

KV Entwurf Integrierter Schaltungen

WiSe25

Digital Filter



Datum: 17. Februar 2026

Name

Gregor Flachs

Inhaltsverzeichnis

1 Einleitung	1
1.1 Allgemeines	1
1.1.1 Tapeouts	1
1.2 Projektbeschreibung	2
2 Theoretische Hintergründe	3
2.1 Finite Impulse Response - Filter	3
2.2 Berechnung der Koeffizienten	4
2.3 Dadda-Multiplizierer	4
3 Umsetzung	6
3.1 Halb-/Volladdierer	6
3.2 Multiplizierer	6
3.2.1 Optimierung der Koeffizienten	7
3.2.2 Effiziente Implementierung	7
3.3 Optionales 2er-Komplement	8
3.4 Spezielle Addierer	8
3.5 Gesamtsystem	9
4 Simulation	11
4.1 Halb-/Volladdierer	11
4.2 Multiplizierer	12
4.3 Optionales 2er-Komplement	13
4.4 Spezielle Addierer	13
4.5 Gesamtsystem	14
Abbildungsverzeichnis	V
Tabellenverzeichnis	VI
Quellcodeverzeichnis	VII

A Filtercharakteristika	VIII
A.1 Tiefpässe	VIII
A.1.1 Grenzfrequenz 0.125	VIII
A.1.2 Grenzfrequenz 0.375	IX
A.1.3 Grenzfrequenz 0.625	IX
A.1.4 Grenzfrequenz 0.875	X
A.2 Hochpässe	X
A.2.1 Grenzfrequenz 0.125	X
A.2.2 Grenzfrequenz 0.375	XI
A.2.3 Grenzfrequenz 0.625	XI
A.2.4 Grenzfrequenz 0.875	XII
A.3 Matlab-Skript zur Generierung	XII
B Bilder der Implementierungen	XVI
B.1 SkyWater Technologies 130nm PDK	XVI
B.2 GlobalFoundries 180nm PDK	XVII

Kapitel 1

Einleitung

1.1 Allgemeines

Alle Dateien des Projekts, inklusive der Simulationsdateien, sind in den zugehörigen GitHub-Repositories zu finden. Das Projekt wurde bei insgesamt zwei Tapeouts von Tiny Tapeout (<https://tinytapeout.com/>) und wafer.space (<https://wafer.space/>) eingereicht.

1.1.1 Tapeouts

Tiny Tapeout Sky 25b

- Projekt: <https://github.com/FlachsGregor/jku-sky-DigitalFilter>
- Fläche: $160 \mu\text{m} \times 100 \mu\text{m}$
- Platzierungsdichte: 60%
- Taktfrequenz: 50 MHz

wafer.space GF180MCU Run 1

- Projekt: https://github.com/iic-jku/gf180mcu-jku-projects/tree/main/macros/digital_filter
- Fläche: $200 \mu\text{m} \times 150 \mu\text{m}$
- Platzierungsdichte: 60%
- Taktfrequenz: 25 MHz

1.2 Projektbeschreibung

Das Projekt ist ein konfigurierbarer digitaler FIR-Filter dritter Ordnung für 8-Bit Daten. Es können sowohl die Grenzfrequenz als auch das Verhalten des Filters verändert werden. Die Abtastfrequenz für die Daten ist gleich der Taktfrequenz des Systems und das Ergebnis der Filterung ist nach einem Taktzyklus verfügbar. Die Ein- und Ausgangswerte werden vom Filter als nicht vorzeichenbehaftet interpretiert. Da die Ausgangswerte dieselbe Bitbreite wie die Eingangswerte besitzen, kann es zu leichten Verzerrungen des Ausgangssignals kommen. Die Ein- und Ausgänge des Systems sind in Tabelle 1 zu sehen.

Eine Konfiguration des Filters besitzt drei Bits (xyz). Das höchstwertigste Bit (x) bestimmt dabei das Verhalten, welches dem eines Tiefpasses ('0') oder dem eines Hochpasses ('1') entsprechen kann. Die anderen zwei Bits bestimmen die normierte Grenzfrequenz über die Beziehung $\omega_g = b0.yz1$. Somit ergeben sich vier mögliche Grenzfrequenzen, nämlich 0.125, 0.375, 0.625 und 0.875. Nach einem Rücksetzen ist der Filter als Tiefpass mit Grenzfrequenz $\omega_g = 0.125$ konfiguriert.

Signal	E/A	Breite	Beschreibung
CLK	E	1	Systemtakt
nRST	E	1	Asynchroner Reset (Low-aktiv)
enconfig	E	1	Neue Konfiguration übernehmen (High-aktiv)
configin	E	3	Konfigurationseingang
datain	E	8	Dateneingang
dataout	A	8	Datenausgang

Tabelle 1: Ein-/Ausgänge des digitalen Filters

Kapitel 2

Theoretische Hintergründe

2.1 Finite Impulse Response - Filter

Ein Finite Impulse Response (FIR) - Filter ist ein meist digitaler Filter, dessen Impulsantwort zeitlich beziehungsweise in der Anzahl an Werten begrenzt ist. Im Gegensatz zu diesen Filtern stehen die Infinite Impulse Response (IIR) - Filter, welche eine unendlich ausgedehnte Impulsantwort besitzen. Die Struktur eines FIR-Filters kann dabei auch interne Rückkopplungen beinhalten, wenn die dadurch entstehenden, nicht im Ursprung liegenden Polstellen exakt durch Nullstellen kompensiert werden. Im Folgenden wird jedoch angenommen, dass das Vorhandensein einer Rückkopplung synonym mit der unendlichen Ausdehnung der Impulsantwort ist, also ein FIR-Filter keine Rückkopplung besitzt, da dies auf die entwickelten Filter zutrifft. Mit dieser Annahme besitzt die Übertragungsfunktion eines FIR-Filters N -ter Ordnung im z -Bereich folgende allgemeine Form:

$$H(z) = b_0 + b_1 \cdot z^{-1} + \dots + b_N \cdot z^{-N} = \frac{b_0 \cdot z^N + b_1 \cdot z^{N-1} + \dots + b_N}{z^N}$$

Aus dieser Übertragungsfunktion kann man ablesen, dass ein beliebiger FIR-Filter nur Polstellen im Ursprung besitzen kann. Da eine Polstelle im Ursprung auch innerhalb des Einheitskreises liegt, ist die Stabilität eines FIR-Filters immer gegeben. Weiters kann man mit der Übertragungsfunktion die Differenzengleichung des Systems für ein allgemeines Eingangssignal bilden:

$$y[n] = b_0 \cdot x[n] + b_1 \cdot x[n-1] + \dots + b_N \cdot x[n-N]$$

Wenn man nun als Eingangssignal einen Dirac-Impuls $\delta[n]$ verwendet, erhält man als Ausgangssignal eine Folge von Koeffizienten der Übertragungsfunktion. Die Impulsantwort eines FIR-Filters beinhaltet also die Koeffizienten der zugehörigen Übertragungsfunktion.

2.2 Berechnung der Koeffizienten

Zur Berechnung der Koeffizienten eines FIR-Filters gibt es mehrere Möglichkeiten. Dazu gehören unter anderem die Fenstermethode, die Frequenzabtastmethode, die Methode des kleinsten mittleren quadratischen Fehlers und der Parks-McClellan-Algorithmus. Für dieses Projekt wurde allerdings eine andere, einfachere Methode gewählt. Bei dieser bildet man das gewünschte Frequenzspektrum und transformiert dieses mit der inversen diskreten Fouriertransformation in den Zeitbereich. Im Zeitbereich werden alle Teile der Impulsantwort auf 0 gesetzt, die nicht als Koeffizienten benötigt werden. Diese Methode kann nun iterativ fortgeführt werden, indem man die gekürzte Impulsantwort wieder mit der diskreten Fouriertransformation in den Frequenzbereich transformiert, das erhaltene Spektrum etwas anpasst und nach der Transformation zurück in den Zeitbereich wieder alle nicht benötigten Teile der Impulsantwort auf 0 setzt.

Um die Berechnung möglichst einfach zu halten, wurde die Methode ohne iterative Optimierung verwendet, also einfach als inverse diskrete Fouriertransformation des gewünschten Spektrums umgesetzt. Für die Transformation wurde das Spektrum mit 16 Werten definiert, da dies die kleinste mögliche Anzahl ist, die benötigt wird, um die gewünschten Grenzfrequenzen darstellen zu können. Die geringe Anzahl an Werten spielt insofern keine Rolle, da die Anzahl der verwendeten Werte kaum Einfluss auf die schlussendlichen Koeffizienten hat. Für die Quantisierung der Koeffizienten wurden 9 Bit und eine Darstellung mit dem 2er-Komplement gewählt, um für den Betrag der Koeffizienten eine Auflösung von 8 Bit zu erreichen. Eine weitere Eigenschaft dieser Methode ist, dass für den Tiefpass mit Grenzfrequenz ω_g der Hochpass mit Grenzfrequenz $1 - \omega_g$, zumindest mit den exakten Koeffizienten, ein Quadrature Mirror Filter ist, also, bei der Normierung auf $\frac{\omega_s}{2}$, das Spektrum des Hochpasses gespiegelt um $\frac{1}{2}$ identisch dem Spektrum des Tiefpasses ist.

2.3 Dadda-Multiplizierer

Ein Dadda-Multiplizierer ist ein spezieller, binärer Multiplizierer, der das Aufaddieren der partiellen Produkte optimiert. Denn im Gegensatz zum Wallace-Multiplizierer, bei dem die partiellen Produkte in jedem Schritt so weit wie möglich zusammengefasst werden, erfolgt die Reduktion beim Dadda-Multiplizierer nach bestimmten Regeln, die sowohl die benötigten Gatter als auch

die Laufzeit reduzieren. Die partiellen Produkte werden dabei meist nur soweit vereinfacht, dass nach der Reduktion zwei Binärzahlen vorliegen, die mit einem normalen Addierer aufaddiert werden können. Man kann das Verfahren aber auch auf die letzten zwei Binärzahlen anwenden und die gesamte Addition nach dem Dadda-Prinzip umsetzen.

In der Vereinfachung werden die Bits der partiellen Produkte als Punkte dargestellt, die übereinander liegen. Hierbei wird jede Schicht an Punkten als Stufe bezeichnet. Die Reduktion erfolgt nun in mehreren Schritten, in welchen die maximale Stufe immer weiter anhand der Folge d_j reduziert wird, wobei für die Folge d_j die Bedingungen $j \geq 1$, $d_1 = 2$ und $d_{j+1} = \lfloor 1.5 \cdot d_j \rfloor$ gelten und man sie theoretisch um das Folenglied $d_0 = 1$ erweitern könnte. Innerhalb eines Schrittes werden mit Halb- und Volladdierern alle Spalten, beginnend beim niederwertigsten Bit, so verringert, dass die Höhe der Spalte c_i die maximale Stufe d_j nicht überschreitet. Dabei werden folgende Regeln verwendet: Wenn die Höhe einer Spalte kleiner oder gleich der maximalen Stufe ist, also $c_i \leq d_j$, dann wird die Spalte nicht verändert. Ist die Höhe der Spalte genau um 1 größer als die maximale Stufe, also $c_i = d_j + 1$, dann werden zwei Bits der Spalte mit einem Halbaddierer aufaddiert und der Überlauf der nächsten Spalte zugeschrieben. Falls die Höhe der Spalte um mindestens 2 größer ist als die maximale Stufe, also $c_i \geq d_j + 2$, dann werden drei Bits der Spalte mit einem Volladdierer aufaddiert, der Überlauf der nächsten Spalte zugeschrieben und die Spalte muss noch einmal überprüft werden.

Nachdem alle Spalten reduziert wurden, wird im nächsten Schritt die maximale Stufe auf das nächstkleinere Folenglied d_{j-1} verringert und dasselbe Prinzip noch einmal angewandt. Ein Beispiel für die Reduktion einer 4-Bit \times 4-Bit Multiplikation auf 2 Stufen ist in Abbildung 1 zu sehen.

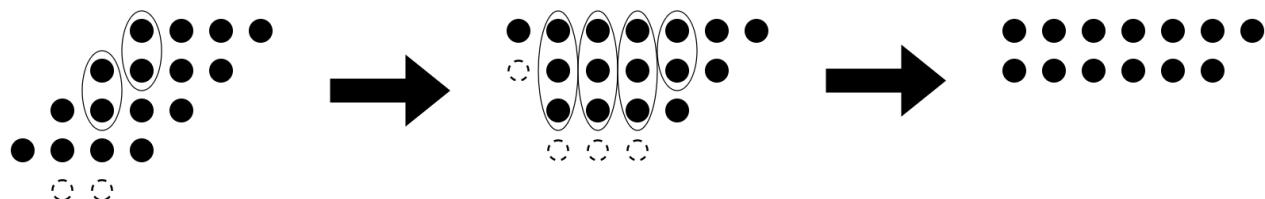


Abbildung 1: Reduktion einer 4-Bit \times 4-Bit Multiplikation

Kapitel 3

Umsetzung

3.1 Halb-/Volladdierer

Der Halbaddierer und der Volladdierer stellen die Basis der Multiplizierer und auch anderer Teile des Filters dar. Die Umsetzung erfolgt in ihrer einfachsten Form, nämlich anhand der Gleichungen, die in Tabelle 2 zu sehen sind.

$$\begin{array}{lll} \text{Halbaddierer:} & s = a \oplus b & c_{out} = a \cdot b \\ \text{Volladdierer:} & s = (a \oplus b) \oplus c_{in} & c_{out} = a \cdot b + (a \oplus b) \cdot c_{in} \end{array}$$

Tabelle 2: Gleichungen des Halb- bzw. Volladdierers

Der Grund für diese Art der Implementierung liegt in der stark beschränkten Fläche, welche für das Design zur Verfügung steht. Daher müssen, vor allem die Multiplizierer, im Bezug auf die Fläche stark verkleinert werden, was unter anderem durch das Verwenden dieser einfachen Halbaddierer beziehungsweise Volladdierer erreicht wird.

3.2 Multiplizierer

Ein Multiplizierer, der aus reiner Kombinatorik aufgebaut ist, kann nur unter größerem Flächenaufwand implementiert werden. Allerdings müssen die Multiplizierer des Filters nicht unbedingt generisch sein, da einer der beiden Faktoren immer ein Koeffizient und somit schon im Vorhinein bekannt ist. Wenn man nun Koeffizienten verwendet, welche eine gewisse Ähnlichkeit besitzen, kann man diese Ähnlichkeiten nutzen, um gewisse Teile der Multiplizierer einzusparen.

3.2.1 Optimierung der Koeffizienten

Bei der Optimierung der Koeffizienten wird prinzipiell auf zwei Methoden zurückgegriffen. Bei der ersten Methode werden die Bitmuster der Koeffizienten verglichen und einzelne Bits gesucht, die bei allen Koeffizienten, die in einem Multiplizierer verwendet werden, gleich sind. Wenn dies der Fall ist kann nämlich die Multiplikation mit diesem Bit weggelassen werden. Bei der zweiten Methode werden jene Koeffizienten, welche eigentlich negativ sind, bei der Multiplikation als positiv angenommen. Das hat zur Folge, dass in diesen Koeffizienten mehr Nullen vorkommen und auch eine höhere Ähnlichkeit zu den bereits positiven Koeffizienten besteht. Natürlich muss das Ergebnis einer solchen Multiplikation dann im Nachhinein noch invertiert werden, da ansonsten das Vorzeichen falsch wäre. Bei dieser Methode kann es jedoch unter Umständen zu einem Fehler von ± 1 LSB kommen, da eine Multiplikation und eine Invertierung im Binären nicht immer vertauschbar sind, wie folgendes Beispiel zeigt:

$$\begin{aligned} 01001 (9) &\xrightarrow{0.5} 00100 (4) \xrightarrow{\cdot(-1)} 11100 (-4) \\ 01001 (9) &\xrightarrow{\cdot(-1)} 10111 (-9) \xrightarrow{0.5} 11011 (-5) \end{aligned}$$

In Tabelle 3 sind die zusammengefassten Koeffizienten der Multiplizierer zu sehen, wobei ein Bit immer 0 (0), immer 1 (1) oder unbekannt (x), also vom ausgewählten Filter abhängig, sein kann.

Koeffizient 1: 0 xx10 0000
 Koeffizient 2: 0 0x0x 1x10

Koeffizient 3: 0 0001 1011
 Koeffizient 4: 0 000x xxxx

Tabelle 3: Zusammengefasste Koeffizienten der Multiplizierer

3.2.2 Effiziente Implementierung

Die Multiplizierer werden als Eintakt-Variante und nach dem Vorbild der schriftlichen Multiplikation implementiert. Es werden also zuerst durch Multiplikation mit den einzelnen Bits des Koeffizienten die partiellen Produkte berechnet, welche dann im Anschluss verschoben und aufaddiert werden. Durch die Zusammenfassung der Koeffizienten können deren Bits in drei Typen unterschieden werden, nämlich Null-Bits, welche immer 0 sind, Eins-Bits, welche immer 1 sind, und unbekannte Bits, welche vom ausgewählten Filter abhängig sind. Das ist vor allem bei der Bildung der partiellen Produkte hilfreich, da man sich so einige Multiplikationen sparen kann. Bei einem Null-Bit kann man zum Beispiel nicht nur die Multiplikation mit dem anderen

Wert weglassen, sondern auch das ganze partielle Produkt an sich, da es immer null sein wird und somit keinen Einfluss hat. Bei einem Eins-Bit kann die Multiplikation ebenfalls weggelassen werden, weil das Ergebnis immer der andere Wert ist, allerdings muss ein solches partielles Produkt in der späteren Aufsummierung sehr wohl berücksichtigt werden. Die unbekannten Bits können nicht wirklich optimiert werden, da man sowohl die Multiplikation als auch die partiellen Produkte selbst benötigt. Das Aufsummieren der partiellen Produkte ist durch das Wegfallen mancher Teilprodukte bereits erheblich vereinfacht, kann aber durch die Implementierung eines Dadda-Multiplizierers noch weiter optimiert werden.

3.3 Optionales 2er-Komplement

Das optionale 2er-Komplement ist, wie der Name schon sagt, ein Modul, welches von einer 8-Bit Zahl das 2er-Komplement bildet, wenn das zugehörige Einschaltsignal gesetzt ist. Die Umwandlung erfolgt nicht nach dem klassischen Prinzip, wo zuerst alle Bits der Zahl invertiert werden und dann +1 gerechnet wird, da die dafür nötigen Addierer unnötigen Platz benötigen würden. Stattdessen wird ein effizienteres Prinzip verwendet. Wenn das 2er-Komplement berechnet werden soll, werden alle niederwertigen Bits bis zur ersten 1 nicht verändert, und alle höherwertigen Bits werden invertiert. Umgesetzt wird dieses Prinzip mit einer Kette von speziellen, Halbaddierer-ähnlichen Blöcken, die entweder das 2er-Komplement oder das 1er-Komplement einer Zahl ausgeben. Das berechnete Komplement muss, im Falle eines 1er-Komplements, dann wieder auf die ursprüngliche Zahl zurück umgewandelt werden. Der Aufbau, dessen Ansätze in Abbildung 2 zu sehen sind, ähnelt dem eines Addierers, mit dem Unterschied, dass es nur eine Zahl als Eingang gibt. Ebenfalls zu erwähnen ist, dass das Modul den Ausgang um das Vorzeichenbit erweitert, also eine 9-Bit Zahl ausgibt, was auch bedeutet, dass nur positive Zahlen umgewandelt werden können. Das ist allerdings kein Problem, da das Modul nur für die Ausgangswerte des Multiplizierers vom dritten Koeffizienten benötigt wird und diese Werte sowieso immer positiv sind.

3.4 Spezielle Addierer

Um die Ergebnisse der Multiplizierer zusammenfassen zu können, werden weitere Addierer benötigt. Dabei kann in zwei Arten von Addierern unterschieden werden. Die erste Art ist ein Addierer mit zusätzlicher Subtrahierfunktion. Diese wird benötigt, da beim Zusammenfassen der Koeffizienten bei manchen Koeffizienten das Vorzeichen umgedreht wurde. Diese Änderung kann nun berücksichtigt werden, indem das positive Ergebnis eines Multiplizierers, welches eigentlich negativ sein sollte, nicht addiert sondern subtrahiert wird. Eine weitere Eigenschaft

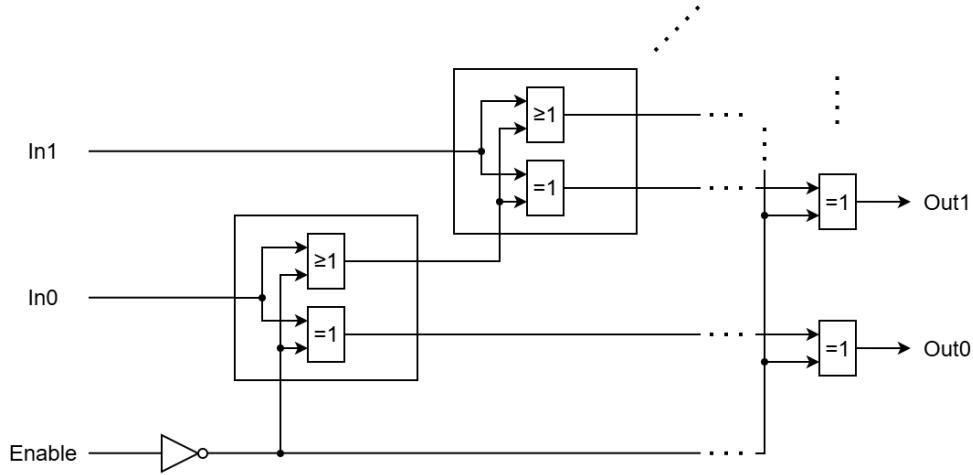


Abbildung 2: Blockschaltbild des optionalen 2er-Komplements

dieser Addierer ist, dass die Bitbreite des Ausgangs genauso groß ist wie die Bitbreite der Eingänge. Dies ist deshalb möglich, weil die Multiplizierer wegen den Koeffizienten nur beschränkte Ergebnisse liefern und somit bei der Addition keine Überläufe passieren können. Die zweite Art der Addierer beschränkt den Ausgang wieder auf 8-Bit. Das wird erreicht, indem Ergebnisse, die negativ sind, auf 0 aufgerundet werden, und Ergebnisse, die den Maximalwert von 8-Bit überschreiten, auf den Maximalwert abgerundet werden. Dadurch können zwar Verzerrungen am Ausgang entstehen, welche aber nicht so schlimm sind, da sie nicht sehr stark sind und nur vergleichsweise selten auftreten. Außerdem ist diese Beschränkung notwendig, da der Ausgang des Filters ebenfalls nur 8-Bit besitzen soll.

3.5 Gesamtsystem

Das gesamte System, dargestellt in Abbildung 3, besteht nicht nur aus den bereits beschriebenen Teilen, sondern auch aus einigen weiteren Komponenten. Ein Beispiel dafür sind die Synchronisationsketten, welche zum Synchronisieren aller Eingangssignale, bis auf den Takt und das Reset-Signal, benötigt werden. Außerdem gibt es für die Konfiguration des Filters ein zusätzliches Speicherregister, welches die aktuelle Konfiguration beinhaltet und über das zugehörige Eingangssignal aktiviert werden kann. Bei einem Filter werden für die Berechnungen auch vergangene Eingangswerte benötigt, weshalb es für die Daten eine Registerkette gibt, welche diese Werte zur Verfügung stellt. Für die Berücksichtigung der Konfiguration bei der Übergabe der Koeffizienten an die Multiplizierer und dem Setzen der Signale zum Invertieren der Ergebnisse der Multiplizierer wird noch etwas Kombinatorik benötigt. Die Bedingungen für die Koeffizienten und die Signale zum Invertieren, welche die Vorlage der Kombinatorik sind, sind in Tabelle 4

zu sehen. In der Tabelle bedeutet eine niedrige Grenzfrequenz, dass $\omega_g = 0.125$ oder 0.375 ist, eine hohe Grenzfrequenz, dass $\omega_g = 0.625$ oder 0.875 ist, eine extreme Grenzfrequenz, dass $\omega_g = 0.125$ oder 0.875 ist, und eine gemäßigte Grenzfrequenz, dass $\omega_g = 0.375$ oder 0.625 ist.

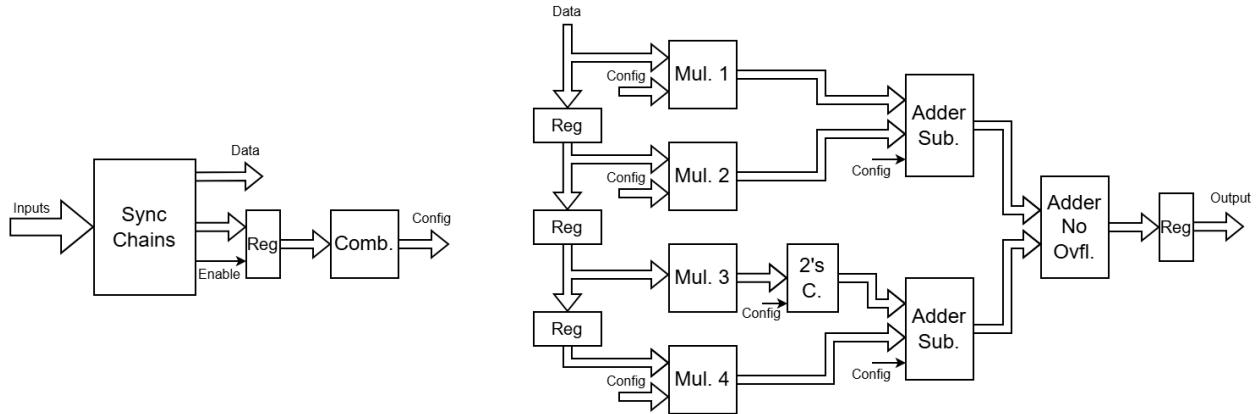


Abbildung 3: Blockschaltbild des Gesamtsystems

	Tiefpass	Hochpass
Koeffizient 1	xy für $\omega_g = 0.xy1_2$	\overline{xy} für $\omega_g = 0.xy1_2$
Koeffizient 2	011 (ω_g extrem) / 100 (ω_g gemäßigt)	
Koeffizient 4	10110 (ω_g extrem) / 01001 (ω_g gemäßigt)	
Subtrahierer 1	0	1
2er Komplement	1 (ω_g hoch) / 0 (ω_g niedrig)	0 (ω_g hoch) / 1 (ω_g niedrig)
Subtrahierer 2	0 (ω_g extrem) / 1 (ω_g gemäßigt)	1 (ω_g extrem) / 0 (ω_g gemäßigt)

Tabelle 4: Bedingungen für die Koeffizienten und Invertierungen

Kapitel 4

Simulation

4.1 Halb-/Volladdierer

Der Halbaddierer und der Volladdierer sind zwei relativ kleine Module und somit auch einfach zu testen. Da sie die Basis für einige weitere Module darstellen, ist es sinnvoll, die beiden Module vollständig zu testen, also alle möglichen Testfälle auf ihre Richtigkeit zu überprüfen. Dies ist aufgrund der wenigen möglichen Testfälle einfach umsetzbar. Ein Beispiel für einen Testfall des Volladdierers ist in Quellcode 1 und das zugehörige Waveform-Diagramm der Simulation in Abbildung 4 zu sehen.

```
initial begin
    ...
    a = 1'b1;
    b = 1'b0;
    ci = 1'b1;
    #5;
    ...
end
```

Quellcode 1: Testfall des Volladdierers

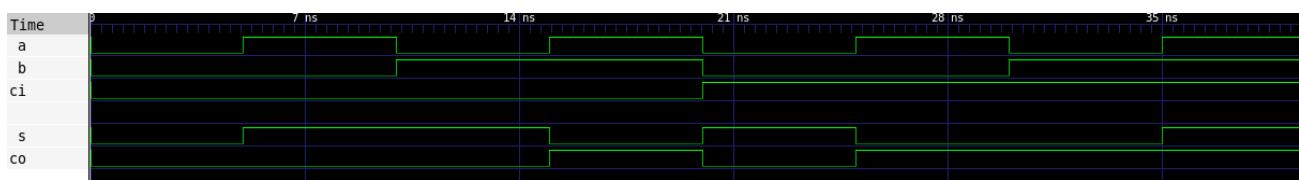


Abbildung 4: Waveform-Diagramm des Volladdierers

4.2 Multiplizierer

Die Multiplizierer sind Komponenten mit sehr vielen möglichen Eingangskombinationen, wodurch für die Testung einige durchzuführende Testfälle herausgesucht werden mussten. Dabei war es wichtig, jeden der möglichen Koeffizienten abzudecken, um eine Simulation zu erhalten, welche alle relevanten Fälle behandelt. Daher werden in der Simulation alle möglichen Koeffizienten durchgelaufen und mit einem bestimmten Datenwert multipliziert. Für den Datenwert wurde bei allen Koeffizienten der Maximalwert gewählt, da bei diesem alle Bits gesetzt sind und somit gleichzeitig überprüft werden kann, ob alle Bits des Dateneingangs bei der Multiplikation berücksichtigt werden. Um überprüfen zu können, ob das Ergebnis des Multiplizierers korrekt ist, wird in der Simulation die Multiplikation auch mit dem arithmetischen Operator `**` durchgeführt, was in einem Vergleichswert für das Ergebnis des Multiplizierers resultiert. Ein Beispiel, nämlich die Testung des ersten Multiplizierers, ist in Quellcode 2 und Abbildung 5 dargestellt.

```

assign c = {coef, 6'b100000};
assign mul = data * c;

initial begin
  ...
  data = 8'hff;

  for (i = 0; i <= 3; i = i + 1) begin
    coef = i;
    #5;
  end
  ...
end

```

Quellcode 2: Testung des ersten Multiplizierers

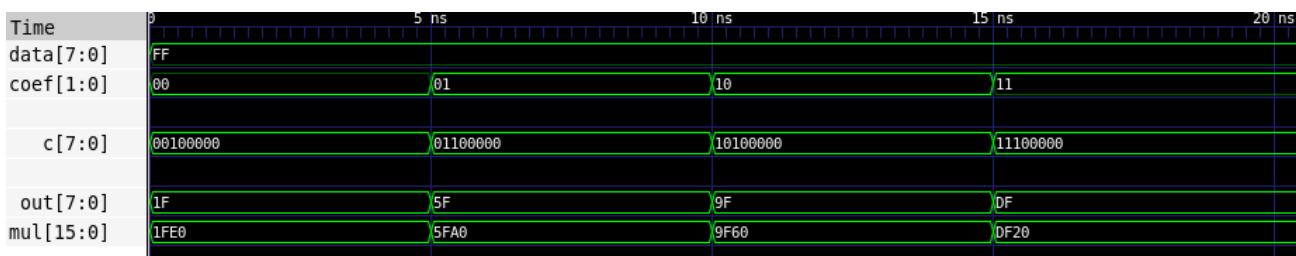


Abbildung 5: Waveform-Diagramm des ersten Multiplizierers

4.3 Optionales 2er-Komplement

Die Simulation des optionalen 2er-Komplements ist relativ simpel. Sie erfolgt mit mehreren zufälligen Werten, wobei auch die Grenzwerte des gültigen Eingangswertebereichs inkludiert sind. Diese Werte werden nacheinander an den Eingang des Moduls angelegt und gleichzeitig wird das Einschaltsignal bei jedem Wert einmal gesetzt und wieder rückgesetzt. Am Ausgang kann dann überprüft werden, ob die Werte korrekt weitergegeben beziehungsweise invertiert wurden. Ein Testfall und das Waveform-Diagramm sind in Quellcode 3 beziehungsweise Abbildung 6 zu sehen.

```
initial begin
    ...
    data = 8'd185;
    en = 1'b0;
    #5;
    en = 1'b1;
    #5;
    ...
end
```

Quellcode 3: Testfall des optionalen 2er-Komplements

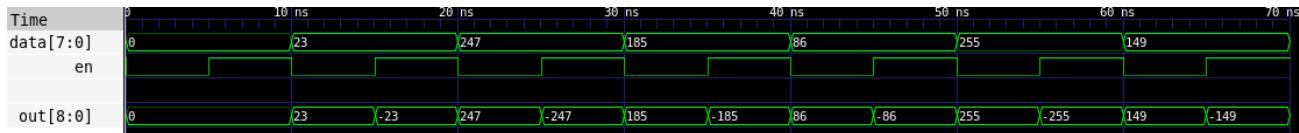


Abbildung 6: Waveform-Diagramm des optionalen 2er-Komplements

4.4 Spezielle Addierer

Die Simulation der Addierer ist ähnlich der Simulation des optionalen 2er-Komplements. Es wird mit mehreren zufälligen Zahlen und den Wertepaaren, welche die Grenzfälle repräsentieren, gearbeitet. Alle Wertepaare werden nacheinander an den Eingang der Module angelegt. Beim Addierer mit Subtrahierfunktion wird noch zusätzlich das Signal zum Subtrahieren während jedem Wertepaar einmal gesetzt und wieder rückgesetzt. Am Ausgang muss dann nur mehr überprüft werden, ob die Addition korrekt durchgeführt wurde. In Quellcode 4 und Abbildung 7 sind als Beispiel ein Testfall und das Waveform-Diagramm des Addierers mit Subtrahierfunktion dargestellt.

```

initial begin
  ...
  a = 9'd179;
  b = 9'd58;
  sub = 1'b0;
  #5;
  sub = 1'b1;
  #5;
  ...
end

```

Quellcode 4: Testfall des Addierers mit Subtrahierfunktion

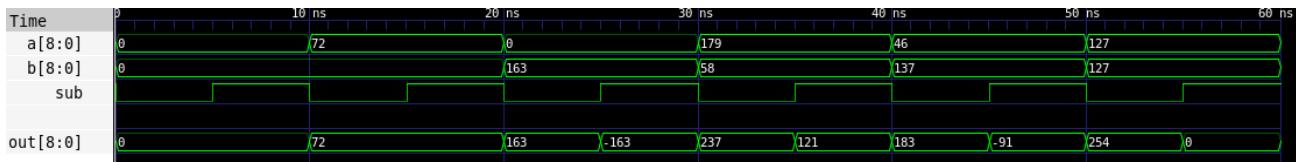


Abbildung 7: Waveform-Diagramm des Addierers mit Subtrahierfunktion

4.5 Gesamtsystem

Das gesamte System wird mit möglichst realistischen Fällen getestet. Für die Simulation werden also unterschiedliche Signale generiert und an den Eingang angelegt. Jedes generierte Signal hat dabei, unabhängig von der Frequenz, die gleiche Dauer in Taktzyklen. Um mehrere Filtertypen zu testen wird auch die Konfiguration des Filters regelmäßig verändert. Dies passiert immer zwischen zwei angelegten Signalen. Zur Kontrolle kann das gefilterte Ausgangssignal mit dem Eingangssignal verglichen und überprüft werden, ob der Filter das gewünschte Verhalten aufweist. Für eine einfache Generierung der Signale und Konfigurierung des Filters werden mehrere Tasks und Konstanten verwendet. In Quellcode 5 ist der typische Ablauf der Testung eines Filters zu sehen und in Abbildung 8 sind die Waveform-Diagramme einiger unterschiedlich konfigurierter Filter dargestellt.

```

initial begin
  ...
  change_config(lowpass, wg_0625);

  gen_rect_signal(w_0_04);
  gen_trian_signal(w_0_50);

```

```

gen_sine_signal(w_0_20);
...
end

```

Quellcode 5: Testung eines Filters

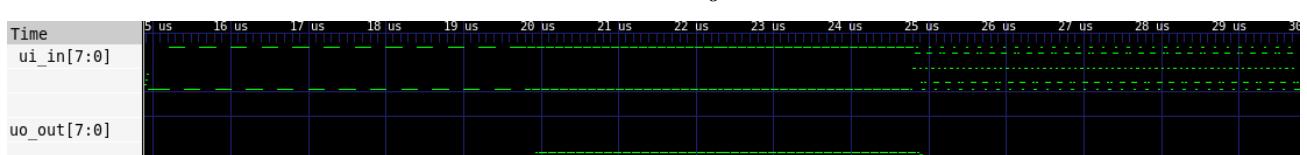
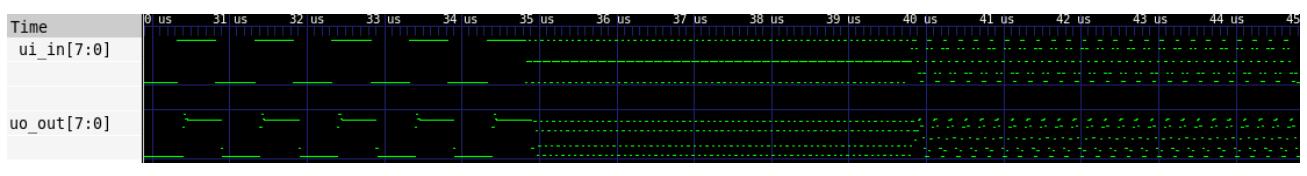
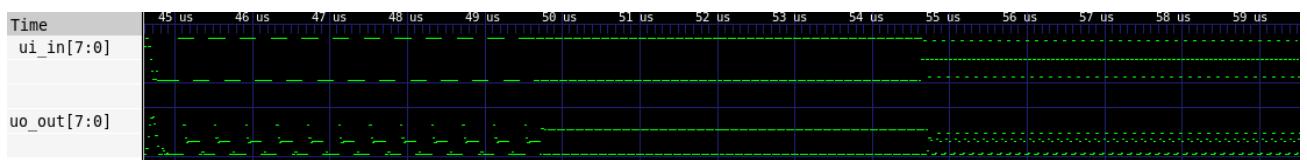
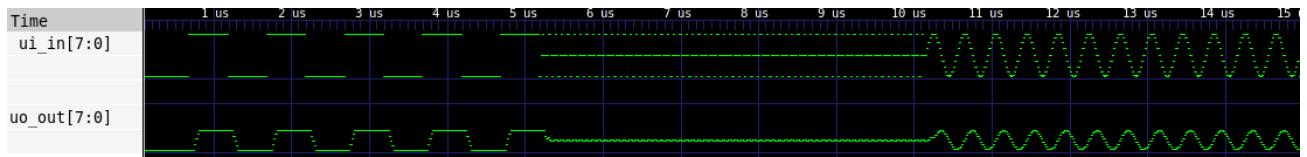


Abbildung 8: Waveform-Diagramme unterschiedlicher Filter

Abbildungsverzeichnis

1	Reduktion einer 4-Bit \times 4-Bit Multiplikation	5
2	Blockschaltbild des optionalen 2er-Komplements	9
3	Blockschaltbild des Gesamtsystems	10
4	Waveform-Diagramm des Volladdierers	11
5	Waveform-Diagramm des ersten Multiplizierers	12
6	Waveform-Diagramm des optionalen 2er-Komplements	13
7	Waveform-Diagramm des Addierers mit Subtrahierfunktion	14
8	Waveform-Diagramme unterschiedlicher Filter	15
9	Bodediagramm des Tiefpasses mit Grenzfrequenz $\omega_g = 0.125$	VIII
10	Bodediagramm des Tiefpasses mit Grenzfrequenz $\omega_g = 0.375$	IX
11	Bodediagramm des Tiefpasses mit Grenzfrequenz $\omega_g = 0.625$	IX
12	Bodediagramm des Tiefpasses mit Grenzfrequenz $\omega_g = 0.875$	X
13	Bodediagramm des Hochpasses mit Grenzfrequenz $\omega_g = 0.125$	X
14	Bodediagramm des Hochpasses mit Grenzfrequenz $\omega_g = 0.375$	XI
15	Bodediagramm des Hochpasses mit Grenzfrequenz $\omega_g = 0.625$	XI
16	Bodediagramm des Hochpasses mit Grenzfrequenz $\omega_g = 0.875$	XII
17	Implementierung mit dem SKY130-PDK	XVI
18	Implementierung mit dem GF180MCU-PDK	XVII

Tabellenverzeichnis

1	Ein-/Ausgänge des digitalen Filters	2
2	Gleichungen des Halb- bzw. Volladdierers	6
3	Zusammengefasste Koeffizienten der Multiplizierer	7
4	Bedingungen für die Koeffizienten und Invertierungen	10
5	Koeffizienten des Tiefpasses mit Grenzfrequenz $\omega_g = 0.125$	VIII
6	Koeffizienten des Tiefpasses mit Grenzfrequenz $\omega_g = 0.375$	IX
7	Koeffizienten des Tiefpasses mit Grenzfrequenz $\omega_g = 0.625$	IX
8	Koeffizienten des Tiefpasses mit Grenzfrequenz $\omega_g = 0.875$	X
9	Koeffizienten des Hochpasses mit Grenzfrequenz $\omega_g = 0.125$	X
10	Koeffizienten des Hochpasses mit Grenzfrequenz $\omega_g = 0.375$	XI
11	Koeffizienten des Hochpasses mit Grenzfrequenz $\omega_g = 0.625$	XI
12	Koeffizienten des Hochpasses mit Grenzfrequenz $\omega_g = 0.875$	XII

Quellcodeverzeichnis

1	Testfall des Volladdierers	11
2	Testung des ersten Multiplizierers	12
3	Testfall des optionalen 2er-Komplements	13
4	Testfall des Addierers mit Subtrahierfunktion	14
5	Testung eines Filters	14
6	Matlab-Skript zur Generierung der Koeffizienten und Bodediagramme	XII

Anhang A

Filtercharakteristika

A.1 Tiefpässe

A.1.1 Grenzfrequenz 0.125

	Koeffizient 1	Koeffizient 2	Koeffizient 3	Koeffizient 4
Exakt	0.125	0.1202424707820	0.1066941738242	0.08641771452282
Quantisiert	0.125	0.1171875	0.10546875	0.0859375
Binär	000100000	000011110	000011011	000010110

Tabelle 5: Koeffizienten des Tiefpasses mit Grenzfrequenz $\omega_g = 0.125$

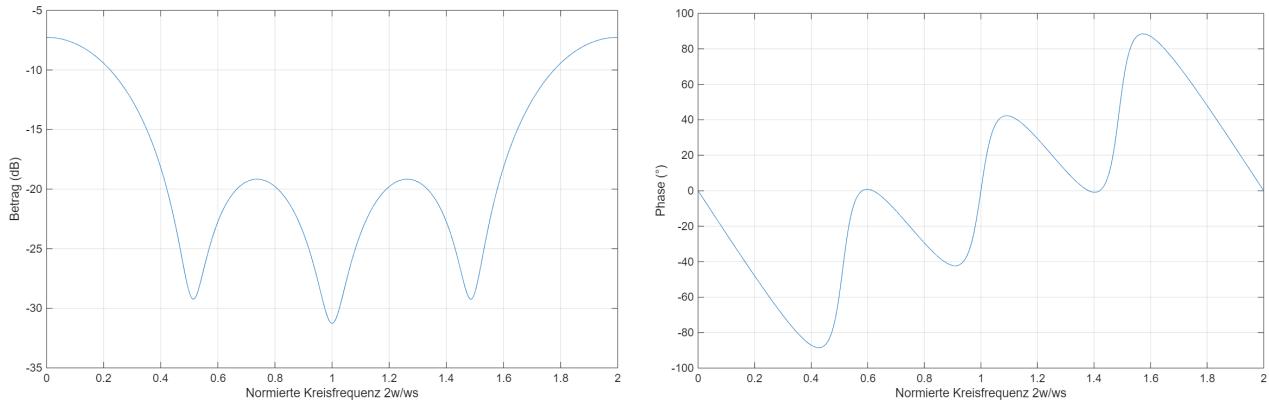


Abbildung 9: Bodendiagramm des Tiefpasses mit Grenzfrequenz $\omega_g = 0.125$

A.1.2 Grenzfrequenz 0.375

	Koeffizient 1	Koeffizient 2	Koeffizient 3	Koeffizient 4
Exakt	0.375	0.2902910037350	0.1066941738242	-0.03579538938464
Quantisiert	0.375	0.2890625	0.10546875	-0.03515625
Binär	001100000	001001010	000011011	111110111

Tabelle 6: Koeffizienten des Tiefpasses mit Grenzfrequenz $\omega_g = 0.375$

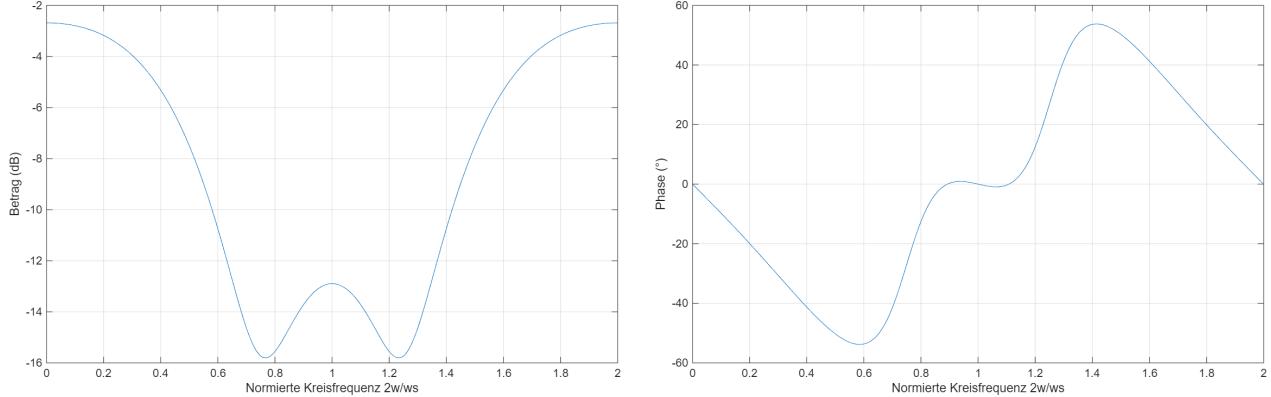


Abbildung 10: Bodediagramm des Tiefpasses mit Grenzfrequenz $\omega_g = 0.375$

A.1.3 Grenzfrequenz 0.625

	Koeffizient 1	Koeffizient 2	Koeffizient 3	Koeffizient 4
Exakt	0.625	0.2902910037350	-0.1066941738242	-0.03579538938464
Quantisiert	0.625	0.2890625	-0.10546875	-0.03515625
Binär	010100000	001001010	111100101	111110111

Tabelle 7: Koeffizienten des Tiefpasses mit Grenzfrequenz $\omega_g = 0.625$

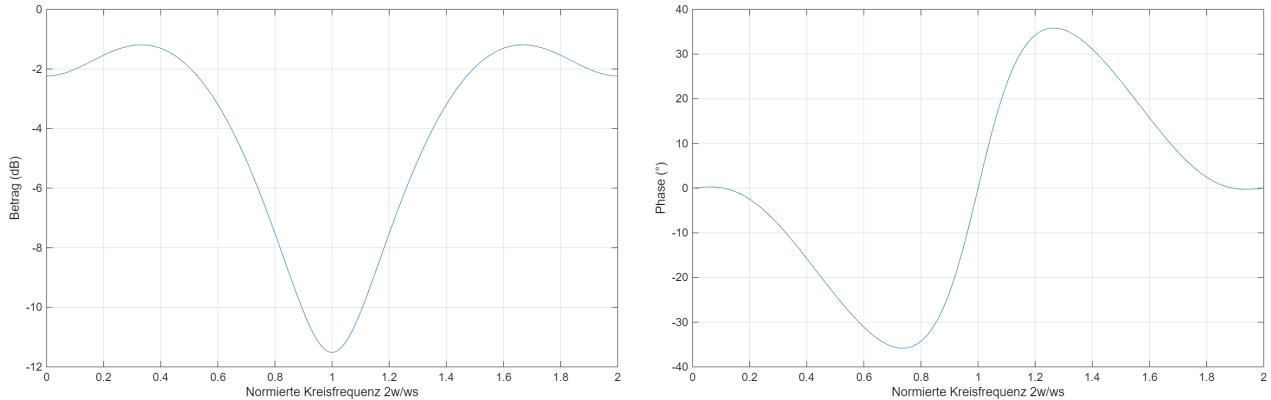
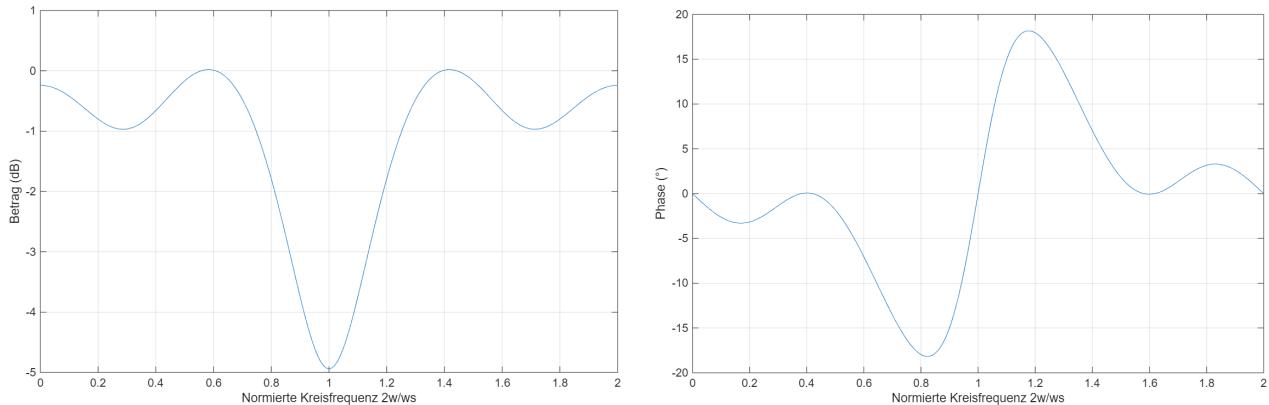


Abbildung 11: Bodediagramm des Tiefpasses mit Grenzfrequenz $\omega_g = 0.625$

A.1.4 Grenzfrequenz 0.875

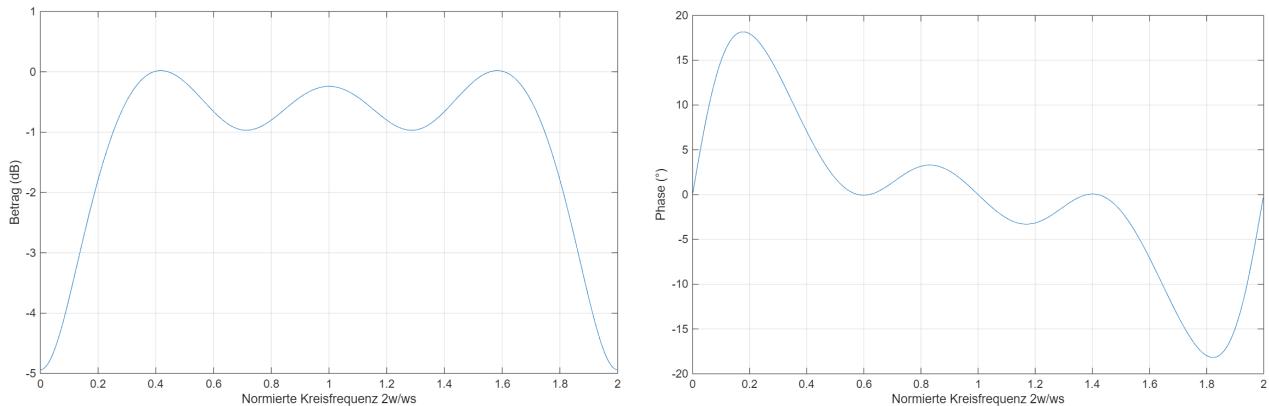
	Koeffizient 1	Koeffizient 2	Koeffizient 3	Koeffizient 4
Exakt	0.875	0.1202424707820	-0.1066941738242	0.08641771452282
Quantisiert	0.875	0.1171875	-0.10546875	0.0859375
Binär	011100000	000011110	111100101	000010110

Tabelle 8: Koeffizienten des Tiefpasses mit Grenzfrequenz $\omega_g = 0.875$ Abbildung 12: Bodediagramm des Tiefpasses mit Grenzfrequenz $\omega_g = 0.875$

A.2 Hochpässe

A.2.1 Grenzfrequenz 0.125

	Koeffizient 1	Koeffizient 2	Koeffizient 3	Koeffizient 4
Exakt	0.875	-0.1202424707820	-0.1066941738242	-0.08641771452282
Quantisiert	0.875	-0.1171875	-0.10546875	-0.0859375
Binär	011100000	111100010	111100101	111101010

Tabelle 9: Koeffizienten des Hochpasses mit Grenzfrequenz $\omega_g = 0.125$ Abbildung 13: Bodediagramm des Hochpasses mit Grenzfrequenz $\omega_g = 0.125$

A.2.2 Grenzfrequenz 0.375

	Koeffizient 1	Koeffizient 2	Koeffizient 3	Koeffizient 4
Exakt	0.625	-0.2902910037350	-0.1066941738242	0.03579538938464
Quantisiert	0.625	-0.2890625	-0.10546875	0.03515625
Binär	010100000	110110110	111100101	000001001

Tabelle 10: Koeffizienten des Hochpasses mit Grenzfrequenz $\omega_g = 0.375$

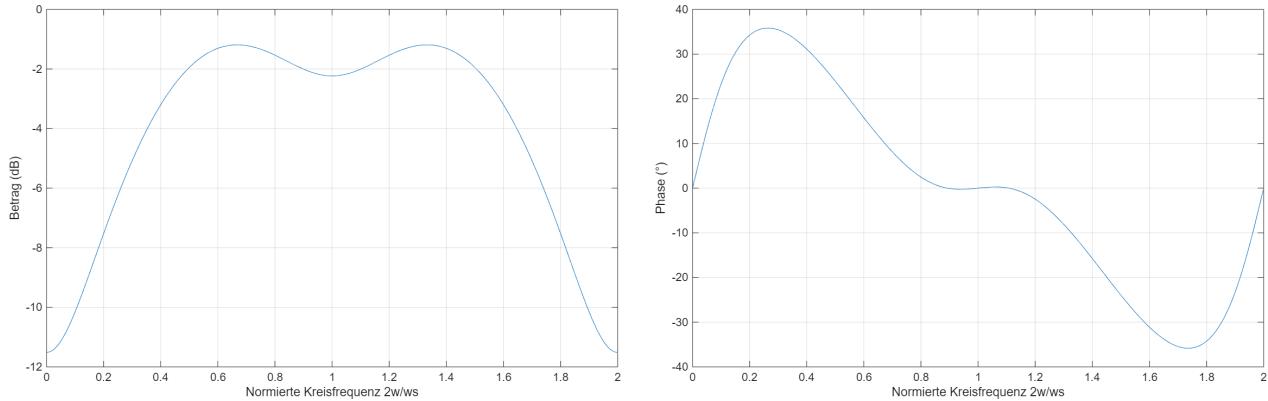


Abbildung 14: Bodediagramm des Hochpasses mit Grenzfrequenz $\omega_g = 0.375$

A.2.3 Grenzfrequenz 0.625

	Koeffizient 1	Koeffizient 2	Koeffizient 3	Koeffizient 4
Exakt	0.375	-0.2902910037350	0.1066941738242	0.03579538938464
Quantisiert	0.375	-0.2890625	0.10546875	0.03515625
Binär	001100000	110110110	000011011	000001001

Tabelle 11: Koeffizienten des Hochpasses mit Grenzfrequenz $\omega_g = 0.625$

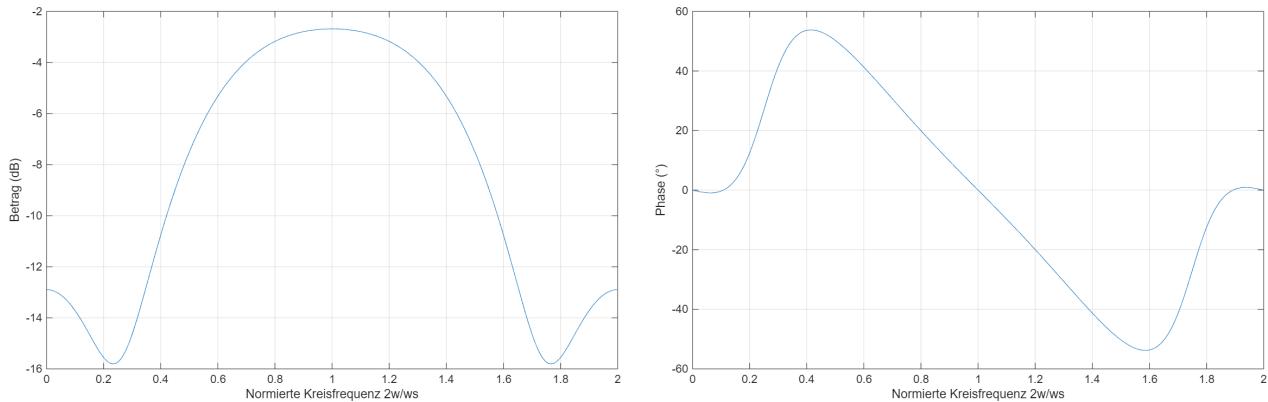
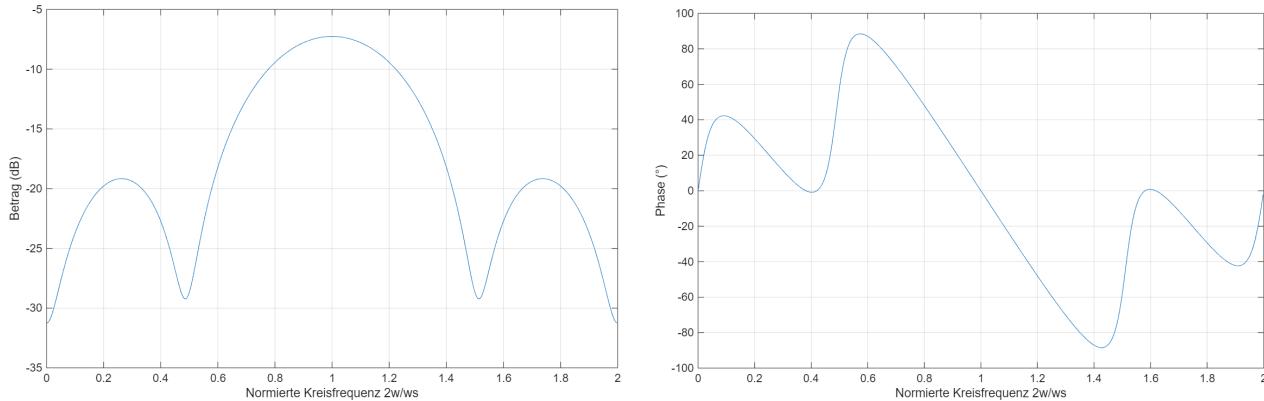


Abbildung 15: Bodediagramm des Hochpasses mit Grenzfrequenz $\omega_g = 0.625$

A.2.4 Grenzfrequenz 0.875

	Koeffizient 1	Koeffizient 2	Koeffizient 3	Koeffizient 4
Exakt	0.125	-0.1202424707820	0.1066941738242	-0.08641771452282
Quantisiert	0.125	-0.1171875	0.10546875	-0.0859375
Binär	000100000	111100010	000011011	111101010

Tabelle 12: Koeffizienten des Hochpasses mit Grenzfrequenz $\omega_g = 0.875$ Abbildung 16: Bodediagramm des Hochpasses mit Grenzfrequenz $\omega_g = 0.875$

A.3 Matlab-Skript zur Generierung

```

close all;
format long;

N1 = 16;
NC = 4;
N = 1024;
x = 0:2/N:2-2/N;
dimp = zeros(1, N);
dimp(1) = 1;

for wg = 0.125:0.25:0.875
    disp(['wg = ', num2str(wg), ':']);
    % Calculate exact coefficients
    ispec1=[ones(1,N1*wg/2),zeros(1,N1*(1-wg)),ones(1,N1*wg/2)];
    ispec2=[zeros(1,N1*wg/2),ones(1,N1*(1-wg)),zeros(1,N1*wg/2)];

```

```
hl = real(ifft(ispecl));
hh = real(ifft(ispech));
bl = hl(1:NC);
bh = hh(1:NC);

% Calculate quantized coefficients
blx = abs(bl);
bxh = abs(bh);
blt = zeros(1, NC);
bht = zeros(1, NC);
blb = zeros(NC, 9);
bhb = zeros(NC, 9);
for i = 1:8
    for j = 1:NC
        if 2^(-i) <= blx(j) - blt(j)
            blt(j) = blt(j) + 2^(-i);
            blb(j, i+1) = 1;
        end
        if 2^(-i) <= bxh(j) - bht(j)
            bht(j) = bht(j) + 2^(-i);
            bhb(j, i+1) = 1;
        end
    end
end
for i = 1:NC
    if bl(i) < 0
        blt(i) = -blt(i);
        k = 9;
        while k >= 1 && blb(i, k) == 0
            k = k - 1;
        end
        for j = 1:k-1
            if blb(i, j) == 0
                blb(i, j) = 1;
            elseif blb(i, j) == 1
                blb(i, j) = 0;
            end
        end
    end
end
```

```
        end
    end
end
if bh(i) < 0
    bht(i) = -bht(i);
k = 9;
while k >= 1 && bhb(i, k) == 0
    k = k - 1;
end
for j = 1:k-1
    if bhb(i, j) == 0
        bhb(i, j) = 1;
    elseif bhb(i, j) == 1
        bhb(i, j) = 0;
    end
end
end
end

% Display coefficients
disp('Tiefpass:');
for i = 1:NC
    disp(['Koeffizient ', num2str(i), ':']);
    disp(['Exakt: ', num2str(bl(i), 15)]);
    disp(['Quantisiert: ', num2str(blt(i), 15)]);
    disp(['Binär: ', num2str(blb(i, :))]);
end
disp([newline, 'Hochpass:']);
for i = 1:NC
    disp(['Koeffizient ', num2str(i), ':']);
    disp(['Exakt: ', num2str(bh(i), 15)]);
    disp(['Quantisiert: ', num2str(bht(i), 15)]);
    disp(['Binär: ', num2str(bhb(i, :))]);
end
disp(newline);
```

```

% Calculate bode plots
yl = filter(blt, 1, dimp);
yh = filter(bht, 1, dimp);
spec1 = fft(yl);
spec2 = fft(yh);
al = 20*log10(abs(spec1));
pl = angle(spec1)*180/pi;
ah = 20*log10(abs(spec2));
ph = angle(spec2)*180/pi;

% Plot bode plots
figure('Name',[ 'ATP0', num2str(wg*1000)], 'NumberTitle','off');
plot(x, al);
xlabel('Normierte Kreisfrequenz 2w/ws');
ylabel('Betrag (dB)');
grid;
figure('Name',[ 'PTP0', num2str(wg*1000)], 'NumberTitle','off');
plot(x, pl);
xlabel('Normierte Kreisfrequenz 2w/ws');
ylabel('Phase (°)');
grid;
figure('Name',[ 'AHP0', num2str(wg*1000)], 'NumberTitle','off');
plot(x, ah);
xlabel('Normierte Kreisfrequenz 2w/ws');
ylabel('Betrag (dB)');
grid;
figure('Name',[ 'PHP0', num2str(wg*1000)], 'NumberTitle','off');
plot(x, ph);
xlabel('Normierte Kreisfrequenz 2w/ws');
ylabel('Phase (°)');
grid;
end

```

Quellcode 6: Matlab-Skript zur Generierung der Koeffizienten und Bodediagramme

Anhang B

Bilder der Implementierungen

B.1 SkyWater Technologies 130nm PDK

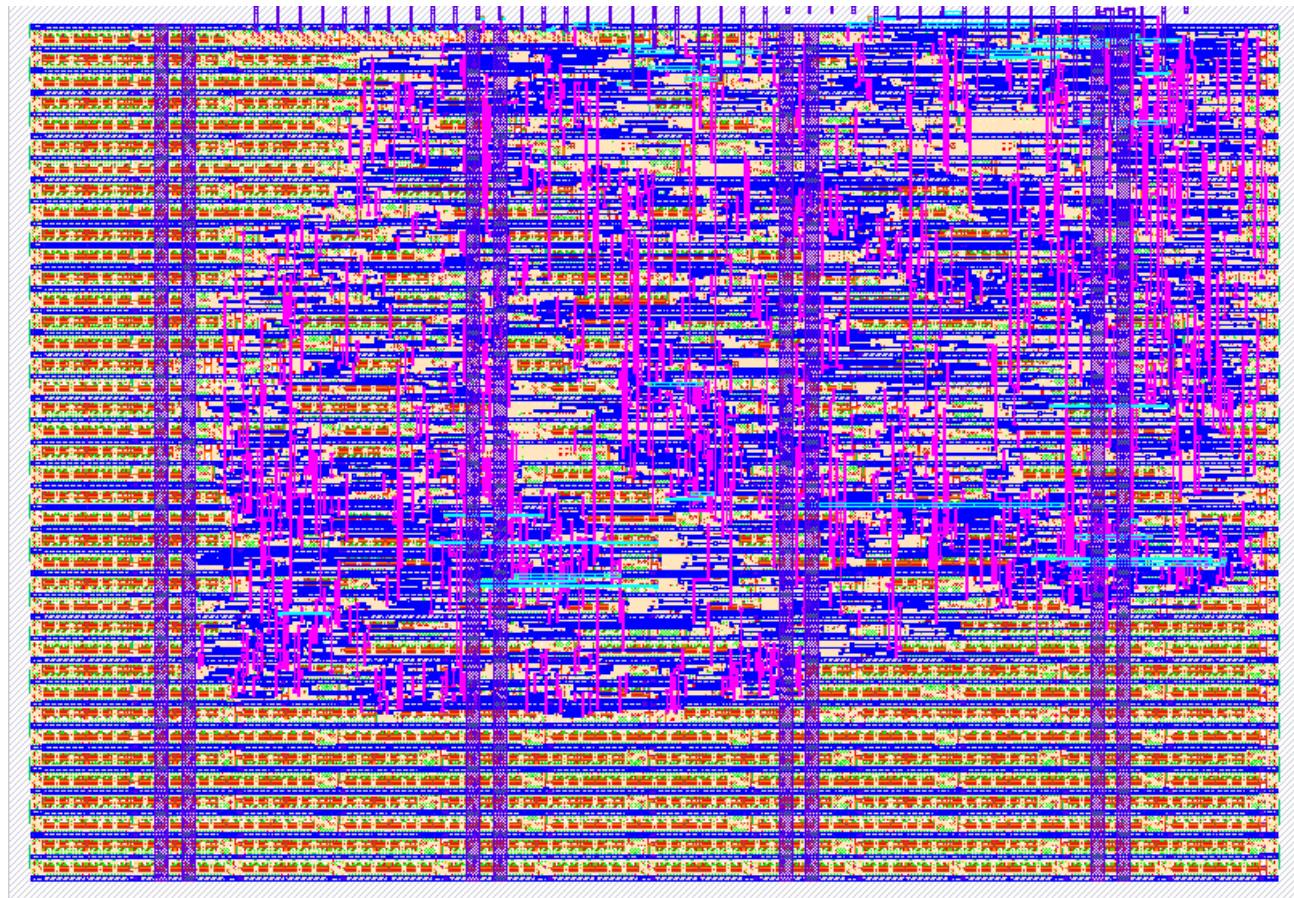


Abbildung 17: Implementierung mit dem SKY130-PDK

B.2 GlobalFoundries 180nm PDK



Abbildung 18: Implementierung mit dem GF180MCU-PDK