Festkomma-Arithmetik – einfacher als man glaubt

Einsatz in eigenen Algorithmen und Bibliotheken

Ferdinand Englberger, Universität der Bundeswehr München

Obwohl immer mehr Mikrocontroller über Gleitkommarechenwerke verfügen, wird Festkomma-Arithmetik in vielen Bibliotheken z. B. für digitale Signalverarbeitung, für neuronale Netze oder in Regelungsalgorithmen eingesetzt. Die Befehlssätze der Prozessoren haben Erweiterungen zum Umgang mit Festkommazahlen und ermöglichen mit SIMD-Instruktionen deren effektive Verarbeitung. Es werden die Grundlagen der Festkomma-Arithmetik erläutert und damit das notwendige Verständnis geschaffen, um diese Arithmetik in eigenen Algorithmen und in Bibliothekfunktionen effektiv einsetzen zu können.

Durch die direkte Unterstützung der Prozessoren von Gleitkomma-Arithmetik ist es für den Entwickler leicht Algorithmen zu implementieren. Aus Effizienzgründen werden hierbei Gleitkommazahlen mit einfacher Genauigkeit eingesetzt. Dabei wird leicht übersehen, dass bei Verwendung dieses Zahlentyps Fehler entstehen können, die zum Verlust des gewünschten Designziels führen. In Abbildung 1 ist dies an einem einfachen Beispiel verdeutlicht. Die Addition von kleinen Zahlenwerten mit anschließender Addition eines großen Werts führt zum gewünschten Ergebnis. Bei der umgekehrten Vorgehensweise werden die kleinen Zahlenwerte im Ergebnis nicht berücksichtigt. Realisiert man FIR-Filter ohne dies zu bedenken, verliert man die Eigenschaft der Linearphasigkeit.

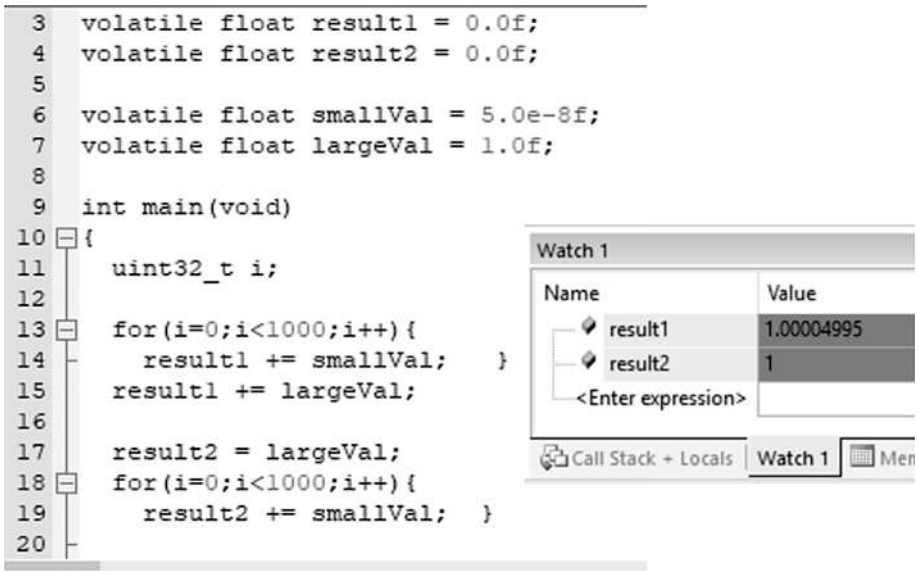

Abb. 1: Problem bei der Nutzung von Gleitkommazahlen.

Bei der Festkomma-Arithmetik geht es darum mit möglichst geringem Aufwand Berechnungen durchzuführen. Dabei wird der Multiplizierer des Prozessors oder eines FPGAs genutzt. Divisionen sind aufgrund höherer Rechenzeiten zu vermeiden. Abbildung 2 zeigt ein einfaches Beispiel für eine Festkommaberechnung. Für `result1` (Ganzzahlarithmetik) wird eine Berechnung mit Multiplikation und

Division genutzt, für `result2` (Festkomma-Arithmetik) wird die Division durch eine Schiebeoperation ersetzt, die in modernen Prozessoren als Bestandteil eines Speicherbefehls integriert ist. Festkomma-Arithmetik ist somit ein optimiertes Rechenverfahren, bei dem Divisionen nur mit Werten durchgeführt werden, die sich als Zweierpotenz darstellen lassen.

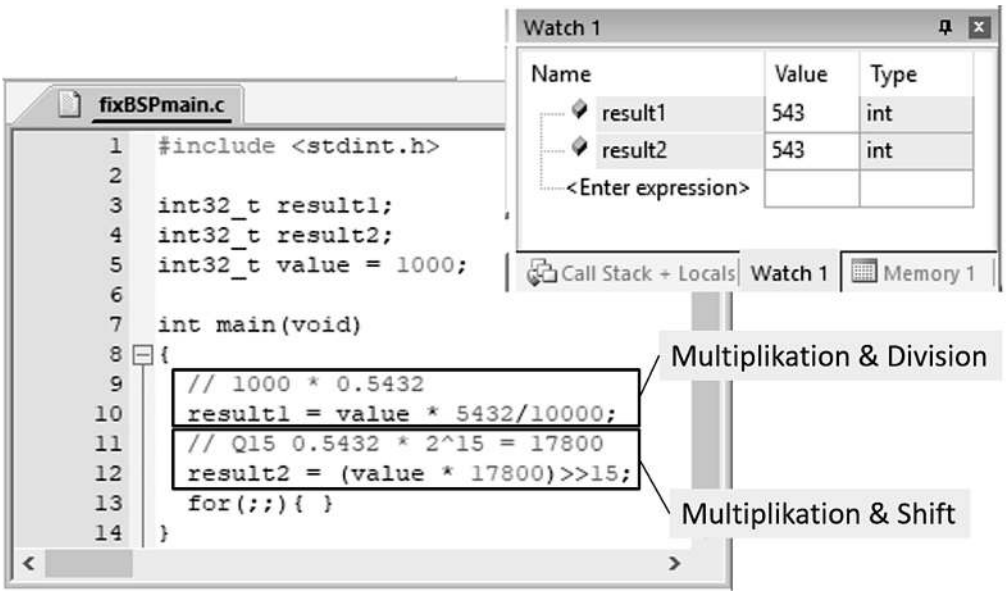

Abb. 2: Beispiel für Festkomma-Arithmetik.

Darstellung der Zahlenwerte

Abbildung 3 zeigt den Aufbau von Festkommazahlen. Mit *x* wird in *Qx* die Position des Kommas angegeben. Die Nachkommastellen werden als *fractional bits* bezeichnet. Benötigt man Zahlenwerte mit einem Wertebereich von größer als eins, werden zusätzlich *integer bits* benötigt. Da jeder Zahlenwert *x* möglichst exakt dargestellt werden soll, werden so wenig wie möglich *integer bits* n_i für die Festlegung eines Zahlenwerts genutzt.

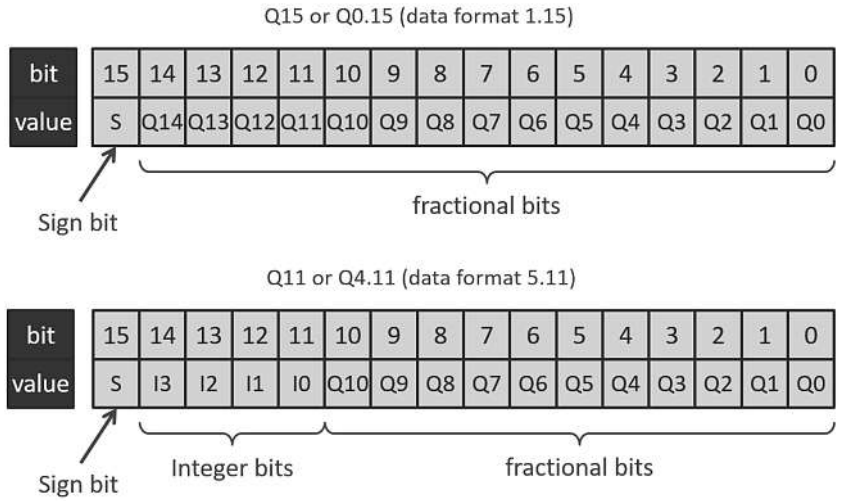

Abb. 3: Darstellung von Festkomma-Zahlen.

Die benötigte Anzahl von *integer bits* n_i in einem Datenwort x mit n Bits errechnet sich dadurch, dass vom Betrag des Zahlenwerts der Zweierlogarithmus gebildet und auf ganze Bits aufgerundet wird.

$$n_i = \mathrm{ceil}\big(\mathrm{ld}(|x|)\big)$$

Die Anzahl der Nachkommastellen P einer QP-Zahl ergibt sich zu:

$$P = n - 1 - n_i$$

Der äquivalente Integerwert x_i einer Festkommazahl ergibt sich, indem der Zahlenwert x durch die Auflösung der Zahl 2^{-P} geteilt und gerundet wird.

$$x_i = \mathrm{round}\left(\frac{x}{2^{-P}}\right) = \mathrm{round}(x \cdot 2^P)$$

Bei Berechnungen müssen folgende einfache Regeln beachtet werden:

- Bei Additionen müssen die Kommas übereinanderliegen.
- Bei Multiplikationen ergibt sich die neue Position des Kommas aus der Addition der Kommapositionen der Operanden, z. B. `Q15·Q15 = Q30`. Die Wortbreite des Ergebnisses ist die Summe der Stellen der beiden Operanden.
- Weitere Additionen werden mit der Multiplikationsergebniswortbreite durchgeführt.
- Die Wortbreite des Endergebnisses wird auf die gewünschte Breite reduziert. Dabei wird häufig Sättigungsarithmetik und Rundung verwendet.

Wie Abbildung 4 zeigt, befindet sich in der Ergebnisvariablen ein Bereich mit dem gewünschten Ergebnis und der gewünschten Genauigkeit. Zusätzlich sind *fractional bits* vorhanden, die bei Additionen zur Verbesserung des Ergebnisses dienen. Der Guard Bereich wird genutzt, um fehlerhafte Ergebnisse zu vermeiden, wenn der Ergebnisbereich bei der Durchführung von Additionen nicht ausreicht.

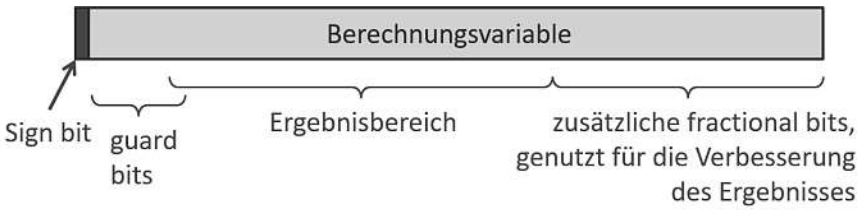

Abb. 4: Format der Ergebnisvariablen.

Die Problematik der Sättigungsarithmetik und die Notwendigkeit des Guardbereichs, wird in Abbildung 5 verdeutlicht. Algorithmen sind häufig so dimensioniert, dass ein bestimmter Wertebereich für das Ergebnis erwartet wird. Dies gilt z. B. für digitale Filter. Trotz dieser Dimensionierung können Zwischenergebnisse diesen Bereich überschreiten. Dabei sollen jedoch keine Fehler durch Zahlenbereichsüberschreitung entstehen und das Endergebnis soll möglichst exakt sein. Im gezeigten Beispiel führt die Addition der beiden Operanden zu einer Änderung des Vorzeichenbits im Ergebnisbereich. Ohne den Guardbereich wäre dies ein gravierender Fehler. Führt man am Ende der Berechnung eine Sättigung durch, erhält man zwar ebenfalls kein korrektes Ergebnis. Der Fehler ist jedoch sehr viel kleiner als ohne Sättigung. Im vorliegenden Beispiel wurde für die Addition von zwei Operanden ein Guardbit benötigt. Bei einer höheren Anzahl von Additionen muss die Anzahl der Guardbits entsprechend angepasst werden, damit ein Verlassen des Ergebnisbereichs zuverlässig erkannt werden kann.

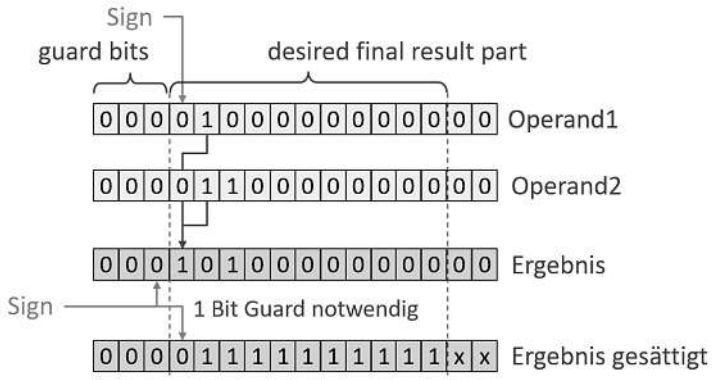

Abb. 5: Funktion des Guard und der Sättigung.

Berechnungen

Abbildung 6 zeigt die Berechnung einer Multiply-Accumulate-Operation (MAC) mit 16 Bit-Zahlenwerten. Bei einem 32 Bit-Prozessor kann der Ergebniswert in einem Register gespeichert werden. Beide Operanden haben das Zahlenformat Q15. Nach der Multiplikation ergibt sich ein Zahlenformat von Q30. Damit steht ein Guard-Bereich von einem Bit zur Verfügung. Um das gewünschte Zielzahlenformat Q15 zu verwenden, muss das Ergebnis um 15 Stellen nach rechts geschoben und dann einer Sättigungsoperation unterzogen werden. Da nur ein Guard-Bit zur Verfügung steht,

kann es bei dieser Zahlendarstellung bei mehr als zwei Additionen zu einer nicht erkannten Zahlenbereichsüberschreitung kommen.

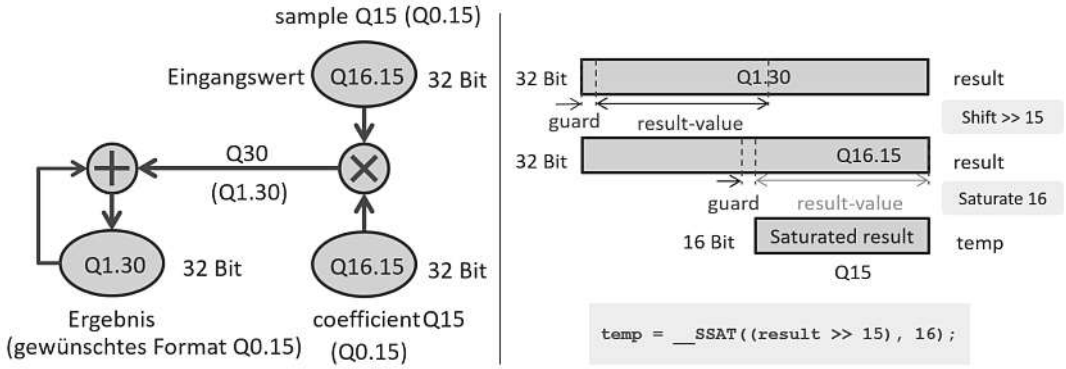

Abb. 6: Durchführung einer MAC-Operation 16·16 bit.

Benötigt man für einen Operanden eine höhere Genauigkeit als 16 Bit, ist ein 64 Bit-Ergebnis (zwei 32 Bit Register) erforderlich. Für moderne 32 Bit-Prozessoren steht für diese Anweisung ein Assembler-Befehl zur Verfügung. Abbildung 7 zeigt die Rechenoperation Q15·Q31. Es wird davon ausgegangen, dass als Ergebniswortbreite lediglich 16 Bit benötigt werden. Damit stehen 17 Guard-Bits zur Verfügung und es könnten bis zu 2^{17} Additionen unter dem Schutz des Guard durchgeführt werden. Diese Lösung hat jedoch den gravierenden Nachteil, dass bei den 64 Bit-Additionen zwei Maschinenbefehle benötigt werden und dass die Schiebeoperationen über zwei Register hinweg durchgeführt werden müssen. Dies führt zu einem erhöhten Rechenaufwand.

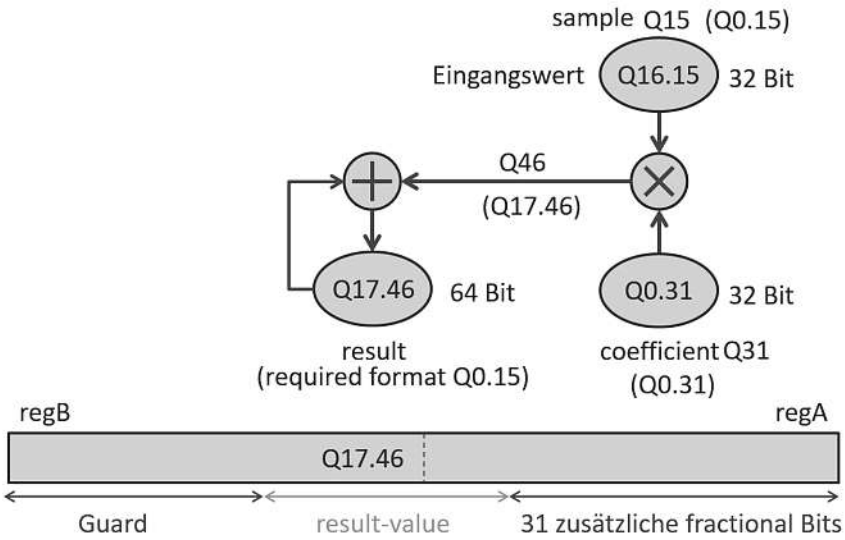

Abb. 7: Durchführung einer MAC-Operation 16·32 bit.

Um den Nachteil zu vermeiden, dass das Ergebnis in zwei verschiedenen Prozessor-Registern gespeichert wird, kann das Zahlenformat des Sample-Operanden angepasst werden. In Abbildung 8 wurde die Position des Eingangswerts auf Q31 festgelegt. Bei einer 32 Bit·32 Bit Multiplikation befindet sich das 16 Bit-Ergebnis nur noch in einem Register. Aufgrund der hohen Anzahl von zusätzlichen *fractional bits* außerhalb des Ergebnisbereichs, kann auf die Nutzung des zweiten Registers verzichtet werden. In der in Abbildung 8 gezeigten Darstellung steht allerdings nur ein Guardbit zur Verfügung. Werden mehr Guardbits benötigt, kann dies durch die Platzierung in der Sample-Variablen angepasst werden. In Abbildung 9 wurde die Position so verändert, dass nun drei Guard-Bits zur Verfügung stehen.

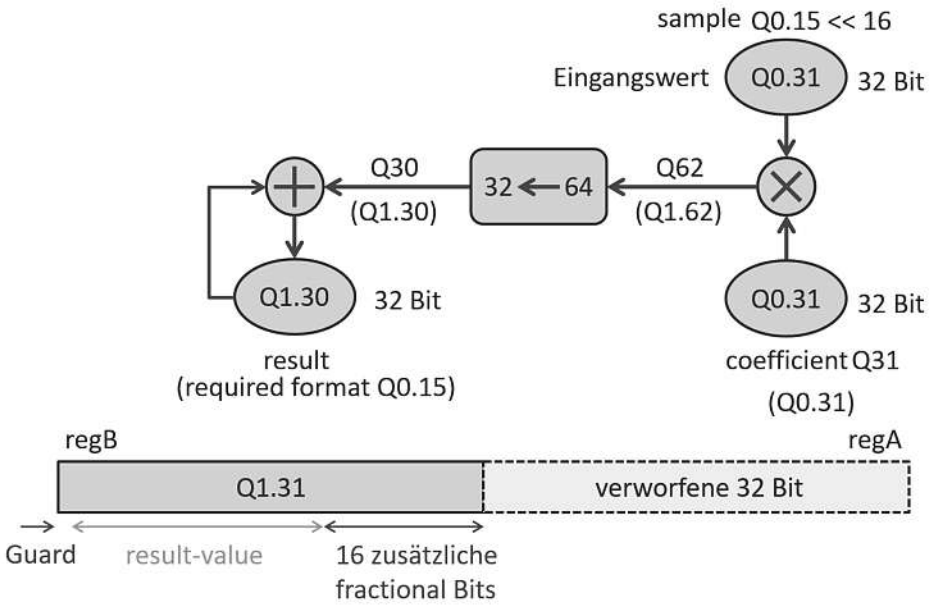

Abb. 8: Durchführung einer MAC-Operation 16·32 bit (optimiert).

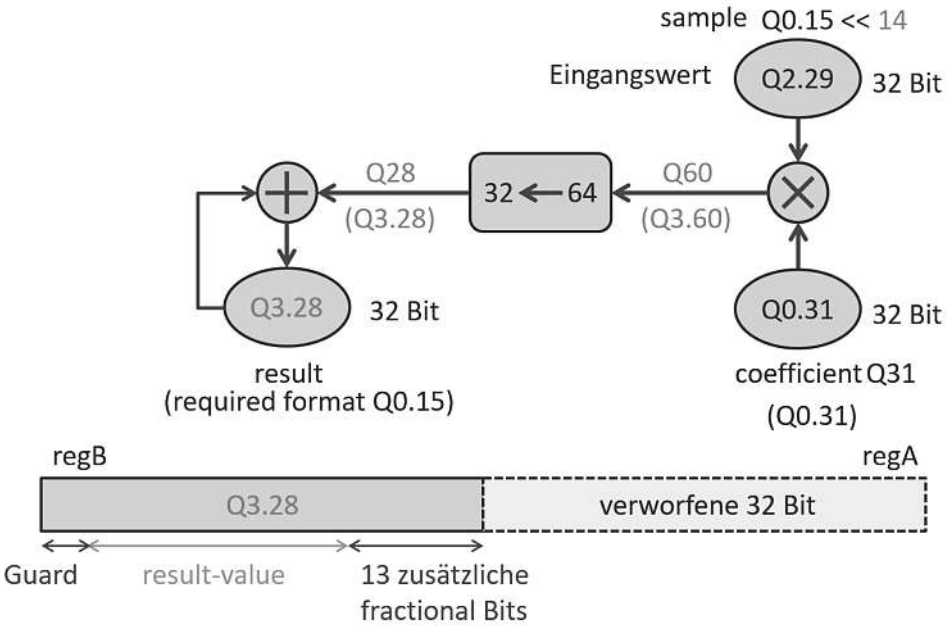

Abb. 9: Durchführung einer MAC-Operation 16·32 bit (optimiert).

FPGA und ASIC

In der bisherigen Darstellung wurde die Funktionalität des Guard-Bereichs herausgehoben. Die zusätzlichen *fractional bits* wurden nicht näher betrachtet. Für eine Lösung mit einem Prozessor ist diese Vorgehensweise ausreichend, da dieser Daten verarbeitet, die ein Vielfaches seiner Wortbreite sind. Bei einem FPGA oder ASIC wird man jedoch versuchen nur die notwendigen Ressourcen zu belegen. Wenn zusätzliche *fractional bits* das Ergebnis nicht beeinflussen können, so können sie weggelassen werden. Die Überlegung ist dabei ähnlich wie bei der Bestimmung des Guard-Bereichs. Bei zwei Additionen ist ein zusätzliches Bit zu berücksichtigen, bei vier Additionen zwei Bits und bei 2^n Additionen n Bit. Abbildung 10 zeigt exemplarisch wie bei einem FPGA vorzugehen ist. Im Beispiel sollen sieben MAC-Operationen mit 10 Bit-Werten durchgeführt werden. Es soll eine Lösung gefunden werden, die bei minimalen Ressourceneinsatz zu einem exakten und fehlerfreien Ergebnis führt. Sieben Additionen erfordern drei Guard Bits um einen Zahlenüberlauf zu verhindern und drei zusätzliche *fractional bits,* um möglichst exakte Ergebnisse zu erhalten. Multipliziert man zwei 10 Bit-Werte erhält man ein 20 Bit-Ergebnis mit einem Guard-Bit und 9 zusätzlichen *fractional bits*. Für die weitere Verarbeitung müssen zwei Guard-Bits ergänzt und es können sechs *fractional bits* weggelassen werden. Mit der nun erzeugten 16 Bit-Variablen kann die Aufsummation durchgeführt werden. Abgeschlossen wird die Berechnung durch einen Sättigungsvorgang.

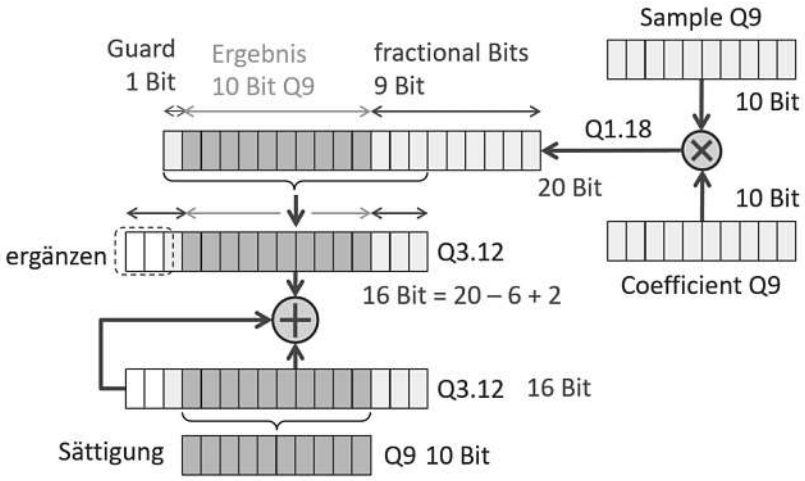

Abb. 10: Planung eines Algorithmus für den FPGA.

Zusammenfassung
Festkomma-Arithmetik ist eine effiziente und ressourcensparende Methode zur Implementierung von Algorithmen. Hierbei können spezielle Fähigkeiten eines Prozessors wie SIMD effektiv eingesetzt werden.

Literatur

[1] ARM Ltd, CMSIS - Cortex Microcontroller Software Interface Standard, http://www.arm.com/products/processors/cortex-m/cortex-microcontroller-software-interface-standard.php

Autor

Prof. Dr.-Ing. Ferdinand Englberger ist Professor für Embedded Systems und Digitale Signalverarbeitung an der Universität der Bundeswehr München in der Fakultät für Elektrotechnik und Technische Informatik. Seine Lehr- und Forschungsgebiete sind Embedded Systems, Robotik, System on a Chip und Digitale Signalverarbeitung.

Email: ferdinand.englberger@unibw.de