BEUTH HOCHSCHULE
FÜR TECHNIK
BERLIN
University of Applied Sciences

Analog-Digital-Umsetzer und digitale Signale

Laborbericht

angefertigt von

Robby Kozok, Nic Frank Siebenborn, Pascal Kahlert

in dem Fachbereich VII – Elektrotechnik - Mechatronik - Optometrie –
für das Modul Digitale Signalverarbeitung III
der Beuth Hochschule für Technik Berlin im Studiengang
Elektrotechnik - Schwerpunkt Elektronische Systeme

Datum 29. November 2015

Lehrkraft
Prof. Dr.-Ing Marcus Purat Beuth Hochschule für Technik

Inhaltsverzeichnis

Kapitel 1

Einlesen von Signalen

Im Ersten Versuch soll der Umgang mit der Software VisualDSP++ erlernt werden. Dies geschieht mit einem Projekt, welches die Signale am Eingang des Codec zum Ausgang des Codec durchreicht. Diese werden dann am PC visualisiert um den Eingang und des Ausgang vergleichen zu können.

Entsprechend der Aufgabenstellung wurde ein Projekt angelegt und der kompilierte Code auf das EVB (Evaluationsboard) übertragen. Wir legten entsprechend der Aufgabenstellung ein Signal an den rechten Kanal des ADC1 an.
Dieses Signal wurde, wie alle weiteren mit dem *VI FFT & FRF & Scope* aufgezeichnet.

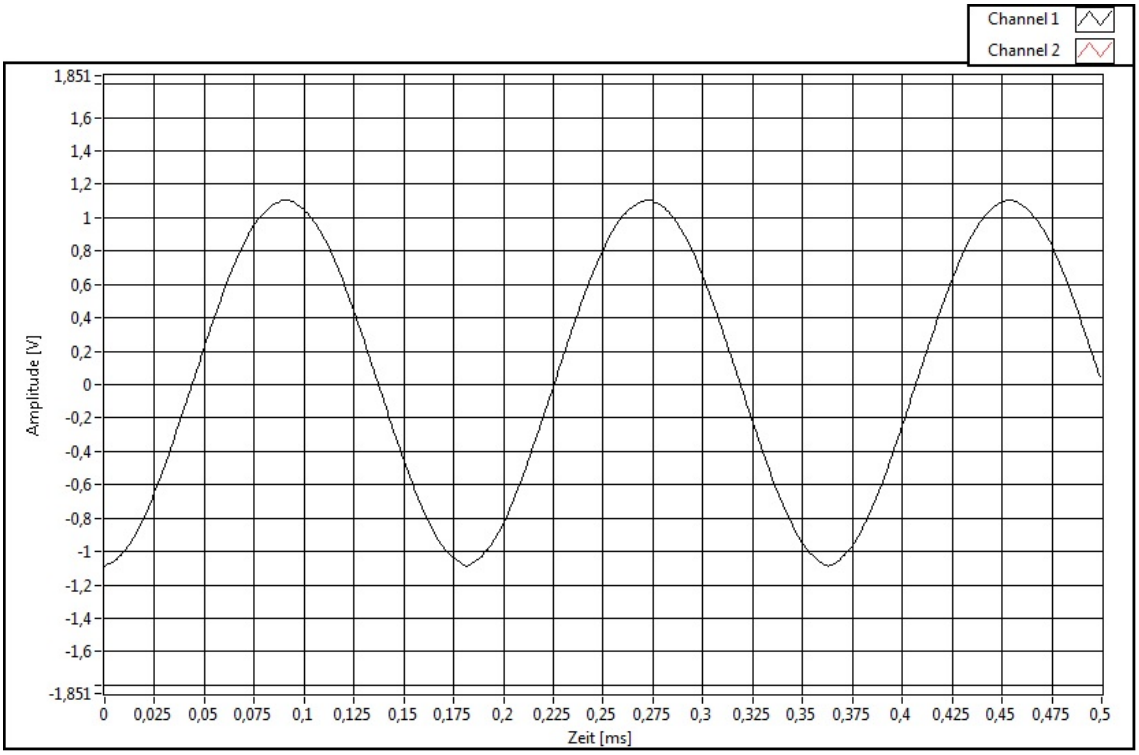

Abbildung 1.1: Darstellung des Signals am Eingang des Codec.

In der Vorbereitung wurde die Funktion copyData() erstellt, die sich wie folgt aufbaut:

```
#include "codeclib.h"
#include "copydata.h"


/*      @function       copyData
*       @brief          Kopiert die Audiodaten des Kanals "Internal ADC R0" aus dem DMA-Lesepuffer
*                               in den Speicherbereich iInput schreibt.
*       @param          input   Adresse der ersten Speicherstelle von input
*       @param          pWrite  Zeiger auf die Adresse von input
*       @param          size    Anzahl der übergebenen Werte
*       @return         void
*/
void copyData(int *input, int **pWrite, int size) {

        //Nimm den 5ten Wert aus iDMARxBuffer und schreibe diesen in den input.
        **pWrite = iDMARxBuffer[4];

        //Inkrementiere den Wert der zuletzt beschriebenen Adresse
        (*pWrite)++;

        // Wenn die obere Grenze size erreicht wurde müssen wir auf die Startadresse zurückspringen.
        if(*pWrite == input + size)
        {
                *pWrite = input;
        }

}
```

copydata.c

Wie man leicht sieht werden die Werte die der Codec liefert kopiert und in den iDMARx-Buffer fortlaufend geschrieben, bis dieser voll ist. In dem Fall wird an der ersten Position des Buffer erneut angefangen und die alten Werte werden überschrieben. Software VisualDSP++ biete die Möglichkeit den DSP zu debuggen.

Stoppt man nun den Durchlauf kann man, wie in der Aufgabenstellung beschrieben die aktuellen Werte des iDMARxBuffer ausleiten und mit MatLab visualisieren.

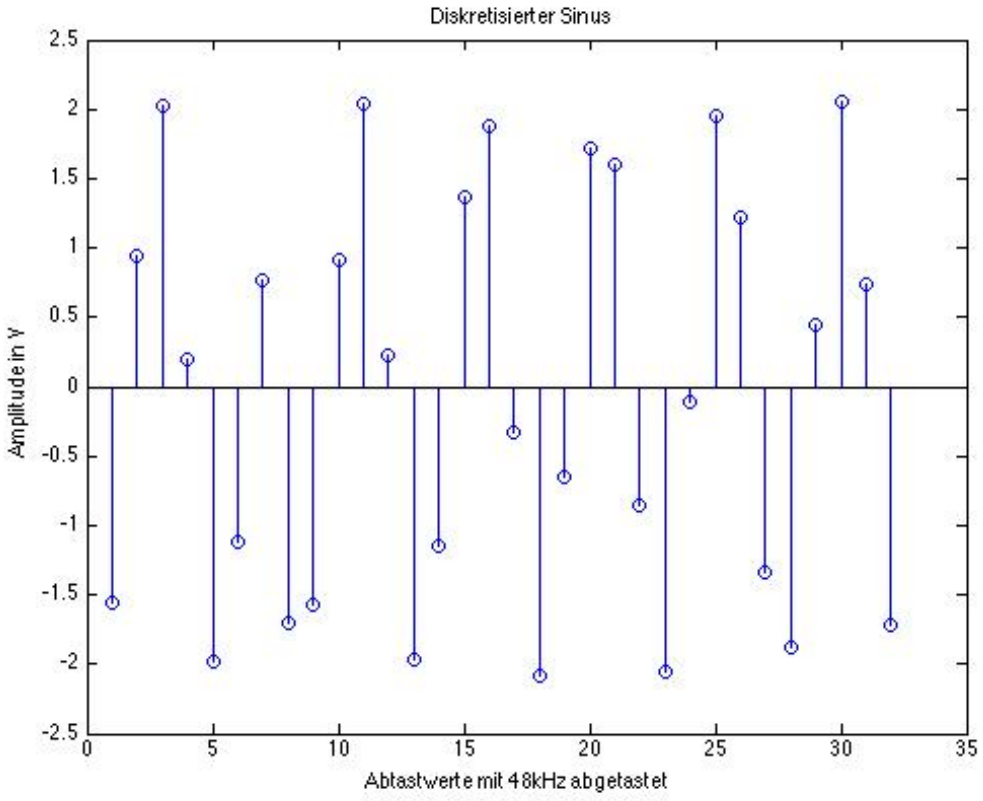

Abbildung 1.2: Darstellung der Datenreihe aus den Werten des DSP.

1.1 Auswertung

Aus Abbildung 1.1 war ein Sinussignal zu erwarten, in Abbildung 1.2 ist diese Sinussignal wieder zu erkennen.

Die Amplitude des Signals beträgt 1446560512 in der Registerdarstellung des DPS. Um diesen Wert exakt ermitteln zu können, wurde die Matlab-Funktion max() verwendet Für die normierte Kreisfrequenz wird der Ansatz

$$\omega_0 = \frac{2\pi}{N} \tag{1.1}$$

verwendet und führt bei 18 Messwerten in 2 Perioden (es folgt $N = \frac{18}{2}$) zu

$$\omega_0 = \frac{2\pi}{9}$$

Da die Diskreten Amplitudenwerte im 32-bit-Integer Format vorliegen, müssen sie in eine Spannung umgerechnet werden, hierzu sind die Daten des EVB erforderlich. Der Codec hat eine Auflösung von 24-bit bei einer Spitze-Spitze-Spannung $V_{ss} = 6,16V$.Es gilt zur

Berechnung der Amplitudenspannung V_{Amp}

$$V_{Amp} = \frac{V_{ADC} * V_{ss}}{2 * 2^{Registerbreite-1}} \tag{1.2}$$

Dies ergibt bei einem Registerwert von 1446560512 eine Spannung von $V_{ss_{max}} = 2,0579V$

Es ist an dieser Stelle nicht nachweisbar ob es sich bei dem gemessenen Maximalwert auch um das tatsächliche Maximum handelt, da der Abtastzeitpunkt nicht unbedingt der Zeitpunkt des lokalen Hochpunktes war. Auch ist die Frequenz des Ursprungssignals nur näherungsweise zu ermitteln, da die Abtastfrequenz kein ganzzahliges vielfaches der Signalfrequenz ist. Um letztgenannten Fehler zu minimieren wurden in erster Annäherung die Abtastwerte über 4 Perioden ermittelt. Die Frequenz des abgetasteten Sinus lässt sich mit

$$f_0 = \frac{f_s}{N} \tag{1.3}$$

zu $f_0 = 5,3kHz$ berechnen, da der Codec mit 48kHz abtastet.

Aus der Messung ergibt sich grafisch eine Amplitude von $V_{ss} = 1,1V$ und eine Frequenz von 5,5kHz. Die Abweichungen sind u.A. der Messung aus o.g. Gründen und des Ablesens geschuldet. Eine weitere Fehlerquelle bildet die interne Schaltung des EVB, so wie der Verstärker zwischen Eingang und ADC.

Kapitel 2

Ausgeben von Signalen

In dieser Aufgabe sollte der DSP ein Sinussignal generieren. Die Frequenz sollte dabei per Taster in 200Hz Schritten einstellbar sein. Die drei auf dem DSP angebrachten LEDs sollten die Frequenzen 200Hz, 400Hz und 4kHz darstellen.
Zur Realisierung dieser Aufgabe wurde in der Vorbereitung ein Funktion genSinus erstellt, die bei jedem Aufruf den nächsten Wert des Sinussignals ausgibt.

```
1  #include <stdio.h>
2  #include <math.h>
3
4  #define PI 3.141592653 //Definition der Zahl Pi
5  #define FABTAST 48000  //Definition der Abtastrate des Codec
6
7  /*      @function genSinus
8  *       @brief  Diese Funktion generiert fortlaufend einen Sinussignal.
9  *
10 *       @param  A ist die Amplitude des gewünschten Sinussignal
11 *       @param  Freq200 ist die gewünschte Frequenz des Sinussignal in Schritten von 200Hz.
12 *       @return Ist der aktuell berechnete Wert des Sinussignals
13 */
14 float genSinus(float A, short Freq200)
15 {
16         float sinusValue;
17         float omegaNormNeu;
18         static float omegaNorm = 0;
19
20         //Die Stellung des Zeigers wird neu berechnet.
21         omegaNormNeu = 2 * PI * ((Freq200 * 200.)/FABTAST);
22         omegaNorm += omegaNormNeu;
23
24         //Ermittlung des Sinuswertes anhand des neuen Zeigers.
25         sinusValue = A * sin(omegaNorm);
26
27         //Bei Ueberlauf von omegaNorm wird das Signal um 2Pi zurückgesetzt
28         if(omegaNorm > 2 * PI)
29         {
30                 omegaNorm -= 2 * PI;
31         }
32
33         return sinusValue;
34 }
```

gensinus.c

Die Funktion ist so Implementiert, dass sie den jeweiligen Zeitpunkten einen entsprechenden Sinuswert zuweist und diesen zurück weißt. Als Übergabeparameter sind zum einen die Amplitude und zum anderen die Frequenz vorgesehen wobei die Frequenz in 200Hz Schritten übergeben wird.
Das neue ω_{Norm}, also wird entsprechend mit

$$\omega_{Norm} = 2\pi * f \tag{2.1}$$

ermittelt und normiert.

Der Aufruf durch den Timer-Interrupt stellt dabei die Periodizität sicher. Das alte ω_{Norm} wird dann mit dem neuen ω_{Norm} addiert und mit der Funktion sin aus math.h wird der entsprechende Sinuswert ermittelt. Die Skalierung erfolgt durch multiplikation mit A, da sin() einen Normierten Wert zwischen 0 und 1 zurück gibt.
Um die LEDs und Taster entsprechend nutzen zu können wurde die main.c angepasst.

```
1  #include <ccblkfn.h>
2  #include "isr.h"
3  #include "codeclib.h"
4
5  void main(void)
6  {
7          // initialize AD1836
8          start_AD1836();
9
10         // !! set control register so that PF5 ...
11         // 8 are enabled as input, all LED PF are directed as output
12         // !! and all LED are switched off
13         ssync();
14
15         //Anfang Modifizierter Code
16         *pFIO2_DIR = 0xFFFF;     //Setze alle LED als Ausgaenge.
17         *pFIO0_DIR &= 0xFF0F;    //Setze die Taster als Eingang.
18         *pFIO0_INEN |= 0x00F0;   //Aktiviere den Input Buffer fuer die Taster.
19         //Ende Modifizierter Code
20
21         // loop forever
22         while(1) {
23                 idle(); // go asleep and wake up when external
24                                 //interrupt and ISR have been
25         }
26 }
```

main.c

Die Modifikation wurde vor der Dauerschleife eingefügt um die Taster als Eingang und die LEDs als Ausgang zu konfigurieren. Außerdem wurde über die programmable Flags der Interrupt der Taster erlaubt, sodass ein Tastendruck die isr aufruft. Des weiteren musste entsprechend die isr angepasst werden.

Kapitel 3

Verarbeiten von Signalen

Anhang A

Quelltext-Dateien

Abbildungsverzeichnis