Digitale Audiosignalverarbeitung auf dem Raspberry Pi

David Bau[1], Dieter Leckschat[2], Christian Epe[2]

[1] *Hochschule Düsseldorf, 40476 Düsseldorf, Deutschland, Email: david_bau@gmx.de*

[2] *Hochschule Düsseldorf, 40476 Düsseldorf, Deutschland*

Einleitung

Der in den letzten Jahren sich immer größerer Beliebtheit erfreuende Mini-Computer *Raspberry Pi* stellt eine verhältnismäßig große Menge Rechenleistung für wenig Geld und auf sehr kleinem Raum zur Verfügung. Diese drei Faktoren (Preis, Leistung, Größe) machen ihn auch interessant für die Verwendung als DSP-Alternative. Leider gibt es bis dato kaum verwendbare Ansätze dafür. Grund dafür mag sein, dass der Raspberry Pi nicht für einen solchen Zweck ausgelegt ist. Es muss eine zufriedenstellende Lösung für eine AD/DA-Wandlung gefunden werden, sowie eine Echtzeit-Audioverarbeitung auf dem Linux-Betriebssystem eingerichtet werden, mit der man DSP-Algorithmen implementieren kann.

Im folgenden soll ein Weg aufgezeigt werden, mit dem auf möglichst einfache Weise eine Umgebung erstellt werden kann, in der DSP-Routinen in C auf dem Raspberry Pi eingesetzt werden können.

Im ersten Abschnitt wird die Problematik zunächst konkretisiert. Danach werden in Abschnitt 2 zwei Möglichkeiten für eine adäquate AD/DA-Wandlung vorgestellt und in Abschnitt 3 eine passende Variante zur Einrichtung einer Lösung zur Signalverarbeitung auf dem Raspberry Pi selbst. In Abschnitt 4 wird ein kurzes Fazit und ein Ausblick für weitere Entwicklungen gegeben.

1. Problemstellung

Das eigentliche Kernproblem kann auf folgendes vereinfachtes Ziel heruntergebrochen werden: Es muss eine Möglichkeit geschaffen werden, analoge Audiosignale zu digitalisieren und dem System auf dem Raspberry Pi als Stream verfügbar zu machen, diesen Stream mit DSP-Routinen auf dem Raspberry Pi verarbeiten und anschließend wieder als analoges Audiosignal ausgeben zu können. Dies ist schematisch in Abb. 1 dargestellt.

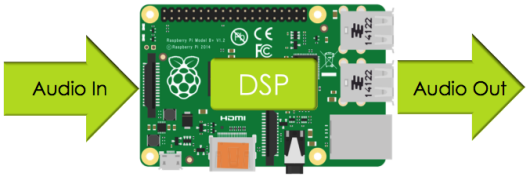

Abbildung 1: Grundkonzept für ein Effektgerät auf Basis des Raspberry Pi

Daraus können zwei essentielle Schritte abgeleitet werden:

Finden einer adäquaten Audio-Hardware

Die Kriterien hierfür sind die folgenden: Grundlegend sollte mindestens ein Stereo-Eingang und -Ausgang zur Verfügung stehen. Der Preis der Hardware sollte möglichst niedrig sein, da es sich um eine kostengünstige DSP-Lösung handeln soll. Die Qualität der Wandler sollte angemessen sein, das heißt je nach Anwendungsgebiet sollte hier mehr Wert gelegt werden. Zudem ist eine niedrige Latenz von Vorteil für viele DSP-Aufgaben.

Einrichten einer Signalverarbeitung

Es muss die Möglichkeit geschaffen werden, auf den Echtzeit Audiostream zugreifen zu können. Dabei soll es möglich sein, einzelne Audiosamples verarbeiten zu können, so dass eine Implementierung von DSP-Algorithmen, beispielsweise in C, möglich ist. Im Fokus steht hier, abgesehen von der Performanz, natürlich eine möglichst einfache Einrichtung und Anwendbarkeit.

2. Hardware

Da der Raspberry Pi lediglich einen PWM-Audioausgang besitzt, ist es erforderlich, die Audio Ein- und Ausgabe mittels zusätzlicher Hardware zu realisieren. Eine Möglichkeit stellt das *Behringer UCA-202* dar, ein preiswertes (∼30 €, Stand März 2016) USB-Audiointerface, welches über Stereo-Eingang und -Ausgang mittels RCA-Stecker verfügt. Für dieses Interface werden keine Treiber benötigt, es kann in der Regel direkt nach dem Anschluss verwendet werden. Die Qualität der Audiowandler ist trotz des niedrigen Preises auf einem guten Niveau.

Abbildung 2: Behringer UCA-202

Eine noch etwas bessere Qualität bietet die *Cirrus Logic Audio Card*. Diese ist unwesentlich teurer (∼40 €, Stand März 2016), bietet jedoch noch mehr Anschlussmöglichkeiten [1], darunter u.a. Stereo-Eingang und Ausgang über 3.5mm Klinke und Digital Ein- und Ausgang über S/PDIF. Die Einrichtung ist etwas

aufwändiger als beim *UCA-202*: Es müssen Hardwaretreiber für das Interface in den Kernel des Linux-Betriebssystems integriert werden. Dies kann durch manuelles Patchen erreicht werden, hierfür sind jedoch gute Linux Kenntnisse nötig. Alternativ bietet der Hersteller ein leicht modifiziertes Raspbian-Betriebssystem auf *Element14* [1], zum Download an, in dem die Treiber bereits integriert sind.

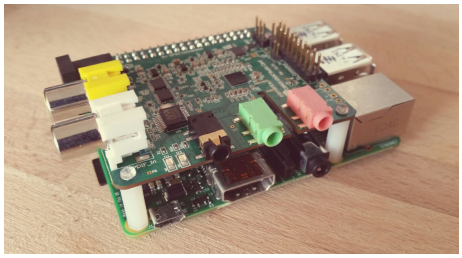

Abbildung 3: Raspberry Pi mit aufgestecketer Cirrus Logic Audio Card

Hardware-Latenz

Bei den Untersuchungen in [6], bei dem ein *Raspian* Betriebssystem auf einem Raspberry Pi 2 Model B installiert war, wurden Latenzmessungen durchgeführt. Dabei wurde die Latenz des Gesamtsystems zwischen Audio-Input und -Output gemessen, während Pure Data (siehe Abschn. 3) in einer Pass-Through Konfiguration ausgeführt wurde. Das *Behringer UCA-202* erzielte hierbei Werte von ∼14ms, die *Cirrus Logic Audio Card* Werte von unter ∼4ms. Bei ersterem könnte die relativ hohe Latenzzeit auf die USB-Übertragung zurückgeführt werden, während die zweite Lösung den Vorteil der systemnahen Einbindung haben dürfte.

3. Audio-Engine / Signalverarbeitung

Um Signalverarbeitung auf dem Raspberry Pi durchführen zu können, muss ein gewisses System eingerichtet werden, welches es ermöglicht, einen konstanten Audiostream (also eine Echtzeitverarbeitung von Audiosamples) zu verwalten und zur Bearbeitung mit DSP-Routinen zur Verfügung zu stellen.

An dieser Stelle wird eine möglichst einfache Variante vorgestellt, für die keine tiefgreifenden Kenntnisse im Umgang mit Linux nötig sind: Der Ansatz hierfür bietet das Programm *Pure Data*, welches auf dem Raspberry Pi installiert wird (Infos hierfür siehe [2]) und als Programm entweder mit oder auch ohne GUI ausgeführt werden kann. Pure Data ist eine sogenannte visuelle Programmiersprache, bei der DSP-Blöcke miteinander grafisch verknüpft werden können, um komplexere DSP-Strukturen zu erzeugen. Der große Vorteil von Pure Data ist, dass es die vollständige Verwaltung der Audio-Engine übernimmt. Das bedeutet, dass es mittels des ALSA-Audio-Treibers[3] die Kommunikation mit dem Audiointerface übernimmt und gleichzeitig den Audiostream zur Bearbeitung zur Verfügung stellt.

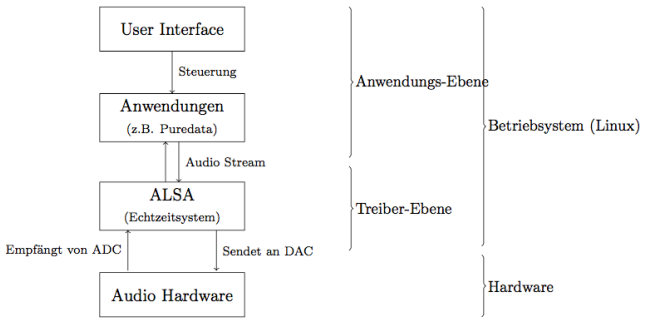

Abbildung 4: Das Audio-System auf dem Raspberry Pi (schematisch)

Eigene DSP-Objekte in Pure Data: *Externals*

Um nun eigene DSP-Routinen verwenden zu können, gibt es in Pure Data die nicht ganz einfach zugängliche, aber mächtige Möglichkeit, eigene DSP-Objekte in der Programmiersprache C zu schreiben. Diese werden *Externals* genannt, da sie nach dem Kompilieren und Einbinden in Pure Data wie normale Objekte verwendet werden können, jedoch eben nicht zu den internen Standart-Objekten gehören. Ein External besteht im Wesentlichen aus einer C-Datei, in der bestimmte von Pure Data vorgegebene Funktionen implemeniert werden müssen (Diese Funktionen sind in `„m_pd.h"` definiert, einer Header-Datei, die im Pure Data Dateisystem vorhanden ist und in das External über `import` eingefügt werden muss). Dazu gehören zum Beispiel Funktionen zum Initialisieren, zum Übergeben von Parametern oder eben eine *DSP-Funktion*, die periodisch von der Audio-Engine von Pure Data aufgerufen wird und der ein Block mit Samples zum Verarbeiten übergeben wird.

```
}

t_int *hsd_biquad_engine_perform(t_int *w)
{
    t_hsd_biquad_engine *x =       (t_hsd_biquad_engine *) (w[1]);  //th
    t_float *in =           (t_float *) (w[2]);                     //in
    t_float *out =          (t_float *) (w[3]);                     //ou
    t_int n =               w[4];                                   //bu

    //get the z-Elements from the data struct
    t_float z1 = x->z1;
    t_float z2 = x->z2;

    t_float u;

    while (n--) {

        // calculate the filter!
        u = *in++ - x->a1*z1 - x->a2*z2;        //Feedback-Path with a1
        *out++ = x->b0*u + x->b1*z1 + x->b2*z2; //Feedforward-Path with

        //shift the z-Elements
        z2 = z1;
        z1 = u;

    }

    //store the z-Elemtents back into the data struct
```

Abbildung 5: Exemplarischer Auszug aus dem C-Code eines Externals (Biquad-Filter): die DSP-Funktion

Man erhält beim Erstellen eines Externals also Zugriff auf die eigentliche Sample-Verarbeitung. Dies bedeutet, dass hiermit eine Möglichkeit gegeben wird, eigene Audio-Algorithmen in C zu erstellen, in Pure Data einzubinden und damit auf dem Raspberry Pi einzusetzen. Nähere Informationen zum Erstellen von Externals sind zu finden in [4] & [5].

4. Fazit & Ausblick

Es wurde gezeigt, dass es in einem bestimmten Umfang möglich ist, mit dem Raspberry Pi auf relativ einfache Weise herkömmliche DSP-Aufgaben zu übernehmen.

Darüber hinaus bietet der Raspberry Pi als DSP jedoch noch mehr Potential. Es ist möglich, herkömmliche DSP-Aufgaben mit den Fähigkeiten eines Einplatinencomputers sowie Pure Data zu kombinieren. Einerseits stellt Pure Data eine leicht zugängliche, schnelle und doch sehr umfangreiche Möglichkeit dar, komplexe DSP-Strukturen zu entwickeln und direkt auf dem Raspberry Pi einzusetzen. Zum anderen werden hierdurch vielseitige Erweiterungsmöglichkeiten geboten. Einige Beispiele wären:

- Anschluss eines Touch-Displays für ein Effektgerät mit X/Y-Pad
- Streamen von Audio über Netzwerk
- Senden und Empfangen von MIDI und OSC
- Ansteuern und Auslesen von Elektronikkomponenten wie LEDs, Sensoren oder Motoren für interaktive Musikinstrumente oder Installationen

Denkbar sind auch andere Alternativen zur Signalverarbeitung als Pure Data. Zum Beispiel beschäftigte sich ein Team der RWTH Aachen mit der Entwicklung einer DSP-Umgebung mithilfe von *JACK* auf einem *Arch Linux* Betriebsystem, bei dem DSP-Routinen in C++ ohne ein Drittprogramm wie Pure Data implementiert werden können[7].

Literatur

[1] Produktseite der *Cirrus Logic Audio Card* auf Element14, Stand März 2016
`https://www.element14.com/community/community/raspberry-pi/raspberry-pi-accessories/cirrus_logic_audio_card`

[2] Informationen und Installationshinweise für Pure Data auf dem Raspberry Pi
`https://puredata.info/docs/raspberry-pi`

[3] Informationen zu ALSA-Treibern
`http://www.alsa-project.org/`

[4] IOhannes m zmölnig: *HOWTO write an External for PureData*, 2014
`http://pdstatic.iem.at/externals-HOWTO/pd-externals-HOWTO.pdf`

[5] Eric Lyon: *Designing Audio Objects for Max/MSP and PD*
ISBN-13: 978-0895797155

[6] David Bau: *Erstellung einer Bibliothek für Audio-Algorithmen auf der Raspberry Pi Plattform*
Bachelorthesis, HS Düsseldorf

[7] Projektseite von Fabian Schlieper `http://dspi.audio/`