POSIV BTE 5476

Signalverarbeitung mit dem STM32F407VGT6 (Cortex-M4F)

Kurzanleitung zur Installation der Entwicklungsumgebung, Aufbau des Basisprojektes und Beschreibung der Hardware zur Signalverarbeitung mit dem Discoveryboard STM32F4.

History: V1.0 14.10.2013, Ivo Oesch

1 Schnelleinstieg

1	Installation von Atollic True Studio (Entwicklungsumgebung)	• Atollic TrueSTUDIO for ARM Version: 4.1.0 Lite (Oder neuer) bei Atollic herunterladen (www.atollic.com/index.php/download/truestudio-for-arm) • Atollic TrueSTUDIO installieren (Während der Installation ist eine Registrierung bei Atollic notwendig) • Atollic TrueSTUDIO starten und einen Workspace erstellen/auswählen
2	Basisprojekt importieren	• Das Basisprojekt 'POSIV_ARM_V1.0.zip' von Moodle herunterladen. • Das Basisprojekt in TrueSTUDIO importieren (Über *File / Import -> General / Existing Projects into Workspace-> Next -> Select Archive File -> Browse* auswählen, mit *finish* importieren)
3	Basisprojekt testen	• Discovery-Board an PC/Laptop anschliessen (Via USB-Anschluss) • In TrueSTUDIO das Basisprojekt im Projekt Explorer auswählen und mit '*Project / Build Project*' compilieren • Im 'Project Explorer' mit der rechten Maustaste auf das generierte Binary 'POSIV_ARM_Basisprojekt.elf' klicken (Bei aufgeklapptem Projekt unter *Binaries* oder *Debug* zu finden) und '*Debug As / Embedded C/C++ Application'* auswählen. (Bei Bedarf mit OK bestätigen) • Programm mit '*Run / Resume*' oder grünem Pfeilknopf starten. (LED 3,4 und 5 sollten jetzt leuchten). • Programm mit rotem Stopp-Knopf oder '*Run / Terminate*' beenden • Wieder auf C/C++ Perspective wechseln (Tab *C/C++ Perspective* anklicken)
4	Eigenen Algorithmus implementieren	• In den Dateien 'SignalProcessing.....c' sind bereits grundlegende Algorithmen (FIR-Filter, FFT, FFT-Filterbank) als Beispiel implementiert, es darf jeweils nur eine dieser Dateien aktiv sein. (Unbenötigte entfernen, oder mit *Rechtsklick / Properties / C/C++ Build / Exclude Resource from Build* aktivieren oder deaktivieren. • Je nach Bedarf die Funktionen **`InitProcessing()`** und **`ProcessBlock()`** anpassen • Die Beispielimplementationen sind jeweils für 32-Bit Fliesskomma, Q31 und Q15 implementiert, einfach die passende Implementation auswählen und die restlichen löschen oder deaktivieren.
5	Informationen einholen	• Die Dokumentation der zur Verfügung gestellten DSV- und Mathfunktionen der CMSIS/DSP Library ist unter Moddle (DokumentationCMIS_DSV.zip) zu finden. • Informationen zum Discoveryboard sind unter www.st.com/stm32f4-discovery zu finden
6	Hinweise	• Signalpegel am Eingang dürfen +/- 1.5 V nicht überschreiten • Bei DC-Kopplung sind am Eingang nur 0-3V erlaubt • TrueSTUDIO Lite ist auf 32k Codegrösse limitiert • Analog in: PC1 und PC2, Analog out: PA4 und PA5.

2 Basisprojekt

Im Basisprojekt 'POSIV_ARM_Basisprojekt' werden die beiden A/D Wandler ADC1 und ADC 2 zum Einlesen der Samples, und die beiden D/A-Wandler DAC1 und DAC2 zum Ausgeben der Samples verwendet. Alle Wandler werden durch den Timer 2 getriggert, die Samplerate kann somit durch Timer2 festgelegt werden. Die Samples selbst werden per DMA zwischen den Wandlern und dem Speicher transferiert, wobei Double-Buffering verwendet wird. Das heisst, dass jeder DMA-Kanal zwei Buffer besitzt, einer wird für den aktuellen Transfer verwendet, der andere kann von Algorithmus bearbeitet werden. Sobald der aktuelle Buffer voll, resp. leer ist, wechselt der DMA-Kontroller automatisch zum anderen Buffer und löst einen Interrupt aus. Dies geschieht völlig autonom, ohne Benutzereingriff.

Im Verzeichnis lib finden sich die Bibliotheken, darunter auch die DSP-Bibliothek des CMSIS. Unter src findet sich der Code des Projektes, welcher aus folgenden Dateien besteht:

2.1 startup_stm32f40xx.s

Startup, initialisiert die C/C++ Runtime-Umgebung und die Interruptvektoren nach einem Reset, in dieser Datei üblicherweise keine Änderungen vornehmen.

2.2 stm32f4xx_conf.h

Konfigurationseinstellungen, steuert die Compilationsvorgänge von Bibliotheksfunktionen, in dieser Datei üblicherweise keine Änderungen vornehmen.

2.3 system_stm32f4xx.c

Initialisiert das Discoveryboard, in dieser Datei üblicherweise keine Änderungen vornehmen.

2.4 stm32f4xx_it.c / stm32f4xx_it.h

In dieser Datei sind die Interrupthandler abgelegt, für dieses Projekt relevant sind die DMA-Interupthandler **`DMA1_Stream5_IRQHandler()`**, **`DMA1_Stream6_IRQHandler()`**, **`DMA2_Stream0_IRQHandler()`** und **`DMA2_Stream2_IRQHandler()`**.
Diese Handler werden aufgerufen, sobald der entsprechende DMA-Kanal einen Datenblock transferiert hat. Prozessblock wird erst aufgerufen, wenn alle 4 Kanäle (2 Input und 2 Output) mit einem neuen Block von Samples bereit sind. Diese Synchronisation läuft über die vier Variablen `TransferCompleteADC1`, `TransferCompleteADC2`, `TransferCompleteDAC1` und `TransferCompleteDAC2`.
Zudem findet eine Konsistenzprüfung statt, um sicherzustellen, dass alle 4 Kanäle synchron zueinander laufen. Bei Synchronisationsverlust wird das System angehalten und die Leds zum Blinken gebracht. In dieser Datei sind normalerweise keine Änderungen vorzunehmen.

2.5 config.h

In dieser Datei können Einstellungen wie Samplerate und Blockgrösse (Anzahl Samples in einem Datenblock) definiert werden.

`NUMBER_OF_CHANNELS` legt die Kanalanzahl fest (Standardmässig 2)

`BLOCK_SIZE` legt die Blockgrösse fest (Anzahl gesammelter Samples, welche gemeinsam verarbeitet werden sollen. Je grösser die Blöcke, desto effizienter arbeitet das System und desto länger die Verzögerung des Systems (Zusätzliche Gruppenlaufzeit). Die Blockgrösse darf auch 1 sein.

`FS` definiert die Samplingfrequenz des Systems in Hz. Sampleraten bis 1MHz sind möglich, bis 100kHz problemlos (Settlingtime des DA-Wandlers ist 6us.). Die Samplerate muss ein ganzzahliger Teiler der Systemtaktfrequenz von 84Mhz sein. (Nicht passende Werte werden auf den nächst passenden gerundet).

2.6 DSPMain.c

Hier werden Ein-und Ausgänge, AD- und DA Wandler, der Timer für die Samplingfrequenz, die DMA Kon-

troller für den Daten- (Sample) transfer und die dazugehörigen Interrupts initialisiert. In dieser Datei sind normalerweise keine Änderungen vorzunehmen. Main verbleibt nach der Initialisierung in einer Endlosschlaufe (In welcher IdleFunction() aufgerufen wird). Die eigentliche Signalverarbeitung erfolgt in den Interruptroutinen.

2.7 SignalProcessing.h

Definiert die Schnittstelle der signalverarbeitenden Algorithmen. In dieser Datei sind normalerweise keine Änderungen vorzunehmen. Die Schnittstelle besteht aus den Funktionen InitProcessing(), ProcessBlock() und IdleFunction().

2.7.1 void InitProcessing(void)

Die Funktion InitProcessing() wird von main() vor dem Start des Systems aufgerufen, hier können lokale Initialisierungen für den jeweiligen Algorithmus vorgenommen werden.

2.7.2 void ProcessBlock(uint16_t *Channel1_in, uint16_t *Channel2_in, uint16_t *Channel1_out, uint16_t *Channel2_out)

Die Funktion ProcessBlock() wird jedesmal aufgerufen, wenn ein Block von Daten zur Verarbeitung zur Verfügung steht. Diese Funktion wird innerhalb des DMA-Interrupts aufgerufen, und muss beendet werden, bevor der nächste Block an Daten bereit ist, sonst gehen Daten verloren, oder das System wird asynchron.

Argumente:

Channel1_in: Zeiger (Array) auf die Samples von Input 1 (Blockgrösse in config.h definiert)
Channel2_in: Zeiger (Array) auf die Samples von Input 2 (Blockgrösse in config.h definiert)
Channel1_out: Zeiger (Array) auf die Samples für Output 1 (Blockgrösse in config.h definiert)
Channel2_out: Zeiger (Array) auf die Samples für Output 2 (Blockgrösse in config.h definiert)

Die Samples werden als vorzeichenlose 16-Bit-Zahlen im Bereich von 0 bis 65535 (2^{16}-1) geliefert und erwartet. Zur Konvertierung in/von vorzeichenbehafteten Werten (Bei Signalen im Bereich von -1.5V...1.5V) muss 32768 (2^{15}) subtrahiert/addiert werden.

2.7.3 void IdleFunction(void)

Die Funktion IdleFunction() wird von main() nach Initialisierung und Start des DMA/Interruptsystems endlos aufgerufen. Hier kann Code eingefügt werden, welcher parallel zur Signalverarbeitung laufen soll. (Jedoch tiefere Priorität als die Signalverarbeitung hat).

2.8 SignalProcessingFFT.c

Gibt das Spektrum des Eingangssignals als Zeitsignal aus.

Hier wird der FFT-Algorithmus aus der DSP-Library von ARM benutzt um das Eingangssignal in den Frequenzbereich zu transformieren. Anschliessend wird der Betrag des so gewonnenen Spektrums bestimmt, dieser exponentiell gemittelt und als Zeitsignal wieder ausgegeben. Dieses Beispiel zeigt die Verwendung der FFT Routine und der vektoriellen Betragsfunktion der DSP Library. Zudem wird die benötigte Sinustabelle (Twiddlefactors) manuell in InitProcessing() generiert. (Die Tabelle aus der Library kann nicht benutzt werden, weil sonst das Codelimit von 32KB überschritten würde).

Das Beispiel ist für die Formate float32 (32-Bit Fliesskomma), Q31 (32 Bit Fractional) und Q15 (16 Bit Fractional) implementiert, wobei jeweils nur ein Format ausgewählt sein darf.

Folgende Makros können angepasst werden:

`FFT_SIZE:` Definiert die FFT-Grösse, erlaubte Werte sind Zweierpotenzen von 16 bis 4096.

`MAKEFFT_FLOAT, MAKEFFT_Q31, MAKEFFT_Q15:` Nur eines dieser Makros darf definiert sein, damit wird das gewünschte Zahlenformat für den Algorithmus festgelegt.

2.9 SignalProcessingFFTFilterbank.c

Filtert das Eingangssignal im Frequenzbereich.

Hier werden der FFT und der IFFT-Algorithmus aus der DSP-Library von ARM benutzt, um eine Filterbank zu implementieren. Das Zeitsignal wird in den Frequenzbereich transformiert, gewichtet (Mit dem gewünschten Amplitudengang multipliziert) und wieder in den Zeitbereich zurück transformiert. Um Artefakte zu minimieren wird das Zeitsignal überlappend verarbeitet, d.h. sobald 25% der FFT-Grösse durch neue Samples ersetzt worden sind, wird der Algorithmus erneut ausgeführt, und mit dem Resultat des letzten Durchlaufs überlagert. (Siehe D. von Grünigen, Digitale Signalverarbeitung Bausteine, Systeme, Anwendungen, Fotorotar 2008, Kap. 6.3.5 DFT-Filterbank). Das gefilterte Signal wird anschliessend wieder ausgegeben..

Dieses Beispiel zeigt die Verwendung der FFT/IFFT Routinen der DSP Library. Zudem wird die benötigte Sinustabelle (Twiddlefactors) manuell in InitProcessing() generiert. (Die Tabelle aus der Library kann nicht benutzt werden, weil sonst das Codelimit von 32KB überschritten würde).

Das Beispiel ist für die Formate float32 (32-Bit Fliesskomma), Q31 (32 Bit Fractional) und Q15 (16 Bit Fractional) implementiert, wobei jeweils nur ein Format ausgewählt sein darf.

Das Makro **`FFT_SIZE`** wird automatisch auf 4***`BLOCK_SIZE`** definiert, da der Algorithmus fix mit 4-facher Überlappung arbeitet. **`BLOCK_SIZE`** muss eine Zweierpotenz in Bereich von 4 bis 1024 sein (**`BLOCK_SIZE`** wird in config.h festgelegt)

Folgende Makros können angepasst werden:

`MAKEFFT_FLOAT, MAKEFFT_Q31, MAKEFFT_Q15:` Nur eines dieser Makros darf definiert sein, damit wird das gewünschte Zahlenformat für den Algorithmus festgelegt.

2.10 SignalProcessingFFTIFFT.c

Transformiert das Signal vor der Ausgabe in den Frequenzbereich und direkt wieder zurück.

Hier werden der FFT und der IFFT-Algorithmus aus der DSP-Library von ARM benutzt, es wird einfach ein Sampleblock mittels FFT in den Frequenzbereich transformiert und gleich anschliessend wieder mittels IFFT in den Zeitbereich zurücktransformiert. Dieses Beispiel zeigt die Verwendung der FFT/IFFT Routinen der DSP Library. Zudem wird die benötigte Sinustabelle (Twiddlefactors) manuell in InitProcessing() generiert. (Die Tabelle aus der Library kann nicht benutzt werden, weil sonst das Codelimit von 32KB überschritten würde).

Das Beispiel ist für die Formate float32 (32-Bit Fliesskomma), Q31 (32 Bit Fractional) und Q15 (16 Bit Fractional) implementiert, wobei jeweils nur ein Format ausgewählt sein darf.

Das Makro **`FFT_SIZE`** wird automatisch auf **`BLOCK_SIZE`** definiert, da der Algorithmus fix einen Sampleblock verarbeitet. **`BLOCK_SIZE`** muss eine Zweierpotenz in Bereich von 16 bis 4096 sein (**`BLOCK_SIZE`** wird in config.h festgelegt)

Folgende Makros können angepasst werden:

`MAKEFFT_FLOAT, MAKEFFT_Q31, MAKEFFT_Q15:` Nur eines dieser Makros darf definiert sein, damit wird das gewünschte Zahlenformat für den Algorithmus festgelegt.

2.11 SignalProcessingFIRFilter.c

Filtert das Eingangssignal mit einem FIR-Filter.

Hier wird der FIR-Filteralgorithmus aus der DSP-Library von ARM benutzt. es wird einfach ein Kanal gefiltert und wieder ausgegeben, der andere Kanal wird direkt wieder ausgegeben.

Das Beispiel ist für die Formate float32 (32-Bit Fliesskomma), Q31 (32 Bit Fractional) und Q15 (16 Bit Fractional) implementiert, wobei jeweils nur ein Format ausgewählt sein darf.

Folgende Makros können angepasst werden:

`MAKEFIR_FLOAT, MAKEFIR_Q31, MAKEFIR_Q15:` Nur eines dieser Makros darf definiert sein, damit wird das gewünschte Zahlenformat für den Algorithmus festgelegt.

3 DSP-Library

Von ARM wird als Bestandteil der CMSIS-Library (http://www.arm.com/products/processors/cortex-m/cortex-microcontroller-software-interface-standard.php) eine Sammlung von DSV-Routinen mitgeliefert. Die HTML-Dokumentation dieser DSP-Funktionen kann von Moodle heruntergeladen werden, die ganze Library kann (Nach Registrierung) auch direkt bei ARM heruntergeladen werden, eine Liste der verfügbaren Funktionen ist im Anhang zu finden.

Im Basisprojekt ist die DSP-Library bereits eingebunden (Im Verzeichnis lib zu finden).

4 Discovery-Board STM32F4-Discovery

Das Discoveryboard besteht im wesentlichen aus dem Mikrocontroller STM32F407VGT6 (32-bit ARM Cortex-M4F core mit FPU, 1 MB Flash, 192 KB RAM) sowie etwas Peripherie und Stiftleisten, auf welche die Ports des Prozessors herausgeführt sind. Weitere Unterlagen können unter www.st.com/stm32f4-discovery heruntergeladen werden.

Im POSIV-Modul werden im wesentlichen die integrierten 12-Bit AD und DA-Wandler des STM32F407VGT6 benutzt. Weitere IO-Ports können nach Belieben eingesetzt werden.

4.1 Pinbelegung:

Eingänge:
PC1: ADC in 11 (Channel1_in)
PC2: ADC in 12 (Channel2_in)

Hinweis: Die Spannungen an diesen Pins müssen im Bereich von 0V-3V liegen, sonst wird der Prozessor beschädigt!

Ausgänge
PA4: DAC 1 out (Channel1_out)
PA5: DAC 2 out (Channel2_out)

Hinweis: Durch diese Konfiguration können Codec und Accelerometer nicht mehr benutzt werden (Doppelbelegung)

PD12: Grüne LED
PD13: Orange LED
PD14: Rote LED
PD15: Blaue LED

PB15: Takt für Antialiasingfilter auf Adapterboard, $F_g = F_{Takt}/100$

PE7: Auf Adapterboard geführt, für Zeitmessungen oder Zustandsanzeigen
PE9: Auf Adapterboard geführt, für Zeitmessungen oder Zustandsanzeigen
PE11: Auf Adapterboard geführt, für Zeitmessungen oder Zustandsanzeigen
PE13: Auf Adapterboard geführt, für Zeitmessungen oder Zustandsanzeigen

5 Adapterboard

Zum einfacheren Experimentieren wird ein Adapterboard eingesetzt. Dieses bietet im wesentlichen folgende Funktionalität:

Direkter Anschluss von Klinkensteckern, Pegelwandlung von +/-1.5V nach 0V-3V (Mittels RC-Hochpass, F_g bei etwa 1Hz) sowie Antialiasingfilter.

Pegelwandlung und Antialiasingfilter können bei Bedarf überbrückt werden, die Grenzfrequenz der Antialiasingfilter kann entweder auf fix 22kHz eingestellt, oder durch einen programmierbaren Takt durch Timer 1, 8 oder 12 festgelegt werden. ($F_{Takt} = 100 * F_g$)

6 Anhang

6.1 Konfiguration Adapterboard

Auf dem Adapterboard können folgende Einstellungen vorgenommen werden:

Wahl der **Eingangsvorverarbeitung**: Mit Jumper P1 für Channel1_in, mit Jumper P4 für Channel2_in:

Stellung DC: Eingang direkt mit Portpin des Prozessors verbunden, erlaubter Eingangspannungsbereich: 0V-3V, negative Spannungen sind verboten!

Stellung AC: Eingang wird über RC-Hochpass (Fg ca. 1Hz) geführt, erlaubter Eingangspannungsbereich: -1.5V-1.V, DC wird unterdrückt!

Stellung AAF: Eingang wird über RC-Hochpass (Fg ca. 1Hz) und Antialiasingfilter geführt, erlaubter Eingangspannungsbereich: -1.5V-1.V, DC wird unterdrückt!

Wahl der **Ausgangsnachbearbeitung**: Mit Jumper P6 für Channel1_out mit Jumper P9 für Channel2_out:

Stellung DC: Ausgang direkt mit Portpin des Prozessors verbunden, Ausgangspannungsbereich: 0V-3V, negative Spannungen sind nicht möglich!

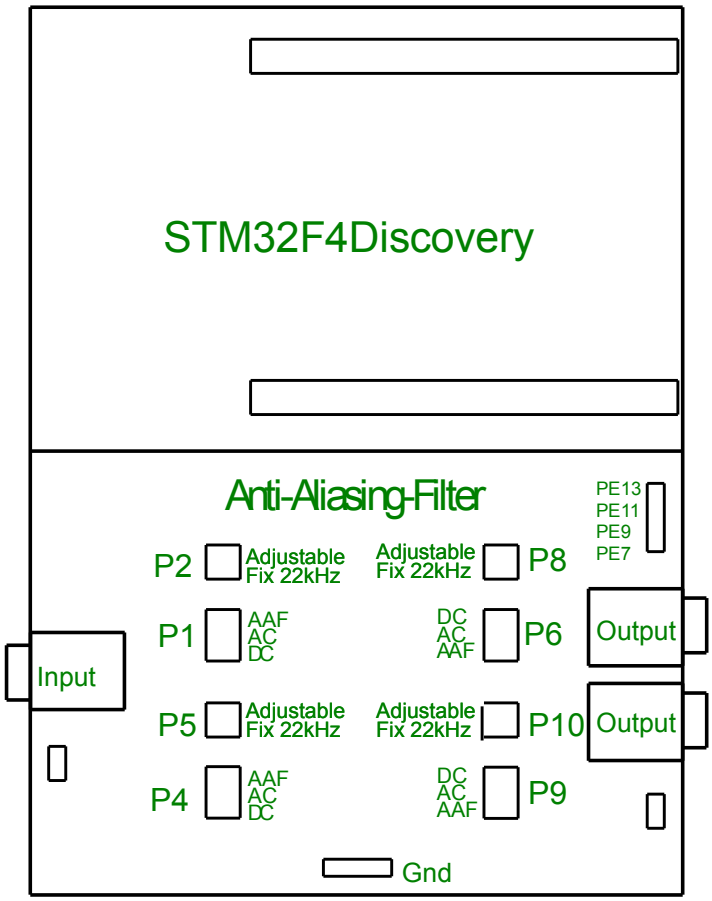
Stellung AC: Ausgang wird über RC-Hochpass (Fg ca. 1Hz) geführt, Ausgangspannungsbereich: -1.5V-1.V, DC wird unterdrückt!

Stellung AAF: Ausgang wird über RC-Hochpass (Fg ca. 1Hz) und Antialiasingfilter geführt, Ausgangspannungsbereich: -1.5V-1.V, DC wird unterdrückt!

Einstellung der **Grenzfrequenz** der Antialiasingfilter: Mit Jumper P2 für Channel1_in, P5 für Channel2_in, P8 für Channel1_out, P10 für Channel2_out :

Stellung Fix 22kHz: Grenzfrequenz auf 22kHz festgelegt (durch C3, C7, C12 und C16).

Stellung Adjustable: Grenzfrequenz kann mit Timer 1, Timer 8 oder Timer 12 festgelegt werden, die Timerfrequenz muss dabei das 100-fache der gewünschten Grenzfrequenz sein.



6.2 Fractional Zahlen

Bei Berechnungen im Fractionalformat muss auf die Wahl der richtigen Datentypen geachtet werden:

Q15-Multiplikation:

```
q31_t mul1
mul1 = (q31_t) x * (q31_t)x;  // 15Bit*15Bit => 30Bit
result =  (q15_t) __SSAT(mul1 >> 15, 16); // 30Bit >> 15 => 15Bit
```

Q31-Multiplikation:

```
q31_t out1
out1 = ((q63_t) inA1 * inB1) >> 32; // 31Bit * 31Bit => 62Bit
                                    // 31Bit * 31Bit >> 32 => 30Bit
out1 = __SSAT(out1, 31);
out1 = out1 << 1;  // 30Bit << 1 => 31Bit
```

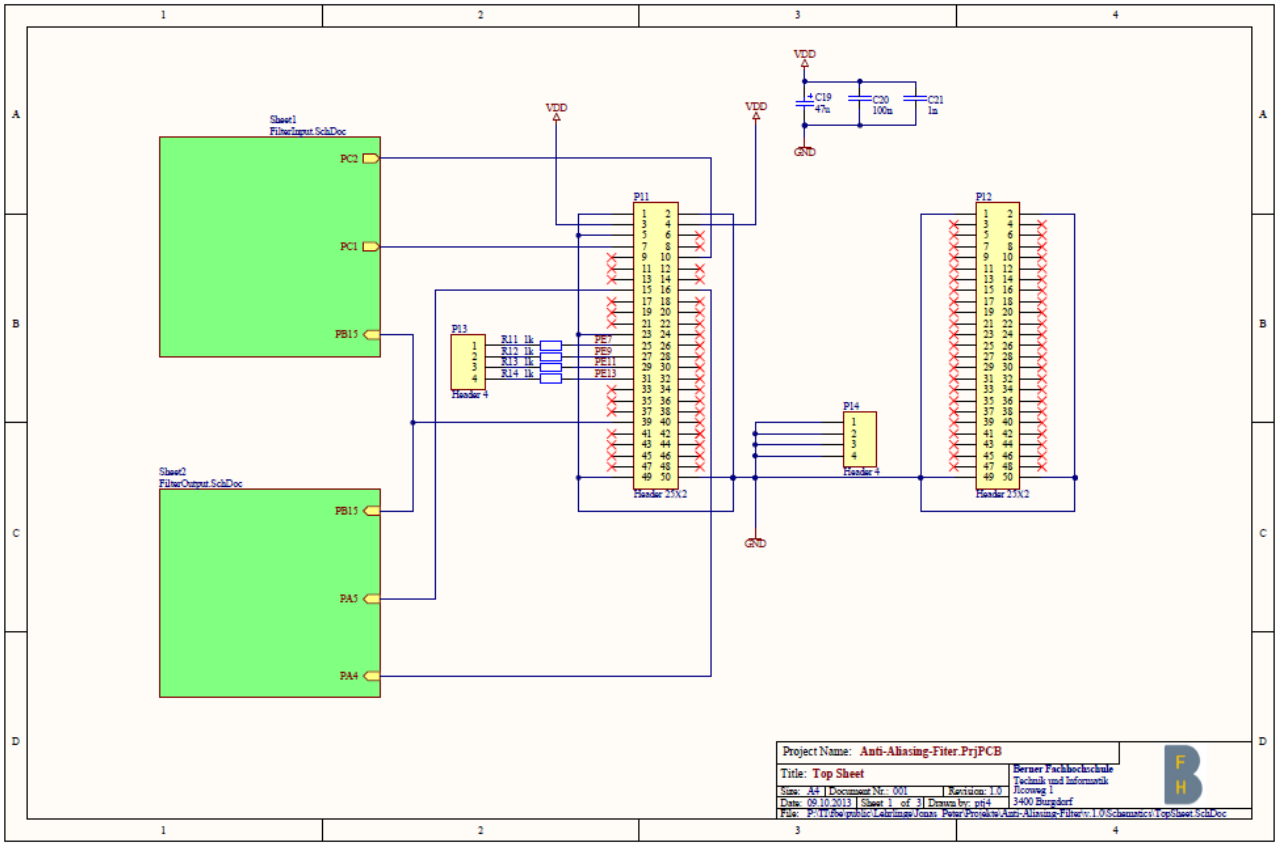
6.3 Technische Daten

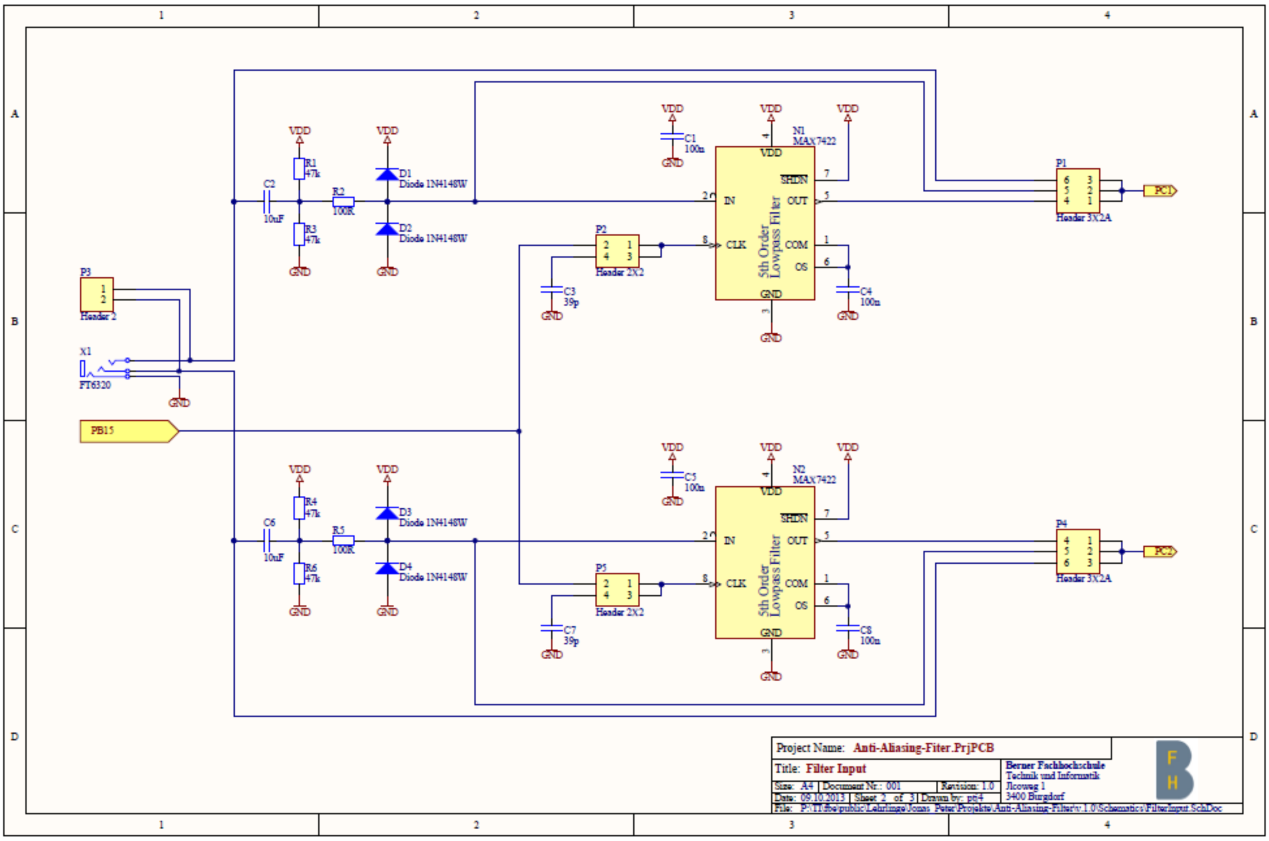
Eingangsspannungsbereich: 0V - 3V (0V ergibt 0, 3V ergibt 65535)

Ausgangsspannungsbereich: 0V - 3V (0 ergibt 0V, 65535 ergibt 3V)

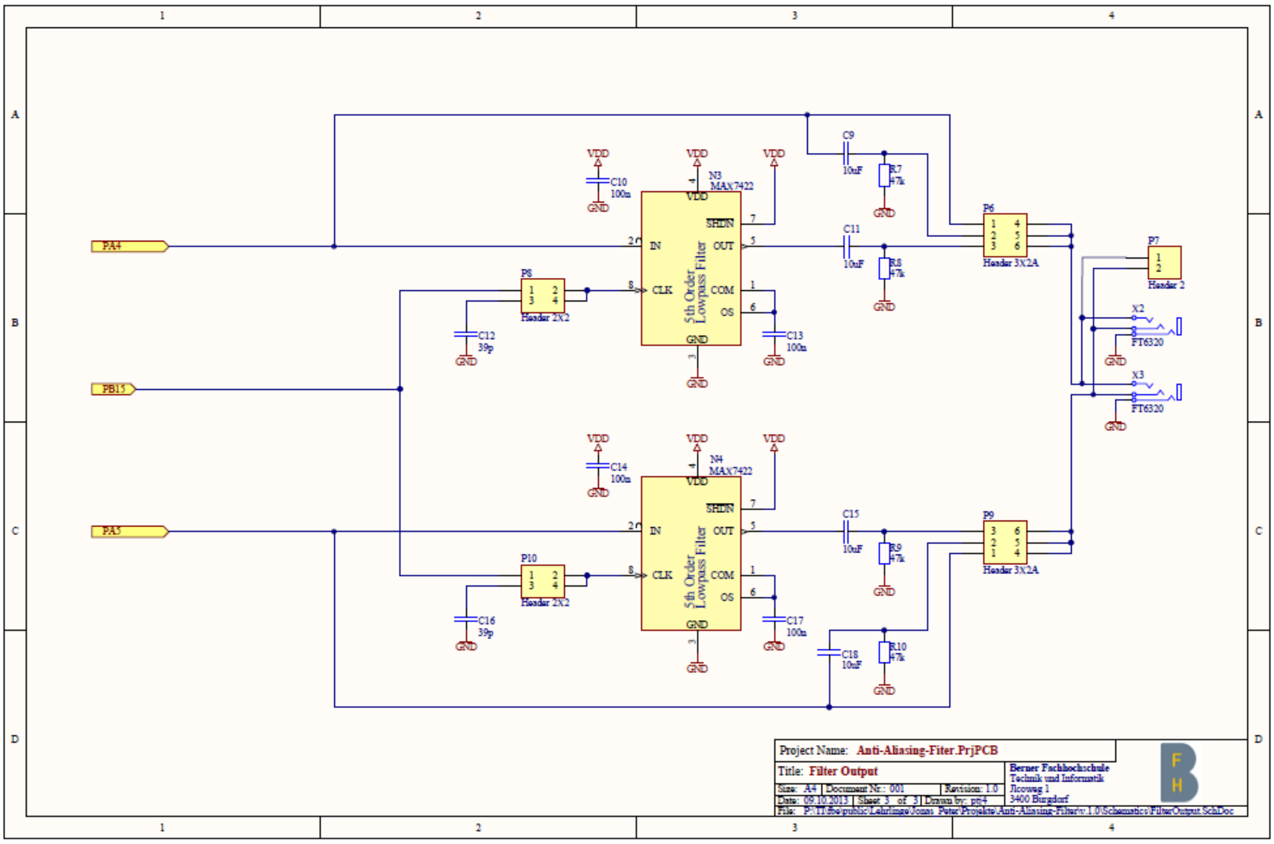
Samplingfrequenz:	Bis 100kHz problemlos, mit Einbussen bis 1MHz (Settling time DAC 6us) Systemtakt (84MHz) muss ganzzahlig durch Samplingfrequenz teilbar sein.
Systemtakt:	84MHz
Rechenleistung:	1024Punkte-FFT in float: 1.2ms, in Q31: 1.5ms, in Q15: 0.77ms
Antialiasingfilter:	Einstellbar von 1Hz bis 45kHz, Elliptisch, 5. Ordnung, 50dB Dämpfung

6.4 Schema Adapterprint





VDD
GND
R1 47k
R2 100K
R3 47k
C2 10uF
D1 Diode 1N4148W
D2 Diode 1N4148W
C1 100n
N1 MAX7422
5th Order Lowpass Filter
IN
OUT
CLK
COM
OS
SHDN
P1 Header 3X2A
PC1
P2 Header 2X2
C3 39p
C4 100n
P3 Header 2
X1 FT6320
PB15
R4 47k
R5 100K
R6 47k
C6 10uF
D3 Diode 1N4148W
D4 Diode 1N4148W
C5 100n
N2 MAX7422
P4 Header 3X2A
PC2
P5 Header 2X2
C7 39p
C8 100n
Project Name: Anti-Aliasing-Fiter.PrjPCB
Title: Filter Input
Size: A4
Document Nr.: 001
Revision: 1.0
Date: 09.10.2013
Sheet 2 of 3
Drawn by: ptj4
Berner Fachhochschule
Technik und Informatik
Jlcoweg 1
3400 Burgdorf
File: P:\TI\fbe\public\Lehrlinge\Jonas_Peter\Projekte\Anti-Aliasing-Filter\v.1.0\Schematics\FilterInput.SchDoc



PA4
PB15
PA5
VDD
GND
C10 100n
N3 MAX7422
5th Order Lowpass Filter
IN
OUT
CLK
COM
OS
SHDN
P8 Header 2X2
C12 39p
C13 100n
C9 10uF
R7 47k
C11 10uF
R8 47k
P6 Header 3X2A
P7 Header 2
X2 FT6320
X3 FT6320
C14 100n
N4 MAX7422
P10 Header 2X2
C16 39p
C17 100n
C15 10uF
R9 47k
P9 Header 3X2A
C18 10uF
R10 47k
Project Name: Anti-Aliasing-Fiter.PrjPCB
Title: Filter Output
Size: A4
Document Nr.: 001
Revision: 1.0
Date: 09.10.2013
Sheet 3 of 3
Drawn by: ptj4
Berner Fachhochschule
Technik und Informatik
Jlcoweg 1
3400 Burgdorf
File: P:\TI\fbe\public\Lehrlinge\Jonas_Peter\Projekte\Anti-Aliasing-Filter\v.1.0\Schematics\FilterOutput.SchDoc

6.5 Anpassung der DSP-Library

Da TrueSTUDIO Lite auf 32KB Codegrösse limitiert ist, können die Sinustabellen für die FFT nicht direkt aus der Library verwendet werden, sondern müssen zur Laufzeit generiert werden. Dazu müssen in der Datei Libraries/CMSIS/DSP_Lib/Source/CommonTables/arm_common_tables.c alle Tabellen entfernt werden (Am einfachsten Tabellen mit #if 0 / #endif einschliessen). Der Code zur Generierung der Tabellen ist in der Datei bei den Tabellen jeweils angegeben.

6.6 Literatur

Informationen zum Discoveryboard und links zu den Prozessordokumentationen:

www.st.com/stm32f4-discovery

DSP-Library (CMSIS)

http://www.arm.com/products/processors/cortex-m/cortex-microcontroller-software-interface-standard.php

6.7 Übersicht der DSP-Funktionen aus der CMSIS-Library

- Basic Math Functions
  - Vector Absolute Value
  - Vector Addition
  - Vector Dot Product
  - Vector Multiplication
  - Vector Negate
  - Vector Offset
  - Vector Scale
  - Vector Shift
  - Vector Subtraction
- Fast Math Functions
  - Cosine
  - Sine
  - Square Root
- Complex Math Functions
  - Complex Conjugate
  - Complex Dot Product
  - Complex Magnitude
  - Complex Magnitude Squared
  - Complex-by-Complex Multiplication
  - Complex-by-Real Multiplication
- Filtering Functions
  - High Precision Q31 Biquad Cascade Filter
  - Biquad Cascade IIR Filters Using Direct Form I Structure
  - Biquad Cascade IIR Filters Using a Direct Form II Transposed Structure
  - Convolution
  - Partial Convolution
  - Correlation
  - Finite Impulse Response (FIR) Decimator
  - Finite Impulse Response (FIR) Filters
  - Finite Impulse Response (FIR) Lattice Filters
  - Finite Impulse Response (FIR) Sparse Filters
  - Infinite Impulse Response (IIR) Lattice Filters
  - Least Mean Square (LMS) Filters
  - Normalized LMS Filters
  - Finite Impulse Response (FIR) Interpolator
- Matrix Functions
  - Matrix Addition
  - Matrix Initialization
  - Matrix Inverse
  - Matrix Multiplication
  - Matrix Scale
  - Matrix Subtraction
  - Matrix Transpose
- Transform Functions
  - Complex FFT Functions
  - Radix-8 Complex FFT Functions
  - DCT Type IV Functions
  - Real FFT Functions
  - Complex FFT Tables
  - RealFFT
- Controller Functions
  - Sine Cosine
  - PID Motor Control
  - Vector Clarke Transform
  - Vector Inverse Clarke Transform
  - Vector Park Transform
  - Vector Inverse Park transform
- Statistics Functions
  - Maximum
  - Mean
  - Minimum
  - Power
  - Root mean square (RMS)
  - Standard deviation
  - Variance
- Support Functions
  - Vector Copy
  - Vector Fill
  - Convert 32-bit floating point value
  - Convert 16-bit Integer value
  - Convert 32-bit Integer value
  - Convert 8-bit Integer value
- Interpolation Functions
  - Linear Interpolation
  - Bilinear Interpolation