

HÖHERE TECHNISCHE BUNDESLEHRANSTALT

HOLLABRUNN

Höhere Abteilung für Elektronik – Technische Informatik

Klasse / Jahrgang: 5BHEL	Gruppe: 2b	Übungsleiter: Dipl.-Ing. Reisinger
Übungsnummer: II / 4	Übungstitel: Digitale Filter	
Datum der Übung: 15.12.2015	Teilnehmer: Graussam, Holzinger, Mischling, Schwarzmann	
Datum der Abgabe: 12.01.2016	Schrittführer: Graussam	Unterschrift:

	Beurteilung
Aufgabenstellung	
Dokumentation	
Messschaltungen	
Messtabellen	
Berechnungen	
Programmlistings	
Auswertung	
Diagramme	
Berechnungen	
Simulationen	
Schlussfolgerungen	
Kommentare	
Inventarliste	
Messprotokoll	
Form	
Summe	

1 Allgemeiner Teil

Titel der Übung: Digitale Filter
Übungsnummer: II / 4
Übungsplatz: T10
Datum der Übung: 15.12.2015
Klasse: 5BHEL
Schriftführer: Lukas Graussam
Übungsteilnehmer: Lukas Graussam
Thomas Holzinger
David Mischling
Philipp Schwarzmann

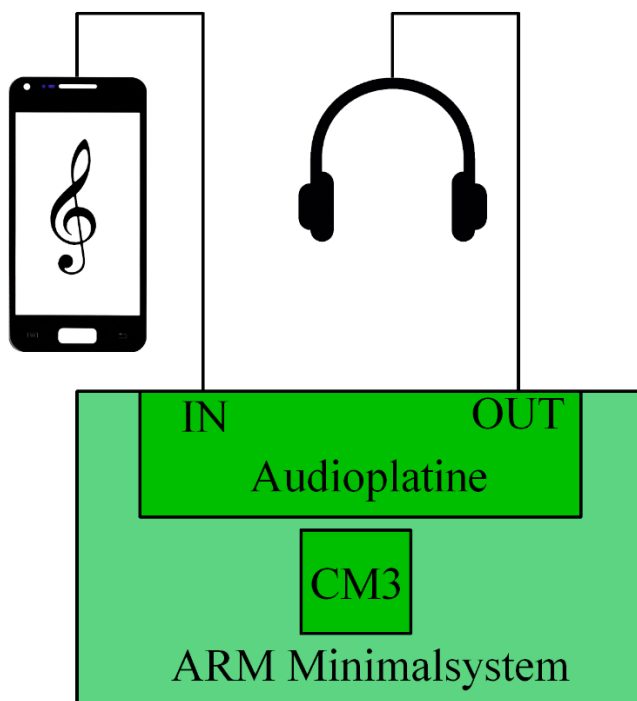
Inhaltsverzeichnis

1	Allgemeiner Teil	2
2	Funktionstest	3
2.1	Messschaltung	3
2.2	Programmcode	3
2.3	Ergebnis	4
3	FIR-Filter	4
3.1	Aufgabenstellung	4
3.2	Messschaltung mit Bode100	5
3.3	Matlab Design	5
3.4	Filteraufbau	7
3.5	Programmcode	8
3.6	Messung	9
3.7	Zeitmessung	10
3.7.1	Messschaltung Zeitmessung	11
3.7.2	Ergebnis	12
4	IIR-Filter	13
4.1	Aufgabenstellung	13
4.2	Messschaltung mit Bode100	13
4.3	Matlab Design	13
4.4	Filteraufbau	16
4.5	Programmcode	16
4.6	Messung	17
4.7	Zeitmessung	18
5	Q15 – Rechnungen	19
6	Erkenntnisse	19

2 Funktionstest

Zu Beginn der Übung wurde uns vom Übungsleiter ein ARM Minimalsystem mit einer Audioplatine ausgehändigt. Die Funktion des Systems sollte mit einem Programm getestet werden, welches die Daten vom Eingang ohne Veränderung am Ausgang ausgibt. Am Eingang sollte eine Audioquelle (in unserem Fall ein Smartphone) und am Ausgang Kopfhörer angeschlossen werden. Wenn man die abgespielte Musik ohne Veränderung am Ausgang mit den Kopfhörern hört, ist der Test erfolgreich.

2.1 Messschaltung



2.2 Programmcode

Vom Übungsleiter wurde uns ebenfalls ein fertiges Keil-uVision Projekt zur Verfügung gestellt, welches die Daten vom Eingang direkt am Ausgang ausgibt. Auf der nächsten Seite folgt der Programmcode der für uns wichtigen Funktion, welche mit der Abtastrate von 8kHz aufgerufen wurde.

```
void process_data(void)
{
    in_1=Audio_IN[0];           //Audiowerte in eine hilfsvARIABLE speichern
                                //fuer filter-Upro
    in_2=Audio_IN[1];

    out_1 = in_1;
    out_2 = in_2;               //Ausgang=Eingang, Durchverbindung

    Audio_OUT[0] = out_1;       //bearbeiteten Werte auf die DMA2 Adresse
                                //spielen
    Audio_OUT[1] = out_2;
}
```

2.3 Ergebnis

Der Test verlief bei uns ohne Probleme, die abgespielte Musik war mit den Kopfhörern einwandfrei hörbar.

3 FIR-Filter

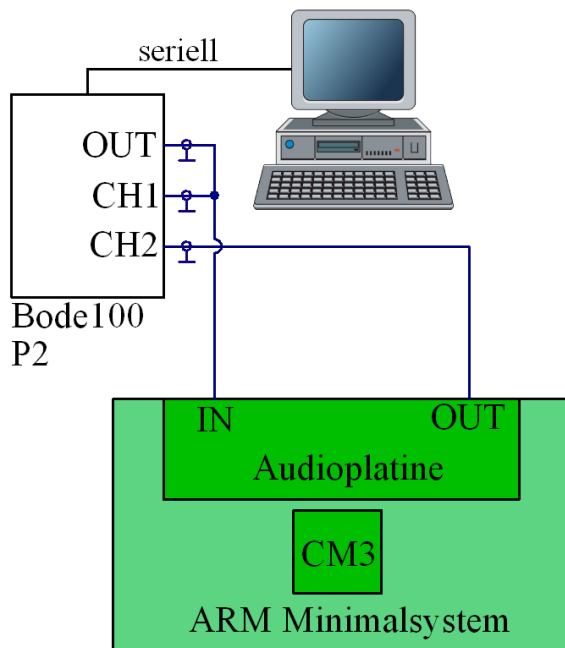
3.1 Aufgabenstellung

Die erste Aufgabe war es, einen FIR-Filter mit Hilfe von Matlab zu designen und diesen dann zu implementieren. Anschließend sollte das Filter mit dem Netzwerkanalysator Bode 100 (P2) gemessen werden und mit der Matlab Berechnung verglichen werden. Zum Abschluss sollte noch die Dauer des Interrupts gemessen werden, damit ersichtlich ist, wieviel Zeitreserve zwischen den Interrupts bleibt.

Das Filter sollte folgende Eigenschaften besitzen:

- FIR-Hochpass
- 4. Ordnung
- $f_g=1020\text{Hz}$
- $f_s=8\text{kHz}$

3.2 Messschaltung mit Bode100



3.3 Matlab Design

Der vorgegebene Filter wurde mit Hilfe des FDA-Tools von Matlab designt. Um das FDA-Tool zu öffnen reicht es in Matlab den Befehl „`fdatool`“ einzugeben. Folgende Einstellungen wurden vorgenommen:

- Response Type: Highpass
- Design Method: FIR – Equiripple
- Specify Order: 4
- Frequency Specifications:
 - $F_s = 8000 \text{ Hz}$
 - $F_{\text{stop}} = 980 \text{ Hz}$
 - $F_{\text{pass}} = 1020 \text{ Hz}$
- Magnitude Specifications (Um das Verhältnis von Sperr- (W_{stop}) zu der Durchlassdämpfung (W_{pass}) festzulegen):
 - $W_{\text{stop}} = 50$
 - $W_{\text{pass}} = 1$

Auf der nächsten Seite folgt ein Screenshot unserer Konfiguration, nachdem „Design Filter“ gewählt wurde.

Filter Design & Analysis Tool - [untitled1.da*]

File Edit Analysis Targets View Window Help

Current Filter Information

Structure: Direct-Form FR
Order: 4
Stable: Yes
Source: Designed

Store Filter ...
Filter Manager ...

Magnitude Responses (dB)

Frequency (kHz)

Filter Order: 4

Response Type: ☐ Lowpass ☒ Highpass ☐ Bandpass ☐ Bandstop

Design Method: ☐ IR ☒ FR

Specify order: ☒ Specify order: ☐ Minimum order

Options: Density Factor: 20

Frequency Specifications

Units: Hz

Fs: 8000 Fstop: 980 Fpass: 1020

Magnitude Specifications

Enter a weight value for each band below.

Wstop: 50 Wpass: 1

Design Filter

Designing Filter ... Done

Der nächste Schritt war es, aus dem designten Filter den Header zu exportieren, der die Filter-Koeffizienten enthält, wobei wir folgendermaßen vorgegangen sind:

Targets → Generate C Header → Einstellungen unverändert lassen →
Generate → Speicherort auswählen

Als Speicherort bietet sich logischerweise der Ordner des uVision-Projekts an. Hier der wesentliche Ausschnitt des generierten Header Files:

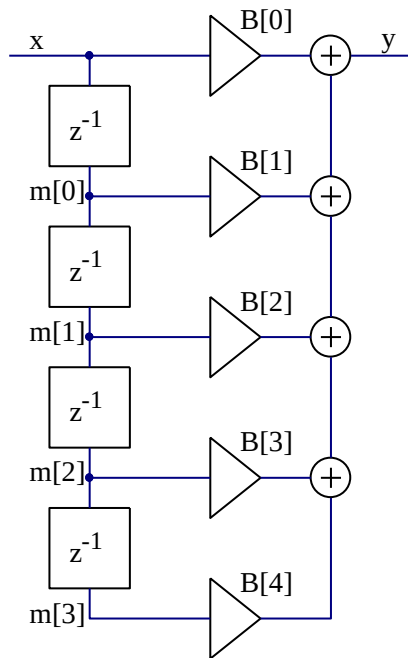
```
/* General type conversion for MATLAB generated C-code */
#include "tmwtypes.h"
/*
 * Expected path to tmwtypes.h
 * D:\Programme\MATLAB\R2015b\extern\include\tmwtypes.h
 */
const int BL = 5;
const real64_T B[5] = {
0.1249263527611, -0.4983878933701, 0.7274442981008, -0.4983878933701,
0.1249263527611
};
```

Es mussten im Endeffekt nur 2 Änderungen vorgenommen werden, zum einen wurde das `include` entfernt und zum anderen wurde der Datentyp der Filterkoeffizienten auf `float` geändert:

```
/* General type conversion for MATLAB generated C-code */
//#include "tmwtypes.h"
/*
 * Expected path to tmwtypes.h
 * D:\Programme\MATLAB\R2015b\extern\include\tmwtypes.h
 */
const int BL = 5;
const float B[5] = {
0.1249263527611, -0.4983878933701, 0.7274442981008, -0.4983878933701,
0.1249263527611
};
```

3.4 Filteraufbau

Auf der nächsten Seite folgt eine Grafik des Filteraufbaus. Die Bezeichnungen in der Grafik stimmen mit den Variablennamen im Programm überein.



Die Werte B[0] bis B[4] befinden sich in dem von Matlab generierten Header. Dieser muss zum Projekt hinzugefügt werden. Die Punkte m[0] bis m[3] sind die Zwischenspeicher, die im Programm benötigt werden.

3.5 Programmcode

```
#include "armv20_std.h"
#include "arm_math.h"
#include "Hochpass1.h"
//-----Speicheradressen der Audiowerte-----//
q15_t Audio_IN[2], Audio_OUT[2]; // Jeweils 2 Kanäle fuer IN und OUT

q15_t in_1;
q15_t out_1;

void process_data(void)
{
    float x; // Input als float
    static float m[4]={0}; // Die 4 Zwischenspeicher muessen zu Beginn 0
                           // sein
    float y; // Output als float
    int i; // Fuer die Zaehlschleifen

    LED0=1; // fuer Zeitmessung

    in_1=Audio_IN[0]; // Audiowerte in eine hilfsvARIABLE speichern fuer
                     // filter-Upro
    arm_q15_to_float(&in_1,&x,1); // Input zu float konvertieren

    y = B[0]*x; // 1 Stufe
    for(i=0;i<4;i++)
    {
        y+=B[i+1]*m[i]; // Additionen der restlichen Stufen
    }
    for(i=3;i>=1;i--)
    {
        m[i]=m[i-1]; // Zwischenspeicher aktualisieren
    }
    m[0]=x; // Erster Zwischenspeicher auf den Input
```

```
arm_float_to_q15(&y,&out_1,1);    // Output wieder ins q15 Format
                                   // umwandeln
Audio_OUT[0] = out_1;             // bearbeiteten Werte auf die DMA2
                                   // Adresse spielen

LED0=0; // LED fuer Zeitmessung zuruecksetzen
}
```

Die Funktion „arm_q15_to_float“ befindet sich im erhaltenen Projektordner unter:
Filter_8kHz \ DSP_Lib \ SupportFunctions

und muss ebenfalls zum Projekt hinzugefügt werden. Wir entschieden uns die Rechnungen mit `float` Variablen durchzuführen (siehe Programmcode). Beim Programmieren muss man darauf achten, dass die Zwischenspeicher in der richtigen Reihenfolge aktualisiert werden (siehe Programmcode). Der inkludierte Header „Hochpass1.h“ ist der zuvor von Matlab generierte, der die Filterkoeffizienten enthält. Das setzen der LED0 während des Interrupts wird erst bei der Zeitmessung benötigt. Es wurde nur ein Kanal des Systems genutzt.

3.6 Messung

Nachdem das Programm geflashed war, wurde mit dem Bode100 eine Messung durchgeführt (siehe 3.2. Messschaltung mit Bode100).

Mit der Software des Geräts musste zuerst folgendes eingestellt werden:

Configuration → Device Configuration → beide Channel auf Extern schalten

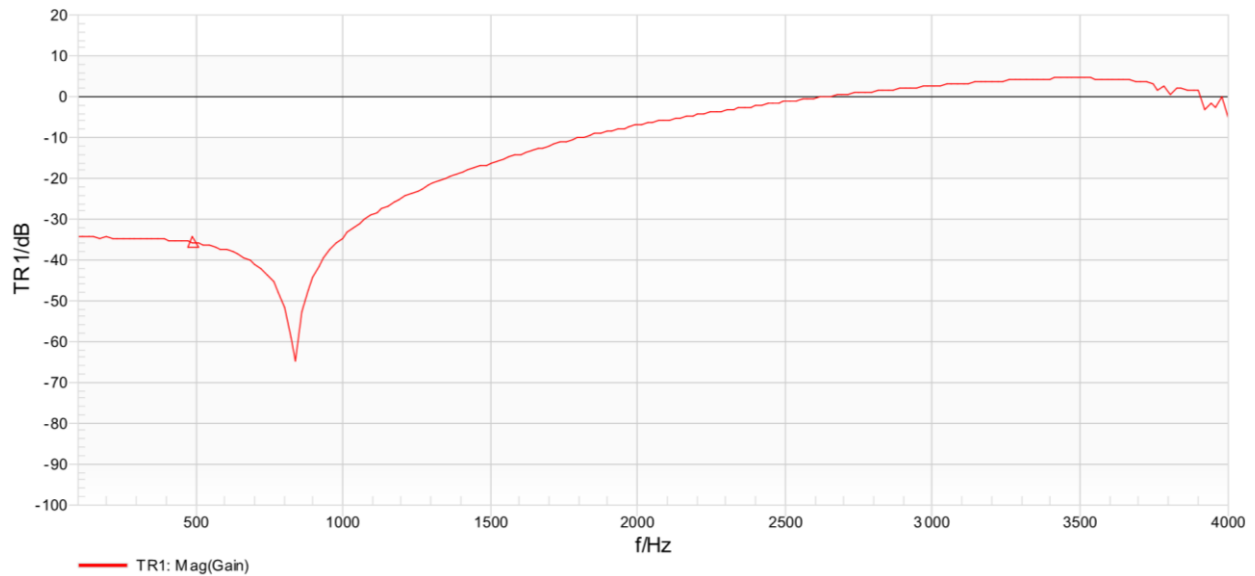
Außerdem sollte „Frequency Sweep“ ausgewählt sein um den Frequenzgang zu erhalten.

Zum Abschluss wurden noch folgende Einstellungen vorgenommen:

Start Frequency: 100 Hz

Stop Frequency: 4 kHz (bis zur halben Abtastrate)

Der gemessene Frequenzgang befindet sich auf der nächsten Seite.



Vergleicht man die Messung mit dem Matlab generierten Diagramm (Seite 6), erkennt man, dass der Verlauf relativ gut übereinstimmt. Die gemessene Sperrdämpfung beträgt ca. -35dB (aus dem Diagramm abgelesen).

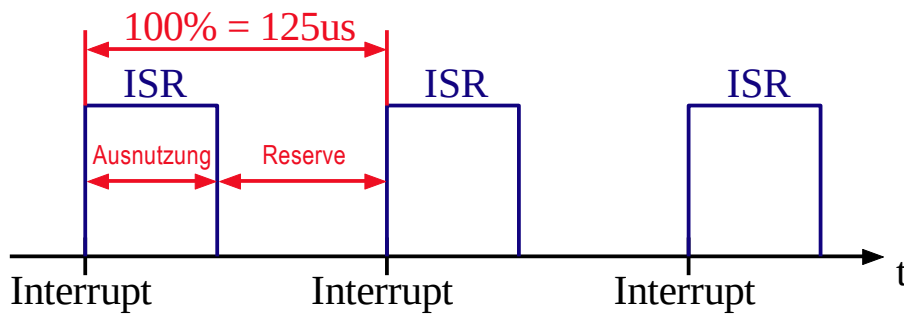
Des Weiteren wurden mit Hilfe der Cursor Messpunkte aufgenommen und mit Matlab verglichen:

Frequenz	Amplitude gemessen	Amplitude Matlab
[Hz]	[dB]	[dB]
1020	-33,00	-31,64
1500	-16,11	-15,56
2000	-7,00	-6,50
3500	4,56	5,20

Wie zusehen ist stimmen auch die Werte relativ gut überein.

3.7 Zeitmessung

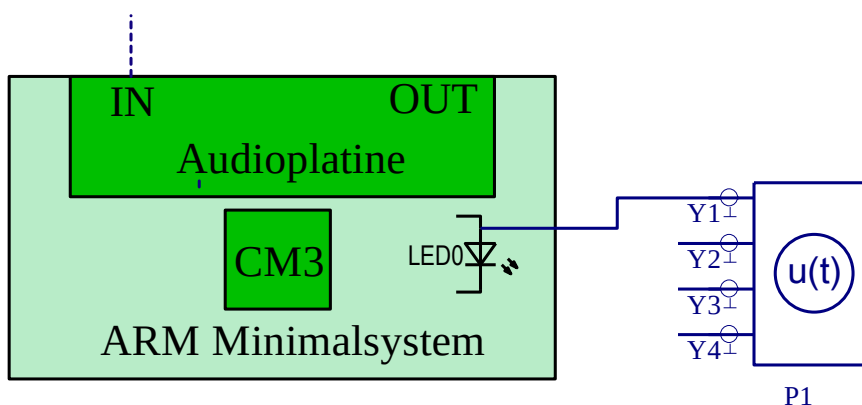
Der nächste Schritt war es, die Zeitreserve zu messen, die zwischen den Interrupts bleibt, bzw. die Zeitausnutzung in Prozent. Auf der nächsten Seite folgt eine Prinzip Zeichnung des zeitlichen Ablaufs. Die Interrupts erfolgen in unserem Fall in einem Abstand von 125us, was sich aus der Abtastrate von 8kHz ergibt.



Um die Zeit zu messen wurde am Beginn der `process_data` Funktion die „LED0“ auf 1 gesetzt und am Ende der Funktion wieder auf 0 (siehe Seite 8/9 Programmcode). Dabei war zu beachten, dass bei dem ausgehändigten uVision Projekt im Hauptprogramm die LEDs in einer Dauerschleife verändert wurden. Diese Programmzeile musste entfernt werden:

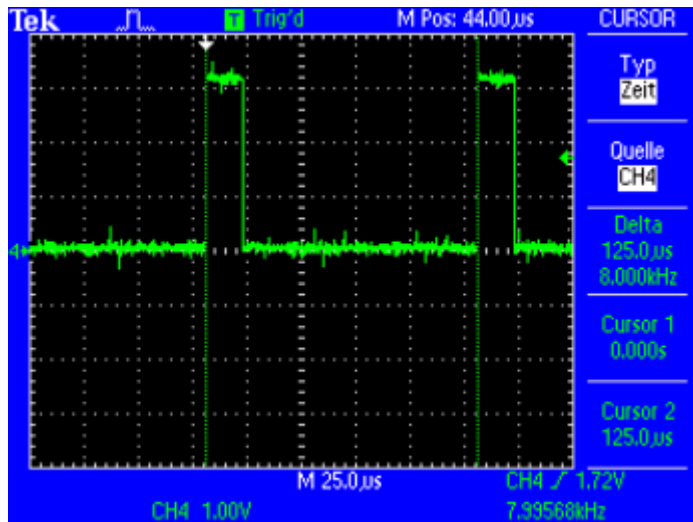
```
int main (void)
{
    char x=0;
    SetSystem(); //hw_config.h
    //RCC-DeInit, HSE-config, PLL-Config, Sysclock=PLL, DMA-Init, etc
    //Set_SysCLK(SysCLK_HSI);
    //SysCLK=HSI = 8MHz
    NVIC_init(14,1);
    init_leds_switches(); //Leds und Schalter initialisieren
    set_leds(0); //Leds auf 0 setzen
    Start_Audio_Processing(); //Audioadapter init
    while(1)
    {
        //set_leds(x);
        x++;
        wait_ms(1000);
    }
}
```

3.7.1 Messschaltung Zeitmessung

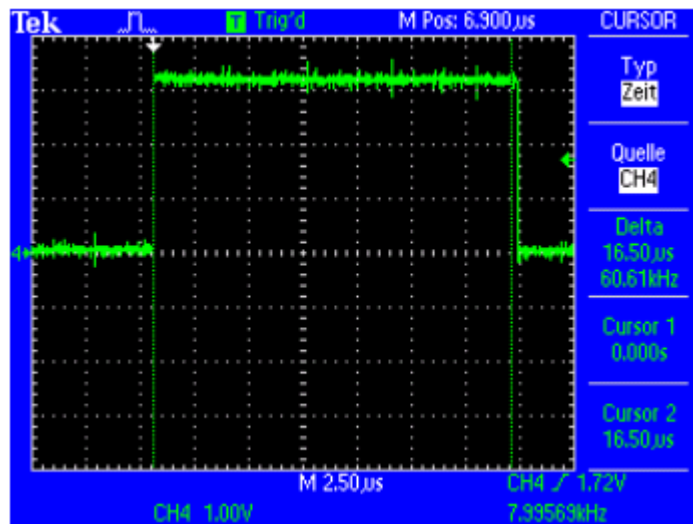


3.7.2 Ergebnis

Hier das Bild der ersten Messung, welches den Ablauf der Interrupts zeigt (vgl. mit Seite 11 oben). Mit den Cursors wurde die Dauer zwischen den Interrupts gemessen, welche wie erwartet 125µs betrug (8kHz).



Hier die Messung der Zeitdauer des Interrupts:



Der rechte Cursor der Messung befindet sich nicht genau auf der Flanke, da die ISR-Dauer der Messung etwas schwankte. Gemessen wurde eine Zeit von 16,5µs.

$$\text{Ausnutzung} = \frac{t_{ISR}}{t_{ges}} = \frac{16,5\mu s}{125\mu s} = \sim 13,2\%$$

$$\text{Reserve} = 100\% - \text{Ausnutzung} = \sim 86,8\%$$

4 IIR-Filter

4.1 Aufgabenstellung

Die zweite Aufgabe war es, einen IIR-Filter zu designen und diesen dann ebenfalls zu implementieren. Anschließend sollte das Filter wieder mit dem Netzwerkanalysator Bode 100 (P2) gemessen werden und mit der Matlab Berechnung verglichen werden. Zum Abschluss sollte noch, genauso wie beim FIR, die Dauer des Interrupts gemessen werden, damit ersichtlich ist, wieviel Zeitreserve zwischen den Interrupts bleibt.

Das Filter sollte folgende Eigenschaften besitzen:

- IIR-Hochpass
- 2. Ordnung
- $f_g=1020\text{Hz}$
- $f_s=8\text{kHz}$

4.2 Messschaltung mit Bode100

Die Messschaltung war identisch wie beim FIR-Filter (siehe: 3.2 Messschaltung mit Bode100; Seite 5).

4.3 Matlab Design

Folgendes wurde im FDA-Tool eingestellt (Informationen zum FDA-Tool - Seite 5)

- Response Type: Highpass
- Design Method: IIR – Butterworth
- Specify Order: 2
- Frequency Specifications:
 - Units: Hz
 - $F_s = 8000\text{ Hz}$
 - $F_c = 1020\text{ Hz}$

Auf der nächsten Seite folgt ein Screenshot unserer Konfiguration.

Filter Design & Analysis Tool - [untitled.fda *]

File Edit Analysis Targets View Window Help

Current Filter Information

Structure: Direct-Form II
Order: 2
Stable: Yes
Source: Converted

Store Filter ...
Filter Manager ...

Magnitude Response (dB)

Magnitude (dB)

Frequency (kHz)

Response Type

Lowpass
Highpass
Bandpass
Bandstop

Design Method

Butterworth
Equiripple

Filter Order

Specify order:
Minimum order

Options

There are no optional parameters for this design method.

Frequency Specifications

Units: Hz
Fs: 8000
Fc: 1020

Magnitude Specifications

The attenuation at cutoff frequencies is fixed at 3 dB (half the passband power)

Design Filter

C file generated

Zuerst wurde direkt nach dem Klick auf Design Filter der C-Header exportiert. Dabei kam es zu dem Problem, dass die Filterkonstanten als mehrdimensionales Array generiert wurden. Dies kann wie folgt umgangen werden:

Im FDA-Tool nachdem auf „Design Filter“ geklickt wurde:

Edit → Convert to single section

Hat man dies getan, sollte im FDA-Tool die „Current Filter Information“ (links oben) so aussehen wie auf dem Screenshot auf Seite 14. Generiert man nun den C-Header, erhält man wieder einfache Arrays für die Filterkoeffizienten. Beim generieren des C-Headers wurde wieder genauso vorgegangen wie beim FIR-Filter (Seite 7).

Hier der wesentliche Ausschnitt des generierten Headers:

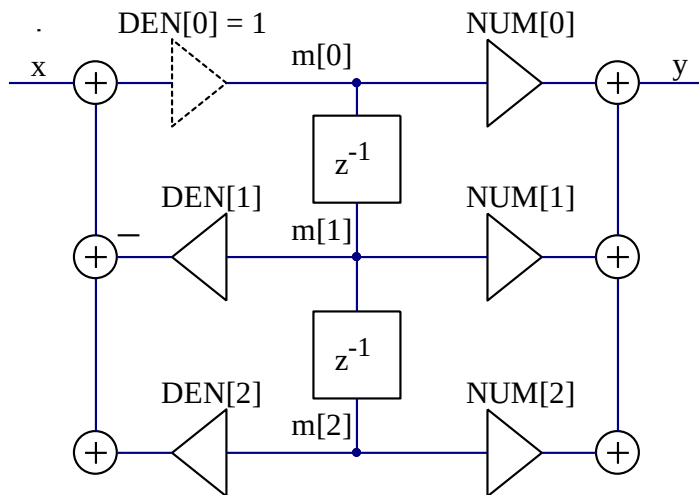
```
/* General type conversion for MATLAB generated C-code */
#include "tmwtypes.h"
/*
 * Expected path to tmwtypes.h
 * D:\Programme\MATLAB\R2015b\extern\include\tmwtypes.h
 */
const int NL = 3;
const real64_T NUM[3] = {
    0.5623828833246,    -1.124765766649,    0.5623828833246
};
const int DL = 3;
const real64_T DEN[3] = {
    1,    -0.9230886066229,    0.3264429266756
};
```

Es musste wieder das `include` entfernt und der Datentyp der Filterkoeffizienten auf `float` geändert werden:

```
/* General type conversion for MATLAB generated C-code */
// #include "tmwtypes.h"
/*
 * Expected path to tmwtypes.h
 * D:\Programme\MATLAB\R2015b\extern\include\tmwtypes.h
 */
const int NL = 3;
const float NUM[3] = {
    0.5623828833246,    -1.124765766649,    0.5623828833246
};
const int DL = 3;
const float DEN[3] = {
    1,    -0.9230886066229,    0.3264429266756
};
```

4.4 Filteraufbau

Hier der Aufbau unseres IIR-Filters. Die Bezeichnungen stimmen wieder mit den Variablennamen im Programm überein.



Die Werte DEN[0] bis DEN[2] und NUM[0] bis NUM[2] befinden sich in dem von Matlab generierten Header. Dieser muss zum Projekt hinzugefügt werden. Die Punkte m[0] bis m[2] sind die Zwischenspeicher, die im Programm benötigt werden.

4.5 Programmcode

```
#include "armv20_std.h"
#include "arm_math.h"
#include "Hochpass2_IIR.h"
//-----Speicheradressen der Audiowerte-----//
q15_t Audio_IN[2], Audio_OUT[2]; // Jeweils 2 Kanäle fuer IN und OUT

q15_t in_1;
q15_t out_1;

void process_data(void)
{
    float x; // Input als float
    static float m[3]={0}; // Die drei Zwischenspeicher muessen zu Beginn 0 sein
    float y = 0; // Output als float
    int i; // Fuer die Zaehlschleifen

    LED0=1; // fuer Zeitmessung

    in_1=Audio_IN[0]; // Audiowerte in eine hilfsvARIABLE speichern fuer filter-Upro
    arm_q15_to_float(&in_1,&x,1); // Input zu float konvertieren
```

```

m[0] = x;      // Den ersten Zwischenspeicher vor den 2 Subtraktionen auf
                den Eingangswert setzen (DEN[0]=1 - keine Verstärkung)
m[0] -= DEN[1]*m[1]; // Erste Subtraktion
m[0] -= DEN[2]*m[2]; // Zweite Subtraktion
for(i=0;i<3;i++)   // 0 - 2 fuer die 3 Additionen
{
    y += NUM[i]*m[i]; // Additionen fuer Ergebnis
}

m[2]=m[1]; //Zwischenspeicher aktualisieren
m[1]=m[0];
m[0]=x;

arm_float_to_q15(&y,&out_1,1); // Output wieder ins q15 Format

Audio_OUT[0] = out_1; //bearbeiteten Werte auf die DMA2 Adresse spielen

LED0=0; // LED fuer Zeitmessung zuruecksetzen
}

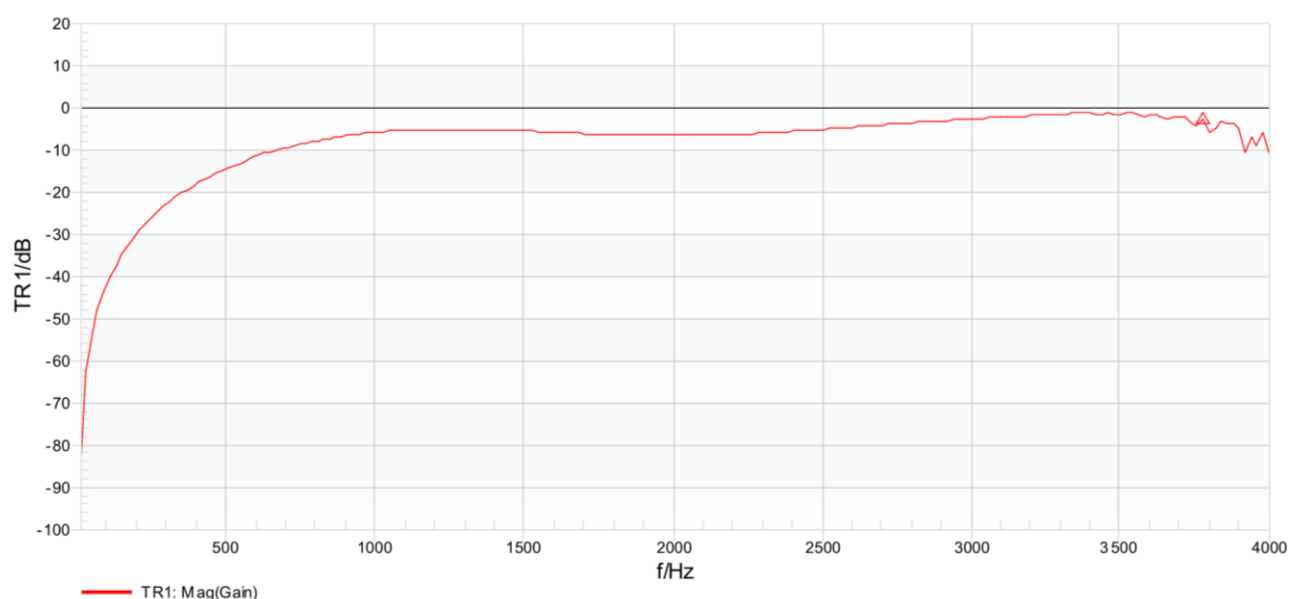
```

Die Rechnungen wurden wieder im `float` Format durchgeführt. Der inkludierte Header „Hochpass2_IIR.h“ ist der zuvor von Matlab generierte, der die Filterkoeffizienten enthält.

Das setzen der LED0 während des Interrupts wird erst bei der Zeitmessung benötigt. Es wurde nur ein Kanal des Systems genutzt.

4.6 Messung

Mit dem Bode100 wurde folgender Frequenzgang gemessen: (Einstellungen siehe Seite 9 – 3.6. Messung)



Der Hochpass ist klar zu erkennen und vergleicht man die Kurve mit dem Matlab generierten Diagramm, erkennt man, dass sie relativ gut übereinstimmt.

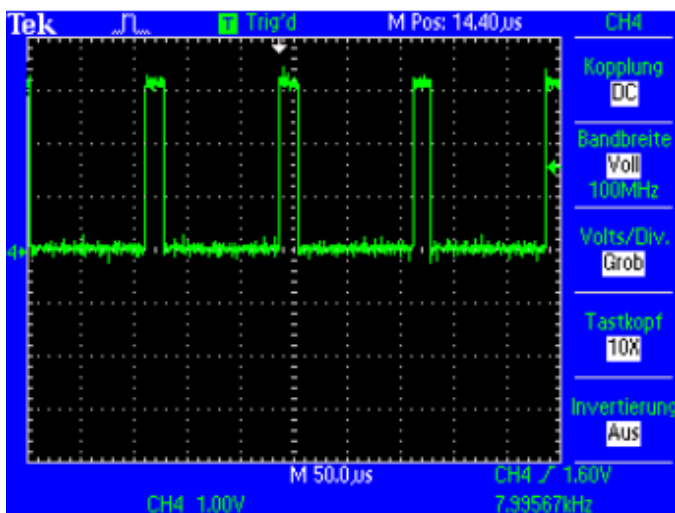
Es wurden auch wieder einige Messpunkte mit den Cursors aufgenommen und mit dem Matlab Design verglichen:

Frequenz	Amplitude gemessen	Amplitude Matlab
[Hz]	[dB]	[dB]
1020	-5,69	-3,01
1500	-5,47	-0,65
2000	-6,58	-0,14
3000	-2,58	0,00

Im Vergleich zum FIR Filter sind die Abweichungen der Werte etwas größer. Das liegt daran, dass sich die Fehler logischerweise mehr addieren. Die Abweichungen könnten zum Beispiel entstanden sein, da die Filterkoeffizienten von 32 auf 16 bit gekürzt wurden.

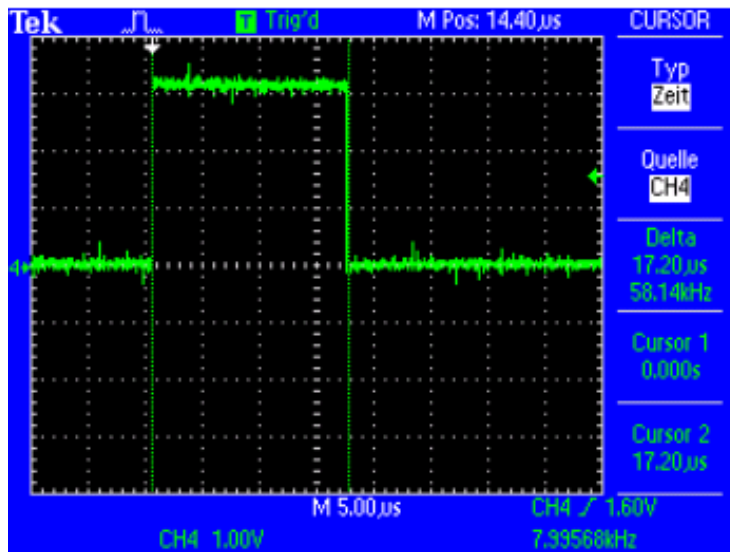
4.7 Zeitmessung

Wie beim FIR-Filter wurde auch beim IIR wieder eine Zeitmessung der ISR durchgeführt. Es wurde wieder die LED0 für die Dauer der ISR auf 1 gesetzt (siehe 4.5. Programmcode auf Seite 16). Es musste wieder darauf geachtet werden, dass das Setzen der LEDs im Hauptprogramm entfernt war. Theorie und Messschaltung siehe 3.7. Zeitmessung auf Seite 10-11.



Hier ein Bild, welches den Ablauf der Interrupts zeigt. Diese erfolgten wieder in Abständen von 125μs (8kHz).

Hier die Messung der Dauer des Interrupts:



Gemessen wurde eine Dauer von 17,2μs, welche jedoch wieder leicht schwankte.

$$Ausnutzung = \frac{t_{ISR}}{t_{ges}} = \frac{17,2\mu s}{125\mu s} = \sim 13,76\%$$

$$Reserve = 100\% - Ausnutzung = \sim 86,24\%$$

5 Q15 – Rechnungen

Die letzte Aufgabe wäre es gewesen, denselben IIR Filter zu realisieren und auszuwerten, wobei jedoch nicht im float sondern im Q15 Format gerechnet werden sollte. Dies war jedoch wegen Zeitmangel nicht mehr möglich.

6 Erkenntnisse

- Theorie: Digitale Filter und Vergleich zu analogen Filtern
- Filterdesign in Matlab
- Implementierung in C
- Digitale Signalverarbeitung wiederholt