Lehrdemonstrationen zur digitalen Filterung auf eingebetteten Systemen

Eduard Funkner Martin Schwarz

2025-06-26

Inhaltsverzeichnis

Einführung **3**

Theoretische Grundlagen **4**
Infinite Impulse Response (IIR-Filter) . . . 4
Differenzengleichung eines IIR-Filters zweiter Ordnung . . . 5
Übertragungsfunktion eines IIR-Filters zweiter Ordnung . . . 5
Stabilität und Pol-Nullstellen-Verteilung . . . 6
Frequenzgang und Phasenverhalten . . . 6

Implementierungsstrukturen biquadratischer Filter **7**
Direktform 1 . . . 7
Direktform 2 (Kanonische Form) . . . 8
Transponierte Direktform 2: . . . 10

Numerische Aspekte und Quantisierungseffekte **12**
Koeffizientenquantisierung . . . 12
Bilineartransformation . . . 12

Implementierungserklärung der Biquad Filter in Arduino **13**
Filter Basisklasse . . . 13
Direktform 1 . . . 13
Private Member-Variablen . . . 13
Konstruktor . . . 14
Filter Methode . . . 14
Direktform 2 . . . 15
Private Member Variablen . . . 15
Konstruktor . . . 15
Filter Methode . . . 16
Transponierte Direktfrom 2 . . . 16
Private Member Variablen . . . 16
Konstruktor . . . 17
Filter Methode . . . 17
Eingabe Koeffiziente . . . 18
Koeffizienten Arrays und Filter Instanziierung . . . 18
Filter Anwendung . . . 18

Einführung

Diese Bachlorthesis dient als eine Lerndemonstration zur Implentierung von digitalen biquadratischen-Filtern auf Mikrocontrollern sowie FPGAs. Im Umfang dieser Lerndemonstration werden digitale Biquad-Filter mithilfe von Matlab- und Pythontools entworfen und im Anschluss auf Hard und Software umgesetzt. Der Prozess der Umsetzung wird zum Verständis dokumentiert sowie genau erklärt, damit Leser dieser Lerndemonstration mithilfe ihrer Hardware für sich den Prozess reproduzieren können.

Theoretische Grundlagen

Infinite Impulse Response (IIR-Filter)

Infinite Impulse Response (IIR) Filter gehören in der Signalverarbeitung zu den grundlegenden Typen der digitalen Filter. IIR-Filter zeichnen sich fundamental durch ihre characteristik aus, bei der Berechnung des aktuellen Ausgangswertes durch die Verwendung des gegenwärtigen und vergangenen Eingabewertes als auch des vorherigen Ausgabewertes. Aufgrund dieser Rückkopplung kann es dazu führen, dass die Impulsantwort von IIR-Filtern theoretisch unendlich lang andauern kann, was sie grundsätzlich von Finite Impulse Response (FIR) Filtern unterscheidet.

Die theoretischen Grundlagen von IIR-Filtern basieren auf der Tatsache, dass die Filter sowohl Nullstellen als auch Polstellen besitzen. Während FIR-Filter ausschließlich Nullstellen aufweisen, ermöglichen die Polstellen von IIR-Filtern eine effizientere Umsetzung bestimmter Filtereigenschaften. Dies führt zu dem Hauptvorteil von IIR-Filter für die Möglichkeit scharfe Übergänge im Frequenzgang mit relativ wenigen Koeffizienten zu erreichen.

Ein immenser Vorteil von IIR-Filtern liegt in der hohen Effizienz bei der Realisierung scharfter Frequenzgänge mit relativ wenigen Koeffizienten, während FIR-Filter für vergleichbare Filtereigenschaften oft hunderte von Koeffizienten benötigen, können IIR-Filter ähnliche Ergebnisse mit deutlich geringerem Rechenaufwand erreichen. Aufgrund dieser Eigenschaft eignen sich IIR-Filter besonders gut im Einsatz von Systemen mit beschränkten Ressourcen oder in Echtzeitverarbeitung.

Biquadratische Filter, kurz als “Biquad” bezeichnet, sind eine spezielle Unterklasse von IIR-Filtern zweiter Ordnung. Durch das Kaskadieren solcher Biquad-Sektionen besteht die möglichkeit komplexe IIR-Filter höhrer Ordnung auf einfache Weise zu realsieren. Das Kaskadieren dieser Sektionen bringt Vorteile in Bezug auf numerische Stabilität, Flexibilät bei der Parameteranpassung und Modularität des Designs.

Die Verwendung von Biquad-Sektionen anstelle von Filtern höherer Ordnung in direkter Form bietet eine Vielzahl von Vorteilen. Die Anfälligkeit für Koeffizientenquentisierung wird enorm gemindert, da die Pole und Nullstellen weniger stark von kleinen Änderungen der Koeffizienten abhängen. Im weiteren ermöglicht die modulare Struktur eine unbahängige Optimierung jeder Sektion, wodurch der Entwurfsprozess vereinfacht wird.

Differenzengleichung eines IIR-Filters zweiter Ordnung

Digitale Filter können mit einer linearen Differenzengleichung beschrieben werden. Die biquadratischen Filter werden mit der Differenzengleichung eines IIR-Filters zweiter Ordnung beschrieben werden:

$$y[n] = \frac{1}{a_0}(b_0 \cdot x[n] + b_1 \cdot x[n-1] + b_2 \cdot x[n-2] - a_1 \cdot y[n-1] - a_2 \cdot y[n-2])$$

Der Wert $y[n]$ bezeichnet den Ausgabewert zum Sample n, während $x[n]$ den Eingangswert darstellt. Der aktuelle Ausgabewert $y[n]$ ergibt sich aus der gewichteten Summe der aktuellen und zwei vergangenen Eingabewerte, subtrahiert mit den zwei gewichteten vergangen Ausgabewerte.

Die Koeffizienten b_0, b_1 und b_2 bestimmen den Einfluss des aktuellen und der vergangenen Eingabewerte (Feedforward), während a_1 und a_2 den Rückkopplungsanteil aus früheren Ausgabewerten (Feedback) modellieren. Im Allgemeinem wird der Koeffizient auf a_0 auf 1 normiert, was zu der vereinfachten Form führt:

$$y[n] = b_0 \cdot x[n] + b_1 \cdot x[n-1] + b_2 \cdot x[n-2] - a_1 \cdot y[n-1] - a_2 \cdot y[n-2]$$

Durch diese Normierung wird die praktische Implementierung vereinfacht, da ein Muliplikationsschritt entfällt.

Übertragungsfunktion eines IIR-Filters zweiter Ordnung

Zur Analyse des Frequenz- und Phasenverhaltens des Filters, wird die Differenzengleichung mittles der z-Transformation in den z-Bereich transformiert und daraus folgt die resultierende Übertragunsfunktion:

$$H(z) = \frac{Y(z)}{X(z)} = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}}$$

Die rationale Funktion eines biquadratischen Filters beschreibt das Verhältnis zwischen dem Ausgang $Y(z)$ zu dem Eingang $X(z)$. Diese Gleichung stellt die normierte Form des Biquads dar, bei welcher der Nennerkoeffizient auf $a_0 = 1$ normiert ist. Für den Fall, dass $a_0 \neq 1$ beträgt müssen die Koeffzienten durch a_0 geteilt werden um die Normalform zu erreichen.

Anhand der Übetragungfunktion können die Pol- und Nullstellen des Filters analyziert werden, um die Stabilität des Biquad-Filters zu vergewissern. Dabei müssen alle Polstellen des Filters innerhalb des Einheitskreises der z-Ebene liegen, beziehungsweise der Betrag jedes Pols kleiner als 1 sein muss.

Befinden sich Pole außerhalb oder direkt auf dem Einheitskreis, kann es dazu kommen dass das Ausgangssignal unkontrolliert schwingt und das System damit instabil wird. Nullstellen haben hingegen lediglichen Einfluss auf die Frequenzcharakteristik und nicht direkt auf die Stabilität.

Stabilität und Pol-Nullstellen-Verteilung

Die Stabilität eines IIR-Filters ist ein kritischer Faktor, wechler für die Implementierung des Filters gegeben sein muss. Ein kausaler IIR-Filter ist genau dann stabil, wenn alle Polstellen seiner Übertragungsfunktion innerhalb des Einheitskreises der komplexen z-Ebene liegen. Mathematisch ausgedrückt muss für alle Polstellen $|p_i| < 1$.

Die Lage der Pole bestimmt nicht nur die Stäbilität, sondern auch das dynamische Verhalten des Filters. Pole die näher am Rand des Einheitskreises liegen führen zu einem resonanten Verhalten mit langen ausschwingendem Verhalten, während Pole näher zum Ursprung hin, ein gedämpfteres Verhalten bewirken. Die Nullstellen hingegen beeinflussen primär dern Verlauf des Frequenzgangs.

Für die praktische Implementierung ist es wichtig zu beachten, dass Quantisierungseffekte sich auf die Lage der Pole auswirken können. Das kann im schlimmsten Fall dazu führen, sodass die Stabilität des Filters beeinträchtigt und der Filter instabil wird. Aus diesem Grund ist bei der Koeffizientenquantisierung besondere Vorsicht geboten, bei Filtern mit Polen nah oder auf dem Rand des Einheitskreises.

Frequenzgang und Phasenverhalten

Die Frequenzgang einew IIR-Filters wird durch die Auswertung der Übertragungsfunktion innerhalb des Einheitskreises erhalten, dafür wird die Substitution $z = e^{j\omega}$ druchgeführt:

$$H(e^{j\omega}) = \frac{b_0 + b_1 e^{-j\omega} + b_2 e^{-2j\omega}}{1 + a_1 e^{-j\omega} + a_2 e^{-j2\omega}}$$

Der Betrag $|H(e^{j\omega})|$ beschreibt die Amplitudencharakteristik, während die Phase $\arg(H(e^{j\omega}))$ das Phasenverhalten wiedergibt. Bei IIR-Filtern ist das Phasenverhlten im Allgemeinen nicht linear, was in manchen Anwendugen als unererwünscht erweist.

Implementierungsstrukturen biquadratischer Filter

Die praktische Implementierung von biquaratischen Filtern erfolgt durch die direkte Umsetzung der Differezengleichungen in Software oder Hardware.

Dafür werden die verzögerten Eingangswerte $x[n-1]$ und $x[n-2]$, sowie die Ausgangswerte $y[n-1]$ und $y[n-2]$ in Speicherelementen, sogenannten Verzögerungsgliedern, gespeichert. Der aktuelle Ausgangswert $y[n]$ wird mittels der gewichteten Summation aller Terme gemäß der jeweiligen Differenzengleichung berechnet.

Bei der numerischen Implementierung sind verschiedene Aspekte zu berücksichtigen, insbesondere die Wortbreite und des Zahlenformats (Festkomma oder Gleitkomma).

Weitere Einflüsse wie Rundungsfehler oder Quantisierungsrauschen können die Filtercharakteristik beeinflussen und im Extremfall die Stabilität des Filters gefährden.

Diese Implementierungsstrukturen bieten die Möglichkeit mehrere Filter-Sektionen aneinander zu Verbinden. Diese Modularität biquadratischer Filter ermöglicht, komplexere Filtersysteme höherer Ordnung, durch die Kaskadierung mehrerer Biquad-Sektionen zu realisieren.

Dadurch können Vorteile im Bezug auf numerische Stabilität, Designflexibiltät und Implementierungseffizienz geschaffen werden, weil jede Sektion unabhängig entworfen und optimiert werden kann.

Direktform 1

Die Direktform 1 stellt die direkteste Umsetzung der biquadratischen Differenzengleichung in einer Signalflussstruktur dar.

Diese Implementierung folgt direkt der mathematischen Beschreibung unter der Verwendung separater Verzögerungsketten für die Eingangs- als auch die Ausgangswerte.

Diese Struktur kann als eine Parallelschaltung eines rein rekusiven Teils (nur Pole) und eines nicht rekursiven Teils (nur Nullstellen) aufgefasst werden.

$$y[n] = b_0 \cdot x[n] + b_1 \cdot x[n-1] + b_2 \cdot x[n-2] - a_1 \cdot y[n-1] - a_2 \cdot y[n-2]$$

Für die Direktform 1 Implementierung werden zunächst alle gewichteten Eingangswerte und deren Verzögerungen zusammengefasst und berechnet:

$$w[n] = b_0 \cdot x[n] + b_1 \cdot x[n-1] + b_2 \cdot x[n-2]$$

Anschließend wird die Rückkoppling der verzögerten Ausgangswerte realisiert:

$$y[n] = w[n] - a_1 \cdot y[n-1] - a_2 \cdot y[n-2]$$

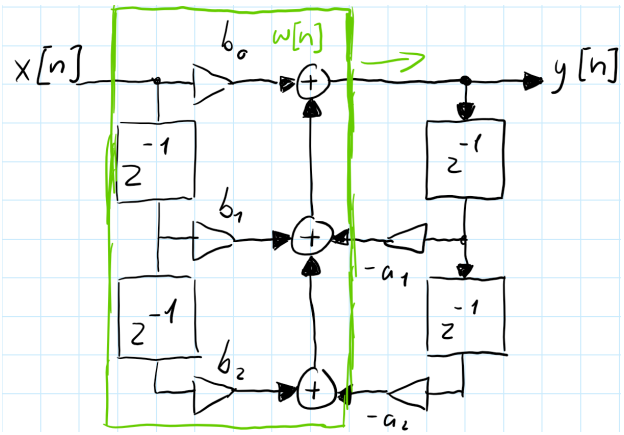

Abbildung 1: Signalflussbild: Direktform 1

In dieser Struktur werden vier Verzögerungselemente verwendet: zwei für die Eingangsverzögerung, $x[n-1]$ und $x[n-2]$, sowie zwei für die Ausgangsverzögerung, $y[n-1]$ und $y[n-2]$. Die Multiplikationen zwischen den Verzögerungselementen und den Koeffizienten erfolgen parallel, wodurch sich eine hohe Verarbeitungsgeschwindigkeit erzielen lässt.

Ein wesentlicher Vorteil der Direktform 1 liegt in ihrer Robustheit gegenüber Koeffizientenquantisierung. Da die Feedforward- und Feedback-Pfade getrennt sind, beeinflussen Quantisierungsfehler in einem Pfad nicht direkt den anderen. Dies führt zu einer besseren numerenrischen Stabilität, besonders bei der Verwendung von Festkomma-Arithmetik.

Die Direktform 1 zeichnet sich durch ihre konzeptionelle Einfachheit und die direkte Korrespondenz zur Differenzengleichung aus. Nachteilhaft bei dieser Struktur ist der erhöhte Speicherbedarf, aufgrund der redundanten Verzögerungselemente. Für Anwendungen mit strengen Speicherbeschränkungen kann dies eine limitierender Faktor sein.

Direktform 2 (Kanonische Form)

Die Direktform 2, oder auch als kanonische Form bekannt, ist eine optimierte Variante der Direktform 1, welche durch die Umordnung der Operationen die Anzahl der benötigten Verzögerungselemente minimiert.

Diese Struktur macht sich die Kommutativität (Vertauschbarkeit) linearer zeitinvarianter (LTI)-Systeme zunutze, indem sie die Reihenfolge von der Vorwärts- und Rückwärtsverarbeitung vertauscht.

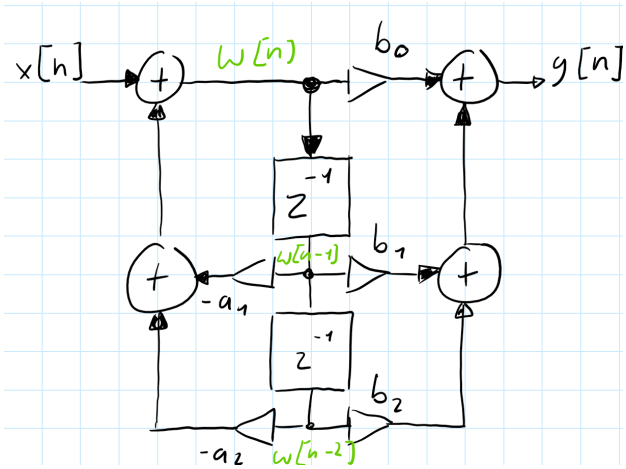

Abbildung 2: Signalflussbild: Direktform 2

Zunächst wird ein Zwischenwert $w[n]$ durch die rekursive Beziehungen erzeugt:

$$w[n] = x[n] - a_1 \cdot w[n-1] - a_2 \cdot w[n-2]$$

Der Zwischenwert $w[n]$ dient als Substituion für die Verzögerungselemente der Ein- und Ausgangswerte.

$$y[n] = b_0 \cdot w[n] + b_1 \cdot w[n-1] + b_2 \cdot w[n-2]$$

In dieser Struktur werden nur zwei Verzögerungselemente für w[n-1] und w[n-2] benötigt, da die rekursive als auch die nicht-rekursive Verarbeitung auf diesleben verzögerten Werte zugreifen. Die Bezeichnung “kanonisch” bezieht sich darauf, dass diese Implementierung die minimal mögliche Anzahl von Verzögerungselementen verwendet und dadurch die Speichereffizienz optimiert.

Die Direktform 2 eignet sich Ideal für Systeme mit begrenzten Speicherressourcen. Der reduzierte Speicherbedarf ist insbesondere bei der Implementierung auf Mikrocontrollern oder in FPGA-Anwendungen von hohem Vorteil, bei welchen Speicher eine limitierte und kostbare Ressource darstellt.

Jedoch weist diese Struktur eine erhöhte Anfälligkeit gegenüber Quantisierungseffekten auf, da die Zwischenwerte $w[n]$ größere Dynamikbereiche aufweisen können als die ursprünglichen Ein- und Ausgangswerte. Dies kann zu einer Verschlechterung des Signal-Rausch-Verhältnisses führen, insbesondere bei der Verwendung von Festkomma-Arithmetik mit begrenzter Wortbreite.

Transponierte Direktform 2:

Die transponierte Direktform 2 kann gebildet werden durch die Anwendung des Transpositionstheorems auf die Direktform 2. Dieses Theorem besagt, dass die Umkehrung aller Signalflussrichtungen bei gleichzeitiger Vertauschung von Eingängen und Ausgängen eine äquivalente Übertragungsfunktion ergibt. Die resultierende Struktur weist oft günstigere numerische Eigenschaften auf, als die ursprüngliche Form.

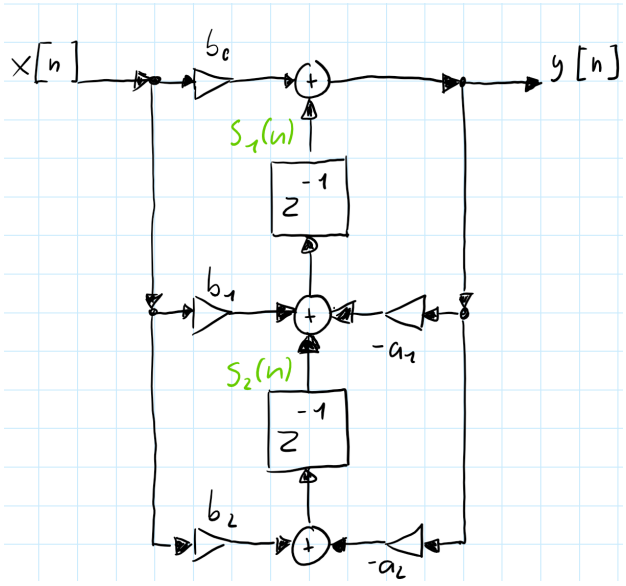

Abbildung 3: Signalflussbild: Transponierte Direktform 2

Die transponierte Direktform 2 wird druch die folgende Differenzengleichungen beschrieben:

$$s_2[n] = b_2 \cdot x[n] - a_2 \cdot y[n]$$

$$s_1[n] = s_2[n] + b_1 \cdot x[n] - a_1 \cdot y[n]$$

$$y[n] = s_1[n] + b_0 \cdot x[n]$$

Bei der transponierten Direktform 2 werden die Zwischenwerte $s_1[n]$ und $s_2[n]$ als Verzögerungselemente verwendet. Diese Struktur eignet sich besonders gut in der Implementierung von Echtzeitanwendung, aufgrund ihrer effizienten Berechnung und der guten Eignung für parallele oder pipelined Hardware-Architekturen.

Ein wesentlicher Vorteil der transponierten Direktfrom 2 liegt in ihrer reduzierten Anfälligkeit gegenüber Koeffizientenquantisierung. Die Rückkopplungsschleifen sind kürzer als in der

Direktform 2, was zu einer verbesserten numerischen Stabilität führt. Darüber hinaus ist die Struktur besonders gut für die Implementierung in Hardware geeignet, da die Berechnungen eine natürliche Pipeline-Struktur ausweisen.

Numerische Aspekte und Quantisierungseffekte

Die praktische Implementierung digitaler Filter auf Systemen mit endlicher Wortlänge birngt verschiedene Herausforderungen mit sich. Diese Effekte können die Filterleistung erheblich beeinträchtigen und müssen bereits in beim Entwerfen berücksichtigt werden. Die wichtigsten numerischen Aspekte sind Koeffizientenquentisierung, Rundungsrauschen, und nichtlineare Quantisierungseffekte.

Koeffizientenquantisierung

Die Koeffizientenquantisierung ist einer der kritischen Aspekte bei der Implementierung digitaler Filter auf Systemen mit begrenzter Wortlänge. Bei der Darstellung der Filterkoeffizienten mit einer endlichen Anzahl von Bits entstehen Quantisierungsfehler, die zu einer Verschiebung der Pole und Nulstellen des Filters führen kann.

Auswirkung der Koeffizientenquantisierung

Die Quantisierung der Koeffizienten führt zu verschiedenen unerwünschten Effekten:

Polverschiebung: Die quantisierten Koeffizienten verschieben die Pole des Filters, was zu einer Änderung der Filtercharakteristik führt. Im schlimmsten Fall können die Pole aus dem Einheitskreis herauswandern und zu einem instabilen Filter führen.

Frequenzgangverzerrung: Die Verschiebung der Pole und Nullstellen führt zur Abweichung im Frequenzgang. Besonders kritisch ist dies bei scharfen Filtern mit hoher Güte.

Stabilitätsverlust: Pole am oder auf dem Rand des Einheitskreises kann bereits durch kleine Quantisierung zur Instabilität zur Instabilität führen. Die Wahrscheinlichkeit der Instabilität steigt mit abnehmender Wortlänge.

Bilineartransformation

Mittels der Bilineartransformation ist es möglich einen bestehenden analogen Filter, anhand seiner Übertragungsfunktion zu digitalisieren.Die analogen Biquad Filter werden aus dem Expirement 4 des Analog Systems Lab Kit PRO entnommen.

Implementierungserklärung der Biquad Filter in Arduino

Für die Implementierung der biquadratischen Filter in Arduino werden die Differenzengleichungen aus den Theoretischen Grundlagen entnommen und in Code umgesetzt.

Filter Basisklasse

```
class Filter
  {
  public:
    virtual ~Filter() {}
    virtual float filter(float) = 0;
  };
```

Diese abstrakte Basisklasse dient als Interface für alle Filterimplementierungsstrukturen. Der virtuelle Destruktor `virtual ~Filter() {}` stellt sicher, dass beim Löschen eines Objekts über einen Basisklassenzeiger der korrekte Destruktor der abgeleiteten Klasse aufgerufen wird. Die rein virtelle Methode `virtual float filter(float) = 0;` definiert die einheitliche Schnittstelle - jede abgeleitete Filterklasse muss eine `filter()` - Methode implementieren, die einen float-Wert entgegennimmt und einen gefilterten float-Wert zurückgibt. Diese Struktur ermöglicht die Verwendung der verschiedenen Filtertypen im Anwendiungsfall einer Kaskadierung.

Direktform 1

Private Member-Variablen

```
private:
  const float b_0;
  const float b_1;
  const float b_2;
  const float a_1;
  const float a_2;
```

```
  float x_1 = 0;
  float y_1 = 0;
  float y_2 = 0;
```

Die Filterkoeffizienten `b_0, b_1, b_2, a_1, a_2` sind als `const` deklariert, da sie nach der Initialisierung unveränderlich bleiben sollen. Dies entspricht der mathematischen Definition eines zeitinvarianten Systems. Die Verzögerungselemente `x_1, y_1, y_2` speichern die vorherigen Ein- und Ausgangswerte. Der Wert `a_0` wird nicht gespeichert, da durch die Normalisierung im Konstruktor alle Koeffizienten bereits durch `a_0` geteilt wurden.

Konstruktor

```
BiquadFilterDF1(const float (&b)[3], const float (&a)[3], float gain)
    : b_0(gain * b[0] / a[0]),
      b_1(gain * b[1] / a[0]),
      b_2(gain * b[2] / a[0]),
      a_1(a[1] / a[0]),
      a_2(a[2] / a[0])
{
}
```

Der Konstruktor verwendet Initialisierungslisten für optimale Performance. Die Koeffizienten werden direkt bei der Objekterstellung normalisiert, indem alle durch `a[0]` geteilt werden. Dies entspricht der Standardform der Differenzengleichung, wo der führende Koeffizient des Nenners auf 1 normiert wird. Der `gain` Parameter wird in die Zählerkoeffizienten eingerechnet, was mathematisch äquivalent zur Multiplikation der gesamten Übertragungsfunktion mit dem Verstärkungsfaktor ist. Die Array-Referenz-Syntax `const float (&b)[3]` stellt sicher, dass exakt drei Elemente übergeben werden müssen.

Filter Methode

```
float filter(float x_0)
{
  float x_2 = x_1;
  x_1 = x_0;

  float y_0 = b_0 * x_0 + b_1 * x_1 + b_2 * x_2 - a_1 * y_1 - a_2 * y_2;

  y_2 = y_1;
```

```
  y_1 = y_0;

  return y_0;
}
```

Diese Implementierung wird durch die Struktur der DF1 umgesetzt. Zuerst werden die Eingangsverzögerungen aktualiseirt: `x_2` erhält den vorherigen Wert von `x_1`, dann wird `x_1` auf den aktuellen Eingangswert `x_0` gesetzt. Die Differenzengleichung wird direkt implementiert: `y[n] = b_0 * x[n] + b_1 * x[n-1] + b_2 * x[n-2] - a_1 * y[n-1] - a_2 * y[n-2]`.
Im Anschluss werden die Ausgangsverzögerungen für die nächste Iteration aktualisiert. Diese Reihenfolge ist kritisch, da die Berechnung vor Aktualisierung der Ausgangsverzögerungen erfolgen muss.

Direktform 2

Private Member Variablen

```
private:
  const float b_0;
  const float b_1;
  const float b_2;
  const float a_1;
  const float a_2;

  float w_0 = 0;
  float w_1 = 0;
```

Die DF2-Struktur benötigt nur zwei Verzögerungselemente `w_0, w_1` anstatt der vier in DF1. Dies reduziert den Speicherbedarf um die 50%. Die `w` Variablen repräsentieren die internen Knotenpunkte der DF2-Struktur, wo sowohl die Rückkopplung als auch die Vorwärtskopplung zusammenlaufen.

Konstruktor

```
BiquadFilterDF2(const float (&b)[3], const float (&a)[3], float gain)
    : b_0(gain * b[0] / a[0]),
      b_1(gain * b[1] / a[0]),
      b_2(gain * b[2] / a[0]),
      a_1(a[1] / a[0]),
```

```
      a_2(a[2] / a[0])
{
}
```

Der Konstruktor ist identisch zur DF1 Implementierung, da die Koeffizientennormalisierung unabhängig von der internen Filterstruktur ist. Die mathematische Übertragungsfunktion bleibt dieselbe, nur die interne Realisierung unterscheidet sich

Filter Methode

```
float filter(float x_0)
{
  float w_2 = w_1;
  w_1 = w_0;
  w_0 = x_0 - a_1 * w_1 - a_2 * w_2;

  float y_0 = b_0 * w_0 + b_1 * w_1 + b_2 * w_2;

  return y_0;
}
```

Die DF2 Implementierung teilt die Berechnung in zwei Phasen: Zuerst wird der interne Zustand `w_0` berechnet, der das Eingangssignal minus der Rückkopplung darstellt. Dies entspricht der Implementierung des Nennerpols der Übertragungsfunktion. Dann wird der Ausgang durch die Vorwärtskopplung mit den `b` Koeffizienten berechnet, was dem Zählerpolynom entspricht. Die Verzögerungselemente werden vor der `w_0` Berechnung aktualisiert, damit die korrekten vorherigen Werte verwendet werden.

Transponierte Direktfrom 2

Private Member Variablen

```
private:
  const float b_0;
  const float b_1;
  const float b_2;
  const float a_1;
  const float a_2;
```

```
  float s_1 = 0;
  float s_2 = 0;
```

Die TDF2 Struktur verwendet Zustandsvariablen `s_1, s_2`, die als "shift register" fungieren. Diese Struktur ist die transponierte Version der DF2, was bedeutet, dass der Signalfluss umgekehrt wird: die Ein- und Ausgänge werden vertauscht, sowohl als auch die Richtung der Verzögerungselemente wird umgekehrt.

Konstruktor

```
BiquadFilterTDF2(const float (&b)[3], const float (&a)[3], float gain)
    : b_0(gain * b[0] / a[0]),
      b_1(gain * b[1] / a[0]),
      b_2(gain * b[2] / a[0]),
      a_1(a[1] / a[0]),
      a_2(a[2] / a[0])
{
}
```

Auch hier ist der Konstruktor identisch zu den anderen Implementierungen, da die Koeffizientennormalisierung eine mathematische Anforderung ist, die unabhängig von der gewählten Realisierungsform gilt.

Filter Methode

```
float filter(float x_0)
{
  float y_0 = s_1 + b_0 * x_0;

  s_1 = s_2 + b_1 * x_0 - a_1 * y_0;
  s_2 = b_2 * x_0 - a_2 * y_0;

  return y_0;
}
```

Bei der TDF2 Implementierung setzt sich der Ausgangswert aus der addition aus dem ersten Zustandsregister `s_1` mit dem verstärkten Eingangswert. Die Zustandsregister werden dann für die nächste Iteration aktualisiert. `s_1` wird zum nächsten Wert von `s_2` addiert mit den gewichteten Ein- und Ausgangswerten. `s_2` wird komplett neu berechnet. Diese Struktur hat den Vorteil, dass der Ausgangswert sehr früh im Berechnungszyklus verfügbar ist, was bei Pipeline Implementierungen sich als vorteilhaft erweisen kann.

Eingabe Koeffiziente

```
const float b_0 = f;
const float b_1 = f;
const float b_2 = f;
const float a_0 = f;
const float a_1 = f;
const float a_2 = f;
```

Bei den Eingabe-Koeffezienten wird der `f` Suffix an die float Literalen angehängt, sodass sicher gestellt wird das die Werte als singel-precision floating-point interpretiert werden, was bei Arduino-System von hoher Wichtigkeit ist.

Koeffizienten Arrays und Filter Instanziierung

```
const float b_coefficients[] = { b_0, b_1, b_2};
const float a_coefficients[] = { a_0, a_1, a_2};
const float gain = 1;

BiquadFilterTDF2 biquad(b_coefficients, a_coefficients, gain);
```

Die Koeffizienten werden in separaten Arrays gespeichert und an den Konstruktor übergeben. Dies ermöglicht eine saubere Trennung zwischen Filterdesign (Koeffizienten) und Implementierung (Filterklasse). Über den `gain` Paramter kann auf den zu Implementierenden Filter eine gewünschte Verstärkung angewandt werden. Bei einem Wert von 1 wird keine zusätzliche Verstärkung angewendet. Bei dieser Filter Instanziierung wird die TDF2 verwendet mir ihrem korosponierenden Klassennamen. Somit wird das Objekt `biquad` durch die Klasse `BiquadFilterTDF2` instanziiert. Innerhalb des Objekts werden die Koeffizienten mit dem Verstärkungsfaktor an die jeweilige Filterklasse übergeben. Der Implementierte Filter steht bereit zur Verwendung.

Filter Anwendung

```
float filtered = biquad.filter(Input);
```

Die direkte Anwendung erfolgt durch den Aufruf der `filter()`-Methode auf dem Filterobjekt `biquad`. Der Eingangswert `Input` wird der Methode übergeben, durch die spezifische Filterimplementierung entsprechend der definierten Differenzengleichung verarbeitet und als transformierter Ausgangswert filtered zurückgegeben.