BEUTH HOCHSCHULE
FÜR TECHNIK
BERLIN
University of Applied Sciences

Realisierung und Analyse des IIR-Filters

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 25. Januar 2016

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

Inhaltsverzeichnis

Kapitel 1

Vorbereitung

In folgender Tabelle sind die DTMF-Frequenzen sowie die normierten Kreisfrequenzen (Abtastfrequenz: 48kHz) zu sehen.

Hz Ω_n	1209 0,158257	1336 0,174881	1477 0,193338	1633 0,213759
697 0,091237	1	2	3	A
770 0,091237	4	5	6	B
852 0,091237	7	8	9	C
941 0,091237	*	0	#	D

Die normierten Kreisfrequenzen errechnen sich nach folgender Formel:

$$\Omega_n = \frac{2 * \pi * f_{DTMF}}{48kHz} \tag{1.1}$$

Nun soll mit folgenden Formeln die Ordnung der FFT bestimmt werden.

$$\Delta\Omega = \frac{2 * \pi}{N} = 2 * \Pi * \frac{\Delta f}{f_T} \tag{1.2}$$

Daraus folgt:

$$N = \frac{f_T}{\Delta f} \tag{1.3}$$

Die minimalste Frequenz Δf befindet sich zwischen den DTMF-Frequenzen 697Hz und 770Hz. Auch sollen laut Vorgabe 3 weitere Stützstellen verwendet werden, das macht insgesamt 4. Daraus ergibt sich folgende Filterordnung.

$$N = \frac{48kHz}{0,25 * (770Hz - 697Hz)} = 2630,14 \tag{1.4}$$

Da wir eine Funktion aus der DSP Library verwenden bei der eine 2er Potenz von N benötigt wird, müssen wir ebenfalls eine Filterordnung mit 2er Potenz verwenden. Dies macht eine Filterordnung von N=4096.

Kapitel 2

Fast Fourier Transformation ohne Fensterung

In dem folgenden Kapitel wird die FFT ohne Fensterung untersucht.

2.1 Aufgabenstellung

In dieser Aufgabe soll die bereits vorhandene Implementierung der FFT untersucht werden. Dafür sollte in der Vorbereitung eine Funktion entwickelt werden, welche das Betragsspektrum ermittelt.

2.2 Durchführung

Zur Überprüfung der FFT sollten zwei unterschiedliche
Dual-tone multi-frequency(Mehrfrequenzwahlverfahren) (DTMF) Frequenzen gewählt werden und jeweils als Sinus-Signal mit einer Amplitude von 100mV auf den Eingang des DSP gegeben werden. Dazu muss der vorhandene Source-Code der ISR.c den Anforderungen der Vorbereitung angepasst werden. Die Anforderungen werden durch die Abbildung 2.1 dargestellt.

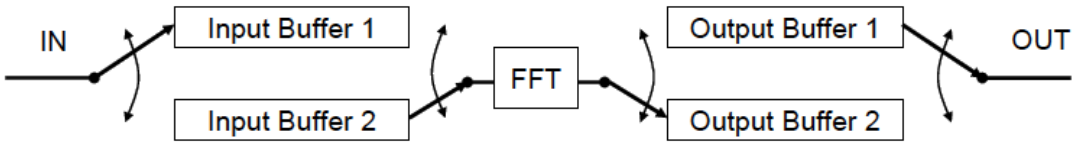

Abbildung 2.1: Segmentverarbeitung mit Wechselbuffer, Quelle: Purat, S. 17

Dabei sollte auf beiden Seiten ein zweiteiliger Buffer implementiert werden, dies hat den Hintergrund, dass die FFT immer über eine bestimmt Anzahl von Werten angewendet wird. Da die FFT aber Länger als $\frac{1}{Abtast} = \frac{1}{48kHz}$ dauert und der Buffer nicht geändert werden darf solange die FFT bearbeitet wird muss hier mit einem Arbeitsbuffer und einem temporären Buffer gearbeitet werden. Dies wurde in dem Source-Code 2.2 implementiert.

```
1 #include <fract.h>
2 #include <ccblkfn.h>
3 #include <sys\exception.h>
4 #include <cdefBF561.h>
5 #include "codeclib.h"
```

```
6  #include "isr.h"
7
8  /* Add your own global variables and definitions here */
9
10 #define TRIGGER_ON      0x7FFFFFFF
11 #define TRIGGER_OFF     0x00000000
12
13 #define INTERNAL_ADC_R1 0x5
14
15 #define INTERNAL_DAC_L0 0x0
16 #define INTERNAL_DAC_R0 0x4
17
18 char cNewFrame = 0;
19 // Pointer für die Buffer
20 fract16 *pInFrame, *pOutFrame, *pInBuffer, *pOutBuffer;
21 // Buffer Arrays
22 fract16 inBuffer1[FRAMELENGTH], inBuffer2[FRAMELENGTH],
23                 outBuffer1[FRAMELENGTH], outBuffer2[FRAMELENGTH];
24
25
26 EX_INTERRUPT_HANDLER(Sport0_RX_ISR)
27 {
28 /* Add your own local variables here */
29
30     static unsigned int iSampleCounter = 0;
31     // confirm interrupt handling
32     *pDMA2_0_IRQ_STATUS = 0x0001;
33
34 /* Add input and output and pointer setting here */
35         // Bei neuem Sample Frame müssen die Pointer umgeschaltet werden
36     if(iSampleCounter == 0){
37         if(pInFrame == inBuffer1){
38             pInBuffer   = inBuffer1;
39             pInFrame    = inBuffer2;
40             pOutBuffer  = outBuffer1;
41             pOutFrame   = outBuffer2;
42         }
43         else{
44             pInBuffer   = inBuffer2;
45             pInFrame    = inBuffer1;
46             pOutBuffer  = outBuffer2;
47             pOutFrame   = outBuffer1;
48
49         }
50         iDMATxBuffer[INTERNAL_DAC_L0] = TRIGGER_OFF;//Trigger on DACL1
51     }
52     // Schreibe den aktuellen Eingangswert in den aktuellen Eingangsbuffer
53     pInBuffer[iSampleCounter] = iDMARxBuffer[INTERNAL_ADC_R1] >> 16;
54     // Schreibe den aktuellen Ausgangswert in den aktuellen Ausgangsbuffer
55     iDMATxBuffer[INTERNAL_DAC_R0] = pOutBuffer[iSampleCounter] << 16;
56     // Zähle Samples hoch
57     iSampleCounter++;
58     // Wenn genug Samples vorhanden sind muss eine neue FFT angestoßen werden.
59     if(iSampleCounter >= FRAMELENGTH){
60         iSampleCounter = 0;
61         iDMATxBuffer[INTERNAL_DAC_L0] = TRIGGER_ON; //Trigger on DACL1
62         cNewFrame = 1;
63     }
64 }
```

Routine zum aufnehmen der Samples und umschalten der Buffer

Zuerst wurden in Zeile 20 - 22 benötigten Pointer und Arrays zum umschalten deklariert.

In Zeile 36 - 49 werden je nach dem zuletzt verwendetet Buffer die Bufferadressen geändert, dies geschieht immer bei der Aufnahme des ersten Samples. Zeile 53 und 55 dienen zum Einlesen des Signals und zum Ausgeben des Spektrums. Sobald genug Samples aufgenommen wurden wird in Zeile 59 - 62 ein Trigger gesetzt und eine Flag für die FFT gesetzt.
Im weiteren musste eine Routine zur Berechnung des Betragsspektrums erstellt werden. Diese sollte in Assembler geschrieben werden. Die Umsetzung ist in dem Source-Code 2.2 zu sehen.

```
1  .section/DOUBLE32 program;
2  .align 2;
3  _winmul:
4          // R0 = Pointer to pInFrame (fract16)
5          // R1 = Pointer to window (fract16)
6          // R2 = FFT Order
7  ._winmul.end:
8  .global _winmul;
9
10 .align 2;
11 _abs2_spec:
12         // R0 = Pointer to abs2_spectrum (fract16)
13         // R1 = Pointer to spectrum (complex fract16)
14         // R2 = FFT Order
15
16         P2 = R2; // FFT Ordnung
17         I0 = R0; // Betragsspektrum - Return
18         I1 = R1; // Komplexes Spektrum - Parameter
19
20         R3.L = 10; // Schiebe Faktor
21
22         NOP;NOP;NOP;NOP;
23
24         R1.L = W[I1++];
25
26         LSETUP(_LOOP_START, _LOOP_END) LC0 = P2; // Schleifendurchlaufanzahl
27         _LOOP_START:
28             A0      = R1.L * R1.L || R1.L = W[I1++];
29             R2.L    = (A0 += R1.L * R1.L) || R1.L = W[I1++];
30             R4.L    = ASHIFT R2.L BY R3.L;
31         _LOOP_END:
32             W[I0++] = R4.L;
33         RTS;
34 ._abs2_spec.end:
35 .global _abs2_spec;
```

Routine zum berechnen des Betragsspektrums

Im ersten Teil, von Zeile 16 - 18 mussten die Eingangsparameter übernommen werden. Diese waren zum einen die Pointer vom Komplexen Spektrum und zum Betragsspektrum, welches nun berechnet werden sollte und die Ordnung der FFT. Zur Skalierung wurde in Zeile 20 ein Schiebe Faktor von 10 festgelegt, dies entspricht einer Multiplikation mit 2^{10}. In Zeile 24 wurde der erste Wert zu Berechnung eingelesen. Zeile 26 ist der beginn der Hardware Schleife diese soll der Ordnung entsprechend durchlaufen werden. Zeile 28 - 29 bilden die Betragsfunktion der komplexen Rechnung ab, diese ist in der Gleichung 2.1 zu sehen.

$$| z | = \sqrt{a^2 + b^2} \tag{2.1}$$

In Zeile 28 wird das Quadrat eines Wertes gebildet und der nächste Wert eingelesen. Zeile 29 realisiert die Bildung des zweiten quadrates, sowie die Addition beider Werte und das Auslesen des nächsten Wertes. Die Skalierung in Zeile 30 wurde bereits in Zeile

20 vorbereitet. Am Ende der Schleife wird der berechnete Wert ausgegeben und der Pointer um eine Stelle erhöht.

2.3 Auswertung

Leck-Effekt

Bevor wir zur Analyse der FFT kommen wollen wir noch über den Leck-Effekt reden, welcher uns im Laufe der Bearbeitung der Aufgaben aufgefallen ist. In Abbildung 2.2 ist zu sehen wie der schwarze Graph eine SI-Funktion darstellt, dies entsteht durch das Tiefpass-Verhalten des Ausganges. Unser Spektrum sollte im Idealfall genau ein Impuls sein, dadurch ist das Ausgangssignal die Impulsantwort des Ausganges, welches eine SI-Funktion ist.

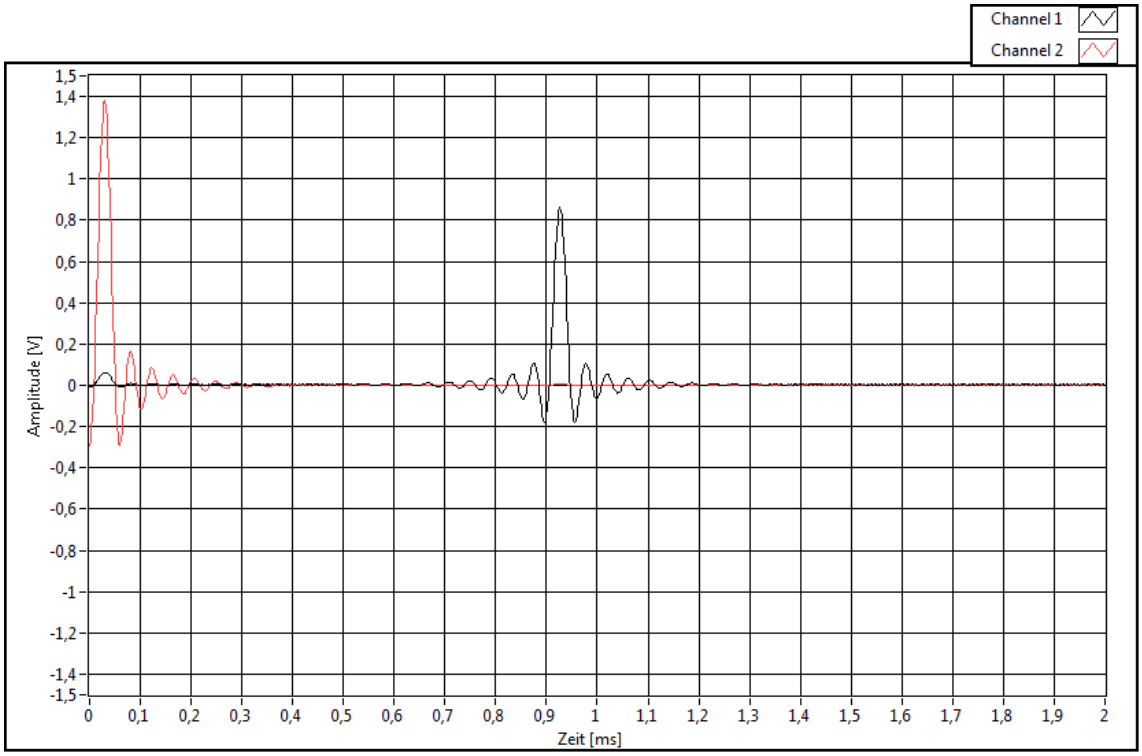

Abbildung 2.2: Ausgangssignal(505 Hz) ohne Leck-Effekt

In Abbildung 2.3 ist zu sehen, dass das Ausgangssignal keine SI-Funktion darstellt.

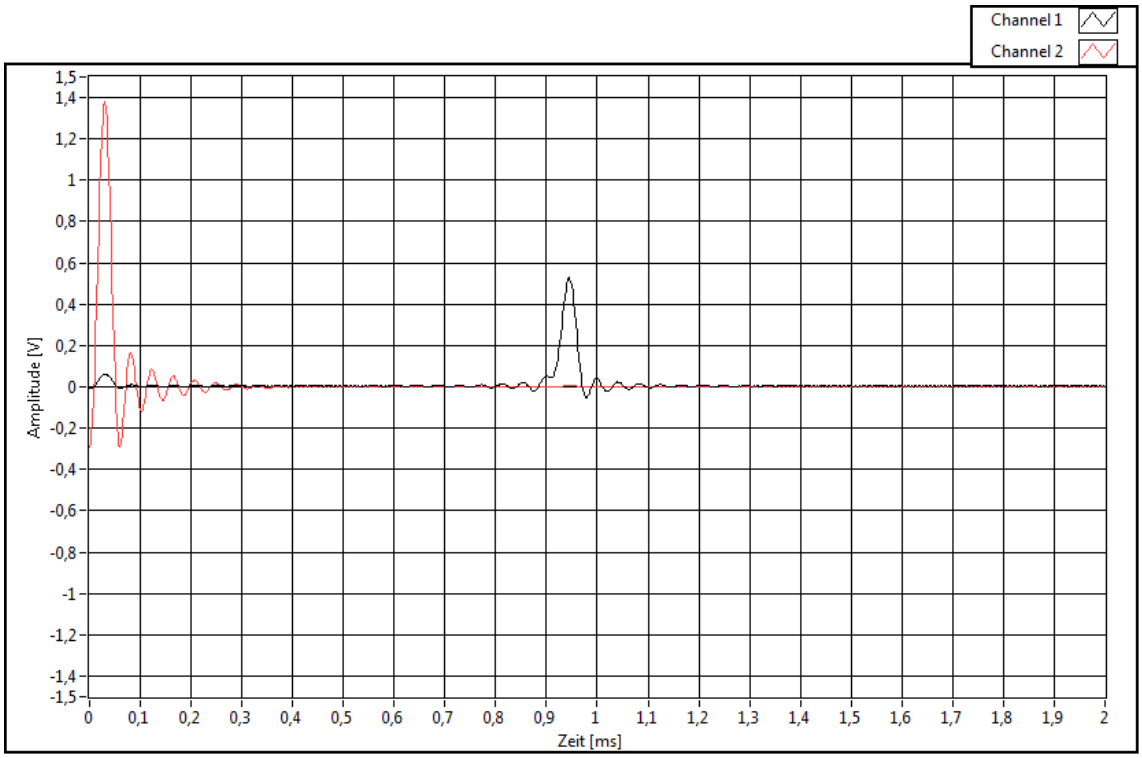

Abbildung 2.3: Ausgangssignal(511 Hz) mit Leck-Effekt

Der Leck-Effekt tritt bei Spektralanalysen auf, welche keinen unendlichen langen Beobachtungszeitraum besitzen, also praktisch bei allen. Durch den Leck-Effekt werden Frequenzen um die eigentliche Frequenz berechnet, welche nicht vorhanden sind. Dieser Effekt kann verhindert werden, indem eine Abtastfrequenz gewählt wird, welche dem ganzzahligen Vielfachen der Signalfrequenz entspricht. Natürlich sind hierbei Ungenauigkeiten zu beachten weshalb die von uns gewählten Frequenzen nicht berechnet wurden, sondern durch ausprobieren ermittelt wurden.

Mit diesem Wissen können wir nun die Analyse der DTMF Frequenzen beginnen. Als Frequenzen haben wir die ersten beiden Zeilenfrequenzen gewählt, diese sind 697 Hz und 770 Hz. Zur besseren Darstellung, haben wir jeweils ein Bild mit passender Achseneinteilung ein Detailbild genommen. Die Achseneinteilung wurde so gewählt das ein Kästchen 1,2 kHz entspricht, somit ist es möglich zu bewerten ob unsere FFT ordentlich arbeitet. In Abbildung 2.4 ist ein Impuls leicht über der Mitte des ersten Kästchens zu sehen. Dies bedeutet, dass wir davon ausgehen können das es sich hierbei um die 697 Hz handeln kann.

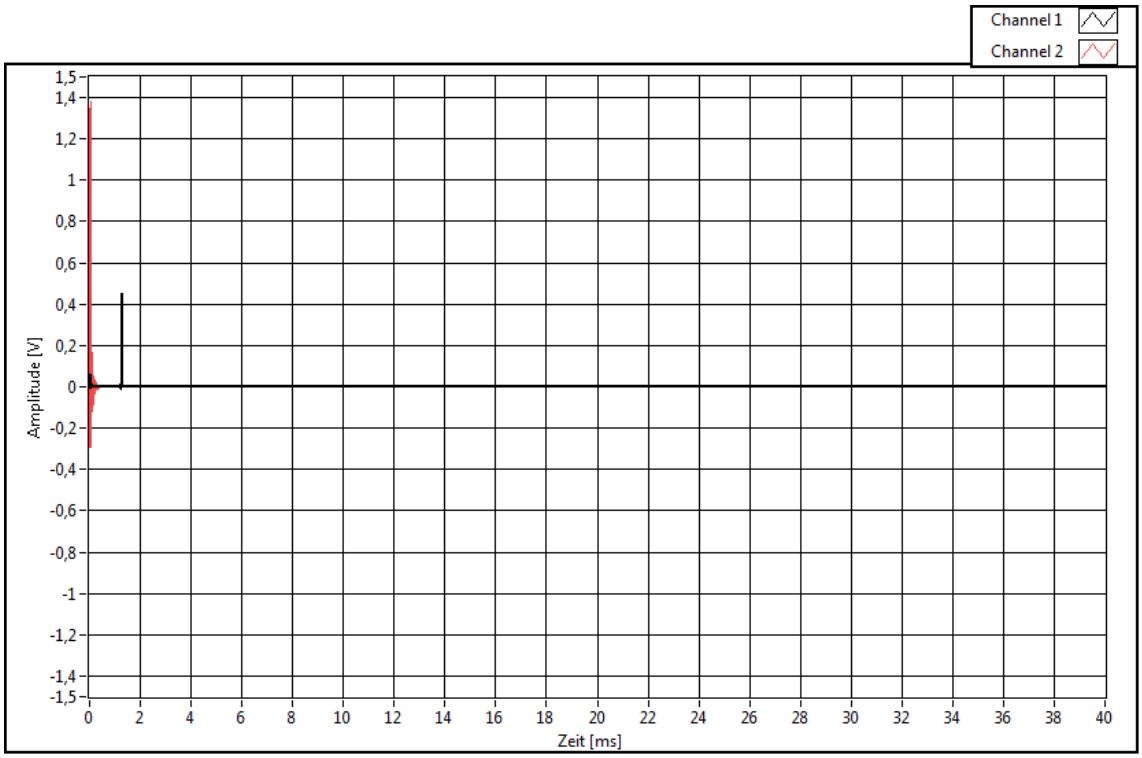

Abbildung 2.4: Ausgangsspektrum bei einem 697 Hz Eingangssignal

Abbildung 2.5 zeigt eine Nahaufnahme des Spektrums, dort ist der Leck-Effekt deutlich zu sehen. Außerdem ist nun zu sehen das sich der Impuls sehr nahe an den 697 Hz bewegt, denn $\frac{1,26ms}{2ms} * 1,2kHz = 0,762kHz$. Dies liegt augenscheinlich erstmal näher bei 770 Hz allerdings werden wir gleich sehen, dass auch das Spektrum des zweiten Signals leicht nach hinten verschoben ist.

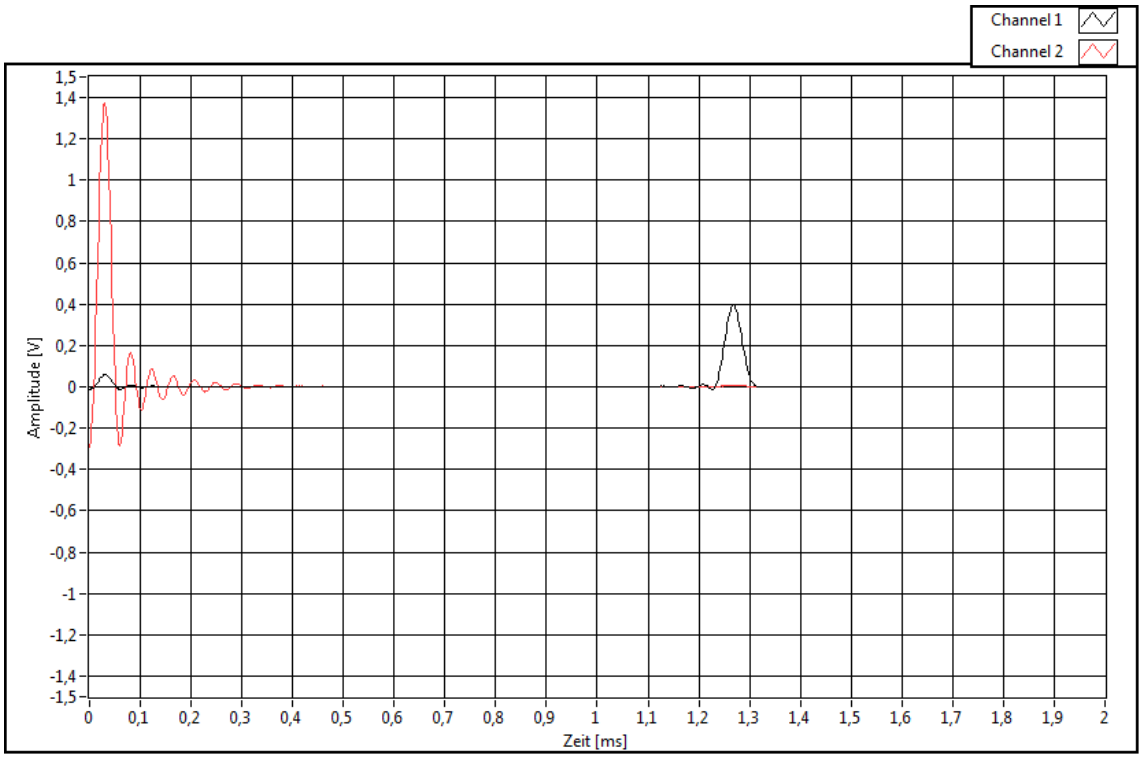

Abbildung 2.5: Ausgangsspektrum bei einem 697 Hz Eingangssignal

Abbildung 2.6 zeigt ein ähnliches Bild wie Abbildung 2.4 und weißt keine Besonderheiten auf.

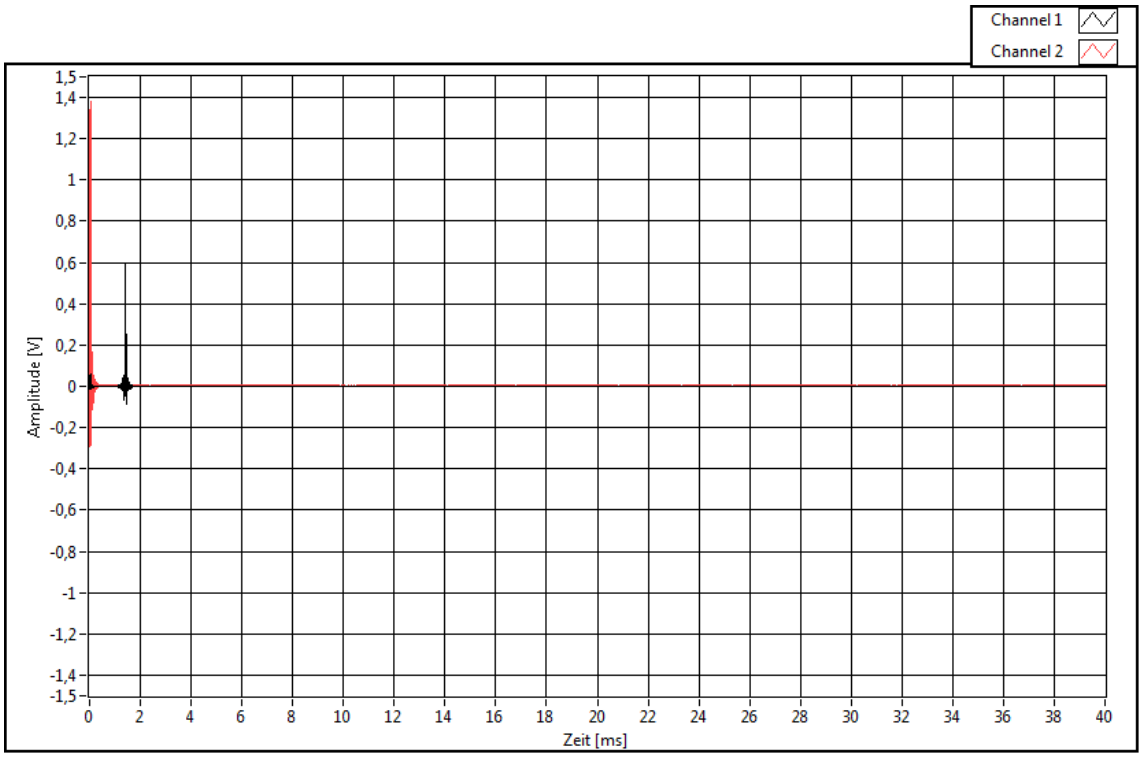

Abbildung 2.6: Ausgangsspektrum bei einem 770 Hz Eingangssignal

In Abbildung 2.7 ist zu sehen das bei 770 Hz kein oder nur ein sehr schwacher Leck-

Effekt auftritt. Außerdem ist zu sehen, dass dieses Spektrum ebenfalls weiter als erwartet nach rechts verschoben ist. An dieser Stelle wäre das Spektrum eines 840 Hz Signals zu erwarten, $\frac{1,4ms}{2ms} * 1,2kHz = 0,84kHz$

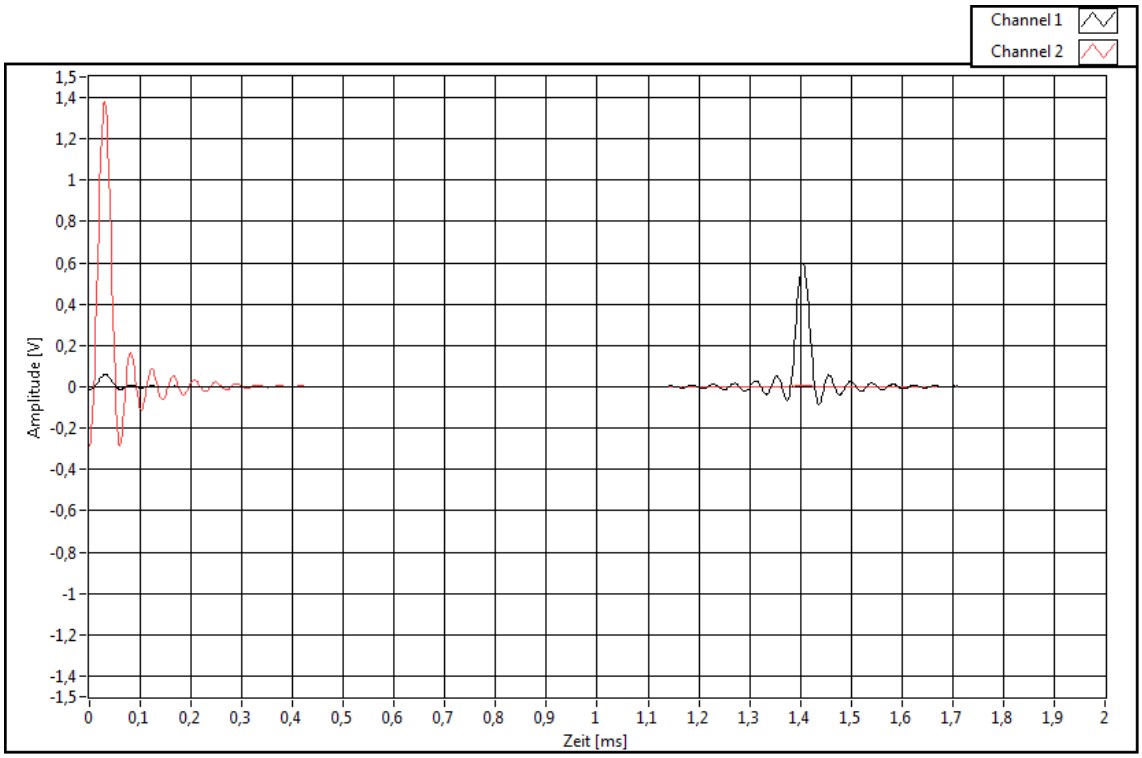

Abbildung 2.7: Ausgangsspektrum bei einem 770 Hz Eingangssignal

Kapitel 3

Fast Fourier Transformation mit Fensterung

Im folgenden Kapitel wird eine FFT mit Fensterung beschrieben.

3.1 Aufgabenstellung

Um den Leckeffekt zu verringern kann man Fensterfunktionen verwenden. Hier soll die FFT nun mit einem gefensterten Signal verwendet werden.

3.2 Durchführung

Um die Fensterung zu realisieren haben wir in der main.c und in process_data.c die entsprechenden Stellen durch Entfernen der Kommentarzeichen aktiviert.

```
1         twidfftrad2_fr16(twiddle_table,FRAMELENGTH);
2         gen_hamming_fr16(window,1,FRAMELENGTH); /* Uncomment to generate Hamming window */
3 //      gen_bartlett_fr16(window,1,FRAMELENGTH);        /* Uncomment to generate Bartlett window */
```

Codeausschnitt der modifizierten main.c

```
1          winmul(pInFrame,window,FRAMELENGTH); /* uncomment to include window function */
```

Codeausschnitt der modifizierten process_data.c

Dies ruft die in der Vorbereitung erstellte Funktion winmul auf (vergleiche hierzu den Abschnitt Vorbereitung). Zum Testen der Funktionalität legten wir ein Sinussignal mit 697Hz und einer Amplitude von 100mV an.

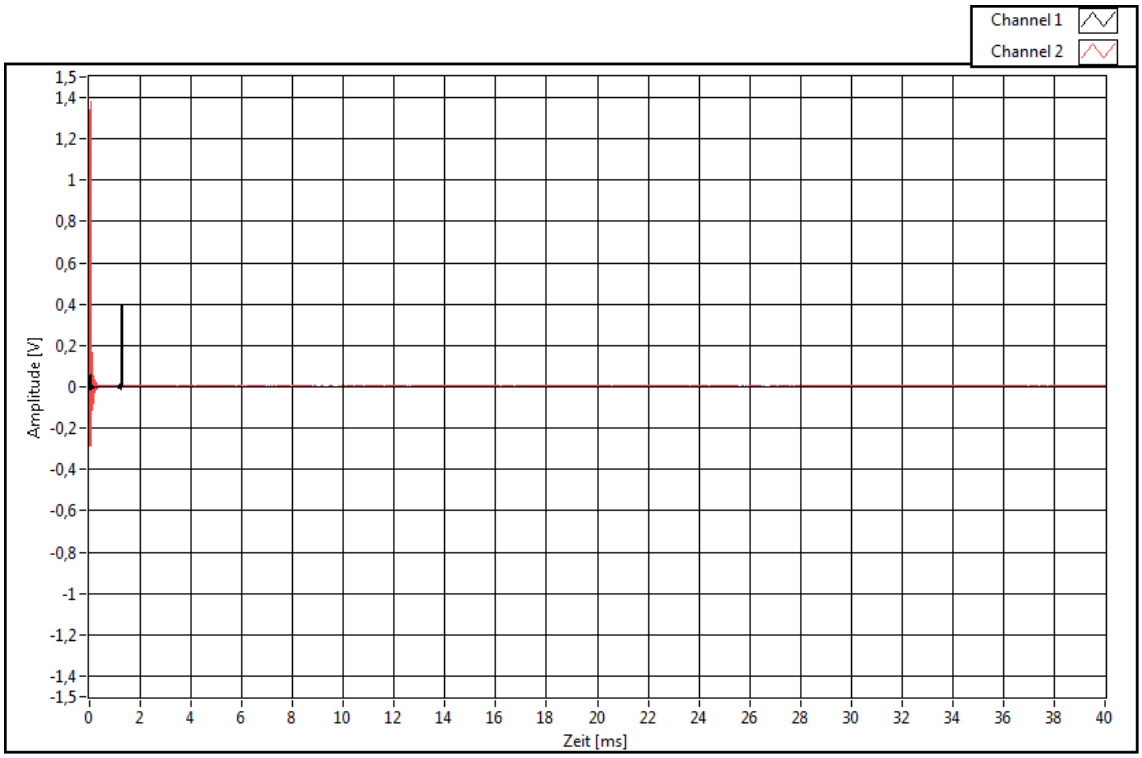

Abbildung 3.1: Hammingfenster 697Hz 100mV

Dies lieferte keinen sichtbaren Unterschied zu den vorherigen Versuchen, weshalb wir weiter hineinzoomten.

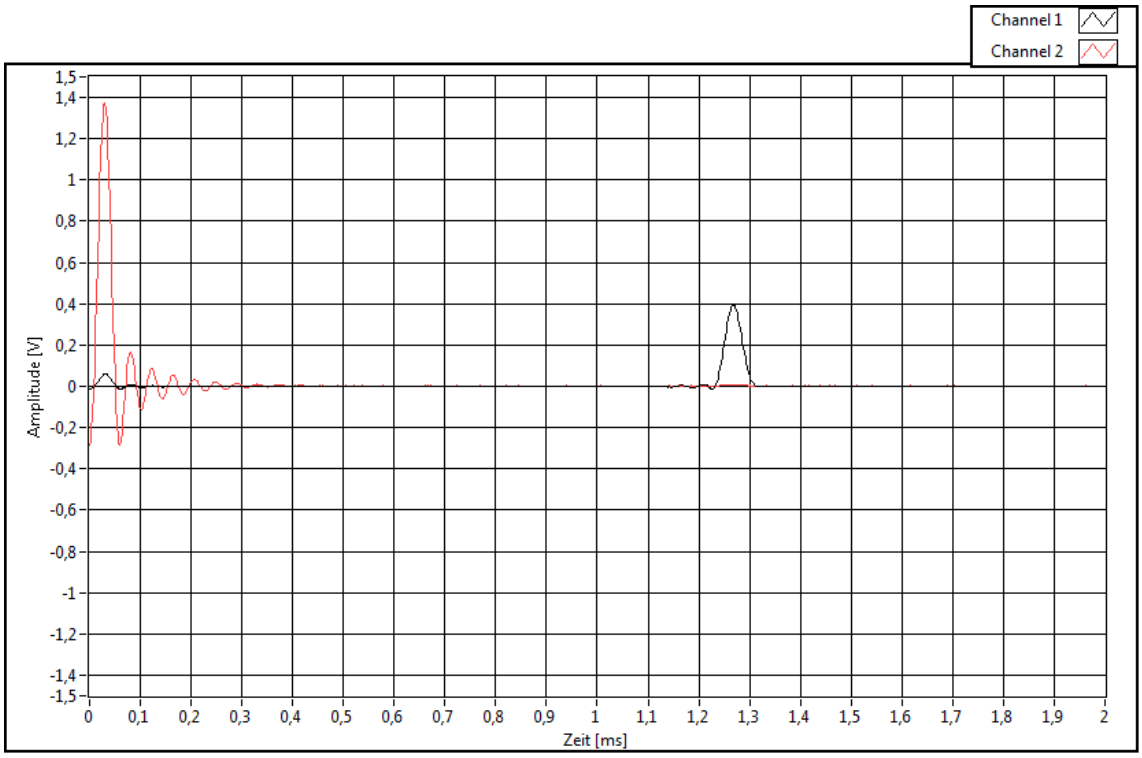

Abbildung 3.2: Hammingfenster 697Hz 100mV gezoomt

Da auch dies keine befriedigend sichtbare Verbesserung hervorbrachte haben wir in Mat-

lab die Werte der FFT mit denen der gefensterten FFT grafisch übereinander gelegt.

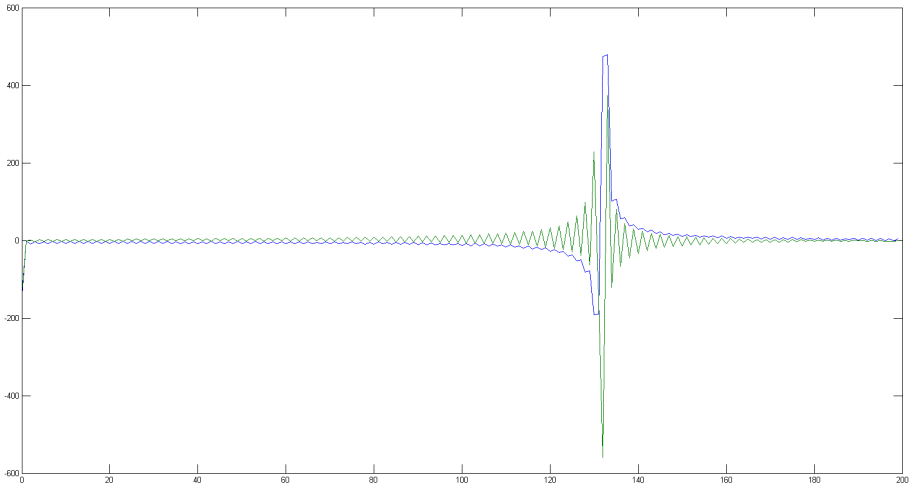

Abbildung 3.3: Ausschnitt des Spektrums des Systems

Es ist zu sehen dass die Werte die in den Speicherstellen des DSPs liegen sich sehr ähneln. Die negativen Werte erklären sich da in den Speicherstellen die komplexen Signalwerte liegen.

3.3 Auswertung

Dieses Ergebnis entspricht nicht unseren Erwartungen deckt sich aber mit den Ergebnissen des Bartlett-Fensters, die hier deshalb nicht extra aufgeführt werden.
Aus Zeitgründen haben wir dies in Absprache mit dem Übungsleites Herrn Professor Purat abgebrochen.
Wir hätten bei den gefensterten FFTs schmalere spektrallinien erwartet.

Kapitel 4

DTMF

Im finalen Teil der Übung wird ein DTMF-Detektor umgesetzt, der eingegebene Töne Dekodiert und entsprechend auf LEDs anzeigt welche Taste gedrückt wurde.

4.1 Aufgabenstellung

Es soll ermittelt werden ob ein Ton eingespielt wird und wenn ein Ton eingespielt wird müssen die zwei spektralen Spitzen detektiert und auf die LEDs kodiert werden.

4.2 Durchführung

Um dies umzusetzen wurde die Funktion process_data() angepasst und die Datei process_data.c um mehrere Funktionen erweitert.
Zuerst wurden die Indizes der DTMF-Frequenzen ermittelt. Grundlage dafür war, dass wir in 4098 Werten 48kHz abbilden. Nach der Formel $N = f_{DTMF} * \frac{4096}{48000}$ errechneten sich die Indize die nun definiert wurden.

```
1  //Zeilenfrequenzen
2  #define dtmf_697 59
3  #define dtmf_770 66
4  #define dtmf_852 73
5  #define dtmf_941 80
6
7  //Spaltenfrequenzen
8  #define dtmf_1209 103
9  #define dtmf_1336 114
10 #define dtmf_1477 126
11 #define dtmf_1633 139
```

Definition der Indizes

In einem nächsten Schritt haben wir eine Funktion erstellt welche den Mittelwert der FFT im Bereich der DTMF-Frequenzen ermittelt, dies ist notwendig da die genaue Frequenz auf Grund der groben Auflösung nicht getroffen wird. Hierfür wird über den oben gegebenen Indize sowie den nächst höheren und niedrigeren gemittelt.

```
1 short dtmf_mittelwert(short dtmf_freq){
2         return (*(pOutFrame + dtmf_freq - 1) + *(pOutFrame + dtmf_freq) + *(pOutFrame + dtmf_freq + 1))/3;
3 }
```

Mittelwert der DTMF Frequenzen

Auf Grund von Rauschen kann es auch während keine DTMF-Frequenz anliegt zu Ausschlägen an diesen Positionen kommen. Um die Mächtigkeit des Rauschens abschätzen zu können wird ein Rauschmittelwert aus den Bereichen außerhalb der DTMF-Frequenzen ermittelt.

Hierbei wird unterhalb oder oberhalb der Frequenzen der Mittelwert über alle Indizes ermittelt, wie es in der Aufgabe gefordert war.

```
short dtmf_noise_mittelwert(rausch_bereich bereich){
        int sum = 0;
        int i = 0;

        switch(bereich){
                case LOWER:
                        for(i = 4; i < 51; i++){ // 50Hz - 600Hz
                                sum = sum + pOutFrame[i];
                        }
                        return sum/(51 - 4);

                case HIGHER:
                        for(i = 149; i < 363; i++){ // 1750Hz - 4250Hz
                                sum = sum + pOutFrame[i];
                        }
                        return sum/(362 - 149);

                default:
                        return 0;
        }
}
```

Bestimmung Rauschmittelwert

Als nächstes haben wir eine Abfrage implementiert, welche diese Funktionen für alle Frequenzen und die Vergleichsbereiche ermittelt.

```
short zeilenWerte[4];   // Mittelwerte der Zeilenfrequenzen
        short spaltenWerte[4];  // Mittelwerte der Spaltenfrequenzen
        short rauschWerte[2];
        //Zeilenmittelwerte
        zeilenWerte[0] = dtmf_mittelwert(dtmf_697);
        zeilenWerte[1] = dtmf_mittelwert(dtmf_770);
        zeilenWerte[2] = dtmf_mittelwert(dtmf_852);
        zeilenWerte[3] = dtmf_mittelwert(dtmf_941);

        //Spaltenmittelwerte
        spaltenWerte[0] = dtmf_mittelwert(dtmf_1209);
        spaltenWerte[1] = dtmf_mittelwert(dtmf_1336);
        spaltenWerte[2] = dtmf_mittelwert(dtmf_1477);
        spaltenWerte[3] = dtmf_mittelwert(dtmf_1633);

        //Rauschmittelwerte
        rauschWerte[0] = dtmf_noise_mittelwert(LOWER);
        rauschWerte[1] = dtmf_noise_mittelwert(HIGHER);
```

Ermittlung der einzelnen Mittelwerte

Nachdem nun die entsprechenden Mittelwerte errechnet sind müssen die Maxima ermittelt werden um herauszufinden ob und welche Frequenzen anliegen. Dabei muss der Rauschmittelwert um das doppelte überschritten werden, um sicher zu gehen, dass das Rauschen bei den DTMF-Frequenzen nicht zufällig größer ist. Dies funktioniert zuverlässig, da die DTMF-Frequenzen sehr Deutliche peaks darstellen und das Rauschen in den meisten fällen 0 ist.

```
short max_mittelwert(short dtmf_mittelwert[], short rausch_mittel){
        short tempMax = 0;
        int i;
        short index = 4;
        for(i = 0; i < 4; i++){
                if(dtmf_mittelwert[i] > tempMax){ //Mittelwert größer als
                //letzter Maximalwert
                        if(dtmf_mittelwert[i] > (rausch_mittel>>1)){ //Mittelwert größer
                        //als Rauschmittelwert
                                index = i;
                                tempMax = dtmf_mittelwert[i];
                        }else{
                                index = 4;
                                tempMax = rausch_mittel>>1;
                        }
                }
        }
        return index;
}
```

Bestimmung der Maxima

Um die Ermittelten Maxima den LEDs zuzuordnen wurde folgende Matrix gebildet. Dabei gilt dass die letzte Spalte sowie die letzte Zeile mit 0 aufgefüllt ist um den Fall das keine 2 DTMF-Frequenzen anliegen abzufangen. Die eingetragenen Zahlen entsprechen der Konfiguration des Registers, was die LEDs ansteuert.

```
const short leds[5][5] = {
                {0x0001, 0x0002, 0x0004, 0x0008, 0x0000}, // 1 2 3 A x --> LED 13 - 16
                {0x0010, 0x0020, 0x0040, 0x0080, 0x0000}, // 4 5 6 B x --> LED 17 - 20
                {0x0100, 0x0200, 0x0400, 0x0800, 0x0000}, // 7 8 9 C x --> LED 5 - 8
                {0x1000, 0x2000, 0x4000, 0x8000, 0x0000}, // * 0 # D x --> LED 9 - 12
                {0x0000, 0x0000, 0x0000, 0x0000, 0x0000}  // x x x x x --> Keine
        };
```

Zuordnung zu den LEDs

Als letztes muss diese Funktionalität nur noch aufgerufen werden und dem entsprechenden Register zugeordnet werden. Der vollständige c-Code ist im Anhang des Dokumentes zu sehen.

```
        testZ = max_mittelwert(zeilenWerte, rauschWerte[0]);
        testS = max_mittelwert(spaltenWerte,rauschWerte[1]);

        *pFIO2_FLAG_D = leds[testZ][testS];
```

Konfiguration Register

4.3 Auswertung

Wie bereits erwähnt ist der Rauschmittelwert meist 0, die Mittelwerte bei den DTMF-Frequenzen ist folgende Beispielhafte Tabelle gemessen worden.

DTMF-Frequenz	Indize -1	Indize	Indize +1	Mittelwert
697	1024	8192	7168	5461,33
770	2048	14336	1024	5802,67
852	2048	14336	1024	5802,67
941	1024	13312	3072	5802,67
1209	19456	0	0	6485,33
1336	1024	13312	3072	5802,67
1477	3072	13312	1024	5802,67
1633	18432	0	0	6144,00

Der Test des Programms lief erfolgreich. Wir legten die DTMF-Töne in einer Dauerschleife an und beobachteten ein einfaches Lauflicht in korrekter Reihenfolge, auch Stichproben-artige Einzeltöne wurden korrekt detektiert. Der Laborbetreuer Herr Professor Purat hat die Funktionalität abgenommen.

Anhang A

Quellcode

```
#include <ccblkfn.h>
#include <filter.h>
#include "isr.h"
#include "tools.h"

complex_fract16 temp[FRAMELENGTH];
complex_fract16 spectrum[FRAMELENGTH];
complex_fract16 twiddle_table[FRAMELENGTH];
fract16 window[FRAMELENGTH];

//Zeilenfrequenzen
#define dtmf_697 59
#define dtmf_770 66
#define dtmf_852 73
#define dtmf_941 80

//Spaltenfrequenzen
#define dtmf_1209 103
#define dtmf_1336 114
#define dtmf_1477 126
#define dtmf_1633 139

//Rauschfrequenzen
typedef enum rausch_bereich{
        LOWER,
        HIGHER
}rausch_bereich;

short dtmf_mittelwert(short);
short dtmf_noise_mittelwert(rausch_bereich);
short max_mittelwert(short[], short);

void process_data()
{
        short i;
        short zeilenWerte[4];   // Mittelwerte der Zeilenfrequenzen
        short spaltenWerte[4];  // Mittelwerte der Spaltenfrequenzen
        short rauschWerte[2];
        short testZ, testS;

        const short leds[5][5] = {
                {0x0001, 0x0002, 0x0004, 0x0008, 0x0000}, // 1 2 3 A x --> LED 13 - 16
                {0x0010, 0x0020, 0x0040, 0x0080, 0x0000}, // 4 5 6 B x --> LED 17 - 20
                {0x0100, 0x0200, 0x0400, 0x0800, 0x0000}, // 7 8 9 C x --> LED 5 - 8
                {0x1000, 0x2000, 0x4000, 0x8000, 0x0000}, // * 0 # D x --> LED 9 - 12
                {0x0000, 0x0000, 0x0000, 0x0000, 0x0000}  // x x x x x --> Keine
        };
```

```
48
49          //winmul(pInFrame,window,FRAMELENGTH); /* uncomment to include window function */
50          rfft_fr16(pInFrame,temp,spectrum,twiddle_table,1,FRAMELENGTH,0,0);
51          abs2_spec(pOutFrame,spectrum,FRAMELENGTH);
52
53          //Zeilenmittelwerte
54          zeilenWerte[0] = dtmf_mittelwert(dtmf_697);
55          zeilenWerte[1] = dtmf_mittelwert(dtmf_770);
56          zeilenWerte[2] = dtmf_mittelwert(dtmf_852);
57          zeilenWerte[3] = dtmf_mittelwert(dtmf_941);
58
59          //Spaltenmittelwerte
60          spaltenWerte[0] = dtmf_mittelwert(dtmf_1209);
61          spaltenWerte[1] = dtmf_mittelwert(dtmf_1336);
62          spaltenWerte[2] = dtmf_mittelwert(dtmf_1477);
63          spaltenWerte[3] = dtmf_mittelwert(dtmf_1633);
64
65          //Rauschmittelwerte
66          rauschWerte[0] = dtmf_noise_mittelwert(LOWER);
67          rauschWerte[1] = dtmf_noise_mittelwert(HIGHER);
68
69          testZ = max_mittelwert(zeilenWerte, rauschWerte[0]);
70          testS = max_mittelwert(spaltenWerte,rauschWerte[1]);
71
72          *pFIO2_FLAG_D = leds[testZ][testS];
73
74  /* Add the code for detection and decoding here */
75  }
76
77
78  short dtmf_mittelwert(short dtmf_freq){
79          return (*(pOutFrame + dtmf_freq - 1) + *(pOutFrame + dtmf_freq)
80          + *(pOutFrame + dtmf_freq + 1))/3;
81  }
82
83  short dtmf_noise_mittelwert(rausch_bereich bereich){
84          int sum = 0;
85          int i = 0;
86
87          switch(bereich){
88                  case LOWER:
89                          for(i = 4; i < 51; i++){ // 50Hz - 600Hz
90                                  sum = sum + pOutFrame[i];
91                          }
92                          return sum/(51 - 4);
93
94                  case HIGHER:
95                          for(i = 149; i < 363; i++){ // 1750Hz - 4250Hz
96                                  sum = sum + pOutFrame[i];
97                          }
98                          return sum/(362 - 149);
99
100                 default:
101                         return 0;
102         }
103 }
104
105 short max_mittelwert(short dtmf_mittelwert[], short rausch_mittel){
106         short tempMax = 0;
107         int i;
108         short index = 4;
109         for(i = 0; i < 4; i++){
110         //Mittelwert groesser als letzter Maximalwert
```

```
111                 if(dtmf_mittelwert[i] > tempMax){
112                 //Mittelwert groesser als Rauschmittelwert
113                         if(dtmf_mittelwert[i] > (rausch_mittel>>1)){
114                                 index = i;
115                                 tempMax = dtmf_mittelwert[i];
116                         }else{
117                                 index = 4;
118                                 tempMax = rausch_mittel>>1;
119                         }
120                 }
121         }
122         return index;
123 }
```

Die von uns angepasste process_data.c

Abbildungsverzeichnis

Abkürzungsverzeichnis

DTMF Dual-tone multi-frequency(Mehrfrequenzwahlverfahren). 3