Digitaltechnik Aufarbeitung

Kajetan Weiß

17. Oktober 2013

Inhaltsverzeichnis

1 Digitale Nachrichten und Information

Die Begriffe Nachricht, Signal, Information und Daten werden in der Informatik Fachsprache häufig verwendet. Hier die Definitionen der Begriffe:

- Das Signal ist eine messbare zeitliche Änderung eines Zustands. Es wird zwischen analogen Signalen und diskreten Signalen unterschieden. Werte analoger Signale können oft nahezu beliebig viele Zustände und Zwischenzustände abbilden. Werte diskreter Signale können endlich zählbar vielen Zuständen zugeordnet werden. Statt diskreter Signale und Werte wird hier auch von digitalen[1] Signalen und Werten gesprochen.

  Beispiele für analoge Signale: Spannungssignal, Stromsignal, Tonsignal, Wasserstand, Federstand

  Beispiele für diskrete Signale: Morsesignal, Flaggensignal, Erfassung von High/Low Spannungssignalen

- Die Nachricht besteht aus Symbolen zur Beschreibung von Information. Die Bedeutung der Symbole muss erlernt werden. Die Nachricht ist also konkret im Gegensatz zur abstrakten Information.

- Die Daten (Singular Datum) bestehen ebenfalls aus Symbolen zur Beschreibung von Information. Wird im Kontext von der Verarbeitung von Information gesprochen wird der Begriff Daten verwendet.

- Information und Nachricht sind nicht dasselbe. Denn der Informationsgehalt ist vom Empfänger abhängig. Liest ein Empfänger eine ihm bekannte Nachricht lernt er nichts neues. Der Informationsgehalt ist also bei einer bekannten Nachricht null. Der Informationsgehalt einer Nachricht ist demnach messbar am Grad der Neuigkeit.

Eine Information muss aus einer Nachricht ermittelt werden. Dazu dient die Informationsvorschrift.

1.1 Analog-Digital-Wandlung einer Messgröße (Quantisierung)

Werte analoger Signale werden in der digitalen Datenverarbeitung auf diskrete Signalwerte abgebildet. Dies vereinfacht die Verarbeitung und ist vergleichsweise kostengünstiger.

[1] digital von digitus lat. Finger

Wie beschrieben kann der Wertebereich analoger Signalwerte überabzählbar groß und unendlich sein. Diskrete Signalwerte sind allerdings immer einem abzählbaren und in diesem Fall endlichen Wertebereich zu zuordnen. Um eine Abbildung analoger Werte auf diskrete zu ermöglichen steht ein diskreter Wert stets für ein Intervall analoger Werte. Also kann von einem diskreten Wert nicht mehr auf genau einen analogen Ursprungswert geschlossen werden. Es entsteht ein Genauigkeitsverlust. Die Aufteilung des kontinuierlichen Wertebereichs eines analogen Signals wird *Quantisierung* und der Genaugkeitsverlust wird *prinzipieller Quantisierungsfehler* genannt. Die einzelnen Intervalle werden als *Quanten* bezeichnet.

Um den maximalen Quantisierungsfehler möglichst klein zu halten empfiehlt es sich logischer Weise alle Quanten gleich groß zu halten. Außerdem gilt: je größer die Auflösung, das heißt je mehr Quanten für das Abbilden des analogen Wertebereichs verwendet werden, desto kleiner wird der maximale Quantisierungsfehler.

Zudem ist wegen ungenauen technischen Messungen nicht sicher ob ein analoger Signalwert, der in unmittelbarer Nähe einer Intervallgrenze liegt, dem richtigen Intervall zugeordnet wird. Der so entstehende Fehler wird *technischer Quantisierungsfehler* genannt. Die kritischen Werte werden in *Unsicherheitsintervallen* auf der jeweiligen Grenze von einem Quantum zum nächsten zusammengefasst.

1.1.1 A/D-Wandlung mit vollständigem Signalumfang

Es sei ein analoges Signal s_a mit kleinstem annehmbarem Wert s_{min} und größtem annehmbarem Wert Wert s_{max}. Die Wertemenge des analogen Signals ist also:

$$M_{sa} = \{s_a \in \mathbb{R} \mid s_{min} \leq s_a \leq s_{max}\}$$

Das analoge Signal s_a soll in ein digitales Signal s_d gewandelt werden. Dies geschieht über die Quantisierung, wobei jedes Quantum gleich groß sei. Dies wird äquidistante Quantisierung genannt. Bei einer Wandlung mit vollständigem Signalumfang wird der niedrigste Wert des digitalen Signals auf den niedrigsten Wert des analogen Signals gesetzt und der höchste Wert des digitalen Signals auf den höchsten Wert des analogen Signals. Die Anzahl der digitalen Werte sei N_d.

Für die Anzahl der Quanten N_Q gilt bei Wandlung mit vollständigem Signalumfang[2]:

$$N_Q = N_d - 1$$

Die konstante Quantum-Größe[3] Δs ist demnach:

$$\Delta s = \frac{s_{max} - s_{min}}{N_Q} = \frac{s_{max} - s_{min}}{N_d - 1}$$

Die Wertemenge des digitalen Signals ist also:

$$M_{sd} = \{s_{di} \mid (s_{di} = s_{min} + i \cdot \Delta s) \wedge (i \in \mathbb{N}) \wedge (0 \leq i \leq N_d - 1)\}$$

[2]Prof. Gemmar benennt die Variable N_Q mit N_I für die Anzahl Intervalle.
[3]Prof. Gemmar nennt die Quantum-Größe Δs Intervallbreite Δs_i.

Abbildung 1.1 zeigt schematisch die Abbildung der analogen Werte auf die entsprechenden digitalen Werte mittels der jeweiligen Schwellwerte sw_i. Beispielsweise werden alle analogen Werte zwischen s_{min} und sw_1 auf den digitalen Wert s_{d1} abgebildet und alle analogen Werte zwischen sw_1 und sw_2 auf den digitalen Wert s_{d2}.

Der maximale Quantisierungsfehler lässt sich über den Schwellwert und die Quantum-Größe bestimmen. Um den maximalen Quantisierungsfehler zu minimieren wird der jeweilige Schwellwert eines Quantums auf den Mittelwert des jeweiligen Quantums gesetzt. Formal ergibt sich für die Schwellwertemenge also:

$$M_{sw} = \left\{ sw_i \mid \left(sw_i = s_{min} + \frac{1}{2}\Delta s + i \cdot \Delta s \right) \wedge (i \in \mathbb{N}) \wedge (0 \leq i \leq N_Q - 1) \right\}$$

Der maximale Quantisierungsfehler F_{Qmax} beträgt also wegen der äquidistanten Quantisierung und der, wie angegeben, gesetzten Schwellwerte:

$$F_{Qmax} = \pm\frac{1}{2}\Delta s$$

1.1.2 A/D-Wandlung mit unvollständigem Signalumfang

Als nächstes wird betrachtet wie eine äquidistante Quantisierung vorgenommen werden kann, ohne dass der minimale und maximale Wert vom analogen Signal Teil der Wertemenge vom digitalen Signal sind. Dies ergibt im Vergleich zur A/D-Wandlung mit vollständigem Signalumfang bei gleich bleibender Anzahl digitaler Werte einen Genauigkeitsvorteil, da der Quantisierungsfehler bei dieser Methode kleiner ist. Die jeweiligen digitalen Werte werden auf die jeweiligen Mittelwerte der jeweiligen Quanten gesetzt. So ergibt sich pro Quantum genau ein digitaler Wert auf den alle analogen Werte des Quantums abgebildet werden. Es gilt also für die Anzahl digitaler Werte N_d und die Anzahl Quanten N_Q:

$$N_Q = N_d$$

Die konstante Quantum-Größe Δs ist demnach:

$$\Delta s = \frac{s_{max} - s_{min}}{N_Q} = \frac{s_{max} - s_{min}}{N_d}$$

Die Wertemenge des digitalen Signals ist also:

$$M_{sd} = \left\{ sd_i \mid \left(sd_i = s_{min} + \frac{1}{2}\Delta s + i \cdot \Delta s \right) \wedge (i \in \mathbb{N}) \wedge (0 \leq i \leq N_d - 1) \right\}$$

Für die Menge der Schwellwerte ergibt sich:

$$M_{sw} = \{ sw_i \mid (sw_i = s_{min} + i \cdot \Delta s) \wedge (i \in \mathbb{N}) \wedge (1 \leq i \leq N_Q - 1) \}$$

Der maximale Quantisierungsfehler F_{Qmax} beträgt wiederum wegen der äquidistanten Quantisierung und der, wie angegeben, gesetzten Schwellwerte:

$$F_{Qmax} = \pm\frac{1}{2}\Delta s$$

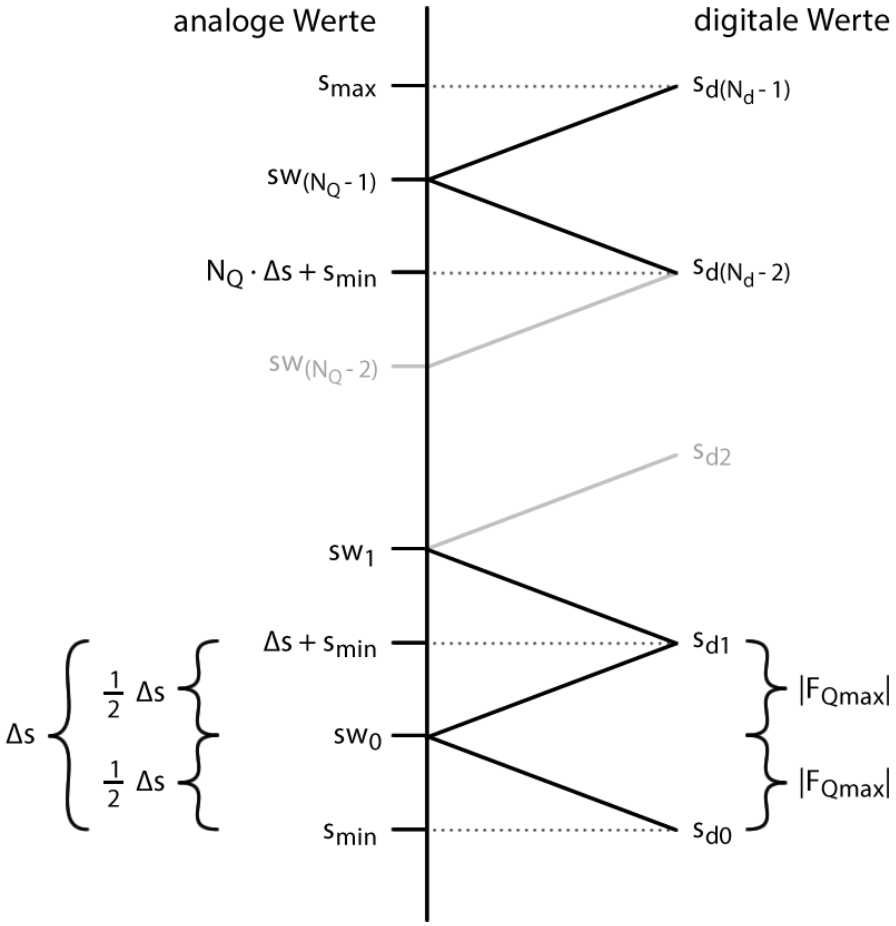

Abbildung 1.1: Analog-Digital-Wandlung mit vollständigem Signalumfang

Abbildung 4.1 zeigt schematisch die Abbildung der analogen Werte auf die entsprechenden digitalen Werte mittels der jeweiligen Schwellwerte sw_i. Beispielsweise werden alle analogen Werte zwischen s_{min} und sw_1 auf den digitalen Wert s_{d0} abgebildet. Alle analogen Werte zwischen sw_1 und sw_2 werden auf den digitalen Wert s_{d1} abgebildet.

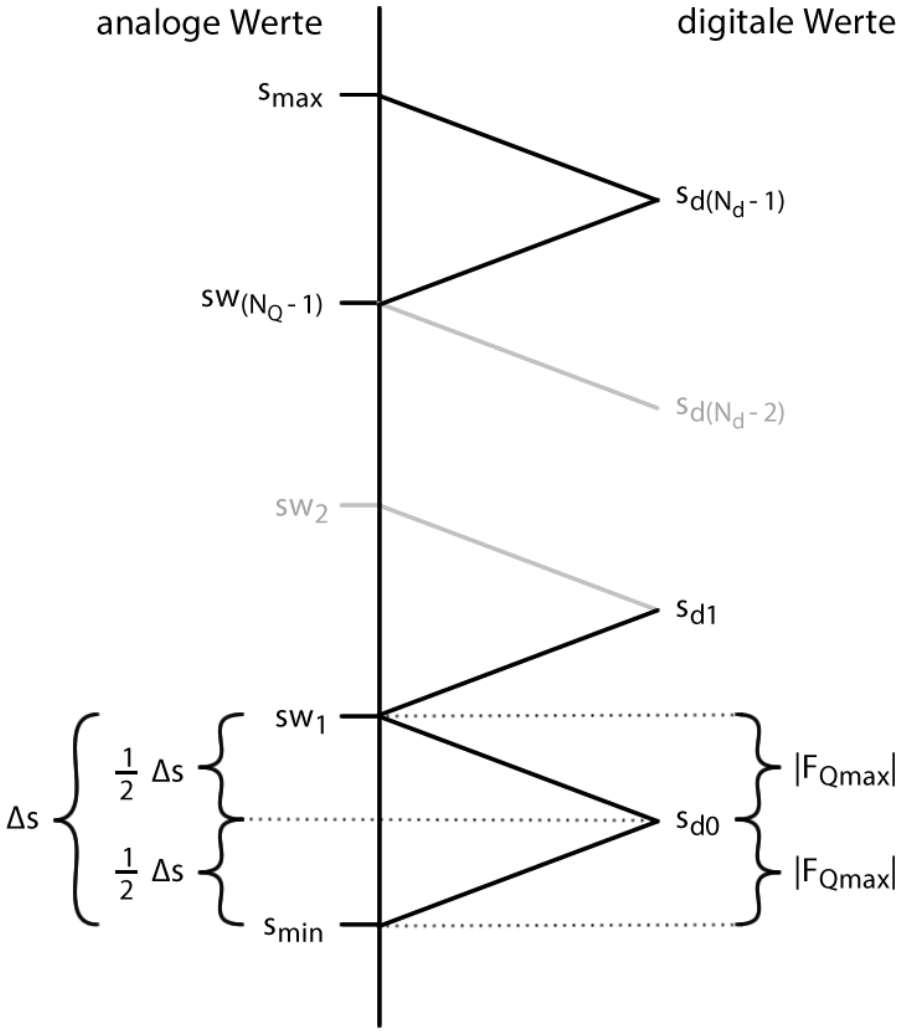

Abbildung 1.2: Analog-Digital-Wandlung mit unvollständigem Signalumfang

1.1.3 Der technische Unsicherheitsbereich

Wegen technisch ungenauer Messungen kann ein Wert eines analogen Signals zu hoch oder zu niedrig eingestuft werden. Dies führt zu einer weiteren Fehlerquelle, wenn ein Wert in unmittelbarer Nähe zu einem Schwellwert gemessen wird. Dementsprechend wird dieser Wert entweder einem zu hohen oder zu niedrigen digitalen Wert zugeordnet. Der Bereich um einen Schwellwert herum, in dem solche kritischen Werte gemessen wer-

den können, wird Unsicherheitsbereich δ genannt. Abbildung 1.3 veranschaulicht diesen Sachverhalt.

Für den tatsächlichen maximalen Fehler[4] F_{max} muss also die Hälfte des Unsicherheitsbereiches zum Quantisierungsfehler addiert werden. Es folgt:

$$F_{max} = F_{Qmax} \pm \frac{1}{2}\delta$$

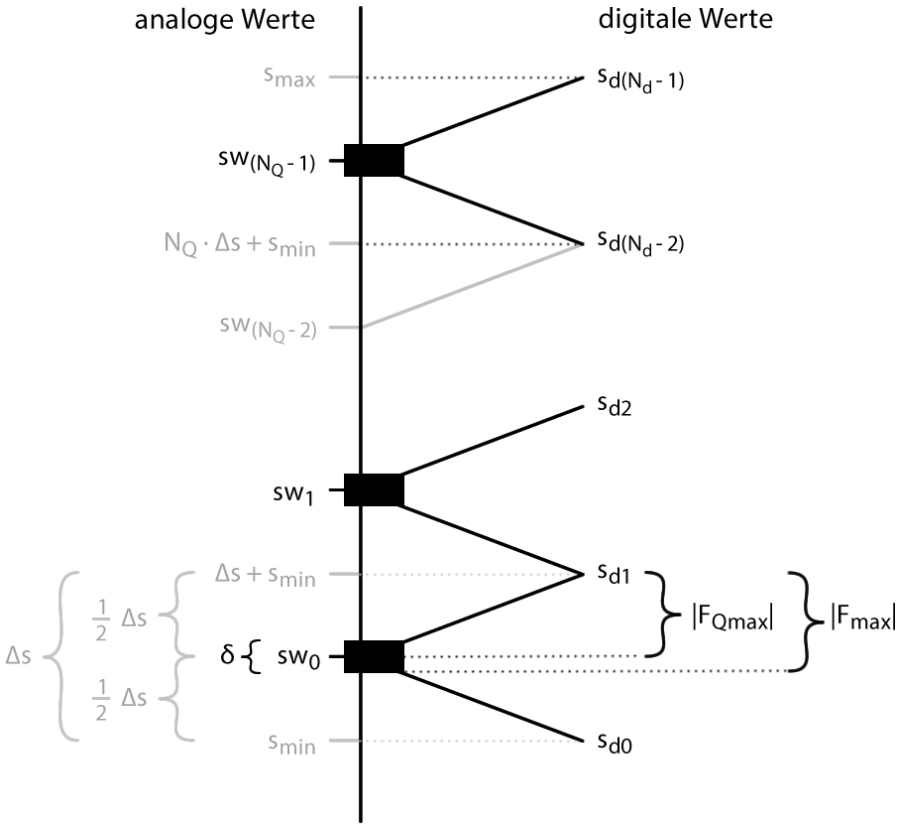

Abbildung 1.3: technischer Unsicherheitsbereich δ um die Schwellwerte

[4]Prof.Gemmar benennt den maximalen Fehler mit F'_{Qmax}.

2 Zahlensysteme

Im folgenden werden Polyadische Zahlensysteme besprochen. Solche Systeme haben die Eigenschaft Zahlen anhand einer Basis in Vielfachen von Potenzen dieser Basis aufzuteilen. Dies erleichtert Operationen auf Zahlen und lässt sich mittels der Potenzschreibweise auch leicht und relativ verständlich aufschreiben. Im Alltag benutzen wir das 10ner Polyadische Zahlensystem mit der Stellenschreibweise.

Ein Beispiel: Die Zahl 7645 ist in Stellenschreibweise im 10ner System geschrieben. Die Zahl hat die Wertigkeit 5 Einer, 4 Zehner, 6 Hunderter und 7 Tausender. Das selbe dargestellt als Addition von Vielfachen von Potenzen über die Basis 10:

$$7645 = 7 \cdot 10^3 + 6 \cdot 10^2 + 4 \cdot 10^1 + 4 \cdot 10^0$$

Gebrochene Zahlen können mittels Nachkommastellen dargestellt werden. Jede Nachkommastelle steht dann für die Potenz eines Anteils einer Einheit.

Ein Beispiel: Die Zahl $10,23$ ist in Stellenschreibweise im 10ner System geschrieben. Die Zahl hat die Wertigkeit ein Zehner, null einer, zwei zehntel und drei hundertstel. Das selbe dargestellt als Addition von Vielfachen von Potenzen über die Basis 10:

$$10,23 = 1 \cdot 10^1 + 0 \cdot 10^0 + 2 \cdot 10^{-1} + 3 \cdot 10^{-2} = 1 \cdot 10^1 + 0 \cdot 10^0 + 2 \cdot \frac{1}{10^1} + 3 \cdot \frac{1}{10^2}$$

Eine Ziffer steht also je nach Stelle, an der sie steht, für ein Vielfaches einer Potenz der Basis. Die Potenz wird durch die Stelle bestimmt an der die Ziffer steht. Das Vielfache der Potenz wird mit dem Symbol der Ziffer bestimmt. Da die Wertigkeit einer Potenz grundsätzlich in die jeweils nächste Stelle übertragen wird, gilt für den Wertebereich einer Ziffer z in Basis B:

$$M_z = \{z \in \mathbb{N} \mid 0 \leq z \leq B - 1\}$$

Im Allgemeinen kann die Wertigkeit einer ganzen Zahl Z mit n Vorkomma-, m Nachkommastellen und den Ziffern z_i in der Basis B folgendermaßen bestimmt werden:

$$\begin{aligned} Z_B &= \sum_{i=-m}^{n} B^i \\ &= z_{n-1} \cdot B^{n-1} + z_{n-2} \cdot B^{n-2} + ... + z_1 \cdot B^1 + z_0 \cdot B^0 + z_{-1} \cdot B^{-1} + ... + z_{-m} \cdot B^{-m} \end{aligned}$$

Über die Zählfunktion werden Arithmetische Operationen vereinfacht. Wird der Wertebereich einer Ziffer um eins überschritten wird die Ziffer auf 0 gesetzt und ein Übertrag[1] auf die nächste Stelle übernommen. Bei Unterschreiten des Wertebereichs um eins

Zählfunktion

[1] engl. carry

erfolgt eine Anleihe[2] auf die nächst höhere Stelle. Die Ausführung von arithmetischen Operationen lässt sich auf die Zählfunktion zurück führen. Die Berechnung für jede Stelle ist auf einen simplen Algorithmus zurückführbar und somit gut für eine automatische Ausführung geeignet.

Eine weitere Besonderheit ist die Operation der Stellenverschiebung. Eine Stellenverschiebung nach links in einem B-System bedeutet eine Division durch B der Zahl. Dies kann bei ganzen Zahlen entweder durch streichen der niedrigsten Stelle geschehen oder bei gebrochenen Zahlen durch verschieben des Kommas nach links. Eine Stellenverschiebung nach rechts bedeutet eine Multiplikation mit B der Zahl. Eine k-fache Verschiebung bedeutet eine Division bzw. Multiplikation um die $k$te Potenz der Basis. Beispiele:

Stellenverschiebung

- $1337_{10} \div 10^1 = 133_{10}$
- $32,1_{10} \cdot 10^2 = 3210_{10}$
- $10110_2 \div 2^1 = 1011_2$
- $111_2 \cdot 2^3 = 111000_2$

2.1 Stellen- und Ziffernaufwand

Der Stellen- und Ziffernaufwand einer Zahl ist von der gewählten Basis und der Größe der Zahl abhängig. Wieviele unterschiedliche Zahlen N bei angegebener Stellenzahl n und Basis B dargestellt werden können lässt sich folgendermaßen berechnen:

$$N = B^n$$

Ist gefragt wieviele Stellen n für N unterschiedliche Zahlen unter Verwendung der Basis B aufzuwenden sind, lässt sich die obige Gleichung folgendermaßen umstellen:

$$n = \lceil log_B(N) \rceil$$

Der Ziffernaufwand gibt an wieviele Zeichen zum Darstellen von einer bestimmten Anzahl unterschiedlicher Zahlen nötig sind. Für jede Stelle sind Anzahl Ziffern in Höhe der Basis notwendig. Der Ziffernaufwand zn_B für N unterschiedliche Zahlen in der Basis B wird also bestimmt mit:

$$zn_B = B \cdot n = B \cdot \lceil log_B(N) \rceil$$

2.2 Umwandlung von Zahlensystemen

Mit Hilfe des Hornerschemas lässt sich ein allgemeingültiger Allgorithmus zur Umwandlung eines beliebigen Zahlensystems in ein anderes herleiten. Das Hornerschema beschreibt das sukzessive Ausklammern eines Faktors. Dieses Verfahren lässt sich gut auf die Potenzschreibweise einer Zahl anwenden.

[2] engl. borrow

Beispielsweise lässt sich die Zahl 2.938 wie folgt zerlegen:

$$\begin{aligned} 2.938 &= 8 + 10 \cdot 3 + 10^2 \cdot 9 + 10^3 \cdot 2 \\ &= 8 + 10 \cdot (3 + 10 \cdot (9 + 10 \cdot 2)) \end{aligned}$$

Anschließend können die einzelnen Stellen durch Division mit der Basis aquiriert werden indem das jeweilige Ergebnis wiederum mit der Basis dividiert wird.

$$\begin{aligned} 8 + 10 \cdot (3 + 10 \cdot (9 + 10 \cdot 2)) \div 10 &= 3 + 10 \cdot (9 + 10 \cdot 2) && \text{Rest } 8 \\ 3 + 10 \cdot (9 + 10 \cdot 2) \div 10 &= 9 + 10 \cdot 2 && \text{Rest } 3 \\ 9 + 10 \cdot 2 \div 10 &= 2 && \text{Rest } 9 \\ 2 \div 10 &= 0 && \text{Rest } 2 \end{aligned}$$

Die Zahl 25_{10} lässt sich in Potenzschreibweise zur Basis 2 folgendermaßen aufschreiben:

$$\begin{aligned} 25_{10} &= 1 + 2 \cdot 0 + 2^2 \cdot 0 + 2^3 \cdot 1 + 2^4 \cdot 1 \\ &= 1 + 2 \cdot (0 + 2 \cdot (0 + 2 \cdot (1 + 2 \cdot 1))) \end{aligned}$$

Anschließend können wiederum die einzelnen Stellen für die Stellenschreibweise zur Basis 2 durch wiederholte Division mit der Basis 2 aquiriert werden.

$$\begin{aligned} 1 + 2 \cdot (0 + 2 \cdot (0 + 2 \cdot (1 + 2 \cdot 1+))) \div 2 &= 0 + 2 \cdot (0 + 2 \cdot (1 + 2 \cdot 1)) && \text{Rest } 1 \\ 0 + 2 \cdot (0 + 2 \cdot (1 + 2 \cdot 1)) \div 2 &= 0 + 2 \cdot (1 + 2 \cdot 1) && \text{Rest } 0 \\ 0 + 2 \cdot (1 + 2 \cdot 1) \div 2 &= 1 + 2 \cdot 1 && \text{Rest } 0 \\ 1 + 2 \cdot 1 \div 2 &= 1 && \text{Rest } 1 \\ 1 \div 2 &= 0 && \text{Rest } 1 \end{aligned}$$

Da durch das Dividieren mit der Basis immer die niedrigste Stelle des aktuellen Ergebnisses aquiriert wird, ist die letzte Stelle die höchste und muss somit als erstes im Ergebnis notiert werden usw. Das Ergebnis der Umwandlung von 25_{10} in das Dualsystem ist also 11001_2.

Allgemein kann folgender Algorithmus angewandt werden um eine ganze Zahl Z_B zur Basis B in die Basis B^* zu wandeln:

a) Teile Z_B durch B^*. Notiere das Ergebnis Z'_B und den Rest.

b) Wiederhole Schritt a mit Z'_B statt Z_B bis das Ergebnis aus Schritt a 0 ist.

c) Notiere die Zahl im B^*-System. Der Rest der letzten Division ist die höchste Stelle der Zahl im B^*-System. Der Rest der vorletzten Division ist die zweit-höchste Stelle der Zahl im B^*-System usw.

Für die Umwandlung echt-gebrochener Zahlen lässt sich ein ähnliches Verfahren anwenden. Hierbei muss die Zahl wiederholt mit der neuen Basis multipliziert werden. Nach jeder Multiplikation wird der Vorkommaanteil notiert. Mit dem Nachkommaanteil wird die Multiplikation fortgesetzt solange bis der Nachkommaanteil 0 ergibt oder der

gleiche Vorkommaanteil, wie zuvor, erreicht wird. Im letzten Fall ergibts sich daraus die Periode. Die Periode lässt sich ab der Stelle mit der Ziffer, die später erneut gefunden wurde, bis zu der Stelle mit der gleichen Ziffer genau feststellen, da sich ab dieser Stelle die vorhergehenden Rechnungen wiederholen würden.

Beispielsweise lässt sich die Zahl $0,375_{10}$ in Potenzschreibweise zur Basis 2 wie folgt zerlegen:

$$\begin{aligned} 0,375_{10} &= 2^{-1} \cdot 0 + 2^{-2} \cdot 1 + 2^{-3} \cdot 1 \\ &= 2^{-1} \cdot (0 + 2^{-1} \cdot (1 + 2^{-1} \cdot 1)) \end{aligned}$$

Die einzelnen Stellen zur Basis 2 erhält man, indem mit 2 multipliziert wird. Die Resultierende Vorkommastelle ist die erste Stelle. Der Nachkommaanteil wird erneut mit 2 multipliziert um die nächste Stelle zu erhalten.

$$\begin{aligned} 2^{-1} \cdot (0 + 2^{-1} \cdot (1 + 2^{-1} \cdot 1)) \cdot 2 &= \underbrace{0}_{Vorkommaanteil} + \underbrace{2^{-1} \cdot (1 + 2^{-1} \cdot 1)}_{Nachkommaanteil} \\ 2^{-1} \cdot (1 + 2^{-1} \cdot 1) \cdot 2 &= \underbrace{1}_{Vorkommaanteil} + \underbrace{2^{-1} \cdot 1}_{Nachkommaanteil} \\ 2^{-1} \cdot 1 \cdot 2 &= \underbrace{1}_{Vorkommaanteil} \end{aligned}$$

Da durch das Multiplizieren mit der Basis immer die höchste Stelle des aktuellen Ergebnisses aquiriert wird, ist die erste Stelle die höchste und muss somit als erstes im Ergebnis notiert werden usw. Das Ergebnis der Umwandlung von $0,375_{10}$ in das Dualsystem ist also $0,011_2$.

Ein Beispiel mit Periode: die Zahl $0,1_{10}$ soll in das Binärsystem gewandelt werden.

$$\left|\begin{array}{rcl} 0,1 \cdot 2 &=& 0,2 \\ 0,2 \cdot 2 &=& 0,4 \\ 0,4 \cdot 2 &=& 0,8 \\ 0,8 \cdot 2 &=& 1,6 \\ 0,6 \cdot 2 &=& 1,2 \end{array}\right|$$

$$0,2 \cdot 2 = 0,4$$

Das Ergebnis ist $0,1_{10} \mapsto 0,\overline{00011}_2$
Ein weiteres Beispiel: $0,2_{10} \mapsto 0,\overline{00111}_2$

Zahlen mit ganzem und gebrochenem Teil werden gewandelt in dem der ganze Teil mit dem Divisionsverfahren und der gebrochene Teil mit dem Multiplikationsverfahren gewandelt wird. Die Ergebnisse der beiden Teilschritte ergeben durch Komma getrennt das Gesamtergebnis.

Ein Beispiel: $19,625_{10}$ soll in das Dualzahlsystem gewandelt werden.

- Wandlung des ganzen Teils:

$$\begin{aligned} 19 \div 2 &= 9 \quad \text{Rest } 1 \\ 9 \div 2 &= 4 \quad \text{Rest } 1 \\ 4 \div 2 &= 2 \quad \text{Rest } 0 \\ 2 \div 2 &= 1 \quad \text{Rest } 0 \\ 1 \div 2 &= 0 \quad \text{Rest } 1 \end{aligned}$$

also ist der ganze Teil $19_{10} \mapsto 10011_2$

- Wandlung des gebrochenen Teils:

$$\begin{aligned} 0,625 \cdot 2 &= 1,25 \\ 0,25 \cdot 2 &= 0,5 \\ 0,5 \cdot 2 &= 1,0 \end{aligned}$$

also ist der gebrochene Teil $0,625_{10} \mapsto 0,101_2$

- Das Ergebnis ist demnach $19,625_{10} \mapsto 10011,101_2$

2.2.1 Zahlenwandlungen mit Basen der Form 2^i

Die fortgesetzte Division ist bei der Wandlung vom Dualsystem in ein anderes 2^i-System durch einfache Stellenverschiebung um i möglich. Der Rest sind dabei die Stellen, welche durch die Verschiebung wegfallen.

Beispielsweise lässt sich die Zahl 1010100110_2 durch Tetradenanordnung leicht in das Hexadezimalsystem[3] wandeln:

$$\begin{aligned} 10\ 1010\ 0110_2 \div 16 &= 10\ 1010 && \text{Rest } 0110_2 = 6_{16} \\ 10\ 1010_2 \div 16 &= 10 && \text{Rest } 1010_2 = A_{16} \\ 10_2 \div 16 &= 0 && \text{Rest } 10_2 = 2_{16} \end{aligned}$$

$$\underbrace{10}_{2}\ \underbrace{1010}_{A}\ \underbrace{0110}_{6}$$

$$10\ 1010\ 0110_2 \mapsto 2A6_{16}$$

Mit der Umkehrfunktion lässt sich die Hexadezimaldarstellung einer Zahl auch in die Dualdarstellung wandeln, indem jede Stelle in vier bits codiert wird. Allgemein kann von vom 2^i-System in das Binärsystem gewandelt werden, indem jeweils eine Stelle der Zahl im 2^i-System in i Stellen des Binärsystems gewandelt werden. Ein Beispiel:

$$\underbrace{A}_{1010}\ \underbrace{F}_{1111}\ \underbrace{F}_{1111}\ \underbrace{E}_{1110}$$

$$AFFE_{16} \mapsto 1010\ 1111\ 1111\ 1110_2$$

Weitere Beispiele:

$$1\ 00\ 10\ 00_2 \mapsto 1020_4$$

$$231_4 \mapsto 10\ 11\ 01_2$$

$$1\ 001\ 000_2 \mapsto 110_8$$

$$55_8 \mapsto 101\ 101_2$$

[3] Hexa = 6; dezi = 10; 16ner System; $16 = 2^4$

2.2.2 Umgang mit Perioden

Je nach Zahlensystem können Zahlen nur mit Periode in Stellenschreibweise geschrieben werden. Die Zahlen können in einen Bruch gewandelt werden. Im Folgenden werden die mathematischen Zusammenhänge erklärt.

Sei p eine periodische Zahl ohne Vorperiode in einem beliebigen Zahlensystem zur Basis B und n sei deren Periodenlänge. Sei o eine ganze Zahl ohne Periode und wie folgt definiert:

$$o = \underbrace{\underbrace{p \cdot B^n}_{\text{Periode als ganzzahliger Anteil}} - p}_{\text{Periode subtrahiert}}$$

o ist also die Darstellung der Periode als Ganzzahl. Durch Umformung gilt:

$$\begin{aligned} o &= p \cdot B^n - p \\ o &= (B^n - 1) \cdot p \\ p &= \frac{o}{B^n - 1} \end{aligned}$$

Beispiel: $0,\overline{1011}_2$ soll in einen Bruch zur Basis 10 gewandelt werden.

$$o = 1011_2; n = 4; B = 2$$

$$\begin{aligned} 0,\overline{1011}_2 &\mapsto \frac{1011_2}{2^4 - 1} \\ 0,\overline{1011}_2 &\mapsto \frac{11_{10}}{15_{10}} \end{aligned}$$

Bei einer Zahl mit Vorperiode der Länge k muss zuerst die Vorperiode durch Multiplizieren mit B^k eliminiert werden. Danach lässt sich der Periodische Anteil berechnen, wie oben angegeben. Das Resultat muss um die Operation auszugleichen mit $\frac{1}{B^k}$ multipliziert und die Vorperiode als Bruch addiert werden. Ein Beispiel: Wandlung der Zahl $0,01\overline{01001011}_2$ in einen Bruch im Dezimalsystem.

1. Eliminiere Vorperiode

$$0,01\overline{01001011}_2 = 0,01 + \frac{1}{2^2} \cdot 0,\overline{01001011}_2$$

2. Wandle periodischen Anteil in Bruch um

$$0,01\overline{01001011}_2 = 0,01_2 + \frac{1}{2^2} \cdot \frac{01001011_2}{2^8 - 1}$$

3. Berechne Ergebnis

$$0,01\overline{01001011}_2 = \frac{1}{4} + \frac{1}{4} \cdot \frac{75}{255} = \frac{11}{34}$$

Mit der Formel $p = \frac{o}{B^n - 1}$ lassen sich auch manche Brüche in Stellenschreibweise umformen. Zum Beispiel kann $\frac{1}{3}$ auch anders geschrieben werden als $\frac{1}{2^2-1}$. Mit Hilfe der Formel kann die Periodenlänge $n = 2$, die Periode als Ganzzahl $o = 1$ und die neue Basis $B = 2$ bestimmt werden. Also ist $\frac{1}{3} \mapsto 0,\overline{01}_2$.

2.3 Addition und Subtraktion im Dualsystem

Nach der Zählfunktion[4] gelten folgende Regeln:

$$\begin{aligned} 0+0 &= 0 \\ 0+1 &= 1 \\ 1+0 &= 1 \\ 1+1 &= 0+ \text{ carry in die nächsthöhere Stelle} \end{aligned}$$

$$\begin{aligned} 0-0 &= 0 \\ 0-1 &= 1- \text{ borrow von der nächsthöheren Stelle} \\ 1-0 &= 1 \\ 1-1 &= 0 \end{aligned}$$

Beim schriftlichen Addieren von Dualzahlen kann folgendes Verfahren angewandt werden:

1. Zähle die 1en in der ersten Spalte.
2. die erste Stelle der Anzahl 1en als Dualzahl wird in dieselbe Spalte geschrieben. Die nachfolgenden Stellen werden in die jeweils nachfolgenden Spalten als carry übernommen.
3. Wiederhole Schritt 1. mit den nachfolgenden Spalten.

Beispiel:

	1	0	0	1	1
+	0	1	0	1	1
				1	
			1		
	1	1	1	1	0

2.3.1 Subtraktion durch Komplementaddition

Das Basis-Komplement $K_B(Z)$ einer Zahl Z mit Anzahl Stellen n ist definiert mit:

$$K_B(Z) = B^n - Z$$

Das Basis-Komplement ist die Umkehrfunktion von sich selbst.

$$\begin{aligned} K_B(K_B(Z)) &= K(B^n - Z) \\ &= B^n - (B^n - Z) \\ K_B(K_B(Z)) &= Z \end{aligned}$$

[4] Siehe Abschnitt 2 unter Zählfunktion

Das Stellenkomplement[5] $K'_B(Z)$ ist definiert über die Differenz jeder Ziffer an jeder Stelle zur höchstwertigen Ziffer. Es gilt also für jede Ziffer z_i an Stelle i:

$$K'_B(Z) : z_i \mapsto ((B-1) - z_i)$$

Das Basis-Komplement lässt sich leicht berechnen indem das Stellenkomplement um eins inkrementiert wird. Beispiel:

$$K'_{10}(123) = 876$$

$$K_{10}(123) = 1000 - 123 = 876 + 1 = 877$$

Werden negative Zahlen mit dem Basis-Komplement dargestellt, kann die Subtraktion zweier Zahlen über die Addition der beiden Zahlen abgearbeitet werden. Dies gilt weil:

$$a - b = a + K(b) - B^n = a - b + B^n - B^n$$

Das Subtrahieren von B^n kann methodisch je nach Fall behandelt werden. Ist der Minuend größer oder gleich dem Subtrahenden wird die führende eins nach der Berechnung gestrichen. Wichtig bei dieser Methode ist, dass Minuend und Subtrahend die gleiche Stellenanzahl besitzen. Die höheren Stellen des Subtrahenden müssen gegebenenfalls mit Nullen aufgefüllt werden und über diese Zahl das Komplement gebildet werden. Beispiel: $M \geq S$

$$\begin{aligned} 8127_{10} - 342_{10} &= 8127 - 0342 \\ &= 8127 + K_{10}(0342) - 10000 \\ &= 8127 + 10000 - 0342 - 10000 \\ &= 8127 + 9658 - 10000 \end{aligned}$$

Addition von 8127 und 9658. Die Subtraktion von 10000 erfolgt über das Streichen der führenden 10000der-Stelle.

	8	1	2	7
+	9	6	5	8
			1	
1				
	7	7	8	5

Ist der Minuend kleiner als der Subtrahend steht das Ergebnis im negativen Basis-Komplement. Das Ergebnis kann also über das Komplement des Zwischenergebnisses $M < S$

[5] Das Stellenkomplement wird auch Einerkomplement genannt.

berechnet werden. Beispiel:

$$\begin{aligned} 342_{10} - 8127_{10} &= 342 + K_{10}(8127) - 10000 \\ &= 342 + 10000 - 8127 - 10000 \\ &= 342 + 1873 - 10000 \\ &= 2215 - 10000 \\ &= -(10000 - 2215) \\ &= -K_{10}(2215) \\ &= -7785 \end{aligned}$$

Die angegbenen Methoden lassen sich auf jedes Basis-System anwenden.
Beispiel: $65_{10} - 23_{10} = 42_{10} \mapsto 1000001_2 - 10111_2 = 101010_2$

dezimal		dual	
	65		100 0001
$K_{10}(23)$	$+77$	$K_2(0010111)$	$+110\ 1001$
	*1*42		*1*010 1010
	+42		**+10 1010**

Beispiel: $12_{10} - 74_{10} = -62_{10} \mapsto 1100_2 - 1001010_2 = -111110_2$

dezimal		dual	
	12		000 1100
$K_{10}(74)$	$+26$	$K_2(1001010)$	$+011\ 0110$
	*0*38		*0*100 0010
$-K_{10}(38)$	**-62**	$-K_2(1000010)$	**-11 1110**

3 Darstellung und Arithmetik rationaler Zahlen

In der Praxis gibt es in digitalen automatischen Rechenanlagen zwei Darstellungsarten für rationale Zahlen: das Festkommaformat und das Fließkommaformat. Den folgenden zwei Abschnitten wird näher auf die einzelnen Formate eingegangen.

3.1 Festkommma-Arithmetik

Die Festkommadarstellung wird verwendet um Zahlen Z im Wertebereich $W_F = \{Z \in \mathbb{Q} \mid -1 < Z < 1\}$ abzubilden. Das höchstwertige Bit wird als Vorzeichenbit VZ verwendet und gibt das Vorzeichen der Zahl an. Die übrigen Bits werden als Mantissenbits verwendet. Die Mantisse gibt alle Nachkommastellen an. Die Vorkommastelle ist immer null und muss deshalb nicht codiert werden. Abbildung 3.1 zeigt Schematisch die Festkommadarstellung.

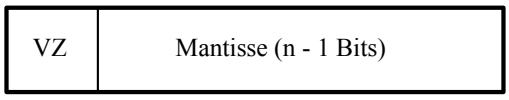

Abbildung 3.1: Festkommadarstellung einer Zahl

Der numerisch kleinste Abstand zwischen zweier Zahlen ist konstant $\Delta Z = 2^{n-1}$. Dies ist ein äquidistantes Zahlenformat.

Der größte Zahlenwert Z_{max} beträgt:

$$\begin{aligned} Z_{max} &= 0,111...1 \\ &= \sum_{i=1}^{n-1} 2^{-i} \\ &= 1 - 2^{-(n-1)} \end{aligned}$$

Der kleinste Zahlenwert Z_{min} beträgt:

$$\begin{aligned} Z_{min} &= -0,111...1 \\ &= -\sum_{i=1}^{n-1} 2^{-i} \\ &= -1 + 2^{-(n-1)} \end{aligned}$$

3.2 Fließkomma-Arithmetik

Bei der Fließkommadarstellung wird zusätzlich ein Exponent zur Mantisse gespeichert. Dem Exponenten wird ein konstanter Wert implizit hinzu addiert um auf ein Vorzeichenbit für den Exponenten verzichten zu können. Die so entstandene Zahl wird Charakteristik einer Fließkommazahl genannt. Abbildung 3.2 zeigt schematisch die Fließkommadarstellung.

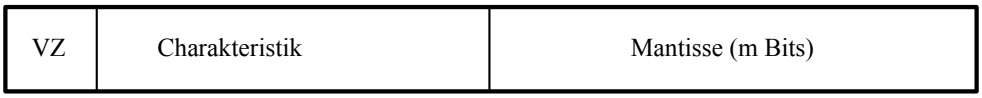

Abbildung 3.2: Fließkommadarstellung einer Zahl

Der Wert einer dualen Fließkommazahl Z mit Mantisse M, Charakteristik C mit konstantem Teil k und Vorzeichen VZ lässt sich also berechnen mit:

$$Z = VZ \cdot M \cdot 2^{C-k}$$

Fließkommazahlen werden normalisiert. Das heißt der Exponent wird so gewählt, dass die Mantisse genau eine Vorkommastelle mit einer Wertigkeit größer null hat. Bei Dualzahlen im Fließkommaformat ist die Vorkommastelle deshalb immer eins und muss nicht gespeichert werden. Eine Ausnahme ist hier die Null. Null hat auch als Vorkommastelle die Ziffer 0 und die Nachkommastellen sind bekanntlicher Weise ebenfalls Null. Deshalb wird die Konvention eingeführt, dass wenn alle Ziffern der dualen Fließkommazahl null sind, der Wert Null gemeint ist.

Beispiel: $+0,010100111_2$ im IEEE 754 32bit float Format:

VZ	Charakteristik (8bit; $k = 2^{8-1} - 1$)	Mantisse (23bit)
+	E + k = -2 + 127 = 125	1,0100 111
0	0111 1101	0100 1110 0000 0000 0000 000

Vor der Addition oder Subtraktion zweier Fließkommazahlen müssen diese erst denormalisiert werden. Der niedrigere Exponent wird immer auf den höherwertigen aufgewertet und die Mantisse um eben so viele Stellen nach rechts verschoben. Durch dieses Verfahren können zwangsläufig Rechenfehler entstehen, da im Computer bei der Denormalisierung bzw. Normalisierung signifikante Stellen der Mantisse verloren gehen können. Beispiele:

$$\begin{aligned} 1,0101E3 + 1,1110111E0 &= 1,0101E3 + 0,0011110111E3 \\ &= (1,0101 + 0,0011110111)E3 \\ &= 1,100011011E3 \end{aligned}$$

$$
\begin{aligned}
1,1101E2 - 1,01E1 &= 1,1101E2 - 0,101E2 \\
&= (1,1101 - 0,101)E2 \\
&= 1,0011E2
\end{aligned}
$$

Nach einer Multiplikation oder einer Division muss das Ergebnis häufig normalisiert werden. Dabei können wiederum zwangsläufig Rechenfehlehler entstehen.

4 Codes

Nachrichten werden in Form von Zeichenketten dargestellt. Zeichenketten wiederum bestehen aus einzelnen Zeichen. Die Zeichen, die für bestimmte Nachrichten verwendet werden, werden im jeweiligen Alphabet[1] festgelegt. Welche Zeichenketten ein Wort einer Nachricht bilden und welche nicht wird in den Produktionsregeln festgelegt.

Ein Wort kann über Konkatenation[2] von Zeichen gebildet werden. Das Symbol der Konkatenation ist „$\circ$“. Wörter einer bestimmten Länge über ein Alphabet werden in Wortmengen zusammen gefasst. Hierzu wird das Alphabetssymbol mit der im Exponenten angegebenen Wortlänge angegeben. Beispiel:
Es sei das Alphabet $A = \{0, 1\}$.
Die Zeichen 0 und 1 sind Elemente von Alphabet A. Kurz: $0, 1 \in A$
Ein Wort über A ist $w = 0 \circ 1 \circ 1 \circ 0 = 0110$
w ist Element von A^4. Kurz: $w \in A^4$

In der Praxis ist das zu verarbeitende Alphabet oft länger als das technisch verfügbare Alphabet. Der Computer kennt auf unterster Ebene nur zwei unterschiedliche Zeichen. Dem Computer steht also ein binäres Alphabet zur Verfügung. Um dennoch umfangreichere Alphabete und Nachrichten, die über dieses gebildet werden, verarbeiten zu können, müssen die Zeichen des Ursprungsalphabets auf Wörter des kleineren Alphabets abgebildet werden. Dieser Umsetzungsprozess wird Codierung genannt. Ein Code c ist also aus dieser Sicht eine eindeutige Zuordnung zwischen Zeichen aus einem Urspungsalphabet U auf Wörter eines anderen Alphabets A.

$$c : U \mapsto A^m$$

$$c : u_i \mapsto a_1 a_2 ... a_m \text{ mit } u_i \in U \wedge a_i \in A$$

Mit einem Alphabet A und m vielen Stellen können maximal $|A|^m$ viele Zeichen codiert werden. Um den minimalen Stellenaufwand S_{min} für n unterschiedliche Zeichen zu bestimmen gilt demnach:

$$S_{min} = \lceil log_{|A|}(n) \rceil$$

Im Folgenden wird beispielhaft berechnet wieviele Bits mindestens nötig sind um die Dezimalziffern $\{0, 1, 2, ..., 9\}$ zu codieren.

$$S_{min} = \lceil ld(10) \rceil = 4$$

Demnach werden mindestens vier bit benötigt um zehn Zeichen über das Binäralphabet abzubilden. Mit vier bit wären allerdings $2^4 = 16$ unterschiedliche Konkatenationen von

[1] Ein Alphabet wird auch Zeichenvorrat genannt.
[2] Konkatenation ist die Aneinanderreihung von Zeichen

Zeichen möglich. Um irgendeine Zuordnung von Binärworten auf die zehn Ziffern zu definieren gibt es

$$\frac{16!}{(16-10)!} \approx 2,9 \cdot 10^{10} \text{ Möglichkeiten}$$

Tatsächlich Verwendung finden nur wenige. Welche Vorteile welche Abbildung bringt, wird in den nachfolgenden Abschnitten besprochen.

4.1 Beschreibungsmerkmale für Codes

Grundsätzlich wird zwischen Alphanumerischen Codes und Numerischen Codes unterschieden. Alphanumerische Codes dienen der Darstellung von Ziffern und anderen Zeichen, wie Buchstaben, Trenn und Sonderzeichen und weitere. Numerische Codes dienen der Darstellung von Zahlen. Numerische Codes lassen sich in Wortcodes und Zifferncodes unterteilen. Bei Zifferncodes wird, wie bei Alphanumerischen Codes, jede Ziffer einzeln codiert. In Wortcodes wird dagegen eine Zahl als Ganzes codiert. Ein Beispiel dafür ist der Dualzahlencode. Der Nachteil dabei ist der Kodieraufwand bei der Wandlung bei der Ein-, Ausgabe. Der Vorteil dabei ist, dass arithmetische Operationen oft direkt möglich sind.

Allgemeine Beschreibungsmerkmale für Codes sind:

- Die **Bewertbarkeit** W**:** Für jede Stelle i in einem Codewort wird eine Stellenwertigkeit w_i definiert. Der Dualzahlencode ist zum Beispiel bewertet.
- Das **Gewicht** G ist die Anzahl der mit 1en belegten Stellen in einem Codewort. Bei einem gleichgewichteten Code haben alle Codewörter das gleiche Gewicht.
- Die **Distanz** D zweier Codewörter ist die Anzahl stellen in denen sich die Codewörter unterscheiden.
- Die **Hamming Distanz** HD ist die minimale Stellendistanz zwischen beliebigen Codewörtern eines Codes.

  $$HD(C) = min(D(cw_a, cw_b)) \text{ mit } cw_a, cw_b \in C \wedge cw_a \neq cw_b$$

- Die **Stetigkeit**: ein Code ist stetig, wenn die Distanz zwischen benachbarten Codewörtern im gesamten Code gleich ist.

  $$C \text{ ist stetig} \Leftrightarrow D(cw_i, cw_{i+1}) = D(cw_j, cw_{j+1})$$

- Die **Redundanz**: ein Code ist redundant, wenn mögliche Kombinationen von Zeichen nicht als Codewörter genutzt werden.
- Die **absolute Redundanz**[3] R_a ist eine Kenngröße eines Codes. Die absolute Redundanz wird über die Anzahl der Kombinationsmöglichkeiten N und die Anzahl

[3] Differenz zwischen Anzahl benötigten Bits für alle Kombinationen und Anzahl verwendeten Bits

Codewörter N_{cw} berechnet. Die Anzahl Kombinationsmöglichkeiten hängt von der Wortlänge len_{cw} des Codes ab. Es gilt:

$$N = 2^{len_{cw}}$$

$$R_a = ld(N) - ld(N_{cw}) \text{ [bit]}$$

Beispiel: Redundanz des BCD-Codes für die Ziffern 0 bis 9

$$N = 2^4$$

$$N_{cw} = 10$$

$$R_a = ld(2^4) - ld(10) \approx 4 - 3,32 = 0,68Bit$$

- Die **relative Redundanz**[4] R_r ist eine Kenngröße eines Codes. Die relative Redundanz wird über die Anzahl der Kombinationsmöglichkeiten N und die absolute Redundanz berechnet. Es gilt:

$$R_r = \frac{R_a}{ld(N)}$$

- Die **Vollständigkeit:** Ein Code wird vollständig genannt wenn die absolute Redundanz null ist. Ist die absolute Redundanz größer null, wird der Code als unvollständig bezeichnet.

$$C \text{ ist vollständig} \Leftrightarrow R_a = 0$$

$$C \text{ ist unvollständig} \Leftrightarrow R_a > 0$$

- **binär-reflektierend:** Ein Code ist binär-reflektierend, wenn die Codewörter aus der Hälfte des Codes gleich der Codewörter der anderen Hälfte des Codes sind mit Ausnahme der vordersten Stelle, die mit jeweils 0 bzw. 1 codiert ist.

- **selbstkomplementierend / selbstinvertierend:** Ein Code ist selbstkomplementierend, wenn das Neuenerkomplement des Dezimalwertes identisch mit dem Einerkomplement des jeweiligen Codewortes des Dezimalwertes ist.

- **prüfbar bzw. gesichert:** Ein Code ist prüfbar, wenn eine Fehlererkennung möglich ist. Dies ist nur mit redundanten Codes möglich. Ausführlich wird dies im Abschnitt 4.3 besprochen.

- **fehlerkorrigierbar:** Ein Code ist fehlerkorrigierbar, wenn eine Fehlererkennung möglich ist und Fehler korrigiert werden können. Dies ist nur mit redundanten Codes möglich. Ausführlich wird dies im Abschnitt 4.3 besprochen.

[4] Verhältnis zwischen nicht benutzten bits zu insgesamt verwendeten bits

4.2 Beispiele für bekannte Ziffern-Codes

Bewertete Ziffern-Codes

8-4-2-1 Code mit den Eigenschaften:

- stetig / einschrittig
- bewertet 8 - 4 - 2 - 1

Dezimalziffer	8-4-2-1 Code
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

Aiken-Code mit den Eigenschaften:

- negationssymmetrisch
- bewertet 2 - 4 - 2 - 1
- selbstkomplementierend

Dezimalziffer	Aiken-Code
0	0000
1	0001
2	0010
3	0011
4	0100
5	1011
6	1100
7	1101
8	1110
9	1111

Excess-3-Code / Stibitz-Code mit den Eigenschaften:

- negationssymmetrisch
- vermeidet 0000 oder 1111
- aus Dualcode mit Addition von 3
- bewertet 8 - (-4) - $\overline{(-2)}$ - $\overline{(-1)}$

Dezimalziffer	Stibitz-Code
0	0011
1	0100
2	0101
3	0110
4	0111
5	1000
6	1001
7	1010
8	1011
9	1100

2 aus 5 mit den Eigenschaften:

- mit gerader Parität
- gleichgewichtig mit $G = 2$
- bewertet 7 - 4 - 2 - 1 - 0

Dezimalziffer	2 aus 5
0	11000 $(= 11_{10})$
1	00011
2	00101
3	00110
4	01001
5	01010
6	01100
7	10001
8	10010
9	10100

Nicht bewertbare Zifferncodes

Gray-Code mit den Eigenschaften:

- binär-reflektierend[a]
- einschrittig
- vollständig

Dezimalziffer	Gray-Code
0	0000
1	0001
2	0011
3	0010
4	0110
5	0111
6	0101
7	0100
8	1100
9	1101
10	1111
11	1110
12	1010
13	1011
14	1001
15	1000

[a] Achsen-Spiegelung der Codewörter

O'Brien mit den Eigenschaften:

- binär-reflektierend[a]
- einschrittig

Dezimalziffer	Gray-Code
0	0000
1	0001
2	0011
3	0010
4	0110
5	1110
6	1010
7	1011
8	1001
9	1000

[a] Achsen-Spiegelung der Codewörter

4.3 Gesicherte Codes

Gesicherte Codes haben die Eigenschaft Fehler bei der Übertragung von Codewörtern zu erkennen. Der Empfänger kann in bestimmten Fällen den Fehler erkennen indem eine Kombination aus $\{0,1\}^n$ empfangen wird, welche kein gültiges Codewort darstellt. Mit bestimmten Verfahren können Fehler teilweise korrigiert werden. Codes können somit in die Klassen nicht fehlererkennbarer, fehlererkennbarer oder fehlerkorrigierbarer Codes eingeordnet werden.

Damit ein Code fehlererkennbar sein kann, muss mindestens ein Bit mehr verwendet werden als nötig. Also ist die notwendige Bedingung für fehlererkennbare Codes, dass die absolute Redundanz größer als eins ist.

$$\text{Code ist fehlererkennbar } \Rightarrow R_a \geq 1 \text{ Bit}$$

Damit ein Code tatsächlich fehlererkennbar ist, muss die Hamming-Distanz größer als eins sein. Die Hinreichende Bedingung ist also $HD(C) > 1$. Es können $HD(C) - 1$ viele

Fehler erkannt werden. Um n_F viele Fehler zu erkennen muss der Code also folgende Hamming-Distanz aufweisen:

$$HD_{Fe}(C) = n_F + 1$$

Für die Fehlerkennung haben sich die Quersummenprüfung und gleichgewichtige Codes etabliert. Bei der Quersummenprüfung wird jedes Codewort um ein Parity Bit ergänzt. Das Parity Bit sorgt bei gerader Parität dafür, dass eine gerade Anzahl 1en im jeweiligen Codewort vorkommt. Bei ungerader Parität sorgt das Parity Bit dafür, dass eine ungerade Anzahl 1en im jeweiligen Codewort vorkommt. Damit sind Einzelfehler erkennbar. Doppelfehler werden nicht erkannt, da sie sich gegenseitig aufheben.

Um n_F viele Fehler korrigieren zu können wird mindestens folgende Hamming-Distanz benötigt:

$$HD_{Fk}(C) = 2 \cdot n_F + 1$$

4.3.1 Blocksicherungsverfahren

Mit Blocksicherungsverfahren ist die blockweise Korrektur von Einfachfehlern pro Block möglich. Doppelfehler werden erkannt, können aber nicht korrigiert werden. Ein Block besteht bei Codewortlänge n_{cw} aus $n_{cw} + 1$ zusätzlichen Parity Bits und einem zusätzlichen Prüfwort der Länge $n_{cw} + 1$. Es kann auf gerade oder ungerade Parität geprüft werden. Bei einem Einfachfehler im Block kann über die fehlerhafte Zeile und fehlerhafte Spalte der Fehler genau lokalisiert und damit korrigiert werden. Beispiele mit Codewörtern der Länge drei und Prüfung auf gerader Parität:

	P	$\mathbf{c_2}$	$\mathbf{c_1}$	$\mathbf{c_0}$	
cw_0	0	1	0	1	✓
cw_1	1	0	1	0	✓
cw_2	1	1	0	0	✓
pw	0	0	1	1	✓
	✓	✓	✓	✓	

Prüfung erfolgreich abgeschlossen; kein Fehler

	P	$\mathbf{c_2}$	$\mathbf{c_1}$	$\mathbf{c_0}$	
cw_0	0	1	0	1	✓
cw_1	1	0	1	0	✓
cw_2	1	1	**1**	0	f
pw	0	0	1	1	✓
	✓	✓	f	✓	

Prüfung erkennt einen Fehler; Fehler ist korrigierbar

	P	$\mathbf{c_2}$	$\mathbf{c_1}$	$\mathbf{c_0}$	
cw_0	0	1	0	1	✓
cw_1	1	0	1	0	✓
cw_2	1	1	1	1	✓
pw	0	0	1	1	✓
	✓	✓	f	f	

Prüfung erkennt zwei Fehler; Fehler sind nicht korrigierbar

	P	$\mathbf{c_2}$	$\mathbf{c_1}$	$\mathbf{c_0}$	
cw_0	0	1	0	1	✓
cw_1	1	0	0	0	f
cw_2	1	1	1	0	f
pw	0	0	1	1	✓
	✓	✓	✓	✓	

Prüfung erkennt zwei Fehler; Fehler sind nicht korrigierbar

	P	$\mathbf{c_2}$	$\mathbf{c_1}$	$\mathbf{c_0}$	
cw_0	0	1	0	1	✓
cw_1	1	0	1	1	f
cw_2	1	1	1	0	f
pw	0	0	1	1	✓
	✓	✓	f	f	

Prüfung erkennt zwei Fehler; Fehler sind nicht korrigierbar

	P	$\mathbf{c_2}$	$\mathbf{c_1}$	$\mathbf{c_0}$	
cw_0	0	0	0	1	f
cw_1	1	0	1	0	✓
cw_2	1	1	0	1	f
pw	0	0	1	1	✓
	✓	f	✓	f	

Prüfung erkennt zwei Fehler; Fehler sind nicht korrigierbar

4.3.2 Systematische Hamming-Codes

Das Verfahren um Hamming-Codes zu erstellen ist vom Blocksicherungsverfahren abgeleitet. Ein einzelnes Codewort ist dabei als Block zu betrachten und zu prüfen. Ein Codewort mit n Stellen setzt sich aus n_p vielen Paritätsbits und n_i vielen Informationsbits zusammen.

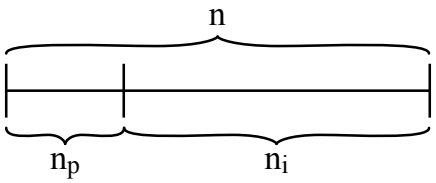

Abbildung 4.1: Schematische Darstellung eines Hamming-Codeworts

Um einen Fehler korrigieren zu können muss jede Stelle auf einen Fehler überprüft werden und es muss ein gültiges Codewort existieren. Somit müssen $n+1$ Fälle erfassen werden. Mit n_p vielen Paritätsbits können 2^{n_p} viele Fälle codiert werden. Damit alle Fälle abgedeckt werden können muss die folgende Ungleichung erfüllt sein:

$$\begin{aligned} 2^{n_p} &\geq n+1 \\ 2^{n_p} &\geq n_p + n_i + 1 \end{aligned}$$

Beispiel: um vier Informationsbits zu sichern werden mindestens drei Paritätsbits benötigt.

$$\begin{aligned} 2^3 &\geq 3+4+1 \\ 8 &\geq 8 \end{aligned}$$

Für die Einfehlerkorrektur bei Blockcodes muss jedes Informationsbit durch mindestens zwei Paritätsbits gesichert werden. Für ein Informationsbit i, dessen zwei Paritätsbits p_a und p_b und deren Paritäten $P(i,p_a)$ und $P(i,p_b)$ kann dann folgende Fallunterscheidung vorgenommen werden:

- $P(i,p_a)$ und $P(i,p_b)$ sind korrekt. Demnach sind p_a, p_b und i korrekt.
- $P(i,p_a)$ und $P(i,p_b)$ sind falsch. Demnach ist i falsch.

- $P(i, p_a)$ ist falsch. $P(i, p_b)$ ist korrekt. Demnach ist p_a falsch.
- $P(i, p_a)$ ist korrekt. $P(i, p_b)$ ist falsch. Demnach ist p_b falsch.

Diese Regel schließt nicht aus, dass ein Paritätsbit für mehrere Informationsbits zur Prüfung der Parität verwendet werden kann. Die Parität wird in diesem Fall nicht über zwei Bits gebildet, sondern über das jeweilige Paritätsbit inklusive der ihm zugeordneten Informationsbits. Damit eine eindeutige Fehlerlokalisierung möglich ist muss auf eine entsprechend günstige Verteilung der Informationsbits auf die Paritätsbits geachtet werden. Wenn über die Fallunterscheidung der Paritäten einem Fall genau ein Fehler zugeordnet werden kann, kann dieser Fehler entsprechend korrigiert werden.

Ein Praktisches Schema zum Finden einer optimalen Verteilung wird im folgenden Abschnitt mit Hilfe eines Beispiels erläutert.

Ein Code mit vier Informationsbits soll gesichert werden. Dazu sind drei Paritätsbits notwendig[5]. Jedes Codewort umfasst also sieben Stellen.

1.	die Stellen werden von 1 bis n durchnummeriert.	1	2	3	4	5	6	7
2.	die Nummerierung wird dual codiert.	001	010	011	100	101	110	111
3.	das j-te Paritätsbit wird an der Stelle eingetragen, die im Dualcode an der j-ten Stelle eine 1 trägt. Die Informationsbits füllen der Reihe nach die übrigen Stellen.	p_1	p_2	i_1	p_3	i_2	i_3	i_4
4.	Dem j-ten Paritätsbit werden diejenigen Informationsbits zugeordnet deren Stelle im Dualcode an der j-ten Stelle eine 1 trägt.	p_1 prüft $\{i_1, i_2, i_4\}$	p_2 prüft $\{i_1, i_3, i_4\}$		p_3 prüft $\{i_2, i_3, i_4\}$			
5.	Die Codewörter werden erstellt. Im Beispiel wird das Codewort für 1010 angegeben. Es wird auf gerade Parität geprüft.	1	0	1	1	0	1	0

Dieses Schema bietet mehrere Vorteile:

- Jedes Informationsbit wird durch mindestens zwei Paritäten geprüft.

[5] siehe oben

- Jedes Paritätsbit wird durch eine Parität geprüft.
- Es entsteht eine optimale Zuordnung der Paritätsbits auf die Informationsbits.
- Fehlercode: Wegen der Anordnung der Paritätsbits wird beim Prüfen der Paritäten die Fehlerstelle direkt über die fehlerhaften Paritäten dual codiert angezeigt. Wenn zum Beispiel die Parität über dem Paritätsbit p_2 fehlerhaft ist, alle anderen jedoch richtig, dann ist das fehlerhafte Bit das zweite. Wenn die Paritäten über p_1 und p_2 fehlerhaft ist, dann ist das dritte Bit fehlerhaft usw.

5 Boolesche Algebra

In diesem Artikel wird die Boolesche Algebra stark in Verbindung mit der Schaltalgebra behandelt. Deshalb werden die Zeichen 0 und 1 für die binäre Grundmenge $B = \{0, 1\}$ verwendet. Weitere Unterschiede in der Notation ergeben sich bei der Besprechung der Operatoren.

5.1 Operatoren

In den folgenden Abschnitten werden die in diesem Artikel verwendeten Operatoren besprochen. Bei booleschen Operatoren kann eine endliche Wertetabelle angegeben werden, da die Grundmenge B endlich ist und die Operatoren endlich viele Elemente behandeln.

5.1.1 Kartesisches Produkt „×“

Das Kartesische Produkt wird über Mengen gebildet. Dabei werden alle Elemente der ersten Menge mit allen Elementen der zweiten Menge in einzelnen Tupeln zusammengefasst.

$$B \times B = (0,0); (0,1); (1,0); (1,1)$$

Das n-fache Produkt einer Menge bezeichnet alle n-fachen Kombinationen aus allen Elementen der Menge.

$$B \times B \times B ... \times B = B^n$$

5.1.2 Negation „NICHT“ „NOT“ „ $\overline{a}$ “

Die Negation ist eine einstellige Operation, welche den Wert invertiert.

$$f_{NOT} : B \mapsto B$$

$$f_{NOT}(a) = \overline{a} \text{ mit } a \in B$$

a	$\overline{a}$
0	1
1	0

5.1.3 Konjunktion „UND“ „AND“ „ · “

Die Konjunktion ist 1, wenn beide behandelten Werte 1 sind, ansonsten 0.

$$f_{OR} : B \times B \mapsto B$$

$$f_{OR}(a, b) = a \cdot b \text{ mit } a, b \in B$$

a	b	$a \cdot b$
0	0	0
0	1	0
1	0	0
1	1	1

5.1.4 Disjunktion „ODER“ „OR“ „ + “

Die Disjunktion ist 1, wenn einer der behandelten Werte 1 ist, ansonsten 0.

$$f_{OR} : B \times B \mapsto B$$

$$f_{OR}(a, b) = a + b \text{ mit } a, b \in B$$

a	b	$a + b$
0	0	0
0	1	1
1	0	1
1	1	1

5.1.5 Äquivalenz „ ↔ “

Die Äquivalenz ist 1, wenn beide behandelten Werte gleich sind, ansonsten 0.

$$f_{eq} : B \times B \mapsto B$$

$$f_{eq}(a, b) = a \leftrightarrow b \text{ mit } a, b \in B$$

a	b	$a \leftrightarrow b$
0	0	1
0	1	0
1	0	0
1	1	1

5.1.6 Antivalenz „XOR“ „ $\nleftrightarrow$ “ „ $\oplus$ “

Die Antivalenz ist 1, wenn beide behandelten Werte ungleich sind, ansonsten 0.

$$f_{an} : B \times B \mapsto B$$

$$f_{an}(a, b) = a \nleftrightarrow b \text{ mit } a, b \in B$$

a	b	$a \nleftrightarrow b$
0	0	0
0	1	1
1	0	1
1	1	0

5.1.7 „*NAND*“

NAND ist die Negation der Disjunktion. Mit *NAND* lassen sich alle logischen Funktionen abbilden. Dies ist bei der Produktion von Hardware von Vorteil.

$$f_{NOR} : B \times B \mapsto B$$

$$f_{NOR}(a, b) = a \ NAND \ b \text{ mit } a, b \in B$$

a	b	$a \ NAND \ b$
0	0	1
0	1	1
1	0	1
1	1	0

5.1.8 „*NOR*“

NOR ist die Negation der Disjunktion. Mit *NOR* lassen sich alle logischen Funktionen ebenfalls abbilden. Dies ist bei der Produktion von Hardware von Vorteil.

$$f_{NOR} : B \times B \mapsto B$$

$$f_{NOR}(a, b) = a \ NOR \ b \text{ mit } a, b \in B$$

a	b	$a \ NOR \ b$
0	0	1
0	1	0
1	0	0
1	1	0

5.1.9 Implikation „→“

Die Implikation von a nach b ist 0, wenn $a = 0$ ist und $b = 1$, ansonsten 1. Die Implikation kann auch geschrieben werden als $(\overline{a} + b)$.

$$f_{im} : B \times B \mapsto B$$

$$f_{im}(a, b) = a \to b = \overline{a} + b \text{ mit } a, b \in B$$

a	b	$a \to b$
0	0	1
0	1	1
1	0	0
1	1	1

5.2 Grundlegende Gesetze

Es gelten folgende grundlegende Gesetze in der booleschen Algebra:

Kommutativ-Gesetze

$$a \cdot b = b \cdot a$$
$$a + b = b + a$$

Assoziativ-Gesetze

$$(a \cdot b) \cdot c = a \cdot (b \cdot c)$$
$$(a + b) + c = a + (b + c)$$

Distributiv-Gesetze

$$a \cdot (b + c) = (a \cdot b) + (a \cdot c)$$
$$a + (b \cdot c) = (a + b) \cdot (a + c)$$

Neutrale Elemente

$$a \cdot 1 = a$$
$$a + 0 = a$$
$$a \cdot a = a$$
$$a + a = a$$

Komplemente

$$a \cdot \overline{a} = 0$$
$$a + \overline{a} = 1$$

Negation der Negation

$$\overline{\overline{a}} = a$$

Gesetze nach De Morgan und Shannon

$$\overline{a \cdot b} = \overline{a} + \overline{b}$$
$$\overline{a + b} = \overline{a} \cdot \overline{b}$$
$$\overline{a \leftrightarrow b} = \overline{a} \nleftrightarrow \overline{b}$$
$$\overline{a \nleftrightarrow b} = \overline{a} \leftrightarrow \overline{b}$$

Absorptionsgesetze

$$a + (a \cdot b) = a$$
$$a \cdot (a + b) = a$$

In diesem Artikel werden folgende Bindungsstärken festgelegt:

1. Negation bindet stärker als Konjunktion
2. Konjunktion bindet stärker als Disjunktion

Das Konjunktionsszeichen „ $\cdot$ “ zwischen Operanden wird zur Vereinfachung der Schreibweise weggelassen.

5.3 Entwurf von Schaltnetzen

Logische Funktionenen werden hardware-technisch in Logikgattern produziert. Wegen der hohen Flexibilität der Booleschen Algebra können verschiedene Gatteranordnungen gleiche Funktionen repräsentieren. Also ist Ziel des Entwurfs eine aufwandsgünstigste Lösung zu finden. Mit der heutigen Halbleitertechnologie werden hochintegrierte Schaltkreise gefertigt. Um den Testaufwand zu vereinfachen werden möglichst regelmäßige Strukturen bevorzugt.

Jede logische Funktionsvorschrift lässt sich als Funktionstabelle[1] oder Term[2] darstellen. Die Funktionsterme dienen als Vorlage für den Gatterbau. Nach den Regeln der Boolschen Algebra können Funktionsterme umgeformt und so in eine optimale Form gebracht werden. Zur Entwicklung des Gatterentwurfs werden in der Praxis SILICON-Compiler Programme eingesetzt.

5.3.1 Definition der Booleschen Normalformen

Ein Boolescher Funktionsterm kann aus mehreren Eingabe-Variablen oder deren Negation und deren logischen Verknüpfungen bestehen.

Die Konjunktive Form (KF) einer logischen Funktion ist die Konjunktion von Disjunktionen. Beispiel:

$$f_{KF}(a,b,c,d) = (a+c)\cdot(\overline{c}+d)\cdot(\overline{a}+c+d)$$

Die Disjunktive Form (DF) einer logischen Funktion ist die Disjunktion von Konjunktionen. Beispiel:

$$f_{DF}(a,b,c,d) = (a\cdot c)+(\overline{a}\cdot d)+(\overline{a}\cdot b\cdot d)$$

Die kanonische Normalform eines Funktionsterms ist die Form in der alle Variablen der Funktion, entweder als solche oder negiert, in einer Disjunktiven Form oder Konjunktiven Form verknüpft sind. Die jeweilige Form wird *Disjunktive kanonische Normalform (DNF)* bzw. *Konjunktive kanonische Normalform (KNF)* genannt. Beispiele:

$$f_{KNF}(a,b,c,d) = \underbrace{(a+\overline{b}+c+d)}_{Maxterm/Faktor}\cdot(a+b+\overline{c}+d)\cdot(\overline{a}+\overline{b}+c+d)$$

$$f_{DNF}(a,b,c,d) = \underbrace{(a\cdot b\cdot c\cdot d)}_{Minterm/Summand} +(\overline{a}\cdot\overline{b}\cdot\overline{c}\cdot d)+(\overline{a}\cdot b\cdot c\cdot d)$$

Die Faktoren der Konjunktiven kanonischen Normalform werden Maxterme genannt. Die Summanden der Disjunktiven kanonischen Normalform heißen Minterme.

[1] Funktionstabelle: auch genannt Schaltbelegungstabelle; Krzl. SBT
[2] Term wird hier synonym für mathematischer Ausdruck verwendet

Jeder logische Term lässt sich in der Konjunktiven Form oder der Disjunktiven Form darstellen. Durch Erweiterung kann jeder Term außerdem in die Disjunktive oder Konfjunktive Normalform gebracht werden. Die Disjunktive Form wird durch „Ausmultiplizieren“ und die Konjunktive Form durch „Ausaddieren“ erstellt.

Eine Disjunktive Formen kann in die Disjunktive Normalform umgeformt werden indem den Summanden das für Konjunktionen neutrale Element $1 = (x + \overline{x})$ je nach fehlender Variable hinzugefügt wird. Anschließend muss wiederum ausmultipliziert werden. Entsprechend kann die Konjunktive Normalform erschlossen werden indem den Faktoren das für Disjunktionen neutrale Element $0 = (x \cdot \overline{x})$ je nach fehlender Variable hinzugefügt wird. Danach wird wieder ausaddiert.

Beispiel: Der Funktionsterm $f = a(b+c)+c(a+\overline{b}c)$ soll in Konjunktive bzw. Disjunktive Form umgeformt werden. Anschließend werden die erhaltenen Formen in die Normalform überführt.

Umformung in Disjunktive Form durch „Ausmultiplizieren“ (siehe Abbildung 5.1):

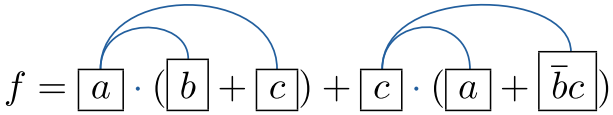

Abbildung 5.1: Ausmultiplizieren

$$\begin{aligned} f &= a(b+c) + c(a+\overline{b}c) \\ &= ab + ac + ca + c\overline{b}c \\ f_{DF} &= ab + ac + \overline{b}c \end{aligned}$$

Umformung in Konjunktive Form durch „Ausaddieren“ (siehe Abbildung 5.2):

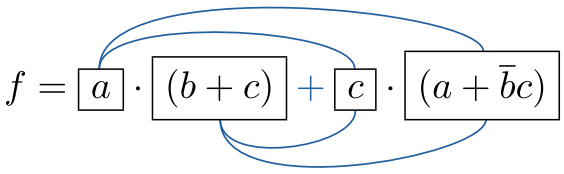

Abbildung 5.2: Ausaddieren

$$
\begin{aligned}
f &= a(b+c) + c(a+\overline{b}c) \\
&= (a+c)(a+(a+\overline{b}c))((b+c)+c)((b+c)+(a+\overline{b}c)) \\
&= (a+c)(a+\overline{b}c)(b+c)(b+a+\underbrace{c+\overline{b}c}_{Absorption}) \\
&= (a+c)(a+\overline{b}c)(b+c)(a+b+c) \\
&= (a+c)(a+\overline{b})(a+c)(b+c)(a+b+c) \\
f_{KF} &= (a+c)(a+\overline{b})(b+c)
\end{aligned}
$$

Herleitung der Disjunktiven Normalform durch Hinzufügen neutraler Elemente der Form $(x+\overline{x})$:

$$
\begin{aligned}
f_{DF} &= ab + ac + \overline{b}c \\
&= ab(c+\overline{c}) + ac(b+\overline{b}) + \overline{b}c(a+\overline{a}) \\
&= abc + ab\overline{c} + acb + ac\overline{b} + \overline{b}ca + \overline{b}c\overline{a} \\
f_{DNF} &= abc + ab\overline{c} + a\overline{b}c + \overline{a}\overline{b}c
\end{aligned}
$$

Herleitung der Konjunktiven Normalform durch Hinzufügen neutraler Elemente der Form $(x \cdot \overline{x})$:

$$
\begin{aligned}
f_{DF} &= (a+c)(a+\overline{b})(b+c) \\
&= (a+c+(b\cdot\overline{b}))(a+\overline{b}+(c\cdot\overline{c}))(b+c+(a\cdot\overline{a})) \\
&= ((a+c+b)\cdot(a+c+\overline{b}))((a+\overline{b}+c)\cdot(a+\overline{b}+\overline{c}))((b+c+a)\cdot(b+c+\overline{a})) \\
&= (a+b+c)(a+\overline{b}+c)(a+\overline{b}+\overline{c})(\overline{a}+b+c)
\end{aligned}
$$

Die Minterme beziehungsweise Maxterme einer logischen Funktion sind eindeutig bestimmt. Demnach besitzt jede logische Funktion genau eine Konjunktive und eine Disjunktive kanonische Normalform. Sie bilden den Ausgangspunkt allgemeingültiger Reduktionsmethoden.

5.3.2 Schaltbelegungstabelle

Logische Funktionen können auch mittels einer Schaltbelegungstabelle spezifiziert werden. Die Min- und Maxterme können aus dieser Tabelle direkt ausgelesen und damit die kanonischen Normalformen erstellt werden. Zur Erstellung der Schaltbelegungstabelle werden für eine n-stellige Funktion alle 2^n Kombinationen der Eingangsvariablen aufgeschrieben und das jeweilige Ergebnis dazu notiert. Damit ist für alle Eingangswerte der Funktion das Ergebnis der Funktion definiert. Beispiel:

Tabelle 5.1: Schaltbelegungstabelle für $f(a, b, c)$

i	0	1	2	3	4	5	6	7
a	0	0	0	0	1	1	1	1
b	0	0	1	1	0	0	1	1
c	0	1	0	1	0	1	0	1
y	0	0	1	0	1	1	0	0
m_i			$\overline{a}b\overline{c}$		$a\overline{b}\overline{c}$	$a\overline{b}c$		
M_i	$(a+b+c)$	$(a+b+\overline{c})$		$(a+\overline{b}+\overline{c})$			$(\overline{a}+\overline{b}+c)$	$(\overline{a}+\overline{b}+\overline{c})$

Die Schaltbelegungstabelle gibt die Belegungen der Variablen an, bei denen der Funktionswert 1 ist. Da eine Disjunktion genau dann 1 ist, wenn eines ihrer Summanden 1 ist, lassen sich die Minterme aus der Schaltbelegungstabelle ablesen. Ein Minterm ergibt sich aus jeder Spalte, in der der Funktionswert 1 ist. Ist die Belegung der Variable in dieser Spalte 1, dann muss im Minterm die Variable als solche angegeben werden. Ist die Belegung der Variable 0, dann muss im Minterm die negierte Variable angegeben werden (vgl. Tabelle 5.1 Spalten $i \in \{2, 4, 5\}$). Im angegebenen Beispiel ist die Disjunktive kanonische Normalform der Funktion $f(a, b, c)$ also:

$$\sum_{i \in \{2,4,5\}} m_i = \sum m(2, 4, 5)$$

$$m_2 + m_4 + m_5 = \overline{a}b\overline{c} + a\overline{b}\overline{c} + a\overline{b}c$$

Für die Herleitung der Maxterme betrachten wir zuerst die negierte Funktion. Die Minterme der negierten Funktion befinden sich an genau den Stellen, an der die Funktion ihre Maxterme hat. Über Negation der negierten Funktion wird wieder die ursprüngliche Funktion erzeugt. Die negierten Minterme der negierten Funktion sind nach De Morgan also die Maxterme der ursprünglichen Funktion.

Mit den Maxtermen kann analog folgendermaßen verfahren werden: Die Schaltbelegungstabelle gibt die Belegungen der Variablen an, bei denen der Funktionswert 0 ist. Ein Maxterm ergibt sich aus jeder Spalte, in der der Funktionswert 0 ist. Ist die Belegung der Variable in dieser Spalte 1, dann muss im Maxterm die negierte Variable angegeben werden. Ist die Belegung der Variable 0, dann muss im Maxterm die Variable als solche angegeben werden (vgl. Tabelle 5.1 Spalten $i \in \{0, 1, 3, 6, 7\}$)[3]. Im angegebenen Beispiel ist die Konjunktive kanonische Normalform der Funktion $f(a, b, c)$ also:

$$\prod_{i \in \{0,1,3,6,7\}} M_i = \prod M(0, 1, 3, 6, 7)$$

$$M_0 \cdot M_1 \cdot M_3 \cdot M_6 \cdot M_7 = (a+b+c)(a+b+\overline{c})(a+\overline{b}+\overline{c})(\overline{a}+\overline{b}+c)(\overline{a}+\overline{b}+\overline{c})$$

[3] also salopp genau anders herum als vorher

5.3.3 Halb- und Volladdierer

Die Addition von Dualzahlen kann über logische Funktionen und damit in logischen hardwaretechnischen Schaltungen abgebildet werden. Ein Halbaddierer berechnet die Summe s von zwei Bits a und b mit Berücksichtigung des Übertrags[4] c. Der Volladdierer berücksichtigt in der Eingabe zusätzlich das carry-Bit.

Halbaddierer SBT

i	0	1	2	3
a	0	0	1	1
b	0	1	0	1
s	0	1	1	1
c	0	0	0	1

$$s_{KNF} = a + b$$
$$c_{DNF} = ab$$

Volladdierer SBT

i	0	1	2	3	4	5	6	7
a	0	0	0	0	1	1	1	1
b	0	0	1	1	0	0	1	1
c	0	1	0	1	0	1	0	1
s	0	1	1	0	1	0	0	1
c	0	0	0	1	0	1	1	1

$$s_{DNF} = \overline{a}\overline{b}c + \overline{a}b\overline{c} + a\overline{b}\overline{c} + abc$$
$$c_{DNF} = \overline{a}bc + a\overline{b}c + ab\overline{c} + abc$$

5.3.4 Redundante Funktionen

In manchen Fällen kommen bestimmte Kombinationen der Belegung der Eingabevariablen nie vor. Folglich müssen Funktionswerte für diese Kombinationen nicht zwingend definiert werden. Der Funktionswert in solchen Fällen ist also beliebig. In der Schalttabelle werden nicht definierte Funktionswerte mit „ * “ gekennzeichnet. Bei der Optimierung können nicht zuerst nicht definierte Funktionswerte frei gewählt werden und können gegebenenfalls für so für eine Optimierung sorgen, die anders nicht möglich gewesen wäre.

[4] Übertrag engl. carry

6 Autorenlogbuch

Datum	Uhrzeit	Logeintrag
tt.mm.Jahr	hh:mm	arbeite schon seit 3 Wochen an diesem Projekt. Heute 17.10.13 kam mir die Idee eines Autorenlogbuchs...
17.10.2013	20:31	Heute wieder nichts gegessen außer ein paar Schokokaffeebohnen. Zu wenig getrunken. Zeit rennt. Überlege mir einen neuen Plan zu erstellen um Zeitmanagement koordinieren zu können. Am Wochenende wollte ich eigentlich weg. Hmm... daraus wird wohl nichts. Nächste Woche nach Karlsruhe? Wird wohl auch schwierig werden. Mal sehen. Mibewo quatschen in der Küche. Werde jetzt erstmal duschen und dann was zu essen machen. Nur noch schnell Daten sichern :)
17.10.2013	21:59	Gut gegessen. Es gab Reis mit Tomatensoße, Möhren und leckere Rohesser von zuhause *YAY* weiter geht's. Will das Kapitel heute noch zu Ende bringen.