programms.

Nun soll auch die Signalverarbeitungsroutine auf 16 Bit umgestellt werden. Dazu existiert bereits eine Assemblerdatei „process\_data\_s.asm“, die die C-Funktion in „process\_data\_s.c“ ersetzt. Modifizieren Sie das Assemblermodul „process\_data\_s.asm“, so dass das Eingangssignal des rechten Kanals um den Faktor 4 vergrößert bzw. verkleinert wird. Verwenden Sie hierzu geeignete Shiftbefehle.

- Benutzen Sie bei der Multiplikation mit dem Faktor 4 sowohl die Shiftbefehl-Variante mit und ohne Sättigung.
- Benutzen Sie für die Division durch den Faktor 4 sowohl den logischen als auch den arithmetischen Shift.

Messen Sie die vier resultierenden Ausgangssignale für ein Sinussignal mit der Amplitude 1V und ca. 500 Hz am Eingang und stellen Sie sie auf dem Oszilloskop dar.

#### **Auswertung:**

- Die Programme (bzw. die notwendigen Änderungen) und die Messergebnisse sind in der Auswertung entsprechend zu kommentieren.
- Ermitteln Sie aus dem gemessenen Phasengang des Gesamtsystems die Signallaufzeit zwischen Eingang und Ausgang. Welche Komponenten aus Abb. 1 des Vorbereitungsblattes haben hierauf maßgeblich Einfluss? Überlegen Sie sich die Laufzeiten der einzelnen Komponenten, schätzen sie ab oder ermitteln sie aus dem Datenblatt des Codecs.

## 2. Übung (FIR-Filter)

In der zweiten Laborübung werden FIR-Filter programmiert. Die für die effiziente Programmierung der entsprechenden Differenzengleichungen notwendigen Multiplikationen bzw. Multiplikationen und Additionen (MAC, Multiply-Accumulate) erfordern eine Beschäftigung mit der Multipliziereinheit des DSP und den Zahlenformaten des DSP. Darüber hinaus sollen die Programmiererfahrungen mit dem DSP erweitert werden.

### 2.1 Realisierung des FIR-Filters:

In dieser Aufgabe soll ein einfaches FIR-Filter (Mittelwertfilter 4. Ordnung) für Stereosignale in vier verschiedenen Implementierungsvarianten

- Fließkommarealisierung in C-Code
- Festkommarealisierung in C-Code
- Festkommarealisierung in Assembler-Code
- Festkommarealisierung in Assembler-Code unter Ausnutzung von SIMD-Befehlen

im Hinblick auf die Recheneffizienz (Befehlszyklen) untersucht werden.

Erstellen Sie ein neues Projekt in Ihrem Gruppenverzeichnis wie in Übung 1. Laden Sie die Templates für die Übung 2 aus dem Netz herunter und legen Sie sie in diesem Projektordner ab:

main.c	Initialisierung des Codecs und Endlosschleife
isr.asm	Interrupt-Service-Routine in Assemblercode
isr.h	Definitionsdatei für isr.asm
process_data.c	Modul mit der Verarbeitungsroutine
process_data.h	Definitionsdatei für process_data.c
fir.asm	Modul mit der FIR-Routine in Assemblercode
fir.c	Modul mit der FIR-Routine in C-code
fir.h	Definitionsdatei für fir.c
codeclib.dlb	Library mit Initialisierungsroutine für den Codec
codeclib.h	Definitionsdatei für codeclib.dlb
adsp-BF561-codec.ldf	Linker Deskription File mit den notwendigen Informationen für den Linker (Speicherbereiche, Objekte, Prozessoren, ...)

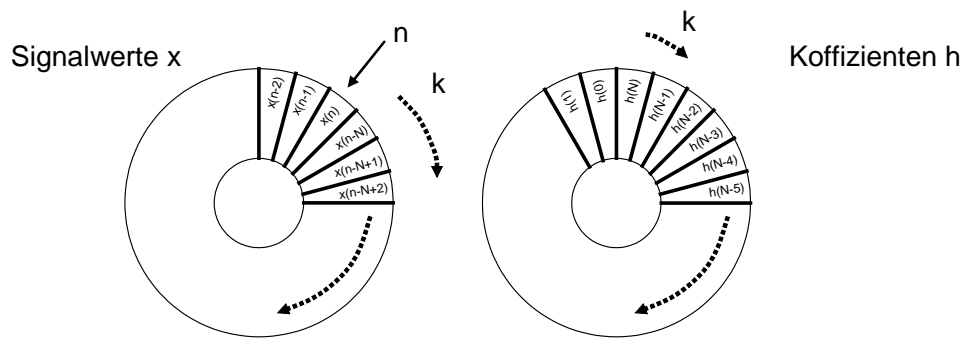
Fügen Sie alle Dateien (ohne die Assemblercode-Datei fir.asm) zu Ihrem Projekt hinzu. Die ISR übernimmt wie in Aufgabe 1.3 das Kopieren der Signalwerte aus dem DMA-Buffer in die short-Variablen sADC1L/R bzw. aus den Variablen sDAC1L/R in den DMA-Buffer. Im Modul process\_data.c sind die Filterstrukturen inkl. der Koeffizienten und die Funktion process\_data() definiert, die wiederum zweimal (einmal für jeden der beiden Stereo-Kanäle) die FIR-Routine aufruft.

In der FIR-Routine fir.c selber erfolgt die Berechnung der Differenzengleichung gemäß folgender Formel:

$$y(n) = \sum_{l=0}^N b_l \cdot x(n-l) = \sum_{k=0}^N x(n-N+k) \cdot h(N-k), \quad b_k = h(k)$$

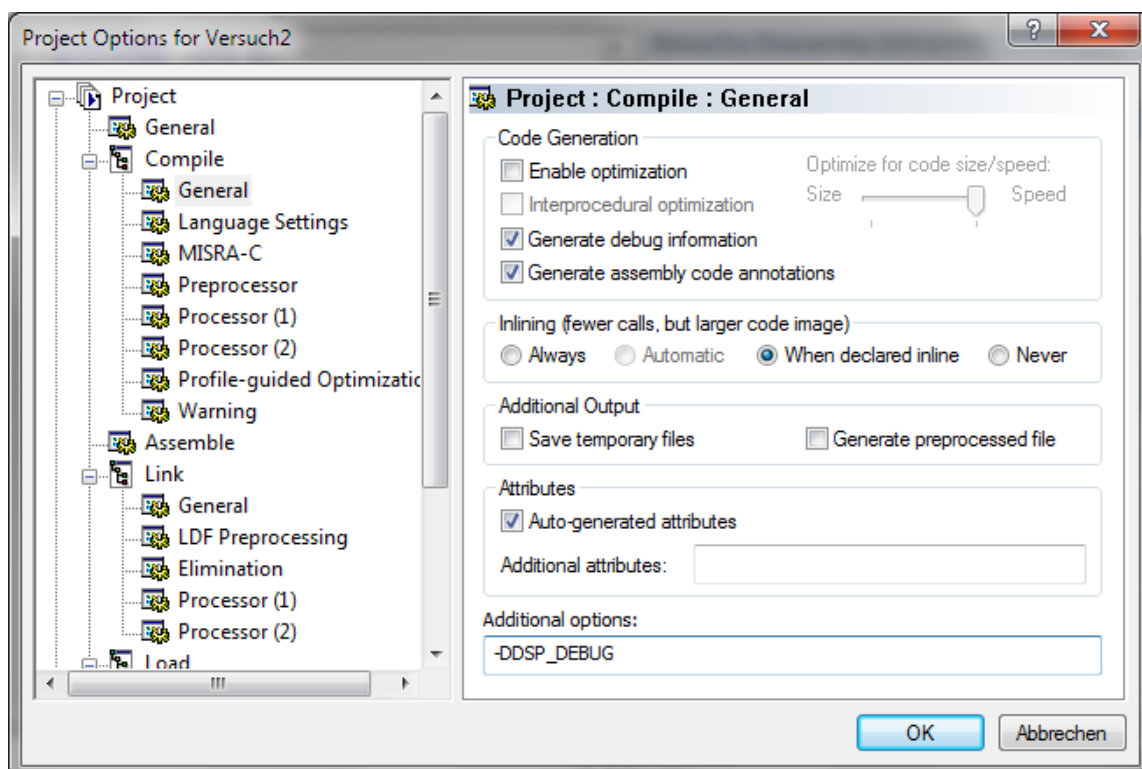
Die für die Filterrealisierung notwendigen Daten sind dabei in einer Struktur zusammengefasst.  $h(k)$  stellt die Impulsantwort dar, deren Werte mit den Koeffizienten  $b_k$  des Filters übereinstimmen. Bei der Implementierung wird für die Signalwerte  $x(n)$  ein zyklisch adressierter Speicher eingesetzt. Die zyklische Adressierung des Speichers ist in der folgenden Abbildung schematisch dargestellt. Da der Speicher für die Signalwerte  $(N+2)$ -mal adressiert wird, werden diese laufend durch neue Signalwerte überschrieben. Die Koeffizienten bleiben hingegen gleich.



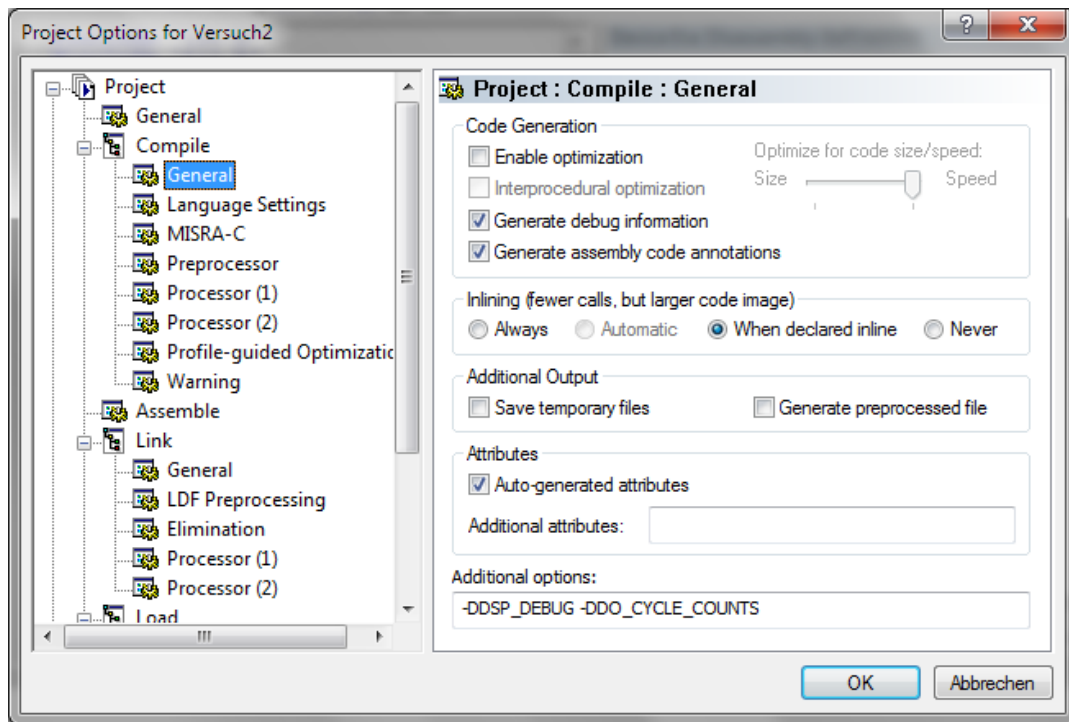


Durch die Abfrage der Umgebungsvariablen `DSP_DEBUG` werden die Teile des Programms, die für die Ansteuerung des Codecs bzw. für die Ermittlung der Zyklenzahl notwendig sind, bei der Kompilierung des Programms ausgeklammert bzw. mit kompiliert, und die Funktion des Filters lässt sich im Debugger einfacher überprüfen. Da die Variable `cNewSample` nun immer gesetzt ist, erfolgt die Verarbeitung der Signalwerte mittels `process_data()` einmal pro Durchlauf der Endlosschleife in `main.c`.

Setzen Sie die Umgebungsvariable `DSP_DEBUG`, indem Sie unter Project, Project Options, Compile, General, Additional Options „-DDSP\_DEBUG“ eintragen (siehe Abbildung) und übersetzen Sie das Programm. Überprüfen Sie die Funktionsweise, indem Sie im Memory-Fenster die Variable `_sADC1L` (Eingang des Filters) manuell verändern (entspricht einem Sprung des Signals am Eingang) und sich die entsprechende Veränderung der Variable `sDAC1L` (Ausgang des Filters) ansehen. Setzen Sie dazu einen *Breakpoint* (BP) hinter den zweiten Aufruf der FIR-Routine `fir()` in `process_data.c`, so dass Sie sich die Veränderung pro (simuliertem) Abtasttakt ansehen können. Vergleichen Sie die vom DSP berechneten Werte mit der von Ihnen in der Vorbereitung berechneten Sprungantwort.



Setzen Sie unter den Compiler-Optionen nun auch noch die Umgebungsvariable DO\_CYCLE\_COUNTS (siehe Abbildung), übersetzen Sie das Programm erneut, und ermitteln Sie die Anzahl der DSP-Zyklen für die Bestimmung der beiden Ausgangswerte. Der Wert wird im Konsolen-Fenster der IDE angezeigt.



Stellen Sie nun die Berechnung auf eine Multiplikation im 1.15-Format um, indem Sie die Koeffizienten entsprechend auf Variablen vom Typ short Integer umstellen und die Programm-Anweisungen im C-Code geeignet modifizieren.

Beispiel:

Fließkommamultiplikation (Fließkommamultiplikation wegen float x):

```
float x=0.1; short a,b; ... b = x*a;
```

Festkommamultiplikation (nur Integeroperationen):

```
short x=3277; short a,b; ... b = x*a >> 15; // x in 1.15, Shift (>>15) korrigiert Format
```

Überprüfen Sie wieder die Funktionsweise wie o.a.

Ermitteln Sie die Anzahl der Programmzyklen für die Festkommavariante in C-Code.

Die Funktion fir(...) soll nun vollständig in Assembler implementiert werden, da der DSP für Rechenoperationen wie oben gesehen spezielle Funktionen bietet, die vom Compiler normalerweise nicht ausgenutzt werden. Ersetzen Sie das C-Modul „fir.c“ im Projekt durch das Assembler-Modul „fir.asm“. In diesem Modul müssen Sie noch die Indexregister I2 und I1, das Basisregister B1 und das Längenregister L1 korrekt definieren, damit die FIR-Routine funktioniert.

Testen Sie wiederum die Funktionalität wie o.a.

Ermitteln Sie die Anzahl der Programmzyklen für die Festkommavariante in Assembler-Code.

Da der DSP mittels SIMD-Operationen zwei MAC-Operationen gleichzeitig ausführen kann, lässt sich die Implementierung für die Stereosignal-Verarbeitung noch weiter beschleunigen, wenn die Daten in geeigneter Weise im Speicher abgelegt werden. Statt einer 16-Bit-MAC-Operation und zwei 16-Bit-Datentransfers werden nun zwei 16-Bit-MAC-Operationen, ein 32-Bit-Datentransfer und ein 16-Bit-Datentransfer in einem Zyklus durchgeführt werden. Die Koeffizientenadressierung bleibt unverändert, zwei 16-Bit-Signalwerte (linker und rechter Kanal) werden jedoch gleichzeitig als ein 32-Bit-Wert aus dem Speicher gelesen werden. Linker und rechter Kanal des ADC1 bzw. DAC1 müssen dazu nur aufeinander folgend im Speicher abgelegt werden. Dies wird durch das Modul `isr.c` sichergestellt. Dementsprechend kann der Aufruf der Funktion `fir_stereo` aus `process_data` durch

```
*(int*)&sDAC1L) = fir_stereo(*(int*)&sADC1L), &firLR);
```

erfolgen, d.h. die Pointer auf den linken Kanal, unter denen eigentlich nur ein 16-Bit-Wert abgelegt ist, werden als Pointer auf einen 32-Bit-Wert interpretiert, so dass gleichzeitig beide Signalwerte als ein 32-Bit-Wert transferiert werden.

Arbeiten Sie nun mit der Stereo-Variante der FIR-Routine, indem Sie die entsprechenden Zeilen ein- bzw. auskommentieren. Definieren Sie in der Funktion `fir_stereo` im ASM-Modul „`fir.asm`“ ebenfalls noch die Indexregister `I2` und `I1`, das Basisregister `B1` und das Längenregister `L1` korrekt, damit die FIR-Routine funktioniert.

Testen Sie wiederum die Funktionalität wie o.a.

Ermitteln Sie die Anzahl der Programmzyklen für die Festkommavariante in Assembler-Code mit SIMD-Ausnutzung.

### **Auswertung:**

- Die Programme (bzw. die notwendigen Änderungen) und die Messergebnisse sind in der Auswertung entsprechend zu kommentieren.
- Schätzen Sie die maximale Ordnung eines FIR-Filters für Stereosignale für die verschiedenen Implementierungsformen ab, wenn die Abtastfrequenz 48 kHz und die Befehlsfrequenz 500 MHz beträgt<sup>4</sup>. Für den C-Code können Sie dabei den Overhead der Initialisierung, Parameterübergabe etc. gegenüber der Ausführungszeit der Schleife in der Messung vernachlässigen und somit die gemessenen Werte einfach proportional entsprechend der Anzahl der Filterkoeffizienten umrechnen. Für den Assembler-Code ist die Ausführungsdauer der Schleife jedoch genau ein Befehlstakt, so dass der Overhead für die Parameterübergabe und Filterinitialisierung bei der Messung zunächst ermittelt werden muss (gemessene Zyklenzahl abzgl. Filterordnung). Dieser Overhead muss bei der Hochrechnung berücksichtigt werden.

### **2.2 Analyse des FIR-Filters:**

Überprüfen Sie jetzt die Funktionsweise des Filters mit Sinussignalen verschiedener Frequenz. Entfernen Sie dazu die Definition der Umgebungsvariablen aus den Compiler-Optionen, damit das Programm in Echtzeit abläuft, und übersetzen Sie erneut.

Legen Sie ein Sinussignal am Eingang an und messen Sie das Ausgangssignal bei unterschiedlichen Frequenzen. Bestimmen Sie die Nullstellen des Filterfrequenzgangs, (hier muss das Ausgangssignal näherungsweise Null sein) und vergleichen Sie diese mit den Werten aus der Vorbereitung.

Messen Sie den Amplitudengang des Filters mit Hilfe des VI's und vergleichen Sie die Darstellung mit Ihrer Vorbereitung.

Geben Sie eine Rechteckfunktion mit geeigneter Frequenz auf das EVB und messen Sie somit (näherungsweise) die Sprungantwort Ihres Filters. Wählen Sie die Frequenz der Rechteckfunktion so, dass der rampenförmige Anstieg der Sprungantwort deutlich zu

<sup>4</sup> Diese Angaben entsprechen der Laborumgebung.

erkennen ist. Versuchen Sie aus der Graphik auch die Verzögerung des Gesamtsystems abzulesen.

Statt des einfachen Mittelwert-Filters soll nun ein aufwändigeres Filter mittels MATLAB entworfen und implementiert werden.

Entwerfen Sie unter MATLAB mit Hilfe des FDA-Tools (Filter Design and Analysis, Aufruf mit „fdatool“ von der MATLAB Kommandozeile) ein FIR-Filter mit den folgenden Parametern:

- Equiripple-Charakteristik
- Passband-Frequenz: 3400 Hz
- Stoppband-Frequenz: 4000 Hz
- Abtastfrequenz 48000 Hz
- Passband-Welligkeit: 2dB
- Stoppband-Dämpfung: 80dB

Exportieren Sie die Koeffizienten im fractional Format in eine C-Header-Datei (Menü-Punkt „Target“) und modifizieren Sie das Modul „process\_data.c“ entsprechend, so dass statt des einfachen Mittelwertfilters nun das mit MATLAB entworfene Filter implementiert wird.

Messen Sie den Amplitudengang und die Sprungantwort des Filters und vergleichen Sie das Ergebnis mit den Ergebnissen der MATLAB-Analyse im FDA-Tool.

#### **Auswertung:**

- Die Messergebnisse sind in der Auswertung entsprechend zu kommentieren.

### 3. Übung (IIR-Filter)

In der dritten Laborübung werden IIR-Filter programmiert. Aus programmiertechnischer Sicht sind hierzu keine weiteren Kenntnisse notwendig. Die besondere Problematik bei der Implementierung der IIR-Filter liegt vielmehr in der korrekten Umsetzung der rekursiven Differenzengleichung, so dass Rundungsfehler möglichst gering gehalten und Überläufe vermieden werden.

#### 3.1 Realisierung des IIR-Filters:

In dieser Aufgabe soll das in der Vorbereitung analysierte IIR-Tiefpassfilter mit verschiedenen Skalierungsvarianten implementiert

- Skalierung am Eingang
- Skalierung am Ausgang
- Gleichmäßige Skalierung
- Optimierte Skalierung

und im Hinblick auf Übersteuerung und Rundungsfehler untersucht werden.

Sortieren Sie die Pole (und damit auch die Nullstellen) wie in der Vorbereitung **nach zunehmender Entfernung** der Pole zum Einheitskreis. Das Filter mit der größten Entfernung zum Einheitskreis (=Filter mit den kleinsten Polradien) werde im Folgenden als  $H_1$ , dasjenige mit der kleinsten Entfernung zum Einheitskreis (=Filter mit den größten Polradien) als  $H_4$  bezeichnet. Für alle Teilsysteme liegen nach der Vorbereitung die Koeffizienten im 1.15-Format vor. Ebenso wurde der Verstärkungsfaktor  $g$  für das Filter berechnet, so dass eine Durchlassverstärkung von 0dB erzielt wird.

Erstellen Sie ein neues Projekt in Ihrem Gruppenverzeichnis wie in Übung 1. Laden Sie die Templates für die Übung 3 aus dem Netz herunter und legen Sie sie in diesem Projektordner ab:

main.c	Initialisierung des Codecs und Endlosschleife
isr.asm	Interrupt-Service-Routine in Assemblercode
isr.h	Definitionsdatei für isr.asm
process_data.c	Modul mit der Verarbeitungsroutine
process_data.h	Definitionsdatei für process_data.c
iir.asm	Modul mit der IIR-Routine in Assemblercode
iir.h	Definitionsdatei für iir.c
codeclib.dlb	Library mit Initialisierungsroutine für den Codec
codeclib.h	Definitionsdatei für codeclib.dlb
adsp-BF561-codec.ldf	Linker Deskription File mit den notwendigen Informationen für den Linker (Speicherbereiche, Objekte, Prozessoren, ...)

Fügen Sie alle Dateien zu Ihrem Projekt hinzu. Die ISR arbeitet wie in Übung 2. Im Modul process\_data.c sind die Filterstrukturen inklusive der Koeffizienten für das IIR-Filter und die Funktion process\_data() definiert, die wiederum die IIR-Routine aufruft. In der IIR-Routine iir.asm selber erfolgt die Berechnung der Differenzengleichung gemäß der Darstellung in der Dokumentation zur Vorbereitung.

In der ersten Variante der Implementierung wird der Verstärkungsfaktor am Eingang angenommen. Dies bedeutet, dass die Filterkoeffizienten wie berechnet im Koeffizientenarray im File process\_data.c abgelegt werden können. Achten Sie beim Eintragen der Filterkoeffizienten auf die richtige Reihenfolge der Teilsysteme (beginnen Sie mit  $H_1$ ) und die richtige Reihenfolge der Filterkoeffizienten für jedes Teilsystem (achten Sie auf den Kommentar im C-Code). Achten Sie auch auf das richtige Vorzeichen der Transversalkoeffizienten. Vor dem Aufruf der IIR-Routine sind die Eingangswerte sADC1L und sADC1R (unbedingt beide Werte skalieren, da wir in diesem Versuch wieder mit Stereo-Signalen arbeiten) entsprechend mit einer Festkommamultiplikation im 1.15-Format zu skalieren.

Das vorgegebene TP-Filter hat eine Grenzfrequenz von etwa 4 kHz. Aufgrund der starken Skalierung des Eingangssignals treten bei dieser Variante keine Übersteuerungsfehler auf. Allerdings lässt sich im Durchlassbereich ein erhöhtes Rauschen feststellen<sup>5</sup>.

- 1) Überprüfen Sie die Funktion des Filters indem Sie mit dem VI den Frequenzgang aufnehmen. Geben Sie ein Sinussignal mit einer Amplitude von ca. 50 mV und einer Frequenz von ca. 2 kHz auf den Eingang des EVB. Messen Sie das Spektrum des Eingangs- und Ausgangssignals (VI, Powerspektrum). Das Rauschen im Durchlassbereich des Filters müsste deutlich zu erkennen sein.

In der zweiten Variante der Implementierung wird der Verstärkungsfaktor am Ausgang angenommen. Dies bedeutet wiederum, dass die Filterkoeffizienten wie berechnet im Koeffizientenarray abgelegt werden können. Nach dem Aufruf der IIR-Routine sind die Ausgangswerte sDAC1L und sDAC1R entsprechend mit einer Festkommamultiplikation im 1.15-Format zu skalieren.

- 2) Aufgrund der starken Verstärkung des Eingangssignals durch die Teilsysteme treten bei dieser Variante starke Übersteuerungsfehler auf. Überprüfen Sie zunächst, mit einem 2 kHz Sinussignal sehr kleiner Amplitude (ca. 10 mV), ob das Filter bereits übersteuert. Wenn hier bereits eine Übersteuerung auftritt, dann melden Sie sich bitte beim Betreuer. Wenn keine Übersteuerung auftritt, erhöhen Sie die Signalamplitude sukzessive und überprüfen Sie, bis zu welcher Amplitude noch kein Übersteuerungsfehler auftritt<sup>6</sup>. Dokumentieren Sie die größte Amplitude ohne Übersteuerung sowie das zugehörige Ausgangssignal in Ihrem Protokoll. Nehmen Sie auch das Ausgangssignal, bei dem gerade eine Übersteuerung auftritt, mit in das Protokoll auf. Überprüfen Sie die Funktion des Filters mit Sinussignalen geeigneter Frequenzen und Amplituden (die nicht zu einer Übersteuerung führen). Die Messung des Frequenzgangs mit dem VI ist hier aufgrund der Übersteuerungsfehler nicht aussagekräftig.

Für die dritte Variante der Implementierung wird der Verstärkungsfaktor gleichmäßig verteilt auf alle Teilsysteme angenommen. Multiplizieren Sie dementsprechend die Transversalkoeffizienten der Teilsysteme mit dem Verstärkungsfaktor, den Sie in der Vorbereitung ermittelt haben.

- 3) Überprüfen Sie wiederum die Funktion des Filters mit Sinussignalen geeigneter Frequenz. Es treten immer noch Übersteuerungsfehler auf, allerdings erst ab einer deutlich größeren Amplitude als bei Skalierung am Ausgang (zweite Variante). Stellen Sie fest, bis zu welcher Amplitude diese Fehler noch nicht auftreten, indem Sie zunächst mit einer kleinen Signalamplitude (ca. 50mV) das Filter betreiben und diese sukzessive erhöhen. Geben Sie dazu ein Sinussignal mit einer Amplitude von ca. 50 mV und einer Frequenz von ca. 2 kHz auf den Eingang des EVB. Messen Sie das Spektrum des Eingangs- und Ausgangssignals. Vergleichen Sie das Spektrum mit dem der Variante 1 (Skalierung am Eingang).

In der letzten Variante der Implementierung wird der Verstärkungsfaktor an das jeweilige Teilsystem angepasst, d.h. der Verstärkungsfaktor für die erste Stufe wird genau so groß gewählt, dass am Ausgang der Stufe keine Übersteuerung auftritt. Als Kriterium wird hierbei der Maximalwert des Amplitudengangs des Filters  $\max\{|H| \} = H_{\max}$  angesetzt. Liegt also am Eingang des Filters ein Sinussignal mit Amplitude 1 (Maximalwert für gebrochene Zahlendarstellung), so darf am Ausgang ebenfalls nur eine Amplitude von 1 auftreten. Der Verstärkungsfaktor für die erste Stufe ergibt sich also als

<sup>5</sup> Da der DSP in jeder Stufe mit 32 Bit Genauigkeit arbeitet, ist dieses zwar recht gering, es lässt sich jedoch deutlich im Spektrum erkennen.

<sup>6</sup> Beachten Sie, dass das Filter häufig nicht mehr in einen stabilen Zustand zurückkehrt, wenn einmal Übersteuerungsfehler aufgetreten sind. Sie müssen den DSP dann zurücksetzen!

$$g_1 = \frac{1}{\max\{|H_1|\}}.$$

Bei der zweiten Stufe kann wiederum berücksichtigt werden, dass die erste Stufe bei der Frequenz, bei der die zweite Stufe die maximale Verstärkung aufweist, bereits dämpfen kann. Daher ergibt sich der Verstärkungsfaktor für die zweite Stufe als

$$g_2 = \frac{1}{\max\{g_1 \cdot |H_1| \cdot |H_2|\}}.$$

Für die dritte Stufe gilt entsprechend

$$g_3 = \frac{1}{\max\{g_1 \cdot g_2 \cdot |H_1| \cdot |H_2| \cdot |H_3|\}}.$$

usw. Somit lassen sich sukzessive alle Verstärkungsfaktoren an die Teilsysteme so anpassen, dass bei der gewählten Reihenfolge keine Übersteuerung am Ausgang der jeweiligen Teilsysteme auftritt und trotzdem die Verstärkungsfaktoren möglichst groß gewählt werden und damit das in nachfolgenden Stufen auftretende Rundungsrauschen minimiert wird.

Die Optimierung soll mittels MATLAB durchgeführt werden. Schreiben Sie dazu die Koeffizienten für jedes Teilsystem in einen Koeffizientenvektor, d.h. für ein Teilsystem mit der Übertragungsfunktion

$$H_i(z) = \frac{z^2 + b_1 \cdot z + b_2}{z^2 + a_1 \cdot z + a_2}$$

geben Sie in MATLAB ein

$$H_i = [1 \ b_1 \ b_2 \ 1 \ a_1 \ a_2];$$

Berechnen Sie dann den Amplitudengang des ersten Teilsystems mit

$$H = \text{abs}(\text{freqz}(H_1(1:3), H_1(4:6)));$$

Hieraus folgt der Verstärkungsfaktor

$$g_1 = 1/\max(H);$$

Für die zweite Stufe setzen Sie den Amplitudengang des skalierten ersten Teilsystems zu

$$H = g_1 * H;$$

und berechnen den Amplitudengang Gesamtsystems ohne zweiten Skalierungsfaktor zu

$$H = H * \text{abs}(\text{freqz}(H_2(1:3), H_2(4:6)));$$

und dementsprechend den Skalierungsfaktor für die zweite Stufe als


$$g_2 = 1/\max(H);$$

Dies wiederholen Sie, bis auch die Skalierungsfaktoren g3 und g4 berechnet sind:

$$\begin{aligned} H &= g_2 * H; \\ H &= H * \text{abs}(\text{freqz}(H_3(1:3), H_3(4:6))); \\ g_3 &= 1/\max(H); \\ H &= g_3 * H; \end{aligned}$$

```
H=H.*abs(freqz(H4(1:3),H4(4:6)));  
g4=1/max(H);
```

Ermitteln Sie den neuen Satz von Filterkoeffizienten, indem Sie die in der Vorbereitung ermittelten Koeffizienten mit den Gewichtungsfaktoren für jedes Teilsystem multiplizieren.

- 4) Wiederholen Sie die Messungen, die Sie für Variante 3 durchgeführt haben, d.h. überprüfen Sie die maximale Eingangsamplitude  bei der noch keine Übersteuerungen auftreten, und messen Sie das Spektrum des Ausgangssignals zur Überprüfung des Rundungsrauschens.

#### **Auswertung:**

- Die Messergebnisse sind in der Auswertung entsprechend zu kommentieren.

#### **3.2 Analyse des IIR-Filters:**



- 1) Ermitteln Sie die 3dB-Grenzfrequenz des Tiefpass-Filters, indem Sie die Frequenz eines Sinussignals solange vergrößern, bis der Wert der Ausgangsamplitude etwa 70% der Ausgangsamplitude bei einer Frequenz von 50Hz erreicht. Messen Sie den Amplitudengang des Filters mit Hilfe des VI's und überprüfen Sie die 3dB-Grenzfrequenz hiermit.

- 2) Entwerfen Sie unter MATLAB mit Hilfe des FDA-Tools (Filter Design and Analysis, Aufruf mit „fdatool“ von der MATLAB Kommandozeile) ein IIR-Bandpass-Filter mit den folgenden Parametern:

- Butterworth-Charakteristik
- Stopband-Frequenzen: 2000 Hz, 4000 Hz
- Passband-Frequenz: 2900, 3100 Hz
- Passband-Welligkeit: 1dB
- Stopband-Dämpfung: 60dB

Bestimmen Sie aus den Koeffizienten der Filter entsprechend der Vorbereitung und dem ersten Teil der 3. Übung die Filterkoeffizienten mittels optimaler Skalierung für den DSP.

Messen Sie den Amplitudengang und die Sprungantwort des Filters und vergleichen Sie das Ergebnis mit den Ergebnissen der MATLAB-Analyse im FDA-Tool.

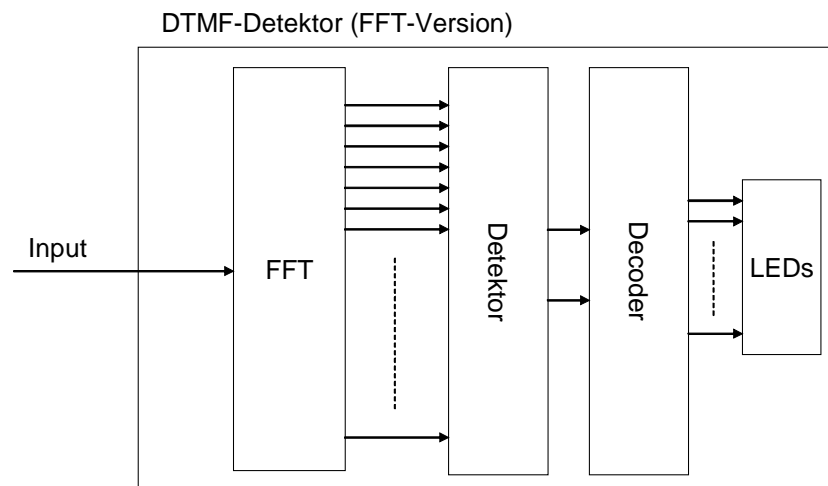
#### **Auswertung:**

- Die Messergebnisse sind in der Auswertung entsprechend zu kommentieren.



#### 4. Übung (DTMF-Detektor mit FFT)

In der vierten Laborübung soll ein DTMF-Detektor (*Dual Tone Multiple Frequency*) entwickelt werden. Zur Realisierung der DFT wird eine FFT eingesetzt. DTMF Detektoren kommen beim sogenannten Mehrfrequenzwahlverfahren (MFV) in Telefonanlagen zum Einsatz. Bei der DTMF-Übertragung werden die Informationen der Telefontastatur mittels einer Kombination zweier Sinustöne im Sprachband codiert. Der Detektor hat die Aufgabe, die Töne herauszufinden und die beiden gefundenen Töne wieder in die Tasteninformation zu decodieren. Die Tasteninformation wird über die LEDs auf dem EVB angezeigt. Die prinzipielle Struktur des Detektors ist in der folgenden Abbildung dargestellt.



##### 4.1 Fast Fourier-Transformation (FFT)

Erstellen Sie ein neues Projekt in Ihrem Gruppenverzeichnis wie in Übung 1. Laden Sie die Templates für die Übung 5 aus dem Netz herunter und legen Sie sie in diesem Projektordner ab:

main.c	Initialisierung des Codecs und Endlosschleife
process_data.c	Modul mit der Verarbeitungsroutine
process_data.h	Definitionsdatei für process_data.c
isr.h	Definitionsdatei für das zu erweiternde Modul isr.c
tools.h	Definitionsdatei für das zu erstellende Modul tools.asm
codeclib.dlb	Library mit Initialisierungsroutine für den Codec
codeclib.h	Definitionsdatei für codeclib.dlb
adsp-BF561-codec.ldf	Linker Deskription File mit den notwendigen Informationen für den Linker (Speicherbereiche, Objekte, Prozessoren, ...)

Fügen Sie alle Dateien und die von Ihnen in der Vorbereitung erzeugten C- und Assembler-Module zu Ihrem Projekt hinzu. Wählen Sie die Segmentlänge (FRAMELENGTH, Datei: isr.h), die der FFT-Ordnung entspricht, gemäß der Vorbereitung.

Die Verarbeitung im Modul process\_data.c ruft die FFT-Funktion `rfft_fr16()` aus der DSP-Library von Analog Devices auf, die sowohl für das (reellwertige) Signal als auch das berechnete (komplexwertige) Spektrum von Zahlenwerten im fractional Format ausgeht. Zur Vermeidung von Überläufen wird das Ergebnis durch die FFT-Ordnung  $N$  geteilt, so dass dieses in der Form

$$X(k) = \frac{1}{N} \cdot \sum_{n=0}^{N-1} x(n) \cdot e^{-jkn\frac{2\pi}{N}} = \frac{1}{N} \cdot \sum_{n=0}^{N-1} x(n) \cdot w_N^{-kn}, \quad k = 0, \dots, N-1$$

ermittelt wird. Die komplexen Twiddle-Faktoren  $w_N^{-kn}$  werden einmalig für die gewählte FFT-Ordnung im Hauptprogramm ermittelt und der Funktion in einem Array übergeben. Das Resultat der FFT steht in der komplexwertigen Variable *spectrum*.

Im nachfolgenden Schritt wird das quadratische Betragsspektrum berechnet. Hierzu wird nach der FFT-Berechnung Ihre in der Vorbereitung in Assemblercode erstellte Funktion `abs2_spec()` für das zu transformierende Signalsegment aufgerufen.



Nehmen Sie auf dem Oszilloskop das mit dem DSP ermittelte Betragsspektrum für zwei unterschiedliche DTMF-Frequenzen auf, die Sie wieder mit dem Funktionsgenerator erzeugen. Wählen Sie hierbei als Amplitude den Wert 100mV. Ändern Sie den Shift in der Assemblerfunktion `abs2_spec`, so dass die beiden Peaks deutlich zu sehen sind.



## 5.2 Fast Fourier-Transformation (FFT) mit Fensterfunktionen

Das Ausschneiden eines Signalsegmentes aus dem eigentlich unendlich langen Sinussignal führt zum Leckeffekt, der sich in deutlichen Maxima neben der eigentlichen Signalfrequenz widerspiegelt. Der Leckeffekt lässt sich verringern, wenn das Signalsegment durch geeignete Fensterfunktionen gewichtet wird. Einfache bekannte Fensterfunktionen sind das dreieckförmige Bartlett-Fenster und das cosinusförmige Hamming-Fenster.

Die Fensterfunktionen können vom DSP durch den einmaligen Aufruf der Funktionen `gen_bartlett_fr16()` bzw. `gen_hamming_fr16()` generiert werden. Entfernen Sie die Kommentarzeichen im Hauptprogramm `main()` und in der Funktion `process_data()`, so dass die Fensterfunktionenwerte generiert werden und Ihre in der Vorbereitung in Assemblercode erstellte Funktion `winmul()` für das zu transformierende Signalsegment vor der FFT-Berechnung aufgerufen wird.

Nehmen Sie für beide Fensterfunktionen wiederum auf dem Scope das mit dem DSP ermittelte Betragsspektrum für zwei unterschiedliche DTMF-Frequenzen auf, die Sie wieder mit dem Funktionsgenerator erzeugen. Wählen Sie hierbei als Amplitude wiederum den Wert 100mV. Passen Sie eventuell den Shift in der Assemblerfunktion `abs2_spec` an.

Im Folgenden wird nur noch mit dem Hamming-Fenster gearbeitet.

## 5.3 DTMF-Detektor

Erweitern Sie die Funktion `process_data()` um die Detektion und die Decodierung.

- Da die FFT das gesamte Spektrum berechnet, ist es möglich, einen Schwellwert für die Detektion der Frequenzen aus den im DTMF-Signal definitiv nicht vorhandenen Spektralanteilen zu ermitteln. Diese stellen ein Maß für das Rauschen des Signals dar. Nur wenn der DTMF-Signalwert deutlich über dem Rauschen liegt, liegt ein DTMF-Signal vor. Bestimmen Sie diesen Schwellwert als Mittelwert der FFT-Spektrallinien zwischen 50 und 600 Hz und zwischen 1750 Hz und 4250 Hz.
- Ermitteln Sie weiterhin die acht relevanten Spektralanteile als Mittelwert über jeweils drei DFT-Bins, die den zu detektierenden Frequenzen am nächsten liegen.

Bestimmen Sie getrennt für die Spalten- und Zeilenfrequenzen jeweils den Maximalwert der relevanten Spektralanteile (Betragsquadrate) und merken sich, bei welcher Frequenz (Zeile / Spalte) dieser auftritt. Vergleichen Sie den mit einem von Ihnen zu bestimmenden Faktor multiplizierten Schwellwert mit dem Maximalwert getrennt für Zeilen und Spaltenfrequenzen.

1) Zur Bestimmung des Faktors ermitteln Sie aus dem DSP-Speicher jeweils für alle zu detektierenden Frequenzen die Maximalwerte und die Schwellwerte und notieren Sie diese für das Protokoll.

Decodierung: Liegt der Maximalwert über dem skalierten Schwellwert (ist also ein DTMF-Signal detektiert), dann ändern Sie die PFs so, dass eine dem DTMF-Signal zugeordnete LED leuchtet. Die Zuordnung ist wie folgt gegeben:

Taste	1	2	3	A	4	5	6	B
LED	13	14	15	16	17	18	19	20

Taste	7	8	9	C	*	0	#	D
LED	5	6	7	8	9	10	11	12

2) Testen Sie die Funktion Ihres Programms, indem Sie an den Eingang eine der Zeilenfrequenzen legen und für die Spaltenfrequenz im Programm eine beliebige Spalte als detektiert annehmen.

3) Wenn die entsprechenden LEDs bei dem o.a. Test leuchten, modifizieren Sie das Programm wieder so, dass sowohl Zeilenfrequenz als auch Spaltenfrequenz erkannt werden müssen. Melden Sie sich beim Betreuer, der dann die Funktion mit realen DTMF-Tönen verifiziert.

### **Auswertung:**

- Kommentieren Sie Messungen und Programme, so dass Ihre Überlegungen und Untersuchungen deutlich werden.

## 5. Übung (DTMF-Detektor mit Goertzel-Algorithmus)

Der DTMF-Detektor aus der vorherigen Übung soll in dieser Übung auf algorithmisch andere Weise umgesetzt werden. Hierzu wird der Goertzel-Algorithmus eingesetzt, der in dem Umdruck zur Vorbereitung auf diese Laborübung kurz beschrieben ist.

### 5.1 Goertzel-Algorithmus

Erstellen Sie ein neues Projekt in Ihrem Gruppenverzeichnis. Verwenden Sie hierzu die Quellen aus der Übung 4 mit der Erweiterung der tools.asm aus der Vorbereitung zur Aufgabe 5 zur Berechnung der DFT-Werte mittels Goertzel-Algorithmus.

Vergleichen Sie die berechneten Werte des Goertzel-Algorithmus mit denen der FFT-Funktion aus der Library für alle 24 zu berechnenden DFT-Frequenzen. Überprüfen Sie diese Werte anhand der Speicherinhalte und nutzen Sie das Hamming-Fenster.

Wenn sichergestellt ist, dass die berechneten Werte stimmen, fahren Sie mit der nächsten Aufgabe fort.

Vergleichen Sie den Rechenaufwand für die FFT-Funktion der Library und Ihre Funktion zur Berechnung der 24 Stützstellen. Gehen Sie hierbei so vor, wie Sie das in der zweiten Übung (FIR-Filter) gelernt haben.

### 5.2 DTMF-Detektor

Ändern Sie die Verarbeitung für die Detektion in der Funktion process\_data(), so dass die acht relevanten Spektralanteile als Mittelwert über jeweils die drei DFT-Bins berechnet werden, die den zu detektierenden Frequenzen am nächsten liegen.

Als Schwellwert für die Detektion soll hier der Mittelwert der nicht als Maximum erkannten Spektralanteile verwendet werden. Bestimmen Sie hierzu wieder getrennt für die Spalten- und Zeilenfrequenzen jeweils den Maximalwert der relevanten Spektralanteile (Betragsquadrate) und merken sich, bei welcher Frequenz (Zeile / Spalte) dieser auftritt. Vergleichen Sie den mit einem von Ihnen zu bestimmenden Faktor multiplizierten Schwellwert mit dem Maximalwert getrennt für Zeilen und Spaltenfrequenzen.

4) Zur Bestimmung des Faktors ermitteln Sie aus dem DSP-Speicher jeweils für alle zu detektierenden Frequenzen die Maximalwerte und die Schwellwerte und notieren Sie diese für das Protokoll.

Die Decodierung kann wie in Übung 4 erfolgen

5) Überprüfen Sie die Funktion des Programms mit realen DTMF-Tönen und lassen Sie den Betreuer dies wieder verifizieren.

### Auswertung:

- Kommentieren Sie Messungen und Programme, so dass Ihre Überlegungen und Untersuchungen deutlich werden.