

Laborversuch 1

Teil 2

Fortsetzung Signalgenerator (Sinus)

Ausgabe eines Sinustones über den Codec TLV320AIC23B auf dem DSP Starter Kit DSK6713 von TI.

Die auszugebenden Sinuswerte liegen als float-Zahlen im Bereich -1 ... +1 in einer z.B. 100 Elemente großen Tabelle im Speicher (sog. Look-up-table).

*Im Unterschied zu Laborversuch 1, Teil 1 soll die Übergabe der Sinuswerte an den McBSP1 des DSP's nicht mehr im Polling-Betrieb sondern im **Interrupt-Betrieb** erfolgen. Der McBSP1 liefert dazu bei freiem Data-Transmit-Register DXR (d.h. nach jedem in das Transmit-Shift-Register XSR übernommenen Datum) das Transmit-Interrupt-Signal XINT1.*

Im optionalen Teil des Laborversuchs kann eine alternative Tonerzeugung mit Hilfe eines digitalen Filters untersucht werden.

1. Einführende Informationen

1.1 CPU-Interrupts

Die CPU des C6713 kann neben dem Reset und dem NMI (Non Maskable Interrupt) über 12 frei belegbare, maskierbare Interrupts in Ihrer augenblicklichen Programmabarbeitung unterbrochen werden. Als auslösende Interrupt-Quellen können Teile der On-Chip-Peripherie wie z.B. Timer, McBSPs, EMIF, EDMA etc., aber auch ein angebundener Hostprozessor oder mehrere spezielle Interrupt-Pins des DSP's dienen (beim TMS320C6713 insgesamt 23 Interrupt-Quellen). Zusätzlich können die maskierbaren CPU-Interrupts auch per Software manuell ausgelöst werden.

Mit Hilfe der Interrupt-Multiplexerregister MUXH und MUXL können diese Interrupt-Quellen beliebig auf die 12 maskierbaren CPU-Interrupts (INT4 – INT15) gelegt werden. Tabelle 1.1 zeigt das Standard-Interrupt-Mapping des DSP's nach dem Einschalten. Die Interrupt-Priorität nimmt mit aufsteigender CPU-Interruptnummer ab (INT4 hat die höchste Priorität, INT15 die niedrigste).

Zusätzlich zu diesem Interrupt-Mapping muss der gewünschte CPU-Interrupt und der Interrupt-Betrieb generell freigegeben werden.

Den CPU-Interrupts werden über die Interrupt-Vektor-Tabelle sogenannte Interrupt-Vektoren (Interrupt-Service-Fetch-Packets, 1 ISFP = acht 32-Bit-Instruktionen) zugeordnet.

SS 2012

Fakultät für Technik, Studiengänge EIT/TI

Dipl.-Ing.(FH) Felix Becker

Die Interrupt-Bearbeitung erfolgt dann meist durch Interrupt-Service-Routinen, die über die entsprechenden Interrupt-Vektoren in der Interrupt-Vektor-Tabelle aufgerufen werden. (siehe auch [Infoblatt](#) ,Code Composer Studio...‘, Kapitel ,Interrupts in C‘)

| DSP INTERRUPT NUMBER | INTERRUPT SELECTOR CONTROL REGISTER | DEFAULT SELECTOR VALUE (BINARY) | DEFAULT INTERRUPT EVENT |
|----------------------------|--|--|-------------------------------|
| INT_00 | – | – | RESET |
| INT_01 | – | – | NMI |
| INT_02 | – | – | Reserved |
| INT_03 | – | – | Reserved |
| INT_04 | MUXL[4:0] | 00100 | GPINT4† |
| INT_05 | MUXL[9:5] | 00101 | GPINT5† |
| INT_06 | MUXL[14:10] | 00110 | GPINT6† |
| INT_07 | MUXL[20:16] | 00111 | GPINT7† |
| INT_08 | MUXL[25:21] | 01000 | EDMAINT |
| INT_09 | MUXL[30:26] | 01001 | EMUDTDMA |
| INT_10 | MUXH[4:0] | 00011 | SDINT |
| INT_11 | MUXH[9:5] | 01010 | EMURTDXR |
| INT_12 | MUXH[14:10] | 01011 | EMURTDXTX |
| INT_13 | MUXH[20:16] | 00000 | DSPINT |
| INT_14 | MUXH[25:21] | 00001 | TINT0 |
| INT_15 | MUXH[30:26] | 00010 | TINT1 |

Tabelle 1.1: TMS320C6713 Interrupts (MUXL, MUXH nach Reset)

Für den interrupt-betriebenen Datenaustausch mit dem Codec soll der McBSP1 Transmitter-Interrupt XINT1 (Selector Value: 01110, siehe z.B. Tabelle im [Infoblatt](#) ,Code Composer Studio...‘, Kapitel ,Interrupts in C – CPU-Interrupts‘) über das Interrupt-Multiplexerregister MUXH auf den maskierbaren CPU-Interrupt INT11 gelegt werden. Und im Hinblick auf den nächsten Laborversuch belegen wir auch gleich CPU-INT12 mit dem Receive-Interrupt RINT1 (Selector Value: 01111).

SS 2012

Fakultät für Technik, Studiengänge EIT/TI

Dipl.-Ing.(FH) Felix Becker

| CPU-Interrupts | Priority | Interrupt Mapping | MUXL / MUXH Register Bits |
|----------------|----------|-------------------|---------------------------|
| RESET | Highest | | |
| NMI | | | |
| INT4 | | GPINT4 | MUXL[4...0]: 00100 |
| INT5 | | GPINT5 | MUXL[9...5]: 00101 |
| INT6 | | GPINT6 | MUXL[15,14...10]: 0 00110 |
| INT7 | | GPINT7 | MUXL[20...16]: 00111 |
| INT8 | | EDMAINT | MUXL[25...21]: 01000 |
| INT9 | | EMUDTDMA | MUXL[31,30...26]: 0 01001 |
| INT10 | | SDINT | MUXH[4...0]: 00011 |
| INT11 | | XINT1 | MUXH[9...5]: 01110 |
| INT12 | | RINT1 | MUXH[15,14...10]: 0 01111 |
| INT13 | | DSPINT | MUXH[20...16]: 00000 |
| INT14 | | TINT0 | MUXH[25...21]: 00001 |
| INT15 | Lowest | TINT1 | MUXH[31,30...26]: 0 00010 |

Tabelle 1.2: Geändertes Interrupt-Mapping für INT11 und INT12

```
=> MUXH[31...0] = 0 00010 00001 00000 0 01111 01110 00011
                = 0000 1000 0010 0000 0011 1101 1100 0011
                = 0x08203DC3 hex
```

1.2 Interrupt Service Routine

Dem CPU-Interrupt INT11 wird über das INT11-ISFP in der Interrupt-Vektor-Tabelle (siehe Datei: *vectors2.asm*) die Interrupt-Service-Routine ISR11 zugeordnet (Label: `_isr11_xint1`).

Zu beachten ist der dem Label (C-Funktionsnamen) voranzustellende Unterstrich und das der Registerinhalt von innerhalb eines ISFP genutzten Registers vor dem Überschreiben auf dem Stack gerettet und nach der Benutzung des Registers wieder zurückgespeichert werden muss.

Die Interrupt-Service-Routine (`isr11_xint1`) übernimmt dann die Übergabe der Sinuswerte an das McBSP1 Data-Transmit-Register DXR.

SS 2012

Fakultät für Technik, Studiengänge EIT/TI

Dipl.-Ing.(FH) Felix Becker

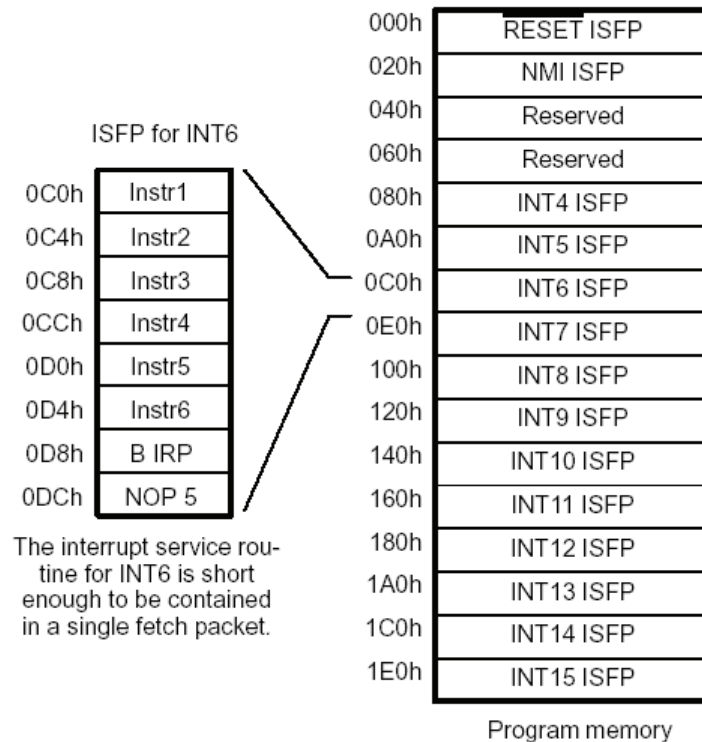


Abbildung 1.1: Interrupt-Vektor-Tabelle mit Byte-Adressangaben („ISFP for INT6“ ist ein Beispiel für die Interrupt-Bearbeitung ohne separate ISR)

Die Interrupt-Service-Routine ist als C-Funktion vom Typ `void` geschrieben und in der Datei `sinetone2.c` enthalten (`interrupt void isr11_xint1(void)`). Einer ISR werden keine Werte übergeben, sie kann allerdings auf globale Variablen zugreifen und diese verändern. Über das Schlüsselwort `interrupt` wird dem C-Compiler mitgeteilt, dass alle Registerinhalte von in dieser Funktion benutzten Registern vor der Verwendung auf dem Stack gerettet und beim Verlassen der Funktion wieder zurückgeschrieben werden müssen. Für den einfachsten Fall ruft eine ISR deshalb auch keine weitere Funktion auf.

1.3 Beispiel einer Interrupt Service Routine

Interrupt Service Routine für CPU-Interrupt 11 (ISR11):
(C-Funktion vom Typ `void` mit Schlüsselwort `interrupt`)

```
interrupt void isr11_xint1(void)
{
    short sample;

    // gaining each sample
    sample = (short)(sine_table[k] * gain);

    // write 16 bit sample to codec left and right channel via
    // one 32 bit word,
    // left channel: upper 16 bits, right channel: lower 16 bits
    // (mcbbsp1_write waits until the write to McBSP1_DXR is possible)
    // (due to the codec sample clock of 32 kHz this write takes place
```

```
// every 1 / 32 kHz = 31,25 µsec)
mcbsp1_write( ((int)sample)<<16 | ((int)sample & 0x0000FFFF) );

// setup the circular buffer pointer
k += steps;
if (k >= tab_size) k -= tab_size;
}
```

1.4 Interruptfreigabe

Die Freigabe (Demaskierung) des CPU-Interrupts INT11 erfolgt mit Hilfe des zugehörigen Bits im Interrupt-Enable-Register IER. Zuvor sollte allerdings das Interrupt-Flag-Register IFR über das Interrupt-Clear-Register ICR gelöscht werden.

Damit die CPU überhaupt auf die maskierbaren Interrupts reagiert, muss im IER zusätzlich das Non-Maskable-Interrupt-Enable-Bit (NMIE-Bit = Bit 2) gesetzt sein.

(Das NMIE-Bit wird zu Beginn eines Non-Maskable-Interrupts gelöscht, um zu verhindern, dass der NMI durch einen anderen Interrupt unterbrochen werden kann. Beim Verlassen der NMI-ISR wird das NMIE-Bit wieder gesetzt. Nach einem Reset muss es allerdings von Hand gezielt gesetzt werden.)

```
ICR = 0x0000FFFF;    // löschen aller Flags
IER |= 0x00000802;    // Demaskierung von CPU-INT11 und
                      // setzen des NMIE-Bits
```

Zum Schluß erfolgt die generelle Freigabe des Interrupt-Betriebs durch Setzen des Global-Interrupt-Enable-Bits (GIE-Bit) im Control-Status-Register CSR.

```
CSR |= 0x00000001;    // setzen des GIE-Bits
```

1.5 Zusammenfassung der für den Interrupt-Betrieb nötigen Maßnahmen und Einstellungen

1. Zuordnung von Interrupts der (OnChip-) Peripherie zu den 12 CPU-Interrupts über die Interrupt-Multiplexer-Register MUXL und MUXH
2. Erstellen der gewünschten Interrupt-Vektoren in der Interrupt-Vektor-Tabelle
3. Erstellen der gewünschten Interrupt-Service-Routinen
4. Löschen des Interrupt-Flag-Registers IFR mit Hilfe des Interrupt-Clear-Registers ICR
5. Freigabe der gewünschten CPU-Interrupts über die entsprechenden Bits im Interrupt-Enable-Register IER
6. Freigabe der Non-Maskable-Interrupts über das NMIE-Bit im Interrupt-Enable-Register
7. Setzen des Global-Interrupt-Enable-Bits (GIE-Bit) im Control-Status-Register CSR.

1.6 Fragen

F 1.1 Welchen Vorteil hat Interrupt-Betrieb gegenüber Polling-Betrieb?

F 1.2 Warum kann jetzt auf das McBSP1-Transmit-Register (DXR) ohne Überprüfung des SPCR-Registers zugegriffen werden?

(Die Fragen sind handschriftlich in der Versuchsvorbereitung zu beantworten!)

2. Durchführung

2.1 Einarbeitung in das vorgegebene Programm *sinetone2.c*



Kopieren Sie sich den Projektordner **Project2.zip** von der Labor-Internetseite auf die lokale Festplatte in das Verzeichnis `c:\users\Ihr_Arbeitsverzeichnis\` bzw. in *Eigene Dateien* (siehe Einführungsdokument ,Organisatorisches') und **entpacken** Sie den Projektordner **in das Unterverzeichnis *eclipseworkspace_v5_1*** (vgl. dazu auch LV1, Teil 1, Kapitel 2.1). Die Labor-Internet-Seite erreichen Sie auf den Labor-PCs im Raum T1.4.01 am schnellsten über das Startmenü:

Start -> Labore -> Signale und Systeme -> Labor Signale und Systeme

Versorgen Sie das Demoboard (DSK6713) mit Spannung (Netzteil) und warten Sie den jetzt ablaufenden **PowerOnSelfTest** (POST) ab. Die LED1 leuchtet dabei für einige Sekunden. Am Ende des Tests blinken die USER-LEDs des Boards einige Male und leuchten nach fehlerfrei abgelaufenem Test dauerhaft.

Starten Sie erst danach die Entwicklungsumgebung Code Composer Studio v5 über das Startmenü:

Start -> Labore -> Signale und Systeme -> Code Composer Studio

Um nicht mit dem ersten Projekt aus dem vorangegangenen Laborversuch in Konflikt zu geraten, bietet es sich an dieser Stelle an, dieses mit Hilfe des zugehörigen Kontextmenüs zu schließen. Klicken Sie dazu mit der rechten Maustaste auf  Project1 im  Project Explorer und wählen Sie den Eintrag Close Project.






Importieren Sie das zweite Projekt indem Sie im Menü **Project** den Eintrag

 Import Existing CCS/CCE Eclipse Project wählen (vgl. LV1, Teil 1, Kapitel 2.3).

Verschaffen Sie sich einen Überblick über die in dieses Projekt eingebundenen Dateien. (Erläuterungen siehe Infoblatt ,Code Composer Studio...')

Sehen Sie sich die Dateien ***sinetone2.c*** und ***vectors2.asm*** genauer an und versuchen Sie die Funktionsweise nachzuvollziehen !!!



Stellen Sie in der Entwicklungsumgebung Code Composer Studio v5 durch Starten des Debuggers eine software-mäßige Verbindung zum Demoboard her. Dadurch wird gleichzeitig die Objekt-Datei *Project2.out* auf das DSK6713 heruntergeladen.

Starten des Debuggers: Bei ausgewähltem/markiertem  Project2 im  Project Explorer entweder im Menü Run den Eintrag  Debug wählen oder in der Symbolleiste den Käfer  anklicken oder den Hot-Key F11 drücken oder mit der rechten Maustaste das zum ausgewählten Projekt gehörige Kontextmenü öffnen und darin Debug As  Code Composer Debug Session wählen.

Die Interrupt-Vektortabelle und der Code werden in den internen Speicher des DSP's geladen, einige Bereiche werden aber auch, wie in der Datei *ds6713.cmd* angegeben, in das externe SDRAM kopiert.

Starten Sie den Sinusgenerator durch einen Klick auf das Symbol  oder wählen Sie im Menü Run den Eintrag Resume oder drücken Sie den Hot-Key F8.






Hören Sie sich den Sinuston über den Kopfhörer an (Pegelsteller für LINE IN 2/3, MAIN MIX und PHONES sowie Schalter TAPE TO PHONES am Mischpult beachten! Siehe dazu auch das Infoblatt „Erläuterungen zur Audioverkabelung“).

Stoppen Sie das Programm durch einen Klick auf das Symbol  oder wählen Sie im Menü Run den Eintrag  Suspend.

2.2 Einige Features von CCS v5

Variables, Expressions und Registers View

Verändern Sie jetzt wieder den Pegel und die Tonhöhe des Sinustones. Im Gegensatz zum letzten Laborversuch aber nicht direkt im C-Code, sondern über die sog. Expressions View, die standardmäßig beim Wechsel in die Ansicht CCS Debug zusammen mit der Variables und Registers View geöffnet wird.



Ändern Sie den Pegel des Sinustons über den Gain-Faktor zunächst auf -12 dBFS und dann auf -36 dBFS. Fügen Sie dazu im Fenster  Expressions die globale Variable `gain` ein und ändern Sie den Wert der Variable direkt in der Spalte Value. Das Einfügen kann auf zwei Arten geschehen. Entweder direkt im Fenster  Expressions über das Symbol  und Angabe der Variable `gain` oder im  C-Quellcodefenster durch markieren der Variable `gain` (zu finden im Deklarationsbereich zu Beginn der Datei) und öffnen des zugehörigen Kontextmenüs mit der rechten Maustaste und Auswahl des Eintrags  Add Watch Expression...

Starten Sie das Programm wieder (Resume) und überprüfen Sie Ihre Änderung mit dem Kopfhörer. Unterbrechen Sie die Programmabarbeitung und testen Sie den zweiten Pegelwert. Testen Sie danach wie sich eine Übersteuerung auf digitaler Ebene anhört, indem Sie den Gain-Faktor auf z.B. 33000 abändern. Stellen Sie anschließend wieder 0 dBFS ein (Gain-Faktor: 32767).

Unterbrechen Sie die Programmabarbeitung und gehen Sie in gleicher Weise mit der Variable `steps` vor. Ändern Sie die Tonhöhe zunächst auf 1600 Hz und dann auf 3200 Hz ab.

Neben der Überwachung und der direkten Wertänderung von Variablen bietet CCS noch einige weitere Features.

Disassembly View

Eines davon ist die Möglichkeit, sich den Inhalt eines Speicherbereichs (beispielsweise mit Quellcode) disassembliert bzw. gemischt mit C-Code darstellen zu lassen. Öffnen Sie dazu über das Menü View ->  Disassembly das Fenster  Disassembly.

Speicherbereich graphisch darstellen lassen

Ein weiteres Feature ist die Möglichkeit, sich den Inhalt eines Speicherbereichs graphisch darstellen zu lassen. In diesem Projekt bietet es sich an, die Sinustabelle, die im externen Speicher liegt, graphisch darzustellen.

Öffnen Sie dazu über das Menü Tools -> Graph -> Single Time das Graph-Properties-Fenster.

Dort müssen Sie unter Data Properties die Startadresse des darzustellenden Speicherbereichs angeben. Die Startadresse, also die Adresse, ab welcher der Linker die Sinustabelle in den Speicher gelegt hat, können Sie dem sogenannten Map-File entnehmen. Diese **Map-Datei** wird beim Linken über die Linkeroption -m standardmäßig erzeugt (falls nicht: Menü Project -> Build Options..., Bereich: C6000 Linker, Blatt: Basic Options).

Nach einem (erfolgreichen) Build-Prozess finden Sie die Datei im Debug-Ordner des Projekts (c:\users\lhr_Arbeitsverzeichnis\eclipse\workspace_v5_1\Project2\Debug\Project2.map).

SS 2012


Fakultät für Technik, Studiengänge EIT/TI



Dipl.-Ing.(FH) Felix Becker

Öffnen Sie die Datei (z.B. in CCS über das Menü File -> Open File...) und suchen Sie die zum Label `_sine_table` gehörende Adresse heraus. Geben Sie die entsprechende achtstellige Zahl als Hexadezimalzahl der Form `0x00000000` im Graph-Properties-Fenster an.

Alternativ lässt CCS aber auch die Angabe des Variablennamens `sine_table` anstelle der Adresse zu.

Weiter muss im Graph-Properties-Fenster unter Data Properties dann noch die Anzahl der aus dem Speicher zu holenden Daten (100) und deren Interpretation (32-bit floating point) angegeben werden.

Nach der graphischen Darstellung erreichen Sie das Graph-Properties-Fenster übrigens auch über das Symbol  in der Symbolleiste des Fensters.


Wenn Sie wiederholt die Graph Properties anpassen, wird jedes mal ein neuer Datensatz aus dem Speicher geholt und ans Ende der Darstellung angehängt. Lassen Sie sich dadurch nicht irritieren und löschen Sie ggf. die komplette Darstellung über das Symbol  in der Symbolleiste des Fensters und lassen Sie über das Symbol  einen neuen Datensatz aus dem Speicher holen und darstellen.


Graphische Eigenschaften ändern:

Über das zur graphischen Darstellung gehörende Kontextmenü können einige graphische Eigenschaften der Darstellung beeinflusst werden (mit der rechten Maustaste in die Grafik klicken und im sich öffnenden Kontextmenü den gewünschten Eintrag wählen).

Dort kann unter Display as beispielsweise die hier sinnvolle Linienart Connected Line with markers gewählt werden. Das geht übrigens auch über den Eintrag Display Properties... (im Kontextmenü ganz unten) und dort in der Rubrik General mit Hilfe von Display data as. Hier kann zusätzlich in der Rubrik Axes unter Y-Axis beispielsweise das Display format Decimal mit zwei Nachkommastellen gewählt werden.

Messfunktion:

Über das Symbol  in der Symbolleiste des Fensters lässt sich der Measurement Marker Mode umschalten, wobei der eine der beiden Modi ein einfaches Ausgeben der X- und Y-Werte des dargestellten Kurvenzugs an der aktuellen Mauszeigerposition im linken oberen Fensterbereich zur Verfügung stellt (dabei sollte der Mauszeiger grob entlang des Kurvenzugs bewegt und nicht in die Darstellung geklickt werden! Ein Klicken bewirkt das Setzen eines Measurement Markers).




Über das Symbol  rechts daneben lassen sich die Eigenschaften der Messfunktion festlegen (sinnvoll: • Snap to Data, • Both).

2.3 Erweiterung des Programms

Wechseln Sie mit Hilfe des Buttons im oberen rechten Bereich der Entwicklungsumgebung zurück in die Ansicht CCS Edit und ändern Sie das C-Programm so ab, dass Sie den Sinuston über den ersten **DIP-Switch** des Demoboards (USER_SW0) ein- und wieder ausschalten können. Gleichzeitig zum Ton soll die erste **LED** (LED0) ein- und wieder ausgeschaltet werden.

Hinweis:

Das Ein- und Ausschalten des Sinustones erfolgt sinnvollerweise durch Verändern des Gain-Faktors. Der Zugriff auf die LED und den Schalter ist im Infoblatt 'DSP Starter Kit DSK6713' erläutert.

Starten Sie Compiler und Linker indem Sie bei ausgewähltem/markiertem  Projekt2 im  Project Explorer entweder im Menü Project den Eintrag  Build All wählen oder mit der rechten Maustaste das zugehörige Kontextmenü öffnen und darin Build Project wählen.

SS 2012

Fakultät für Technik, Studiengänge EIT/TI

Dipl.-Ing.(FH) Felix Becker

Nach erfolgreichem Build-Prozess erhalten Sie im Konsolenfenster die abschließende Meldung:

```
Finished building target: Project2.out
```

```
**** Build Finished ****
```

Quittieren Sie die darauf folgende Reload-Abfrage mit Yes.

Nach dem Wechsel in die Ansicht CCS Debug (Button im oberen rechten Bereich der Entwicklungsumgebung) kann das neue Programm sofort gestartet werden.

Überprüfen Sie das Ergebnis wieder mit dem Kopfhörer.

Sie werden feststellen können, dass beim Ein- und Ausschalten des Sinustons meistens ein **Knacken** zu hören ist. Das liegt daran, dass Sie den Gain-Faktor abrupt ändern und so eine Sprungstelle im Signal erzeugen.

Abhilfe:

Sie dürfen die Gain-Faktoränderung nicht abrupt vornehmen, sondern müssen den Faktor stetig bis zum gewünschten Wert ändern (engl. ramping).

Dazu verändern Sie den Gain-Faktor zum Beispiel bei jedem TX-Interrupt nur um einen Bruchteil. Die Signaländerung zieht sich dadurch in die Länge, es kommt zu keiner Sprungstelle mehr, das Knacken ist weg.

Erweitern Sie jetzt das C-Programm so, dass Sie zwei oder drei Sinustöne verschiedener Tonhöhe über jeweils einen eigenen DIP-Switch (auch gleichzeitig) ein- und wieder ausschalten können (Mäuseklavier). Dazu ist vor der Ausgabe an den Codec eine Summierung der Signale (mit entsprechender Skalierung zur Vermeidung einer Übersteuerung) notwendig.

2.4 Frage

F 2.1 Wie kommt es zu der Sprungstelle beim Ein- und Ausschalten, bzw. was genau ist die Voraussetzung dafür?

Hinweis: Die Sinustabelle wird (unabhängig vom Gain-Faktor) permanent durchlaufen.

(Die Frage ist handschriftlich in der Versuchsvorbereitung bzw. Ihrem Versuchsprotokoll zu beantworten!)

Ab hier folgt der optionale Teil des Laborversuchs (Eigenstudium):

3. Alternative Tonerzeugung mit Hilfe eines digitalen Filters

3.1 Systemfunktion digitaler Filter

Digitale Filter lassen sich allgemein durch eine Übertragungsfunktion der Form

$$H(z) = \frac{Y(z)}{X(z)} = \frac{b_1 + b_2 \cdot z^{-1} + \dots + b_{n_b+1} \cdot z^{-n_b}}{a_1 + a_2 \cdot z^{-1} + \dots + a_{n_a+1} \cdot z^{-n_a}} \quad (1.1)$$

beschreiben.

Diese Notation wird auch beim Filterentwurf in MATLAB benutzt. Die Ordnung des Filters beträgt $n = \max(n_b, n_a)$, wobei für kausale Systeme $n_b \leq n_a$ gilt – also der Nennergrad maßgebend ist.

Die zugehörige Differenzengleichung lautet:

$$a_1 \cdot y(k) + a_2 \cdot y(k-1) + \dots + a_{n_a+1} \cdot y(k-n_a) = b_1 \cdot x(k) + b_2 \cdot x(k-1) + \dots + b_{n_b+1} \cdot x(k-n_b) \quad (1.2)$$

bzw.:

$$a_1 \cdot y(k) = b_1 \cdot x(k) + b_2 \cdot x(k-1) + \dots + b_{n_b+1} \cdot x(k-n_b) - a_2 \cdot y(k-1) - \dots - a_{n_a+1} \cdot y(k-n_a) \quad (1.3)$$

Bei der Differenzengleichung handelt es sich um eine Rechenvorschrift, die gleichzeitig einen direkten Weg für das Vorgehen bei der Implementierung liefert. Der Koeffizient a_1 muss dabei noch den Wert 1 erhalten. Dies wird durch Normierung aller Koeffizienten auf a_1 erreicht. Die Differenzengleichung lautet dann:

$$y(k) = d_1 \cdot x(k) + d_2 \cdot x(k-1) + \dots + d_{n_d+1} \cdot x(k-n_d) - c_2 \cdot y(k-1) - \dots - c_{n_c+1} \cdot y(k-n_c) \quad (1.4)$$

Der gegenwärtige Wert der Ausgangsfolge $y(k)$ wird dabei aus dem gegenwärtigen Wert der Eingangsfolge $x(k)$ sowie aus vergangenen Werten von Eingangs- und Ausgangsfolge berechnet.

Die Übertragungsfunktion lautet dann:

$$H(z) = \frac{Y(z)}{X(z)} = \frac{d_1 + d_2 \cdot z^{-1} + \dots + d_{n_d+1} \cdot z^{-n_d}}{1 + c_2 \cdot z^{-1} + \dots + c_{n_c+1} \cdot z^{-n_c}} \quad (1.5)$$

In der Literatur ist meist ebenfalls diese Form der Darstellung zu finden.

Die Eigenschaften des Systems werden durch die Koeffizienten b_n und a_n bzw. d_n und c_n festgelegt.

Die Systemfunktion (1.5) beschreibt ein rekursives digitales Filter (IIR-Filter), gleichzeitig aber auch den „allgemeinen“ Fall eines digitalen Filters. Setzt man nämlich im Nenner alle Koeffizienten $c_n = 0$, dann bleibt mit dem Zähler der sog. transversale Teil der Systemfunktion übrig. Er beschreibt ein nichtrekursives digitales Filter (FIR-Filter), das in einem weiteren Laborversuch näher untersucht werden soll. Der Nenner enthält demnach den rein rekursiven Teil eines digitalen Filters.

Eine mögliche Struktur ist die Direkt-Form 1:

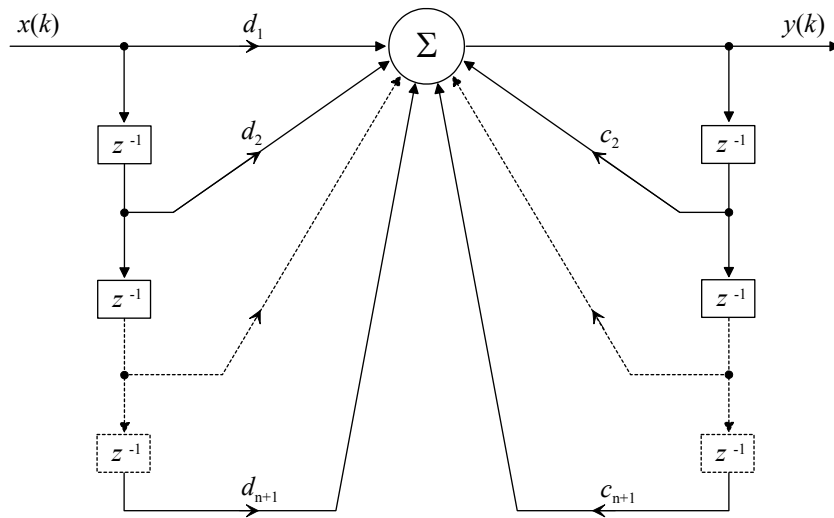


Abbildung 3.1: Allgemeine Filterstruktur Direkt-Form 1

Die Übertragungsfunktion lässt sich aber auch über die Wurzeln der Polynome im Zähler (Nullstellen) und Nenner (Pole) angeben:

$$H(z) = \frac{Y(z)}{X(z)} = k \cdot \frac{(z - z_{o1}) \cdot (z - z_{o2}) \cdot \dots \cdot (z - z_{on_b})}{(z - z_{\infty 1}) \cdot (z - z_{\infty 2}) \cdot \dots \cdot (z - z_{\infty n_a})} \quad (1.6)$$

In MATLAB stehen zur Umformung von der einen in die andere Darstellung (transfer function - $tf \Leftrightarrow zp$ - zero pole gain) folgende Funktionen zur Verfügung: `tf2zp`, `zp2tf`

Es kann aber auch eine getrennte Umformung von Zähler oder Nenner erfolgen. Dafür stehen folgende Funktionen zur Verfügung: `roots`, `poly`

Es besteht dabei folgende Konvention, die bei der Übergabe an andere Funktionen beachtet werden muss (z.B. bei `freqz`, `zplane` ... s.u.):

Die Koeffizienten von Polynomen werden als Zeilenvektor (Form: $1 \times n$) gespeichert und dadurch als solche erkannt. Im Gegensatz dazu werden die Wurzeln eines Polynoms als Spaltenvektor (Form: $n \times 1$) abgelegt.

3.2 Digitales Filter zweiter Ordnung als Alternative zur Look-up-table

Anstelle der Look-up-table zur Tonerzeugung mit einer im Speicher abgelegten Schwingung kann zur Sinussignalerzeugung auch eine rekursive Filterstruktur zweiter Ordnung verwendet werden. Dabei ist es – im Gegensatz zu der bisher behandelten Methode – möglich, jede beliebige Signalfrequenz bis zur halben Abtatsfrequenz ($f_a/2 = f_n$) des Codecs zu erzeugen.

Die **rekursive Filterstruktur zweiter Ordnung in der transponierten Direkt-Form 2** ist in Abbildung 3.2 dargestellt. Dabei hat der Koeffizient a_1 wieder den Wert 1 erhalten. Die Rechen-vorschrift lautet dann:

$$\begin{aligned} y(k) &= d_1 \cdot x(k) + z_1(k-1) \\ z_1(k) &= d_2 \cdot x(k) + z_2(k-1) - c_2 \cdot y(k) \\ z_2(k) &= d_3 \cdot x(k) - c_3 \cdot y(k) \end{aligned} \quad (1.7)$$

SS 2012

Fakultät für Technik, Studiengänge EIT/TI

Dipl.-Ing.(FH) Felix Becker

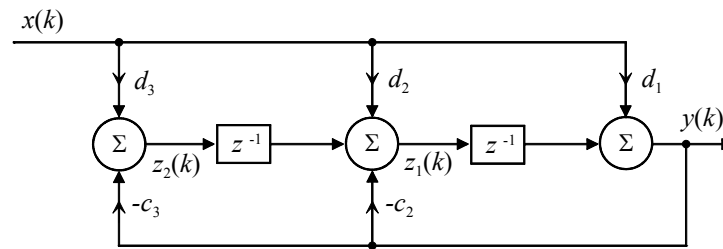


Abbildung 3.2: Rekursives digitales Filter zweiter Ordnung, Direkt-Form 2

Die rekursive Filterstruktur zweiter Ordnung realisiert ein Polpaar, das zur Erzeugung einer fortwährenden Schwingung konjugiert komplex auf dem Einheitskreis der z-Ebene platziert werden muss.

$$z_{\infty 1/2} = p_{1/2} = [\cos(\Omega_s) + j \cdot \sin(\Omega_s); \cos(\Omega_s) - j \cdot \sin(\Omega_s)]$$

Die gewünschte Signalfrequenz f_s wird dabei wie folgt in die normierte Kreisfrequenz Ω_s überführt:

$$\Omega_s = 2 \cdot \pi \cdot f_s / f_a$$

Mit der Vorgabe einer Nullstelle im Ursprung ($z_o = [0]$) ergibt sich für eine Signalfrequenz f_s von z.B. $1/16 \cdot f_a$ der in Abbildung 3.3 dargestellte Pol- Nullstellenplan. Bei einer Abtastfrequenz des Codecs auf dem DSK6713 von 32 kHz erhält man so eine Signalfrequenz von 2000 Hz.

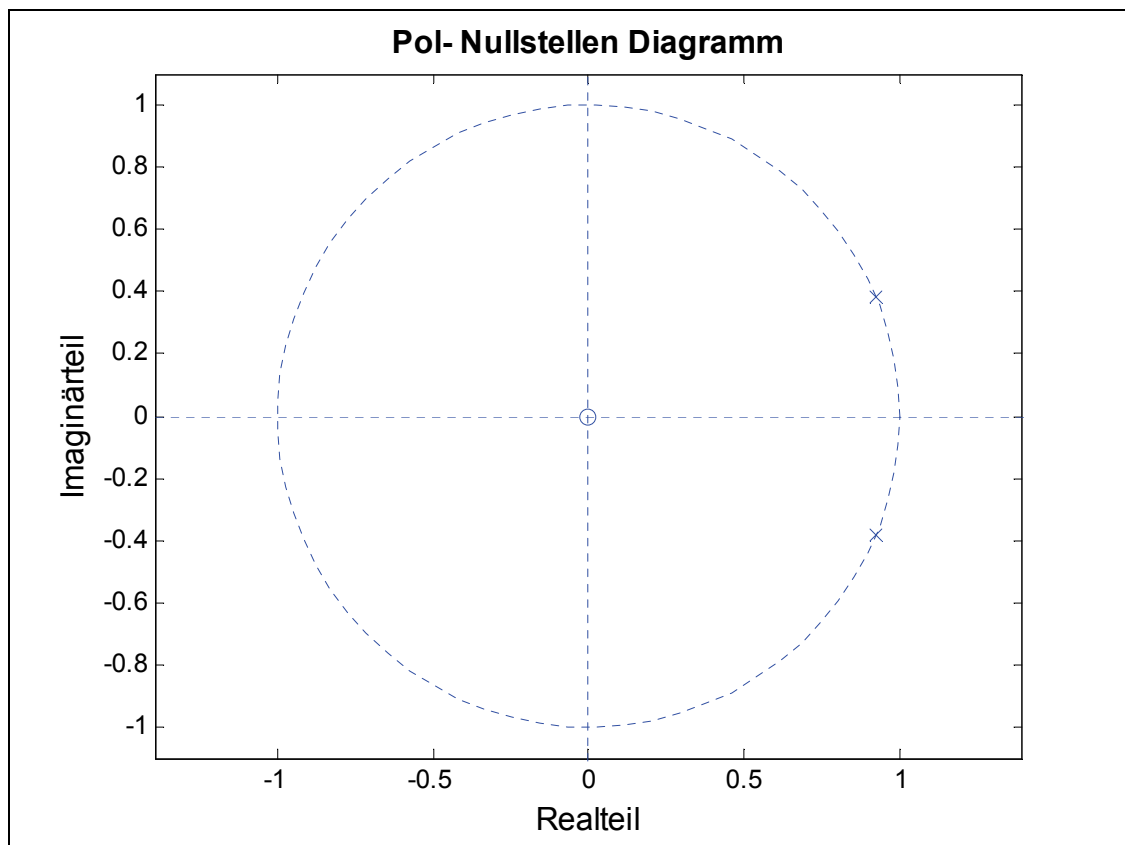


Abbildung 3.3: Konjugiert komplexes Polpaar für $f_s = 1/16 \cdot f_a$

SS 2012

Fakultät für Technik, Studiengänge EIT/TI

Dipl.-Ing.(FH) Felix Becker

Die Koeffizienten erhält man daraus z.B. mit Matlab über die folgenden Aufrufe:

```
fs=2e3
fa=32e3; Ws=2*pi*fs/fa
z=[0]
p=[cos(Ws)+j*sin(Ws); cos(Ws)-j*sin(Ws)]
gain=sin(Ws);
format long
[b, a] = zp2tf(z, p, gain)
format
```

Im Command Window von Matlab erhält man dann folgende Ergebnisse:

```
fs = 2000
Ws = 0.3927
z = 0
p = 0.9239 + 0.3827i
    0.9239 - 0.3827i
b = 0    0.382683432365090    0
a = 1.0000000000000000    -1.847759065022574    1.0000000000000000
```

Regt man ein solches Filter mit einem Dirac an, beginnt sein Ausgangssignal sinusförmig zu schwingen. Zur Simulation des Vorgangs kann in Matlab ein entsprechender Eingangsvektor x vorgegeben und mit der Funktion `filter()` das Ausgangssignal y erzeugt werden:

```
x = [1 zeros(1, 1023)];
y = filter(b, a, x);
```

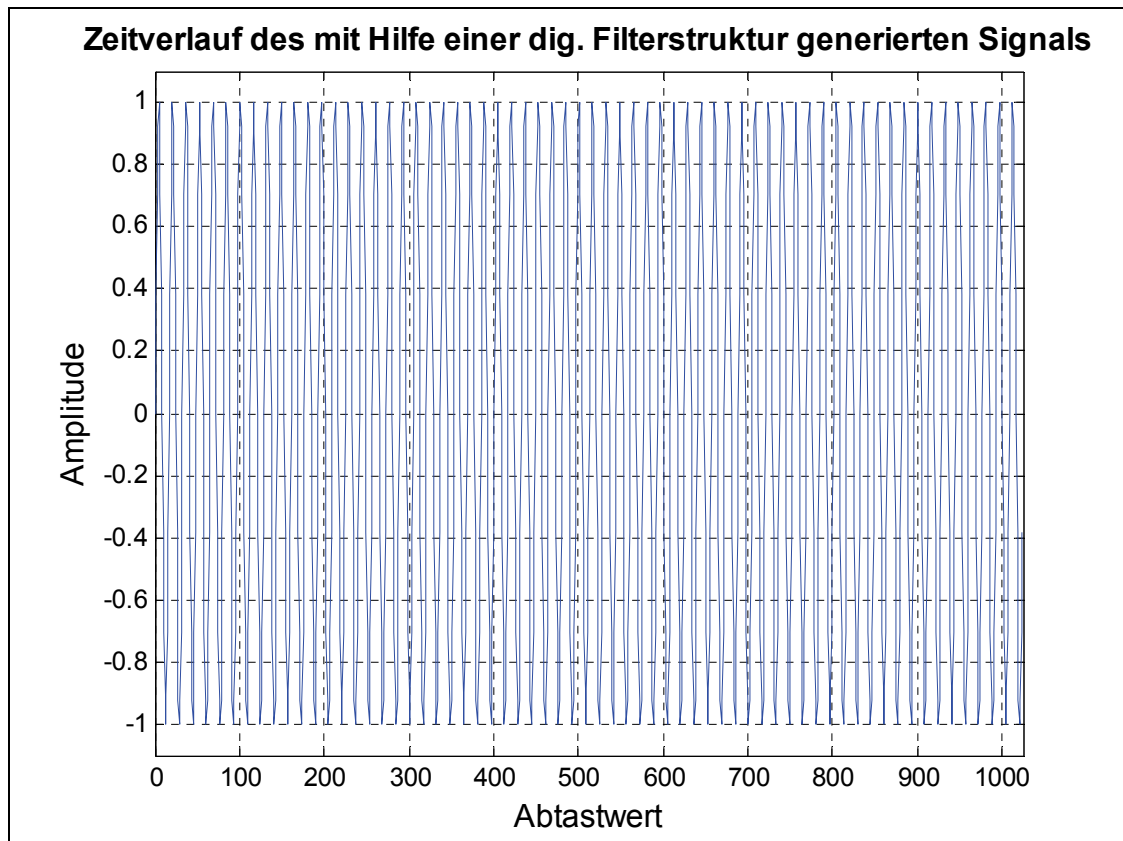


Abbildung 3.4: Zeitverlauf des generierten Signals mit $f_s = 2000$ Hz

Mit Hilfe obiger Vorgehensweise in Matlab erhalten die Koeffizienten b_1 und b_3 den Wert 0 und die Koeffizienten a_1 und a_3 den Wert 1. Damit reduziert sich die **Rechenvorschrift** zu:

$$\begin{aligned}y(k) &= z_1(k-1) \\z_1(k) &= b_2 \cdot x(k) + z_2(k-1) - a_2 \cdot y(k) \\z_2(k) &= -y(k)\end{aligned}\tag{1.8}$$

Als **Startwerte** für die Filterberechnung im DSP und damit die Sinussignalerzeugung werden die Variablen y und z_1 mit 0 initialisiert. Den Dirac zur Anregung kann man – verrechnet mit dem Koeffizienten b_2 – im Zustandsspeicher z_2 ablegen ($z_2 = b_2$).

Die Berechnung kann beispielsweise in der Endlosschleife von `main()` durchgeführt und durch Ändern eines Freigabesteuerwortes (eine global deklarierte Variable) in der Transmit-Interrupt-Routine angestoßen werden. In der Transmit-Interrupt-Routine also beispielsweise durch `calc_start = 1;`. Der Berechnungsstart in `main()` erfolgt dann durch eine if-Abfrage auf diese Variable mit anschließendem Rücksetzen von `calc_start`. In der Transmit-Interrupt-Routine müssen Sie dann anstelle eines Datenwortes aus dem Ringspeicher das Ergebnis der Filterberechnung an den Codec ausgeben (beispielsweise über die global deklarierte Variable `float y;`).

Wegen der während der Berechnung im DSP fortwährenden Skalierung der Variablenwerte in das **32 Bit Fließkommaformat** und natürlich auch wegen der Konvertierung der beiden Koeffizienten in dieses Format, kommt es zu (sich im Laufe der Zeit erhöhenden)

Abweichungen vom wahren Wert (endliche Genauigkeit, siehe Infoblatt ‚Floating Point DSP, Zahlendarstellung‘).

Dies drückt sich bei beabsichtigter Vollaussteuerung der DA-Wandler im Codec (0 dBFS mit Hilfe des Gain-Faktors 32767.0) durch eine sich nach kurzer Zeit einstellende **Übersteuerung** aus (dabei ist die Konvertierung des Ergebnisses von $|y| > 1$ und anschließender Multiplikation mit dem Gain-Faktor in eine Festkommazahl zu beachten!).

Zur Verhinderung einer solchen Übersteuerung kann z.B. die Variable y vor dem Ablegen im Zustandsspeicher z_2 auf Überschreiten des Maximalwertes überprüft und entsprechend korrigiert werden.

Code-Beispiel für die Realisierung mit Hilfe einer Endlosschleife in `main()`, angestoßen über die Freigabe `calc_start = 1;` bei jedem TX-Interrupt.

```
for (;;)
{
    if (calc_start==1)
    {
        calc_start = 0;
        y = z1;
        z1 = z2 - a2*y;
        if (y>1) y=1;
        if (y<-1) y=-1;
        z2 = -y;
    }
}
```



3.3 Frage

- F 3.1 Wie wirkt es sich aus, wenn Sie anstelle der Vorgabe einer Nullstelle im Ursprung ($z_o = [0]$) entweder keine ($z_o = []$) oder zwei Nullstellen im Ursprung ($z_o = [0; 0]$) vorgeben? Achten Sie bei der Untersuchung mit Matlab besonders auf den Beginn des generierten Zeitsignals y und beachten Sie dabei auch den erhaltenen Koeffizientenvektor b .