Lerntext Computergestütztes Experimentieren I

Mladen Ivkovic
mladen.ivkovic@uzh.ch

Herbstsemester 2015

Anmerkung des Autoren

Zweck Dieses Dokument soll Lernmaterial/eine Lernhilfe für die Vorbereitung auf die Prüfung des Moduls Computergestütztes Experimentieren I an der UZH im Herbstsemester 2015 an der UZH sein.

Quellen Eine Quellenangabe gibt es nicht, ich habe nicht nachverfolgt, wo ich welche Information herhabe. (Das Gleiche gilt für die Abbildungen.) Das Allermeiste wurde aber aus den Vorlesungsfolien und dem Buch "Grundlagen der technischen Informatik" von D. Hoffmann entnommen (bzw. übernommen).

Fehler Ich kann und will nicht für die Richtigkeit (oder Aktualität) dieses Dokumentes garantieren. Ich habe es aber mit bestem Wissen und Gewissen geschrieben.

Weitergabe Jegliche nichtkommerzielle Weitergabe und Benutzung, insbesondere zu Lehr- und Lernzwecken, dieses Dokumentes ist meinerseits freigegeben.

Digitale Schaltungen Zum besseren Verständnis von Flipflops könnte diese Website nützlich sein: `http://www.ti.informatik.uni-frankfurt.de/Lehre/TI2-SS2002/flash.html`

Sie beinhaltet ein paar interaktive Flash-Animationen digitaler Schaltungen wie Addierern und Flipflops.

Abbildungen im Anhang Alle Abbildungen, welche vor der Nummerierung ein A enthalten, beispielsweise A16, befinden sich im Anhang.

Danke Mit herzlichstem Dank an Herrn Herbert Bitto für die (zweimalige) Gegenlektüre dieses Dokumentes!

Inhaltsverzeichnis

1. Zahlendarstellung und Boolesche Algebra

1.1. Grundlagen der Zahlendarstellung

1.1.1. Grundlagen

Die gängigste Form der Zahlensysteme sind Stellenwertsysteme. Eine Zahl a wird in Form einer Reihe von Ziffern z_i mit dazugehöriger Potenz der Basis b^i dargestellt. Der Wert der Zahl ergibt sich dann als Summe der Werte aller Einzelstellen: $a = \sum_i z_i b^i$.

Umrechnung in andere Zahlensysteme: Gegeben sei Zahl Z, umzuwandeln in System mit Basis b. Eine angenehme Vorgehensweise gibt uns das **Horner Schema**[1]: Dividiere Z durch b. Der Rest dieser Division ist die letzte Stelle der Zahl in der Basis b (Einerstelle). Dividiere den Quotienten dieser Division wieder durch b. Der Rest dieser zweiten Division ergibt die zweite Stelle der Zahl in der neuen Basis. Wiederhole Divisionen, bis kein Rest mehr.

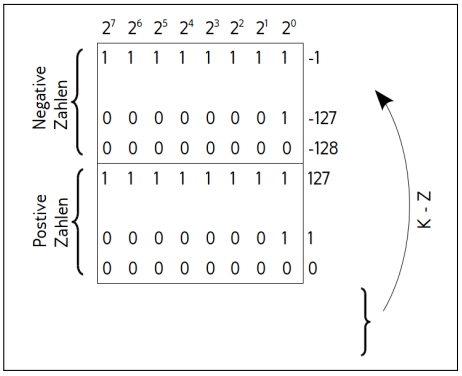


Abb. 1: Darstellung des Zahlenbereichs des Zweierkomplements mit acht Stellen

1.1.2. Addition, Subtraktion, negative Zahlen

Das Dual- oder Binärsystem ist das wichtigste Zahlensystem im Rechner. Oktal- und Hexadezimalsystem werden benutzt, da sie leicht ins Dualsystem umwandelbar und besser zu verstehen als lange 0-1-Kolonnen sind.

Die **Addition** binärer Zahlen funktioniert im Prinzip genau gleich wie die der Dezimalzahlen: Man addiert die einzelnen Stellen miteinander, wobei mit der Einerstelle angefangen wird. Ist die Summe grösser oder gleich der Basis des Systems, hier also 2, so gibt es einen Übertrag, der bei der nächsten Stelle dazuaddiert werden muss. Für die duale Addition gilt allgemein:

$$\begin{aligned} 0 + 0 &= 0 \\ 0 + 1 &= 1 \\ 1 + 0 &= 1 \\ 1 + 1 &= 0 \text{ Übertrag } 1 \\ 1 + 1 + 1 \text{ (vom Übertrag)} &= 1 \text{ Übertrag } 1 \end{aligned}$$

Die **Subtraktion** erfolgt mittels Addition einer negativen Zahl. Für die Darstellung negativer Zahlen in Rechnern werden vier verschiedene Formate benutzt, von denen wir uns hier nur auf das Einerkomplement und das Zweierkomplement beschränken.

[1] Website mit Umrechnungen und Erklärungen: `http://www.arndt-bruenner.de/mathe/scripts/Zahlensysteme.htm`

Beim **Stellen-** oder **Einerkomplement** wird jedes Bit der Zahl komplementiert, also Nullen in Einsen und umgekehrt verwandelt, um eine Zahl zu negieren. Der darstellbare Zahlenraum wird in zwei Teile gespalten: Ist an der höchsten Stelle eine Null, so ist die Zahl positiv; ist es hingegen eine Eins, so ist die Zahl negativ. Die höchste Stelle trägt aber weiterhin zum Wert der Zahl bei, ist also kein gesondertes Vorzeichenbit. Der Nachteil des Einerkomplementes ist, dass es zwei Darstellungen der Null gibt.

Bei der **Zweierkomplement-Darstellung** gibt es zwei Vorgehensweisen:

1. Zuerst muss vereinbart werden, dass wir mit N_{max}-stelligen Dualzahlen rechnen. Somit ist die grösste darstellbare Zahl $Z_{max} = 2^{N_{max}} - 1$ und die grösste positive Zahl per Vereinbarung $P_{max} = 2^{N_{max}-1} - 1$.[2]

   Nun nennen wir K die kleinste nicht darstellbare Zahl, also ist $K = Z_{max} + 1 = 2^{N_{max}}$

   Wir erhalten das Zweierkomplement einer Zahl B, indem wir sie von K abziehen (d.h. die Ergänzung von B zu K, daher Komplement):

   $$A - B = A + (K - B) - K$$

   Es gilt: $P_{max} + 1 \leq K - B \leq Z_{max}$, die negativen Zahlen werden sozusagen "nach oben verschoben". (Vergleiche dazu die Abbildung 1)

2. Man addiert nach der Stellenkomplementierung noch eine 1 dazu.

   Bei der Stellenkomplementierung werden einsen in Nullen umgewandelt und umgekehrt: $B \to \bar{B}: \ 0 \leftrightarrow 1$. Es gilt: $B + \bar{B} = K - 1 \Rightarrow K - B = \bar{B} + 1$

Die Darstellung mit dem Zweierkomplement hat eine eindeutige Null, aber der Zahlenbereich ist unsymmetrisch: Es gibt eine negative Zahl mehr als positive.

1.2. Basisoperatoren

Nun wollen wir (logische) Verknüpfungen zwischen binären Zahlen resp. Variablen einführen und benutzen. Solche Verknüpfungen nennt man Operatoren.

Im Grunde genommen benötigen wir drei Basisoperatoren, um ein vollständiges Operatorensystem zu erhalten:

Konjunktion UND			Disjunktion ODER			Negation		NAND			NOR		
a	b	$a \wedge b$	a	b	$a \vee b$	a	$\bar{a}$	a	b	$\overline{a \wedge b}$	a	b	$\overline{a \vee b}$
0	0	0	0	0	0	0	1	0	0	1	0	0	1
0	1	0	0	1	1	1	0	0	1	1	0	1	0
1	0	0	1	0	1			1	0	1	1	0	0
1	1	1	1	1	1			1	1	0	1	1	0

[2] Vergiss nicht: wir können an der höchsten Stelle sehen, ob die Zahl positiv oder negativ ist. Damit sie positiv ist, muss die höchste Stelle eine Null sein, deshalb $2^{N_{max}-1} - 1$

Schematische Abbildungen der drei Operatoren als elektrische Schaltkreise sind in den Abbildungen A5 - A7 dargestellt. Die entsprechenden Gatter-Symbole sind in der Abb. 2 zu sehen.

Der NAND und NOR - Operator bilden jeweils vollständige Operatorensysteme. Das $(\vee, \wedge, ^-)$-System ist zwar intuitiver, jedoch sind NAND- und NOR-Gatter einfacher realisierbar. Mit dem NAND- und NOR-System lassen sich beliebige disjunktive und konjunktive Ausdrücke darstellen. Die Überführung geschieht durch doppelte Negation und anschliessende Anwendung der DeMorganschen Regeln (siehe nächsten Abschnitt). Ein Problem dabei ist, dass die Operatoren $\overline{\vee}$ und $\overline{\wedge}$ nicht assoziativ sind.

In Schaltkreisen ist die Konversion teilweise noch einfacher durchführbar: Vernachlässigt man Laufzeiteffekte, so können wir zwischen zwei Gattern zwei Negationen hinzufügen: Dadurch wird der Wahrheitswert nicht verändert. Eine Negation nach einem UND-Gatter macht diesen zu einem NAND; Negiert man beide Eingänge eines ODERs, so wird daraus auch ein NAND (siehe DeMorgan-Gesetz) Je eine Negation können wir dann mit je einem Gatter zusammenfassen - und schon haben wir aus UNDs und ODERs NANDs und NORs gemacht.

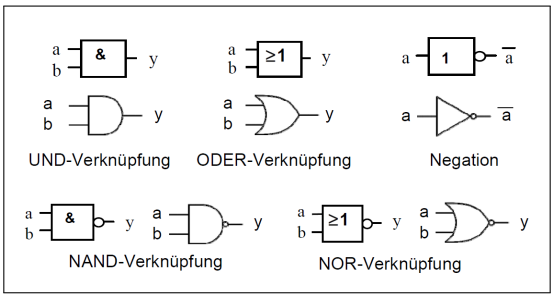


Abb. 2: Schaltsymbole (DIN 40900 Teil 12, ANSI/IEEEStandard 91-1984, IEC-Standard & US-Symbole

1.3. Boolesche Algebra

Als **Boolesche Algebra** bezeichnet man eine abgeschlossene Menge V ($\forall y, b \in V$: $a \otimes b \in V$, $a \oplus b \in V$), auf der für zwei zweistellige Operationen $\oplus$ und $\otimes$ das Kommutativgesetz und das Distributivgesetz gelten und ein neutrales und inverses Element existieren (Huntingtonsche Axiome). Wir ersetzen für unsere Zwecke direkt $\otimes$ mit $\wedge$ bzw. UND und $\oplus$ mit $\vee$ bzw. ODER.

Kommutativgesetz:	$a \wedge b = b \wedge a$	$a \vee b = b \vee a$
Distributivgesetz:	$a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge b)$	$a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c)$
Neutrales Element	$a \wedge 1 = a$	$a \vee 0 = a$
Inverses Element	$a \wedge \bar{a} = 0$	$a \vee \bar{a} = 1$
Assoziativgesetz	$(a \wedge b) \wedge c = a \wedge (b \wedge c)$	$(a \vee b) \vee c = a \vee (b \vee c)$

Idempotenzgesetz	$a \wedge a = a$	$a \vee a = a$
Absorptionsgesetz	$a \wedge (a \vee b) = a$	$a \vee (a \wedge b) = a$
DeMorgan-Gesetz	$\overline{a \wedge b} = \bar{a} \vee \bar{b}$ (NAND)	$\overline{a \vee b} = \bar{a} \wedge \bar{b}$ (NOR)
Gesetz vom Widerspruch		$a \wedge \overline{a} = 0$
Gesetz vom ausgeschl. Dritten		$a \vee \overline{a} = 1$
Gesetz der doppelten Negation		$\overline{\overline{a}} = a$

Ein bemerkenswertes (und hilfreiches) Merkmal der booleschen Algebra ist die Dualität der Gesetze: Ersetzt man nämlich alle Konjunktionen ($\wedge$) und Nullen durch Disjunktionen ($\vee$) und Einsen (oder umgekehrt), so erhält man eine weiterhin gültige Gleichung, nämlich genau die Gleichung des entsprechenden Gesetzes für den gewechselten Operator. Kurz: $\vee, 1 \Leftrightarrow \wedge, 0$

Jede n-stellige boolesche Funktion lässt sich eindeutig mit einer Wahrheitstabelle, welche 2^n Zeilen beinhaltet, beschreiben. (Eine boolesche Variable kann genau zwei Werte annehmen; Kommt eine weitere Variable dazu, so verdoppelt sich die Anzahl der möglichen Kombinationen der Eingangsvariablen.[3]) In einer Wahrheitstabelle werden alle möglichen Kombinationen der Eingabevariablen mit dem entsprechenden Funktionswert aufgeführt.

2. Digitale Schaltungen

2.1. Entwicklung einer logischen Schaltung

Sei nun die Wahrheitstabelle mit n Variablen einer noch unbekannten booleschen Funktion bekannt bzw. gegeben. (Wir gehen davon aus, dass wir die Funktion, welche wir realisieren wollen, zumindest in dieser Form kennen.) Die Wahrheits- bzw. Funktionstabelle weist jeder möglichen Kombination der Variablen einen Wert zu. (Die zweite Spalte der Abb. A2 beispielsweise führt alle möglichen Funktionstabellen mit zwei Variablen auf.) Nun möchten wir eine Schaltung erstellen, welche genau dieses Verhalten besitzt.

Zunächst müssen wir nun aus der Wahrheitstabelle die boolesche Funktion bestimmen. (Die boolesche Funktion ist eindeutig durch die Wahrheitstabelle bestimmt. Wir erhalten also immer äquivalente Ergebnisse.) Dies machen wir mittels der **Konjunktiven- oder Disjunktiven Normalform (KNF oder DNF)**.

- **DNF**: Aus der Funktionstabelle einer Funktion erhält man die Minterme, indem man in allen Zeilen mit dem Funktionswert 1 jeweils alle Eingangsvariablen mit

[3] Vielleicht ist es durch dieses kleine Beispiel besser ersichtlich: Nehmen wir an, wir kennen alle möglichen Kombinationen unserer n Variablen. Führen wir jetzt eine weitere Variable a_{n+1} ein, so können wir alle bereits bekannten Kombinationen der n Variablen durchgehen, während $a_{n+1} = 0$ ist, und ebenso nochmals alle bekannten Kombinationen durchgehen, während $a_{n+1} = 1$ ist. Wir sehen: Mit der Einführung einer neuen Variable hat sich die Anzahl möglicher Kombinationen verdoppelt.

$\wedge$ verknüpft und dabei Eingangsvariablen mit dem Wert 0 negiert.[4] Durch die disjunktive[5] Verknüpfung dieser Minterme kann ein boolescher Funktionsausdruck in DNF hergeleitet werden.

Beispiele:

$$\text{Disjunktion/ODER in der DNF: } a \vee b = (\overline{a} \wedge b) \vee (a \wedge \overline{b}) \vee \underbrace{(a \wedge b)}_{\text{Minterm}}$$
$$\text{Konjunktion/UND in der DNF: } a \wedge b = a \wedge b$$

Kurz gefasste Merkhilfe für die DNF: Welche Kombinationen von Eingangsvariablen ergeben 1? Bei jeder Kombination: Verbinde alle Eingangsvariablen mit einem UND ($\wedge$). Jede Eingangsvariable, die in dieser Kombination den Wert 0 hat, komplementiert in diesen Term schreiben. Verbinde dann alle Terme (pro Kombination 1 Term) mit einem ODER. Fertig.

- **KNF**: Aus der Funktionstabelle einer Funktion erhält man die Maxterme, indem man in allen Zeilen mit dem Funktionswert 0 jeweils alle Eingangsvariablen mit $\vee$ verknüpft und dabei Eingangsvariablen mit dem Wert 1 negiert.[6] Durch die konjunktive Verknüpfung dieser Maxterme kann ein boolescher Funktionsausdruck in KNF hergeleitet werden.

  Beispiele:

$$\text{Disjunktion/ODER in der KNF: } a \vee b = a \vee b$$
$$\text{Konjuntktion/UND in der KNF: } a \wedge b = (a \vee b) \wedge (\overline{a} \wedge b) \wedge \overbrace{(a \vee \overline{b})}^{Maxterm}$$

Bequemerweise wählt man diejenige Normalform, welche weniger Arbeit macht. Nun haben wir die logische Funktion als boolescher Ausdruck, jetzt müssen die Operatoren nur noch mit dem entsprechenden Gatter in der Schaltung ersetzen und die richtigen Verbindungen machen.

Eine Tabelle mit häufig benutzten Funktionen und ihrer Realisierung mit NAND- und NOR-Gattern ist in Abb. A8 dargestellt.

[4] Ein Produktterm $K(x_1, ..., x_n)$ heist Implikant einer Boolescher Funktion $f(x_1, ..., x_n)$, wenn für jede beliebige Belegung B der Variablen$\{x_1, ..., x_n\}$ gilt: Wenn $K(B) = 1$, dann ist auch $f(B) = 1$, aber nicht zwangsweise umgekehrt. Ein Implikant einer booleschen Funktion heisst Minterm, wenn jede Variable der Funktion genau einmal in ihm vorkommt, sei es invertiert oder nicht. Es müssen also pro Minterm alle Variablen vorkommen!

[5] Eselsbrücke für Konjunktion/Disjunktion: Alphabetisch. **d**isjunktion = **o**der, **k**onjunktion = **u**nd. Tipp: Merke dir nur eines, und leite dir daraus her, wie das andere ist.

[6] Eine Disjunktion $D(x_1, ..., x_n)$ heist Implikat einer Boolescher Funktion $f(x_1, ..., x_n)$, wenn für jede beliebige Belegung B der Variablen$\{x_1, ..., x_n\}$ gilt: Wenn $D(B) = 0$, dann ist auch $f(B) = 0$, aber nicht zwangsweise umgekehrt. Ein Implikat einer booleschen Funktion heisst Maxterm, wenn jede Variable der Funktion genau einmal in ihm vorkommt, sei es invertiert oder nicht. Es müssen also pro Maxterm alle Variablen vorkommen!

2.2. Kombinatorische und Sequentielle Schaltkreise

Kombinatorische Schaltkreise sind logische Funktionen, welche zu jedem Zeitpunkt durch ihre Eingangsvariablen bestimmt sind.. Zu deren Untersuchung und Beschreibung der Eigenschaften und des Verhaltens ist die Boolesche Algebra hervorragend geeignet. Der Zustand aller Ausgänge ist eindeutig von den Zuständen aller Eingänge bestimmt.

Sequentielle Schaltkreise hingegen haben eine Art Gedächtnis bzw. Speicher. Wird der Speicherwert als Eingabewert für den nächsten Schritt verwendet, so ist der Schaltkreis nicht nur von den momentanen Zustand ihrer Eingangsvariablen bestimmt, sondern auch vom Zustand der vorhergehenden Eingangsvariablen, welche den Zustand des Speichers bestimmt haben. Dies wird dadurch realisiert, dass gewisse Ausgänge wieder in Eingänge zurückgeführt werden (Stichwort Rückkopplung). Diese Rückführungen implementieren die Funktionalität eines Gedächtnises.

Auf diese Weise funktionieren die sogenannten Flipflops, welche wir und in den nächsten Kapiteln näher anschauen wollen. Zunächst jedoch machen wir noch den Unterschied zwischen synchronen und asynchronen Schaltkreisen:

Werden alle Zustandsspeicher von einem oder mehreren zentralen Synchronisationssignal(en) **T** (Takt, englisch: Clock) gesteuert, so spricht man von einem **synchronen Schaltwerk**. Anderenfalls spricht man von einem **asynchronen Schaltwerk**. Asynchrone Schaltwerke arbeiten ohne einen zentralen Takt: Sie reagieren sofort auf jede Änderung der Eingangs- und Zustandsvariablen und sind sehr störempfindlich.

Die Synchronisation über einen Takt kann flankengesteuert und pegelgesteuert sein. Bei der **Flankensteuerung** wird nur während der positiven ($0 \to 1$) oder der negativen ($1 \to 0$) Taktflanke die Eingabewerte übernommen. Bei der **Pegelsteuerung** hingegen ist der Speicher während einer Takthälfte transparent, während der anderen speichert er. Die Eingänge wirken sich nur auf den Zustand aus, wenn der Takt z.B. den Wert 1 hat. Ist der Takt 0, wird der Zustand gespeichert. Pegelgesteuerte Zustandsspeicher werden auch Latches genannt. Im Schaltsymbol wird die Taktflankensteuerung durch ein Dreieck am Takteingang spezifiziert. Bei einer Steuerung mit der negativen Taktflanke wird ein Negationszeichen vor das Dreieck gesetzt (Kreis vor dem Eingang).

2.3. Flipflops

2.3.1. RS-Flipflop

Das wohl einfachste Flipflop ist das sogenannte RS-Flipflop. Dabei stellen wir die folgende Anforderung: Das Flipflop soll zwei Eingänge haben: S für set und R für reset. Ist $S = 1$, so soll der Ausgabewert $Q = 1$ sein; ist $R = 1$, so soll $Q = 0$ gesetzt werden. Sind sowohl $S = R = 0$, so soll der Wert beibehalten werden. Das führt zur Wahrheitstabelle rechts, wobei Q_v der vorhergehende Ausgabewert ist und Q_n der resultierende.

Q_v	S	R	Q_n
Q_v	0	0	Q_v
x	1	0	1
x	0	1	0

Dabei müssen wir eine Nebenbedingung $R \wedge S = 0$ setzen - R und S dürfen niemals gleichzeitig $= 1$ sein. In der Realisierung, dargestellt in Abb. 3, führt dies zu oszillationen.

Will man ein taktgesteuertes RS-Flipflop, so braucht man lediglich das Taktsignal mit einem UND-Gatter jeweils mit dem R- und S-Eingang zu verbinden (siehe Abb. 4).

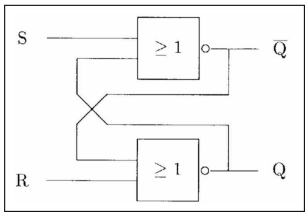


Abb. 3: RS-Flipflop

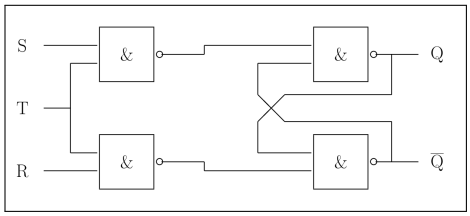


Abb. 4: getaktetes RS-Flipflop

2.3.2. D-Flipflop/D-Latch

Bei einem RS-Flipflop ist stets die Nebenbedingung ($R \wedge S = 0$) zu beachten. Führt man eine Eingangsvariable D bejaht zum S-Eingang und negiert zum R-Eingang, ist diese Bedingung stets erfüllt. Damit erhält man ein sogenanntes **D-Latch** (Siehe Abb. 5 und 6).

$D = 1$ entspricht $S = 1$ und $R = 0$, also wird der Ausgangswert $Q = 1$ sein. Umgekehrt bei $D = 0$: Dies entspricht $S = 0$ und $R = 1$, der Ausgabewert ist $Q = 0$. Wir sehen: $D = Q$.

Der anliegende Eingabewert wird in allen Fällen als Flipflopzustand übernommen und einen Takt lang gespeichert. Das Eingangssignal wird um eine Taktperiode verzögert am Ausgang zur Verfügung gestellt (D-Latch von ‘to delay’ = verzögern; Auch von ‘data’). Soll der Ausgabewert nicht sofort bei Takteingang geändert werden, so kann man das leicht durch die Implementierung eines Master-Slave-Flipflops erreichen (siehe weiter unten).

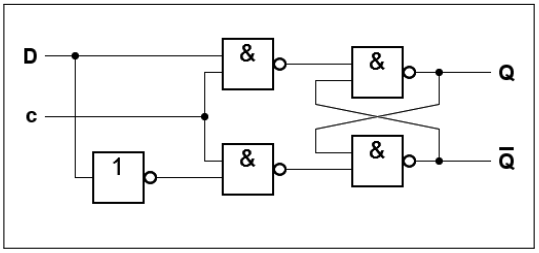


Abb. 5: D-Flipflop, Grundschaltung

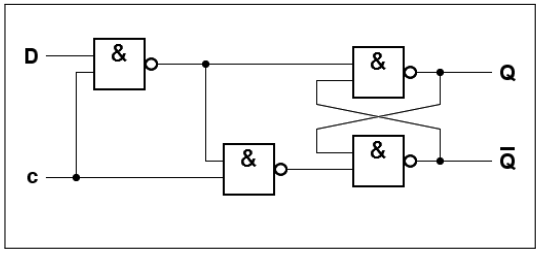


Abb. 6: D-Flipflop, vereinfachte Schaltung

2.3.3. JK-Flipflops

Beim RS-Flipflop war die Eingangsvariablen-Kombination $R = S = 1$ verboten. Als vierte Funktion neben ‘speichern’, ‘setzen’ und ‘rücksetzen’ soll nun bei Eingangskombination $R = S = 1$ der Flipflop-Inhalt komplementiert werden (siehe Tabelle rechts).

Die Eingänge bezeichnet man nun mit J (jump) - Setzeingang und K (kill) - Rücksetzeingang. Das resultierende Flipflop bezeichnet man als JK-Flipflop, dessen Implementierung in Abb. 7 dargestellt ist. Vergleicht man es mit der Schaltung des getakteten RS-Flipflops in Abb. 4, so sieht man, dass ein JK-Flipflop eigentlich ein RS-Flipflop beinhaltet (Abb. 8), nun aber der Ausgabewert Q mit dem Takt K über ein UND-Gatter verbunden ist, während $\bar{Q}$ auf gleiche Weise mit dem Eingang J verbunden ist.

Da Q und $\bar{Q}$ ja komplementär sind, können so die Eingänge des (im JK-Flipflop enthaltenen) RS-Flipflops nie gleichzeitig den gleichen Wert haben, auch wenn die Eingabewerte $J = K = 1$ sind. Daraus resultiert auch die Komplementierung des Ausgabewertes bei $J = K = 1$: Ist momentan $Q = 1$, so kann nur $S = 1$ resultieren; bei $Q = 0$ kann nur $R = 0$ resultieren.

J	K	Q_n
0	0	Q_{n-1}
1	0	1
0	1	0
1	1	$\overline{Q}_{n-1}$

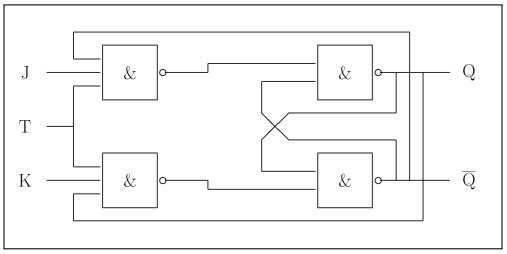


Abb. 7: JK-Flipflop

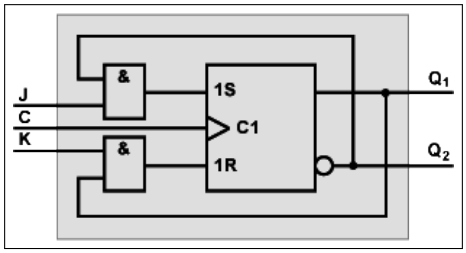


Abb. 8: JK-Flipflop, Darstellung mit RS-Flipflop. C = Takt, $Q_1 = Q$, $Q_2 = \bar{Q}$

2.3.4. Master-Slave-Flipflops

Soll nun ein Ausgabewechsel erst beim Taktflankenwechsel geschehen, also erst wenn der Takt von 1 wieder auf 0 geht, so kann dies durch Zusammenschaltung zweier Flipflops implementiert werden. Beispielsweise erhält man ein flankengesteuertes JK-Flipflop durch die Zusammenschaltung zweier JK-Flipflops, die mit komplementären Taktpegeln gesteuert werden. (Siehe Abb. 9) Das erste Latch wird Master-Flipflop, das zweite Slave-Flipflop genannt. Ein solches Flipflop wird auch als **Master-Slave-Flipflop** bezeichnet.

So funktioniert das Ganze: Während $T = 1$ folgt das erste (= Master-) Latch den Änderungen der Eingangssignale J und K, während das zweite Flipflop verriegelt ist: Wir führen den Takteingang invertiert zum Slave-Latch. So können sich die Ausgabewerte der beiden Flipflops nie gleichzeitig ändern. Ändert sich T von 1 auf 0 (negative Taktflanke), wird nun das Master-Flipflop verriegelt und das Slave-Flipflop übernimmt die Ausgabewerte des Master-Flipflops als Eingabewerte. Unabhängig von den nun auftretenden Änderungen von J und K bleibt der Ausgabewert Q_i gleich dem Wert von denjenigen J und K, die beim 0-1-Wechsel des Taktes anlagen. Dieser Wert wird in das zweite Latch übernommen und dort auch weiter gespeichert, wenn T wieder auf 0 zurückgeht.

Analog dazu gibt es auch D-Master-Slave-Flipflops.

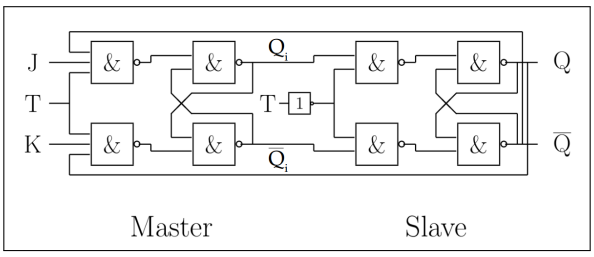


Abb. 9: Master-Slave-JK-Flipflop

2.4. Register und Zähler

2.4.1. Register

Unter einem Register verstehen wir eine parallele Anordnung von Speicherelementen, die synchron über eine gemeinsame Taktleitung betrieben werden. Im Gegensatz zu singulär verbauten Speicherelementen, die jeweils nur ein einzelnes Bit speichern können, dienen Register zur Speicherung und Manipulation vollständiger Datenwörter. Dazu wird das Wort nicht Bit für Bit verarbeitet, sondern jedes Bit gleichzeitig - die Bits werden parallel verarbeitet.

Wir wollen uns zunächst das sogenannte **Schieberegister** genauer anschauen. Ein Schieberegister wird seriell geladen. Der Schieberegister besitzt einen Registereingang. Der Registerinhalt wird in jedem Takt um ein Bit nach links oder rechts geschoben (je nach Schieberegistertyp) und die frei werdende Stelle mit dem am Eingang anliegenden Wert überschrieben. Jedoch verfügt es über die genau gleiche Anzahl Ausgängen wie die Wortbreite Bits hat - der Registerinhalt kann also parallel ausgelesen werden.

Ein Schieberegister kann auf einfache Weise mittels D-Master-Slave-Flipflops implementiert werden. (Siehe Abb. 10) Die Schiebeeigenschaft wird erreicht, indem der jeweilige Ausgang eines Speicherelements direkt mit dem Eingang des nachfolgenden Elements verbunden wird. Bei jedem Takt übernimmt das Flipflop ganz links den anliegenden Wert (Siehe Funktionsweise von D-Flipflops in Kapitel 2.3.2). Die anderen Flipflops übernehmen beim Taktsignal den Zustand des vorhergehenden Flipflops. Auf diese Weise wird die eingegebene Bitfolge ‘geschoben’.

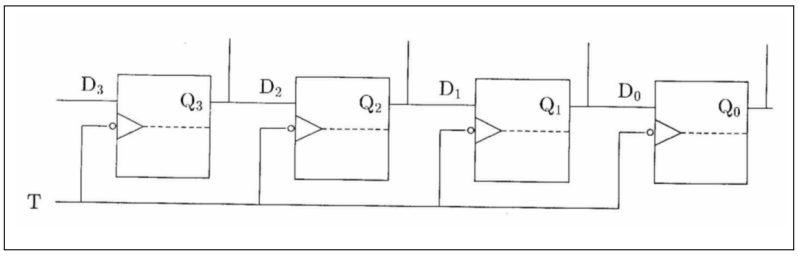


Abb. 10: Schieberegister aus D-Flipflops

Mit Hilfe von Schieberegistern können serielle Datenströme auf einfache Weise parallelisiert werden. Hierzu wird das Schieberegister nacheinander mit allen Bits eines Datenworts gefüllt und anschliessend an den Registerausgängen abgegriffen.

Ein Schieberegister kann auch dermassen implementiert werden, dass er parallel geladen wird, also alle (oder mehrere) Speicherelemente gleichzeitig einen externen Eingabewert übernehmen. Damit er aber immer noch als Schieberegister fungieren kann, muss jeder externe Eingang und die Ausgänge der Flipflops (welche ja als Eingänge für die darauffolgenden Flipflops benutzt werden) über einen Multiplexer (siehe Kap. 2.5) verbunden werden. Das Eingangssignal des Multiplexers ist dann Laden (L) oder Schieben (S), wobei meist $S = \overline{L}$ implementiert ist. (Siehe Abb. 11)

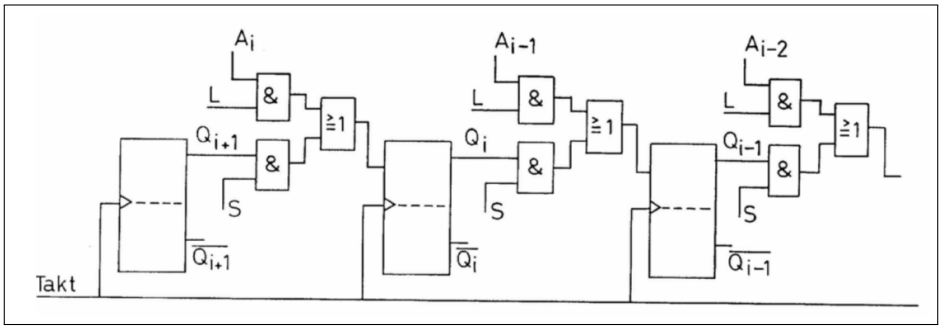


Abb. 11: Schieberegister mit parallelem Laden

2.4.2. Zähler

Interpretieren wir die Ausgabewerte von Flipflops als Bits einer Dualzahl, so lassen sich ziemlich schnell Schaltungen implementieren, welche wie Zähler funktionieren. Dazu wollen wir zwei davon etwas genauer anschauen.

Betrachten wir zuerst einen **asynchronen Zähler aus JK-Flipflops** (siehe Abb. 12). Hierbei reihen wir eine Anzahl von Master-Slave-JK-Flipflops hintereinander ein. Diesen Zähler nennen wir asynchron, da nicht alle Flipflops mit einen gemeinsamen Takt arbeiten. Stattdessen nehmen wir den Ausgabewert eines Flipflops als Takt für den darauffolgenden. Alle Eingänge J und K speisen wir mit dem Wert 1 (Zur Erinnerung: $J = K = 1$ heisst 'komplementiere den momentanen Ausgabewert' für JK-Flipflops). Auf diese Weise wechselt beim Taktsignal[7] das erste JK-Flipflop seinen Ausgabewert. Diesen Ausgabewert wählen wir als Takteingang für das nächste JK-Flipflop. Nach einem weiteren Taktumlauf hat das erste Flipflop wieder den ursprünglichen Ausgabewert, der zweite hingegen hat erst dann einen vollen Taktumlauf ($0 \to 1 \to 0$) erlebt und der Ausgabewert des zweiten Flipflops ändert sich zum ersten Mal. So muss das erste Flipflop immer zwei volle Taktumläufe durchmachen, damit sich beim zweiten der Ausgabewert ändert. Nach weiteren zwei Zustandsänderungen kann ist das zweite Flipflop beim Ursprungszustand; das dritte Flipflop hingegen hat nun zum ersten Mal einen Taktumlauf erlebt und wechselt seinen Ausgabewert. Man kann nun beliebig viele Flipflops aneinanderhängen, wobei jeder darauffolgende zweimal so viele Taktumläufe wie sein Vorgänger

[7] Anm. des Autoren: Wir arbeiten hier mit Master-Slave-Flipflops. Als 'Taktsignal' bezeiche ich hier einen vollen Umlauf $0 \to 1 \to 0$, weil erst dann der Ausgabewechsel beim Slave-Flipflop, das ja auch dem Ausgabewert des gesamten Master-Slave Flipflops darstellt, angekommen ist.

braucht, um seinen Ausgabewert zu ändern. Interpretieren wir die Ausgabewerte der Flipflops als Binärzahl, so fungiert diese Anordnung wie ein Zähler, der bei jedem Taktumlauf um eins grösser wird.

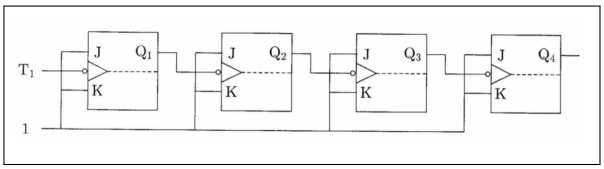


Abb. 12: Asynchroner JK-Flipflop Zähler

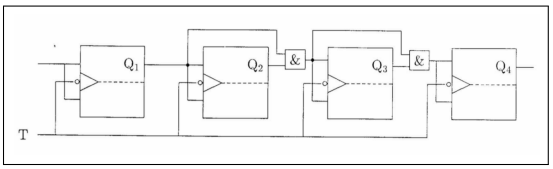


Abb. 13: Synchroner JK-Flipflop Zähler

Anmerkung zu Abb. 12 und 13: Bei dieser Darstellung muss die Zahl von links nach rechts (entgegen der Konvention) gelesen werden.

Bei einem **synchronem Zähler** müssen alle Flipflops über das gleiche Taktsignal angesteuert werden. Wir wollen nun einen solchen wieder aus JK-Flipflops realisieren (siehe Abb. 13). Hierfür leiten wir den Ausgabewert der vorhergehenden Flipflops als beide Eingabewerte des darauffolgenden, während alle Flipflops vom gleichen Taktsignal angesteuert werden. Nun müssen wir aber dafür sorgen, jedes Flipflop nur dann seinen Ausgabewert wechselt, wenn alle vorhergehenden Ausgabewerte = 1 sind. Dies ist einfach realisierbar, indem wir jeden Ausgabewert zuerst mit allen vorhergehenden Ausgabewerten über ein UND-Gatter verbinden, bevor wir es als Eingabewert in das nächste Flipflop leiten.

2.5. Multiplexer und Tristate-Treiber

Zwei wichtige digitale 'Schalter' sind der Multiplexer und der Tristate-Treiber.

Der Multiplexer soll bei zwei Eingängen a und b über das Signal S (*select*) 'entscheiden', ob am Ausgang a oder b ausgegeben wird. Das führt zu der rechts dargestellten reduzierten Wahrheitstabelle. Die Realisierungen sind in den Abb. 14 und 15 zu sehen.

Multiplexer

a	b	S	Ausgang
a	b	0	a
a	b	1	b

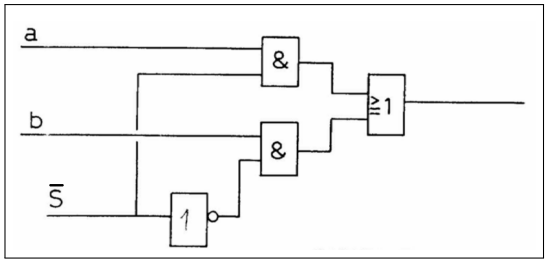


Abb. 14: Multiplexer.

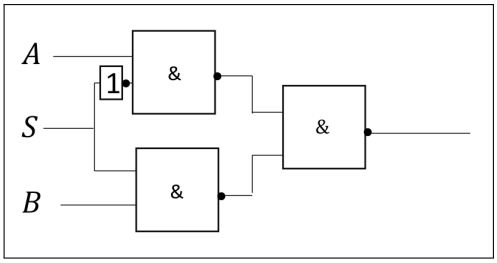


Abb. 15: Multiplexer.

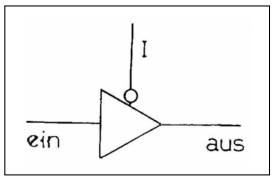


Abb. 16: Schaltbild Tristate-Treiber

Ein Tristate-Treiber hingegen dient dazu, eine Datenquelle von einer Busleitung (mit mehreren Quellen) zu isolieren, um sicher zu stellen dass jeweils immer nur eine Quelle aktiv ist. Der Tristate-Treiber ist kein logisches Gatter, sondern fungiert als eine 'Informations-Diode'. Hierfür werden Transistoren verwendet, welche über eine Eingangsspannung ansteuerbar sind. Erhält der Tristate-Treiber die benötigte Eingangsspannung (Eingang I, 'inhibit'), so lässt er keine Information durch. In der Abb. 16 ist das Schaltsymbol eines Tristate-Treibers zu sehen. 'ein' ist der Eingang, 'aus' die Ausgabe.

2.6. Addierer

Interpretieren wir eine Abfolge von Bits als eine Binärzahl, so lassen sich Schaltungen erstellen, welche zwei solche Binärzahlen addieren können.

Als **Halbaddierer** bezeichnet man die in Abb. 17 dargestellte Schaltung. Diese erstellt die Summe Z von den zwei eingegebenen Zahlen A und B sowie den Übertrag Ü.

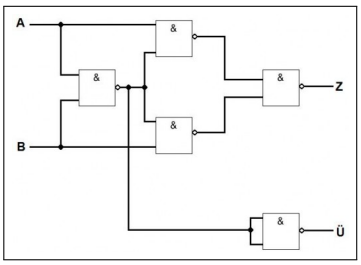


Abb. 17: Halbaddierer

Ein **Volladierer** (Abb. 19) kann eine Bitfolge addieren, welche man als Binärzahl interpretiert. Dazu muss er in der Lage sein, der vorhergehenden Übertrag mitzuberücksichtigen. Die Realisierung erfolgt durch Zusammenhängen von zwei Halbaddierern (siehe Abb. 18): Zuerst wird die Summe der Eingabewerte A und B gebildet, worauf diese zum vorhergehenden Übertrag addiert werden, was der dann ausgegebenen Summe entspricht. Die Überträge dieser zwei Zwischensummen von Halbaddierern werden über ein ODER-Gatter verbunden, so dass der Übertrag dieser Summe für die nächste Addition verwendet werden kann.

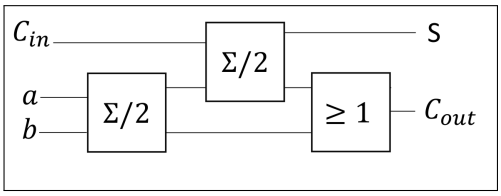


Abb. 18: Volladierer aus Halbaddierern

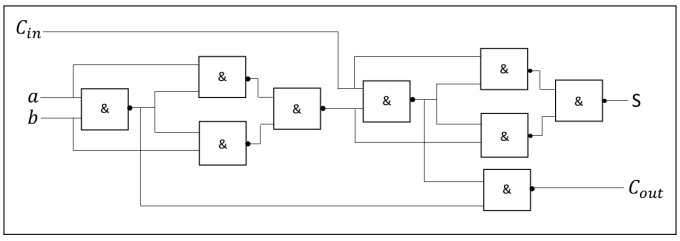


Abb. 19: Volladierer

3. Struktur und Arbeitsweise eines Rechners

3.1. Von-Neumann-Architektur

Die Von-Neumann-Architektur bildet die Grundlage für die Arbeitsweise der meisten heute bekannten Computer.

Ein Von-Neumann-Rechner beruht auf folgenden Komponenten, die bis heute in Computern verwendet werden:

- Rechenwerk: ALU (Arithmetic Logic Unit); führt Rechenoperationen und logische Verknüpfungen durch.
- Steuerwerk: interpretiert die Anweisungen eines Programms und verschaltet dementsprechend Datenquelle, -senke und notwendige ALU-Komponenten; das Steuerwerk regelt auch die Befehlsabfolge.
- Register: Dienen dem Zwischenspeichern von Datenworten.
- Speicherwerk: (Memory) speichert sowohl Programme als auch Daten, welche für das Rechenwerk zugänglich sind.
- Eingabe-/Ausgabewerk: (I/O unit) steuert die Ein- und Ausgabe von Daten, zum Anwender (Tastatur, Bildschirm) oder zu anderen Systemen (Schnittstellen).

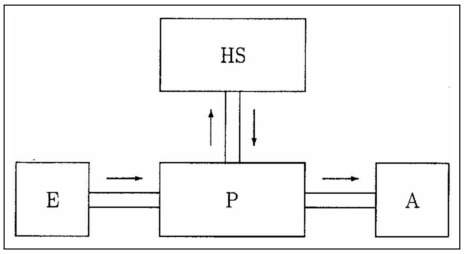


Abb. 20: klassische Von-Neumann Architektur

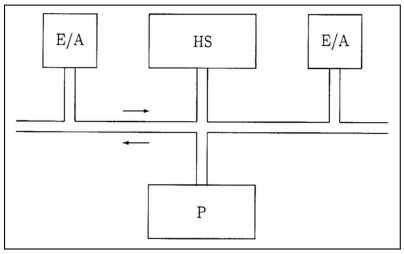


Abb. 21: Unibus mapping, memory mapped I/O

Die Register, das Rechen- und Steuerwerk bilden dabei die zentrale Prozessoreinheit, die CPU (Central Processing Unit).

Bei der klassischen Von-Neumann-Architektur gehen alle Daten über den Prozessor. Der Prozessor ist damit der Engpass im System und wird als von Neumann Flaschenhals (von Neumann bottleneck) bezeichnet, da alle Befehle und Daten durch ihn hindurch müssen.

Ebenfalls könnte man alle Komponenten, den Prozessor inklusive, durch einen Bus (siehe nächstes Kapitel) verbinden. (Unibus mapping[8], siehe Abb. 21) Dadurch wird aber der Bus zum Flaschenhals, da nur eine Quelle gleichzeitig schreibend auf den Bus zugreifen darf.

[8] Wird unibus mapping verwendet, so gibt es nur einen universellen Bus, über den alle Daten laufen. Der Prozessor hat nicht die volle Kontrolle über den Bus wie in Abb. 20, sondern ist ebenfalls mit dem Bus verbunden wie in Abb. 21. Daten auf dem Bus können so unabhängig vom Prozessor laufen. Das ermöglicht Direct Memory Acces DMA und Memory Mappen I/O.

3.2. Bus

Müssen Daten zwischen mehr als zwei Komponenten ausgetauscht werden, so lassen sich zwei prinzipielle Kommunikationsstrukturen unterscheiden, die in der Praxis auch in gemischter Form auftreten. Eine möglichkeit wäre es, alle Komponenten separat miteinander zu verbinden (direkte Kommunikation). Das bietet sich zwar bei hohen Datendurchsätzen an, allerdings quadriert sich die Anzahl der Kommunikationswege mit der Anzahl der verbundenen Komponenten.

Eine andere Möglichkeit besteht darin, einen zentralen Transportweg, der alle Komponenten miteinander verbindet, einzurichten. Dieser Transportweg wird als Bus bezeichnet. Der grosse Vorteil der Bus-Topologie besteht darin, dass man problemlos weitere Komponenten dranhängen kann. Allerdings können keine zwei Komponenten gleichzeitig auf den Bus schreiben, da ansonsten die Daten auf dem Bus zerstört würden. Die klassische Von-Neumann-Architektur sieht für die rechnerinternen Transportwege eine Bus-Topologie vor.

3.3. Memory Mapping

Wird ein sogenanntes Memory Mapping I/O verwendet, so teilen sich die Register der Ein- und Ausgabegeräte den Adressraum mit dem Speicher: Die Form der Adressen ist gleich sowohl für den Speicher als auch für E/A-Geräte. Anstelle eines Schnittstellenzugriffs wird der externe Speicher einer Komponente in den Arbeitsspeicher des Prozessors eingeblendet ("gemappt"). Die I/O-Register von elektronischen Bauelementen, mit denen angeschlossene Hardware gesteuert wird, werden in den Hauptspeicher-Adressraum abgebildet. Der Zugriff auf die Bauelemente kann dann über übliche Speicherzugriffsbefehle geschehen. Einige Adressen repräsentieren dann Speicherzellen, andere E/A-Geräte. So sind beispielsweise in Abb. 23 die oberen Zeilen (grau angezeichnet), welche mit F beginnen, für E/A- Geräte reserviert, während der (grössere) unterere (auch grau eingezeichnet) Teil des Adressraumes für den Hauptspeicher reserviert ist.

In Abb. 22 ist der Aufbau eines Memory Mapped I/O Systems skizziert. Auch wenn von der logischen Seite alle Busse gleich sind, so unterscheiden sie sich durchaus auf der physikalischen Ebene: Der Speicher-Bus beispielsweise muss viel schneller als der E/A-Bus sein, was auf der physikalischen Ebene zu Unterschieden führt.

3.4. Hauptspeicher

Der Arbeitsspeicher oder Hauptspeicher ist die Bezeichnung für den Speicher, der die gerade auszuführenden Programme oder Programmteile und die dabei benötigten Daten enthält. Er ist eine Komponente der Zentraleinheit. Der Arbeitsspeicher des Computers ist ein durch Adressen (in Tabellenform) strukturierter Bereich, der Binärwörter fester Grösse aufnehmen kann. Ein Speicherbaustein dient der flüchtigen[9] Speicherung mehrerer Datenwörter. Zum Zugriff auf die Daten besitzt ein Speicherbaustein eine Reihe von

[9]Flüchtig: Beim Ausschalten ist alles weg! Ein persistenter Sekundärspeicher wie eine Festplatte lässt sich als ein weiteres E/A-Gerät implementieren.

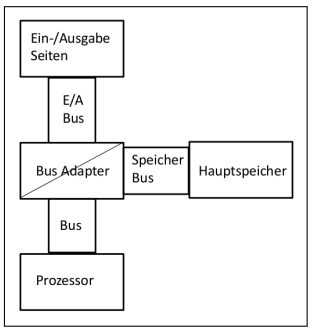


Abb. 22: Skizze eines Memory Mapped I/O - Systems

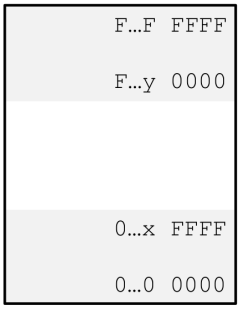


Abb. 23: Darstellung des logischen Adressraumes

Adressleitungen adr_0 , ... , adr_{m-1}, über die jedes abgelegte Datenwort direkt angesprochen werden kann (siehe Abb. A9).

3.5. Struktur eines Datenrechners

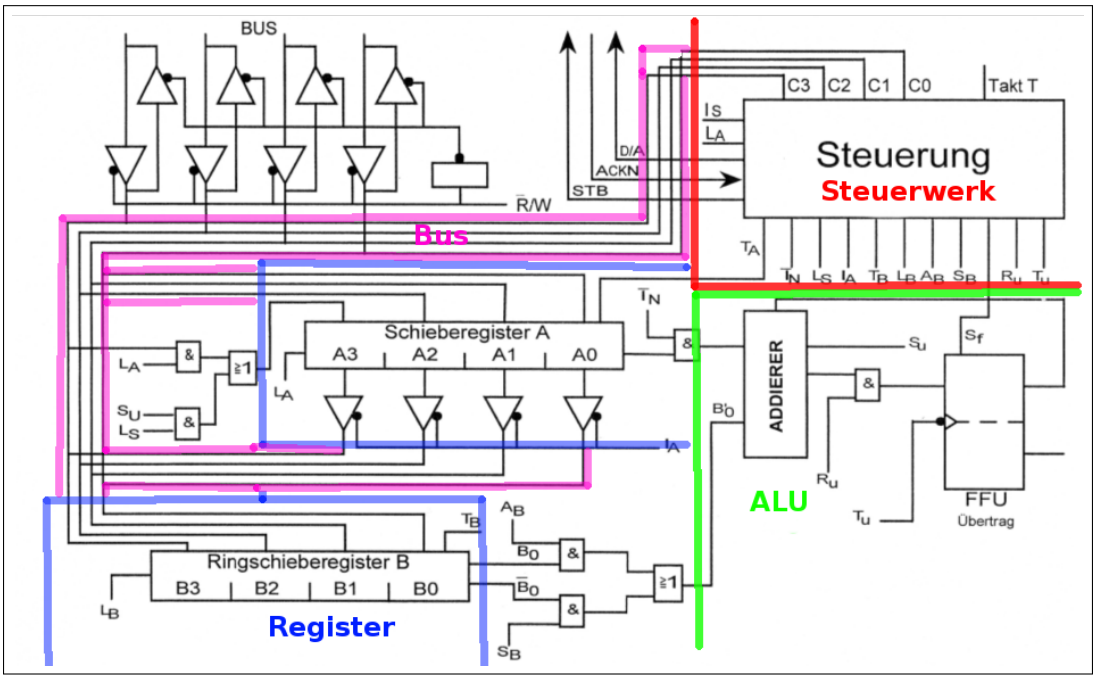


Abb. 24: Struktur eines Digitalrechners

3.6. Rechnerstrukturen: Instruktionsarchitekturen CISC und RISC

Anhand der Instruktionsarchitektur lassen sich gängige Mikroprozessoren grob in **CISC** (Complex Instruction Set Computer) und **RISC** (Reduced Instruction Set Computer) unterteilen.

CISC-Prozessoren stellen einen umfangreichen Befehlssatz zur Verfügung, der die Programmierung komplexer Aufgaben mit einer geringen Anzahl von Befehlen ermöglicht. Intern zerlegt die CPU die komplexen CISC-Befehle in mehrere aufeinander folgende Teilschritte und benötigt hierdurch mehrere Takte, bis ein Befehl vollständig ausgeführt ist. CISC-Prozessoren sind in der Lage, den Inhalt einer Speicherstelle direkt in eine andere zu übertragen, ohne dass der Wert explizit in einem Prozessorregister zwischengespeichert werden muss. Neben den elementaren Adressierungsarten verfügen viele CISC-Prozessoren über komplexe Möglichkeiten der Adressmanipulation.

RISC-Prozessoren beschränken sich auf die Bereitstellung von wenigen elementaren Maschinenbefehlen, die von der CPU in der Regel besonders effizient ausgeführt werden können. Aufgrund des beschränkten Befehlssatzes werden zur Programmierung komplexerer Aufgaben hierdurch längere Befehlssequenzen benötigt. Eines der Hauptmerkmale von RISC-Prozessoren ist die sogenannte Load-Store-Architektur. Der Datentransfer in dieser Architektur ist nur noch zwischen dem Hauptspeicher und den internen CPU-Registern möglich. Soll der Inhalt einer Speicherstelle an eine andere Stelle im Speicher verschoben werden, so sind auf einem RISC-Prozessor stets zwei Schritte notwendig. Im ersten Schritt wird der Inhalt der Speicherstelle ausgelesen und in einem CPU-Register zwischengespeichert. Im zweiten Schritt wird der Registerinhalt an den entsprechenden Zielort im Hauptspeicher zurückgeschrieben.

3.7. Pipelining

Pipelining ist eine Methode zur Leistungssteigerung der Prozessoren. Ein Befehl innerhalb der CPU wird in mehreren Phasen abgearbeitet (Fetch, Decode, Execute, Write). Die Grundidee des Pipelining ist es, dem Prozessor für jede Ausführungsphase eine separate Verarbeitungseinheit einzubauen. Die Einheiten sind so ausgelegt, dass sie völlig unabhängig voneinander operieren können. Ein Befehl wird abgearbeitet, indem er nacheinander alle Einheiten durchläuft.

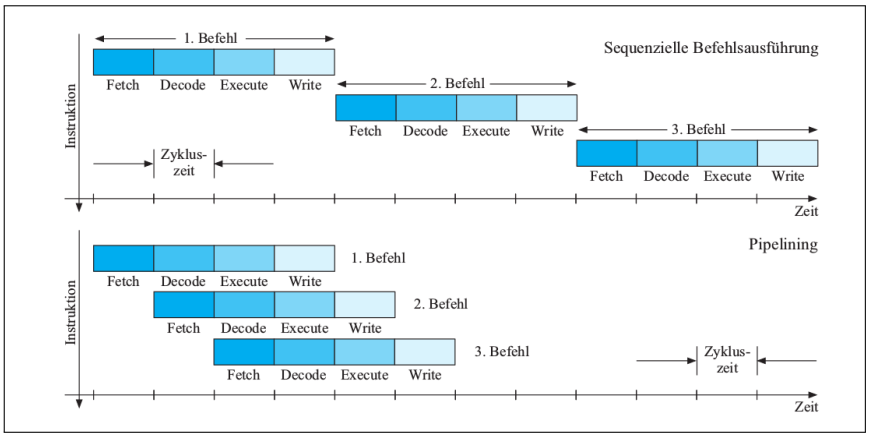


Abb. 25: Pipelining

Jede Einheit nimmt zu Beginn eines Taktzyklus[10] einen Befehl entgegen und reicht ihn zu Beginn des nächsten Taktzyklus an die Folgeeinheit weiter. Die absolute Ausführungszeit eines einzelnen Befehls sinkt nicht, jeder Befehl muss alle Ausführungsphasen nacheinander durchlaufen. Der Effizienzgewinn kommt erst bei der Ausführung langer Befehlsfolgen zum Tragen: Durch die verschränkte Ausführung zu jedem Zeitpunkt sind stets alle Funktionseinheiten ausgelastet. (Siehe Abb. **??**)

Modernere Pipelining-Architekturen beinhalten Superpipelining-Architekturen (die Befehlsausführung in noch feinere Teilschritte zerlegen, so dass jeder Befehl genau einen Takt Zeit benötigt; denn so führen die einzelnen Funktionseinheiten immer einfachere Operationen aus und lassen sich dadurch erheblich optimieren) und Superskalartechnik (den Befehlsdurchsatz durch die Mehrfachauslegung der Funktionskomponenten erhöhen: Gleichzeitig mehrere Pipelines in einem Prozessor laufen lassen) (Siehe Abb. A10).

Ein Problem der Pipeline-basierten Prozessoren liegt in bedingten Sprüngen: Soll z.B. eine while-Schlaufe erst dann unterbrochen werden, wenn ein mitlaufender Zähler einen bestimmten Wert erreicht, so kommt es zu sogenannten Pipeline-Hazards.

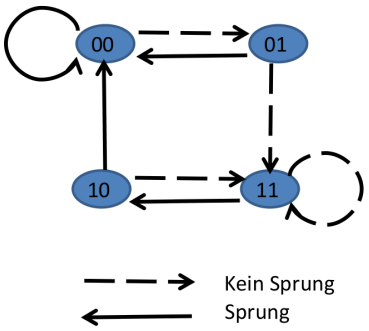


Abb. 26: 2-Bit Prädiktion. Bedeutung der 2-Bit Stati: siehe Tabelle rechts

Sprung-status	Bedeutung	Vorhersage: springen?
00	Sprung 2 oder mehrere Male ausgeführt	ja
01	Sprung zum ersten Mal nicht ausgeführt	ja
10	Sprung zum ersten Mal wieder ausgeführt	nein
11	Sprung 2 oder mehrere Male nicht ausgeführt	nein

Während der Vergleich-Befehl (ist Zähler $\geq$ Wert ?) in der Decode-Stufe noch analysiert wird, ist die Fetch-Stufe bereits damit beschäftigt, den nächsten Befehl einzulesen. Welcher Befehl dies sein muss, ist zu diesem Zeitpunkt allerdings noch überhaupt nicht klar, da die hierfür benötigte Information erst dann vorliegt, wenn der Vergleich-Befehl die Execute-Stufe passiert hat. Dieses Problem lässt sich mit unterschiedlichen Strategien bekämpfen:

- Einfügen von Wartezyklen:

  Die Befehlsaufnahme der ersten Pipeline-Stufe so lange verzögert wird, bis die Verzweigungsbedingung eines vorangegangenen Sprungbefehls vollständig ausgewertet ist. Kann sowohl hardware- als auch softwareseitig implementiert werden.

[10] Taktzyklus, nicht Takt! Ein Befehl kann so in einer Einheit bspw. in mehreren Takten abgearbeitet werden. Wichtig ist, dass alle Taktzyklen gleich lang sind.

- Spekulative Befehlsausführung:

  Nach einem Sprungbefehl wird derjenige Folgebefehl des Programmzweigs in die Pipeline eingespeist, der als wahrscheinlicher gilt. Stellt sich nach der Abareitung des Sprungbefehls heraus, dass mit der falschen Befehlssequenz weitergearbeitet wurde, so wird die Pipeline geleert (pipeline flush) und die fälschlicherweise geänderten internen Zustände der CPU werden annulliert. Erst danach erfolgt das Weiterarbeiten mit der richtigen Befehlssequenz. Im Falle der dynamischen Sprungvorhersage (bis 98% Trefferquote) analysiert der Prozessor die Sprunghistorie einer Verzweigungsbedingung und berechnet hieraus einen spekulativen Wert für das zukünftige Sprungverhalten. Die Vorhersage wird hardwareseitig in einer speziellen Tabelle abgespeichert, der sogenannten Branch prediction table. Eine populäre Methode ist die in Abb. 26 skizzierte 2-Bit-Prädiktion, die den Vorhersagewert genau dann wechselt, wenn ein Sprung zweimal hintereinander falsch prädiziert wurde.

4. Software Engineering

4.1. Wozu Software Engineering?

Beim unsystematischen "Drauf-Los" Programmieren stösst man bald auf Grenzen für die Komplexität der zu bewältigenden Aufgabenstellung. Die Anforderungen an die Software werden jedoch immer grösser, entsprechend steigt die Komplexität der Programme. Bei grösseren Projekten ist man auf Teamarbeit angewiesen und damit auf eine systematische Aufteilung der Arbeitsprozesse und der entsprechenden Programmteile.

Beim Software Engineering versucht man nun systematische Methoden für die Erstellung von Programmen zu verwenden, ähnlich zu den Konstruktionsmethoden der Maschinen-, Bau- und Elektroingenieure. Dieses Ziel ist nur schwer zu erreichen. Generell ist nicht immer klar, wie man Software "billiger, besser und schneller" produzieren kann. Heute steht hauptsächlich die Reduzierung der Gesamtkosten (Total Cost of Ownership – TCO) im Vordergrund. Diese beinhaltet nicht nur die Anschaffungskosten, sondern auch Unterhalt, Produktivität, Supportaufwand, etc.

Ziel des Software Engineering ist es, die Qualität der Programme zu steigern, die Benutzerfreundlichkeit zu verbessern, Programme an sich ändernde Bedingungen leicht anzupassend und portabel zu machen sowie die Lesbarkeit und Dokumentation zu verbessern. Effizienz ist meist zweitrangig: "The task of the software development team is to engineer the illusion of simplicity".

Die **Komplexität der Software** hat mehrere Ursachen: So kann das Problem an sich schon komplex sein; Der Programmherstellungs- und Entwicklungsprozess kann sich durch die Grösse und Arbeitsweise als schwierig erweisen; Software an sich kann sehr flexibel geschrieben werden, was zu Schwierigkeiten führt, wenn mehrere Programmierer am gleichen Code arbeiten. Man muss zwischen inhärenter, also von der Aufgabenstellung bestimmter Komplexität, und willkürlicher Komplexität, welche von den Programmierern eingeführt wird, unterscheiden. Die ihnärente Komplexität muss von

uns beherrscht werden und kann nicht umgangen werden; Die willkürliche Komplexität hingegen soll durch ein systematisches Vorgehen minimiert werden. Dabei wird das Gesamtsystem in Teilsysteme unterteilt und komplexe Systeme von funktionierenden einfacheren Systemen aus aufgebaut.

4.2. Lebenszyklen und Phasen von Softwareprojekten

Zur Bewältigung des gesamten Projektes wird dieses in Phasen unterteilt (siehe Abb. 27).

Im sogenannten Wasserfallmodell werden die Phasen der Reihe nach durchlaufen. Meist ist jedoch ein iterativer Prozess nötig. Im Wasserfallmodell beschränkt sich die Iteration aber auf benachbarte Phasen.

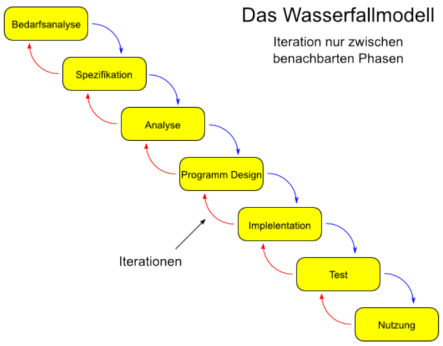


Abb. 27: Darstellung des Wasserfallmodells

Im Allgemeinen muss ein Kompromiss zwischen einem reinen "Top - Down" Vorgehen und einem iterativen Vorgehen gefunden werden. Der sogenannte Lifecycle im Software Engineering nimmt daher oft die Form einer Spirale an. Bevor einen Schritt weiter gegangen wird, sollte das Projekt validiert (erzeugen wir das richtige Produkt?) und verifiziert (machen wir das Produkt richtig?) werden.

4.3. Komponenten der strukturierten Analyse

Zu einer strukturierten Analyse gehören:

Datenfluss-Diagramm: Datenfluss-Diagramme beinhalten die Darstellung des Datenflusses eines Prozesses: Welcher Teilprozess benötigt und liefert welche Daten? (Elemente eines Datenflussdiagramms: Siehe Abb. A11) Sie dienen zur Wiedergabe der logischen und konzeptionellen Zusammenhänge. Teilprozesse werden dann wiederum in weitere Datenfluss-Diagramme unterteilt. Jede Ebene ist eine inhaltliche Abstaktion der darunterliegenden Ebene. (Siehe Abb. A12)

Daten-Lexikon: Im Daten-Lexikon werden die Datenflüsse und Speicher definiert. Welche Daten beinhaltet ein spezifischer Datenfluss? In welcher Form?

Mini-Spezifikation: Die Mini-Spezifikation sit die Beschreibung der Datentransformation für alle nicht zerlegbaren Aktivitäten. Mini-Spezifikationen beinhalten bspw. den Input/Output und welche Aktionen durchgeführt werden.

Entity Relationship Diagramme: Dienen der statischen Beschreibung der Beziehung der Daten. Beispiel siehe Abb. A13.

Zustandsdiagramme: beschreiben die Dynamik des Systems. Bei der Modellierung der System-Dynamik unterscheidet man zwischen Zuständen und Ereignissen: Ereignisse initiieren Zustandsübergänge, aber sowohl Zuständen wie auch Ereignissen können Aktionen zugeordnet werden.

5. Kommunikationssysteme

5.1. Nachrichtenkodierung

Jede Nachricht lässt sich als eine geordnete Folge von Symbolen aus einem endlichen Zeichensatz auffassen. Jedem Symbol wird im Computer eineindeutig eine Binärzahl zugeordnet. Diese sogenannten Codes sind international normiert. Ein wichtiges Beispiel ist der sogenannte ASCII Code, bei dem die Zeichen durch 7 Bits (128 Möglichkeiten) dargestellt werden (siehe Abb. A14). Codes können durch hinzufügen redundanter Bits fehlererkennende oder fehlerkorrigierende Eigenschaften bekommen.

5.2. Struktur von Kommunikationssystemen

Ein Kommunikationssystem besteht im Allgemeinen aus folgenden funktionalen Komponenten:

Nachrichtenleitung: Soll die Daten unversehrt übertragen. Dabei kann die Nachricht parallel (bspw. 8 Bits gleichzeitig über 8 Drähte) oder seriell (8 Bits hintereinander an einem Draht) erfolgen.

Nachrichtenleitungen können anhand ihrer Struktur unterschieden werden: Als einen dedizierten Weg bezeichnen wir eine Nachrichtenleitung, welche genau zwei Datenstationen miteinander verbindet. Ein allgemeiner Weg (Bus) hingegen verbindet mehr als zwei Datenstationen miteinander.

Ausserdem lassen sie sich auch durch ihr Richtungsverhalten charakterisieren: simplex-Leitungen können nur in eine Richtung Daten übertragen, also ist hierbei keine Antwort möglich. Das ist bei der duplex- bzw. halbduplex-Leitung möglich, wobei bei der halbduplex-Leitung eine gleichzeitige Datenübertragung in beide Richtungen nicht möglich ist.

Nachrichtenvermittler: Leitet Nachrichten weiter, wählt Adressen und Nachrichtenleitungen. Man unterscheidet zwischen Leitungsvermittlern[11] und Paketvermittlern[12]

Datenstationen: Die eigentlichen Einheiten, die die Nachrichten untereinander austauschen.

[11] einer Nachrichtenverbindung wird zeitweilig ein durchgeschalteter Übertragungskanal mit konstanter Bandbreite zugeordnet, der dieser Verbindung dann zur exklusiven Nutzung zur Verfügung steht, auch wenn keine Informationen übertragen werden. Z.B. das Telefonnetz.

[12] Hierbei werden längere Nachrichten in einzelne Datenpakete aufgeteilt. Die Pakete durchqueren als unabhängige und eigenständige Einheiten das Netz (Vgl. Briefpost). Z.B: Internet

Wir wollen nun die Struktur von Kommunikationssystemen anhand des IEEE 488 Bus bzw. des GPIB (General Purpose Interface Bus) anschauen. Dieser wurde dazu entwickelt, um Kommunikation zwischen Computer und Instrumenten zu ermöglichen. Der GPIB hat insgesamt 24 Leitungen (siehe Abb. 28): Davon sind 5 Steuerleitungen, 3 Handshakeleitungen und 8 Datenleitungen. Den restlichen 7 Pins sind spezielle Aufgaben zugewiesen, u.a. das logische Masse-Pin (LOGIC GND) für die Übertragung logischer Signalstandards und sechs weitere Grounds und der SHIELD-Pin zum Beenden der Übertragung, z.B. wenn Rauschsignale gesendet werden.

Jedes mit dem Bus verbundene Gerät muss als Talker, Listener, Talker und Listener oder Controller definiert werden (Siehe Abb. 29). Ein Listener kann Daten empfangen, ein Talker kann Daten verschicken, ein Talker und Listener kann beides. Der Controller kann auch schreiben und empfangen, jedoch kann er auch Befehle an alle Geräte senden wie beispielsweise dass sie Daten annehmen sollen, Daten verschicken sollen oder eine andere Operation einzuleiten. Damit kann er die Richtung der Kommunikation bestimmen.

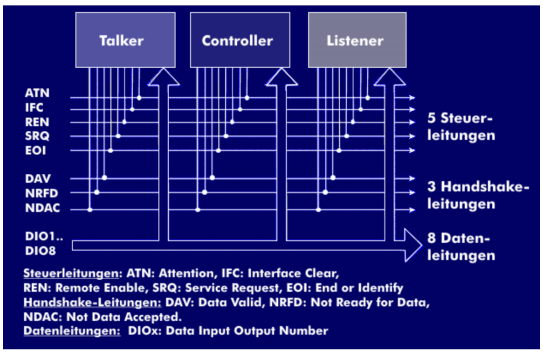


Abb. 28: Leitungsbelegung GPIB

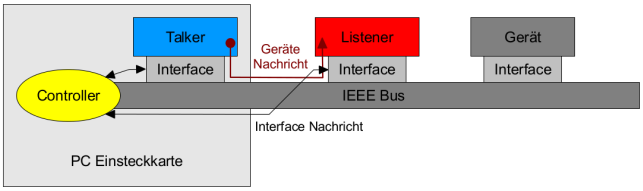


Abb. 29: Schema GPIB

Um eine Datenübertragung anzukünden wird ein sogenanntes Handshake-Verfahren eingeleitet. Dabei muss zuerst der Talker den Bus frei vorfinden, was durch den Zustand der drei Handshakeleitungen angezeigt wird. Wenn der Talker bereit ist, Daten zu senden, wartet er darauf, dass die Listener bereit werden, was über der NRFD-Leitung (Not Ready For Data) angezeigt wird. Sind alle Listener bereit, so gibt der Talker über die DAV-Leitung (Data AVailible) an, dass er Daten senden will. Wenn die Listener das DAV-Signal erhalten, stellen sie wieder die NRFD, aber auch die NDAC-Leitung (Not Data Accepted) um. Erst wenn alle Listener anzeigen, dass sie die Daten erhalten haben, werden die Daten von den Datenleitungen gelöscht.

5.3. Regeln des Nachrichtenaustausches: Protokolle

Ein Protokoll definiert die Regeln und den zeitlichen und logischen Ablauf der Nachrichtenübertragung. Eine grosse Zahl solcher Protokolle sind durch internationale Konventionen festgelegt. Die Protokolle können hardware- und softwaremässig implementiert werden. Insbesondere wird in sogenannten Kommunikations-Prozeduren der exakte Auf-

bau von Nachrichten (Syntax) und die Bedeutung der darin enthaltenen Steuer-, Status- und Fehlernachrichten (Semantik) festgelegt.

Kommunikationsprotokolle können auf verschiedenen Ebenen arbeiten und zu einem sogenannten Protokollstapel zusammengefasst werden (siehe nächstes Kapitel).

6. LAN / WAN

6.1. ISO/OSI Schichtenmodell

Das **ISO/OSI Schichtenmodell** (International Organization for Standardization / Open System Interconnections) definiert eine hierarchische Architektur für den Austausch von Informationen. Die verschiedenen Verbindungsebenen werden in einem sogenannten **Protokollstapel** miteinander verbunden. Höherliegende Schichten müssen sich nicht um die Details und die konkrete Realisierung der darunterliegenden Schichten kümmern.

Man unterscheidet zwischen der **Anwendungsebene**, der **Transportebene** und der **Netzwerkebene**. Diese drei Ebenen sind ihrerseits weiter unterteilt, so dass sich insgesamt 7 Schichten ergeben. (Siehe Abb. 30) Die Kommunikation zwischen den Schichten innerhalb eines Knotens (vertikale Kommunikation) wird zum Beispiel über festgelegte Programmschnittstellen realisiert (API, Application Programming Interfaces). Die Kommunikation zwischen den gleichen Schichten auf unterschiedlichen Knoten (horizontale Kommunikation) wird über standardisierte Protokolle abgewickelt.

Die Funktionen der einzelnen Schichten sind:

1. **Bitübertragung: Physikalische Schicht**

   Eigentliche Datenübertragung; Bitstrom von Knoten zu Knoten.

2. **Sicherheitsschicht/Verbindungsschicht**

   Fasst Folgen von binären Informationen zu Datenpaketen zusammen

3. **Vermittlungsschicht**

   Bestimmung eines optimalen Weges durch die verzweigten Netzwerke (Routing)

4. **Transportschicht**

   Protokolle dieser Schicht bieten dem Anwender eine logische Verbindung zwischen zwei Kommunikationsendpunkten (Peers)

5. **Kommunikationsschicht/Sitzungsschicht**

   Stellt Sprachmittel zur Verfügung, die zur Eröffnung einer Kommunikationsbeziehung (Session, Sitzung), ihrer geordnete Durchführung und ihrer Beendigung notwendig sind.

6. **Darstellungsschicht/Präsentationsschicht**

   Beispielsweise Umrechnung von Zahlen- oder Bildformaten.

7. **Anwendungsschicht**

   Eigentliche Anwendung.

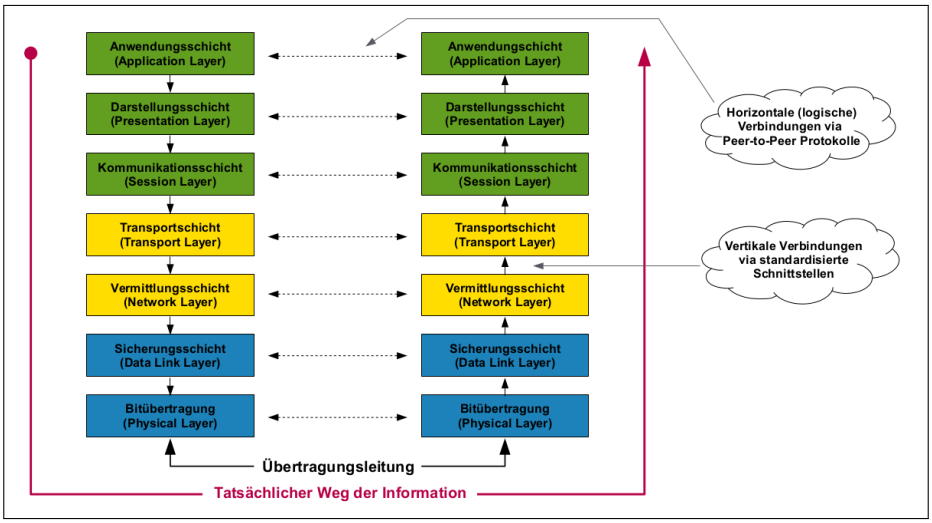


Abb. 30: Graphik des ISO/OSI Schichtenmodells. Grün ist die Anwendungsebene, gelb die Transportebene und blau die Netzwerkebene.

6.2. Beispiel: TCP/IP Protokoll

Das Internet kann als Netz von Netzwerken und Rechnern (Hosts) aufgefasst werden. Zur Übermittlung der Daten zwischen zwei Anwendungsprogrammen müssen folgende Aufgaben erfüllt werden:

Adressierung: Sogenannte IP-Adressen erlauben eindeutige Identifizierung des Zielrechners.

IP-Adressen sind bei IPv4 aus 4 Bytes aufgebaut, aufgeteilt in eine Netzwerkadresse und eine Hostadresse.

Bei IPv6 sind es 8 Blöcke zu 16 Bit (normalerweise hexadezimal dargestellt), aufgeteilt in einen 64 Bit Präfix (kann wieder in eine Netzwerkadresse und Hostadresse unterteilt werden) und einen 64 Bit Interface Indentifier (eindeutiger Name).

Routing: Sogenannte Gateways oder Router übermitteln die Daten an das korrekte Netzwerk.

Jeder Host und jeder Gateway hat eine sogenannte Routing-Tabelle. Sie ordnet dem Netzwerkteil einer IP Adresse ein Gateway zu. Statische Routing-Tabellen können von Hand von Systemadministratoren erstellt werden, dynamische können automatisch von Routing Protocols erstellt werden, so dass sie sich dynamisch an die veränderte Netzsituation anpassen können.

Multiplexing: Sogenannte Protokoll- und Port-Nummern leiten die Daten an den richtigen Prozess und damit an den richtigen Benützer innerhalb des Rechners.

Beim Aufstieg eines IP Paketes auf die Transportebene und anschliessend auf die Anwendungsebene wird der Zielprozess im Rechner durch Multiplexing bestimmt. Hierbei ist ein sogenannter Socket ein Kommunikationsendpunkt. Er ist eindeutig definiert durch die Angabe der IP Adresse + Protokollnummer + Portnummer. Jede Punkt-zu-Punkt Verbindung im Internet ist eindeutig durch ein Socket-Paar definiert.

6.3. DNS

DNS (Domain Name Service) ist ein Protokoll der Anwendungsebene im OSI- Schichtenmodell mit der Aufgabe, die IP Adresse in leichter zu merkende Domainnamen umzuwandeln. Die Realisierung erfolgt in der Form einer weltweit verteilten, hierarchisch strukturierten Datenbank (siehe Abb. A15) mit lokalen Cache-Mechanismen, so dass die lokal benötigten Namensinformationen effizient zur Verfügung stehen. Die Verwaltung der Namensräume werden nach unten delegiert: Jede Domain hat ihren Domain Name Server, dessen Cache mit mindestens den Adressen der weltweit definierten Root Name Servern initialisiert wird. Domains werden beim NIC (Network Information Center) beantragt.

7. Verbindung Rechner-Prozess

7.1. Definition Prozess

Unter einem **Prozess** versteht man einen Vorgang zur Umformung oder zum Transport von Stoff, Energie oder Information.
Unter einem **technischen Prozess** versteht man ein System, dessen Zustandsgrössen überwiegend physikalische Grössen sind, die gemessen, gesteuert und geregelt werden können. D.h. unsere Experimente in der Experimentalphysik und anderen experimentellen Naturwissenschaften sind technische Prozesse.

7.2. Arten der Prozesskopplung

Prozesskopplungsarten werden anhand der Kopplung durch das Kopplungselement (Interface = Schnittstelle) und der Enge der Kopplung an den Prozess unterschieden.

7.2.1. Indirekte Prozesskopplung (Off-line operation)

Bei der indirekten Prozesskopplung wird der Computer fast ausschliesslich als Rechnerelement benutzt. Experimentdaten werden manuell ohne Computerunterstützung gesammelt, um dann mithilfe des Computers bearbeitet und analysiert zu werden. Der

Computer hat also keinen Einfluss auf die Durchführung des Experimentes und die Datenerfassung. Der Mensch ist das (universelle) Kopplungselement; deshalb wird diese Kopplungsart als indirekt bezeichnet. Ein Schema der indirekten Prozesskopplung ist in Abb. A1 dargestellt.

7.2.2. Direkte Prozesskopplung (On-line operation)

Bei der direkten Prozesskopplung bewegen sich Daten direkt zwischen Computer und Experiment. Die technische Schnittstelle ist im Rechner (und gegebenfalls im Expriment) implementiert (vgl. Messstrecke, Kap. 7.6.1). Der Computer verarbeitet die Daten und trifft Entscheidungen anhand des ausgeführten Programmes. Der Vorgang des Experimentes kann nun in Intervallen, welche unter dem Millisekundenbereich liegen, kontrolliert und gesteuert werden. Ein Schema der direkten Prozesskopplung ist in Abb. A2 dargestellt.

7.2.3. Innige Prozesskopplung (In-line operation)

Mit der Entwicklung des Rechners können und werden nun Computer/Mikrocomputer direkt in die Instrumente integriert. Die "intelligenten" Messgeräte werden an den Rechner über den Bus gekoppelt. Somit können das Experiment gesteuert und die Daten erfasst werden, ohne den Computer vorher auf den Prozess programmieren zu müssen. Der Nachteil der innigen Prozesskopplung ist jedoch, dass wenn man die Instumente nicht mehr programmieren muss, man sie oft nicht mehr programmieren kann, um sie den jeweiligen Anforderungen anzupassen. Ein Schema der innigen Prozesskopplung ist in Abb. A3 dargestellt.

7.3. Ausgabe von Steuergrössen

Man unterscheidet zwischen digitalen Signalen mit nur zwei möglichen Zuständen und analogen Signalen mit einem Kontinuum von möglichen Zuständen.

Für digitale Steuersignale wird die Verbindung über eine digitale Ausgabe-Schnittstelle realisiert, welche die Potentialtrennung zwischen Experiment und Rechner, die Pegelanpassung (z.B. 0 .. 5 V $\rightarrow$ -10 .. 10 V), die Synchronisation des Signalaustausches, die Datenzwischenspeicherung und die Erzeugen von Taktsignalen übernimmt.

Für analoge Steuersignale muss der analoge (d.h kontinuierliche) Wert aus der entsprechenden binären Zahlendarstellung im Computer umgewandelt werden, d.h. eine Digital-Analog-Umwandlung stattfinden (DAC = Digital-to-Analog-Converter). Für den Einsatz eines DACs sind Bereich der Ausgangsspannung, Auflösung, Integrale Linearität, differenzielle Linearität, Nullpunktabweichung, Dauer der Einstellzeit eines stabilen Ausgabewertes sowie Zeit- und Temperaturstabilität wichtig. Das Prinzip der technischen Realisierung eines DAC ist in Abb. A16 dargestellt.

7.4. Erfassen von Messgrössen

Auch hier gilt die gleiche Unterscheidung in digitale und analoge Signale.

Digitale Eingangssignale werden über eine digitale Schnittstellenkarte in den Computer eingelesen. Diese kann die Potentialtrennung zwischen Experiment und Rechner, die Pegelanpassung und Signalregenerierung, die Synchronisation des Signalaustausches, die Datenzwischenspeicherung, die Auswertung von Signalflanken, das Zählen (maximale Zählfrequenz) von Impulsen und die Zeitmessung (Auflösung, Stabilität der internen Taktfrequenz) übernehmen.

Die meisten Messgrössen sind jedoch analoger Natur und müssen zur Verarbeitung im Computer zunächst in einen binären Zahlenwert umgewandelt werden (ADC: Analog-to-Digital-Converter).

Die wichtigsten Kenngrössen eines ADCs sind der Messbereich (typisch ± 10V), die Auflösung (Anzahl Bits), die Konversionszeit (bestimmte maximale Abtastrate, siehe später), die Linearität (differentielle, integrale), der Offset (Zahlenwert bei Eingangsspannung = 0) sowie die Zeit und Temperaturstabilität. Ein ADC ist oft mit einem Multiplexer gekoppelt, so dass verschiedene Eingangssignale der Reihe nach von einem einzigen ADC gemessen werden können. Das Prinzip der technischen Realisierung eines ADCs ist in Abb. A17 dargestellt.

Der ADC arbeitet nach dem Prinzip der sukzessiven Annäherung. Dabei baut er bei jedem Schritt einen Vergleichswert neu auf. Einfache sukzessive Approximation setzt dabei pro Schritt ein Bit um. Das Eingangssignal wird mittels Intervallschachtelung eingegrenzt: Man beginnt beim Bit der höchsten Stelle. Bei jedem Schritt wird das entsprechende Bit nur dann dazuaddiert, wenn die Summe mit dem momentanen Wert kleiner ist als der zu beschreibende Wert.

7.5. Kontroll-Optionen

Abfragemöglichkeiten:

Polling: 'Pull'-Prinzip. Der Computer fragt den Prozess regelmässig ab, holt sich die Daten ab. Einfach zu implementieren, aber schlechte Ausnutzung der CPU-Ressourcen. Schnelle Reaktion möglich, aber schwierig mehrere Vorgänge gleichzeitig zu kontrollieren.

Interrupts: 'Push'-Prinzip. Der Prozess benachrichtigt den Computer, dass etwas wichtiges passiert ist. Der Computer unterbricht (interrupt) seine Arbeit um den Prozess zu bedienen. Schnelle Reaktion möglich, ohne die CPU zu belasten, aber schwierig zu implementieren.

Datenübertragungsmöglichkeiten:

Direkter Speicherzugriff: (Direct Memory Access = DMA) Transport der Daten ohne CPU. Ein spezieller DMA Kontroller übernimmt die Kontrolle des Speicherbusses und transferiert die Daten zwischen Speicher und Peripherie.

Programmierte Datenübertragung: Die CPU sorgt für die Übertragung der Daten. Spezielle I/O Befehle oder Memory Mapped I/O möglich (Siehe Kap. 3.3).

7.6. Verarbeitung analoger Messwerte

7.6.1. Messkette

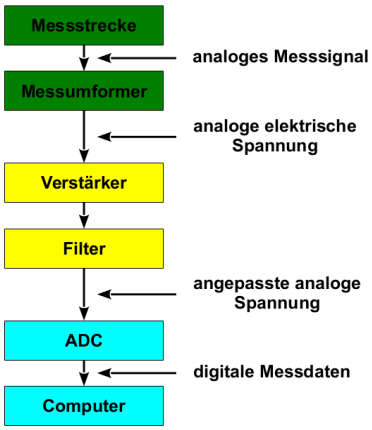


Abb. 31: Messkette: Der Weg vom analogen Messsignal zu digitalen Messdaten

Messen heisst, eine zu messende Grösse als Vielfaches einer allgemein anerkannten Einheitsgrösse derselben physikalischen Dimension zu bestimmen. Dies geschieht durch experimentellen Vergleich mit einer Massverkörperung dieser Einheit.

Der Vergleich findet innerhalb einer Messkette statt (siehe Abb. 31), an deren Ende das Resultat der Messung in computerlesbarer Form zur Verfügung steht. Verstärker und Filter dienen dazu, die Messfehler zu minimieren.

Die Komponenten der Messkette sind:

Messstrecke: Teil des Messobjektes, innerhalb dessen der Messwert durch die Rückwirkung der Messeinrichtung beeinflusst wird. Bei einer direkten Messung findet der Vergleichsvorgang in der Messstrecke statt (z.B. Balkenwaage), bei einer indirekten Messung findet der Vergleich erst später in der Messkette statt.

Messumformer: Er wandelt die physikalische Grösse wie Temperatur, Druck usw. in ein analoges Messsignal um, das in Form einer elektrischen Spannung/Stromstärke übertragen wird. Die Übertragungsfunktion des Messfühlers (Zusammenhang zwischen der Messgrösse und der Spannung) soll umkehrbar eindeutig sein, aber nicht unbedingt linear.

Verstärker: Passt den Spannungsbereich an den Spannungsbereich des ADCs an. Die Verstärkung V sollte so gewählt werden, dass der maximal auftretende, analoge Spannungswert der Messgrösse gleich der maximalen Eingangsspannung des ADCs ist. Der Messfehler in der Amplitude der physikalischen Grösse ist dann minimal ("Minimierung des Fehlers im Amplitudenraum").

Filter: Er wird bei der Erfassung von zeitabhängigen Signalen eingesetzt, um den Frequenzgang des Messsignals so zu verändern, dass Übertragungsfehler minimal wer-

den. Insbesondere soll das Problem der sogenannten Aliasfrequenzen behoben werden (siehe später). Filter und Verstärker sind oft in einem Gerät integriert.

Analog-Digital-Wandler: Er wandelt die Spannung in eine digital dargestellte Zahl. Es gelten alle Einschränkungen des digitalen Computers, insbesondere gibt es eine kleinste und eine grösste Zahl sowie einen endlichen kleinsten Unterschied zwischen zwei Zahlen, der den Messfehler mitbestimmt. Der Unterschied zwischen dem ursprünglichen analogen Signal und dem amplituden-quantisierten Signal führt zu einem sogenannten Quantisierungsrauschen (Fehler im Amplitudenraum, siehe Abb. A18 und A19).

7.6.2. Erfassung zeitabhängiger Signale, Nyquist Abtasttheorem, Aliasfrequenzen

Um ein zeitabhängiges Signal zu erfassen, sollte die Amplitude des Signals in regelmässigen, möglichst kurzen Zeitabständen abgetastet werden. Wichtige Kenngrössen eines ADCs für den Abtastvorgang sind:

Konversionszeit t_C : Zeit für die Umwandlung der Spannung in eine Zahl. Werte zwischen 10 ms und 5 ns.

Aperture Time t_A : Zeit, in der das Signal möglichst konstant gehalten werden sollte (im Allgemeinen ist $t_A << t_C$). Oft wird das Signal in einer "sample-and-hold" Schaltung (analog) zwischengespeichert.

Abtastrate ω_S : Frequenz, mit der der ADC Messungen ausführen kann.

Wie viel Information über das ursprüngliche Signal geht beim Abtasten verloren? Die Antwort liefert das **Abtast-Theorem von Nyquist**: Es geht keine Information verloren, wenn die Abtastfrequenz ω_S (S für Sampling) grösser oder gleich der doppelten Maximalfrequenz ist, die im ursprünglichen Signal enthalten ist (entsprechend einer Fourierzerlegung des Signals): $\omega_S \geq 2 \cdot \omega_{max}$. Die halbe Abtastfrequenz, also die Maximalfrequenz für die Signale, $\frac{1}{2}\omega_S$ wird auch als Nyquist-Frequenz ω_{Ny} oder Nyquist-Grenze bezeichnet[13].

Wenn die Abtastfrequenz zu niedrig gewählt wird, also $\omega_S < \omega_{Ny} = 2 \cdot \omega_{max}$, dann treten sogenannte **Aliasfrequenzen** auf: die Originalfrequenz ω wird auf die Aliasfrequenz $\omega_S - \omega$ "gespiegelt".[14]

Kommen Frequenzanteile vor, die höher als die Nyquist-Frequenz sind, so werden diese als niedrigere Frequenzen interpretiert. Die höheren Frequenzen geben sich sozusagen als eine andere (niedrigere) aus (siehe Abb. 32).

[13] Anm. des Autoren: Ich habe sich widersprechende Definitionen im Internet gefunden. Mal wird die Nyquist-Frequenz als die halbe Samplingfrequenz, mal als doppelte Maximalfrequenz definiert. Hier ist die Definition nach der Vorlesung

[14] Hier könnte es vorkommen, dass die Berechnung der Aliasfrequenz eine negative Frequenz als Ergebnis liefert. Das erscheint zunächst nicht physikalisch sinnvoll, aber das Vorzeichen der Frequenz hat eine andere Bedeutung. Es gibt uns an, wie die Kurve verläuft, während die Frequenz selbst, also wie oft die Welle ihre Periode pro Zeitintervall ausführt, natürlich nur ≥ 0 sein kann. Kleine Erinnerung: $\sin(\omega t) = -\sin(-\omega t)$ und $\cos(\omega t) = \cos(-\omega t)$. Vergleiche dazu auch Abb. A20

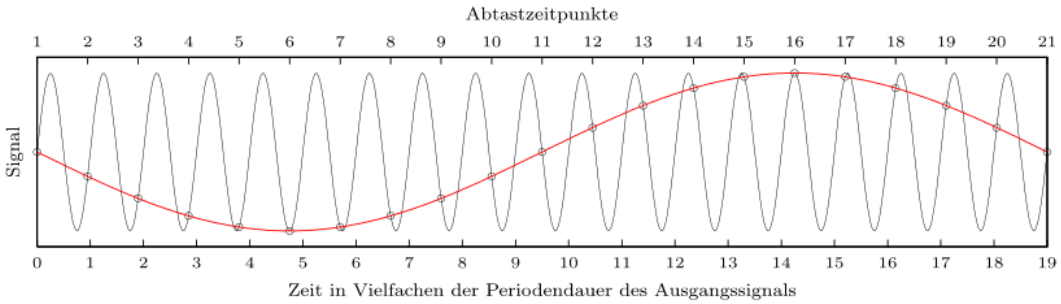


Abb. 32: Veranschaulichung des Alias-Effekts. Ein kontinuierliches Ausgangssignal (schwarze Linie) wird mit einer ungeeigneten Abtastfrequenz $< \omega_S$ diskretisiert. Aus den erhaltenen Messwerten (Kreise) entsteht ein verfälschtes Signal (rote Linie).

Zur Vermeidung solcher Aliasing-Effekte wird das Eingangssignal durch einen Tiefpass gefiltert. Als Tiefpass bezeichnet man in der Elektronik solche Filter, die Signalanteile mit Frequenzen unterhalb ihrer Grenzfrequenz annähernd ungeschwächt passieren lassen, Anteile mit höheren Frequenzen dagegen dämpfen. (Schaltplan: Siehe Abb. A21) Diese Filterung muss vor der Digitalisierung geschehen – eine nachträgliche Korrektur von Alias-Effekten ist nicht mehr möglich.

Zur Prüfung, ob ein Frequenzanteil im digitalisierten Signal echt ist oder von einer Aliasfrequenz herrührt, kann man die Messung bei einer andern Abtastfrequenz ω_{S2} wiederholen. Die echten Frequenzen bleiben dabei fest, während die Aliasfrequenzen auf den neuen Wert $\omega_{S2} \breve{} \omega = 2\omega_{Ny} - \omega$ verschoben werden

7.7. Transientenrekorder

Transientenrekorder sind Systeme zur Datenerfassung, die mit sehr hohen Abtastraten und hohen Speichertiefen arbeiten können. Daher eignen sie sich sehr gut für Messaufzeichnungen im Hochgeschwindigkeitsbereich, insbesondere für nicht periodisch auftretende Signale (Transienten), z. B. bei Crashversuchen. Der aufzuzeichnende Zeitraum kann durch gezieltes Triggern auf ein bestimmtes Ereignis eingegrenzt werden.

Die am Eingang anliegenden Signale werden im Takt der Zeitbasis digitalisiert und hintereinander im Speicher abgelegt. Dabei wird der Speicher zyklisch verwendet (siehe Abb. 33), also wird nach Erreichen des letzten Speicherplatzes beim ersten Speicherplatz fortgefahren und die dort stehenden Werte überschrieben. Gleichzeitig wird auf das Eintreten des Trigger-Ereignisses geprüft. Tritt dieses ein, so wird die Erfassung für eine festgelegte Anzahl von Taktschritten fortgesetzt und dann beendet. Der Speicher enthält dann sowohl vor als auch nach dem Triggerereignis aufgetretene Werte, wobei die Gesamtzahl der Messwerte der vollen Speicherlänge entspricht. Die Werte werden dann durch einen Mikroprozessor oder PC ausgelesen und weiterverarbeitet.

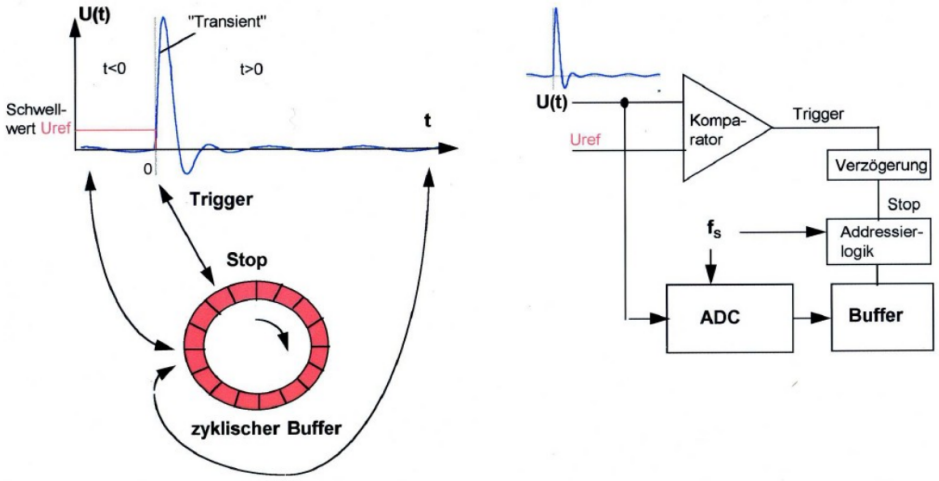


Abb. 33: Schematische Funktionsweise von Transientenrekordern

8. Digitale Signalverarbeitung

8.1. Einführung

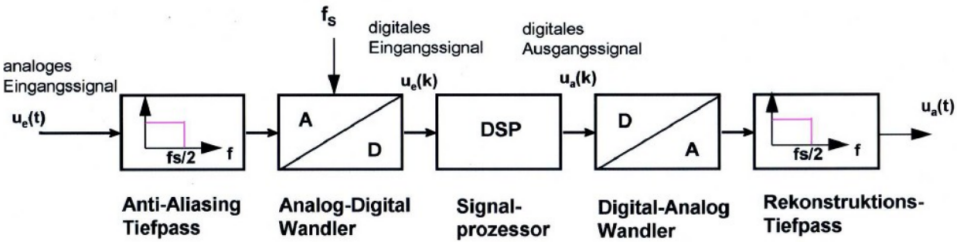


Abb. 34: Gesamtsystem einer digitalen Verarbeitung von analogen Signalen

Das Gesamtsystem einer digitalen Verarbeitung von analogen Signalen ist in Abb. 34 dargestellt. Die zentrale Verarbeitung der Daten kann von einem herkömmlichen Prozessor oder für schnelle Anwendungen von einem sogenannten DSP (Digitaler Signal Prozessor) übernommen werden, deren Befehlssatz speziell an die Bedürfnisse der digitalen Signalverarbeitung angepasst sind und daher höhere Verarbeitungsgeschwindigkeiten erreichen.

Digitale Signalverarbeitung hat viele Vorteile wie besseren Rauschabstand, keine Probleme mit Fertigungstoleranzen wie bei analogen Bausteinen, keine Alterserscheinungen und verlustfreie Datenübertragung. Ihr Einsatz erfolgt nicht nur bei zeitlich veränderlichen Signalen: Die Methoden der digitalen Signalverarbeitung lassen sich auch z.B. auf räumlich veränderliche und auf mehrdimensionale Grössen erweitern, wie z.B. in der Bildverarbeitung (zweidimensional, dreidimensional).

8.2. Fouriertransformationen

8.2.1. Fourierreihen und Fouriertransformationen

Die Theorie der Fourierreihen erlaubt es, ein periodisches Signal (Funktion) mit der Frequenz $\omega_0 = 1/T$ in eine unendliche Summe (Reihe) von harmonischen Schwingungen der Frequenzen $\omega_0, 2\omega_0, 3\omega_0, ...$ zu zerlegen. Unter einer harmonischen Schwingung ist eine Funktion/Signal mit einer Funktionsgleichung der Form $f(t) = A\cos(\omega t)$ oder $f(t) = A\sin(\omega t)$ oder $f(t) = A\cos(\omega t) + B\sin(\omega t) = C\cos(\omega t + \varphi)$ mit $\varphi \in [0, 2\pi[$ zu verstehen.

Die allgemeine Fourierreihe einer solchen beliebigen periodischen Funktion f mit Periode $T = 2\pi/\omega_0$ lautet:

$$f(t) = \frac{a_0}{2} + \sum_{n=1}^{\infty} (a_n \cos(n\omega_0 t) + b_n \sin(n\omega_0 t)) = \sum_{n=-\infty}^{\infty} c_n e^{-i\omega_0 t}$$

Die Koeffizienten a_n, b_n bzw. c_n lassen sich folgendermassen aus f berechnen:

$$a_n = \frac{2}{T}\int_0^T f(t)\cos(n\omega_0 t)dt \quad b_n = \frac{2}{T}\int_0^T f(t)\sin(n\omega_0 t)dt \quad c_n = \frac{2}{T}\int_0^T f(t)e^{-in\omega_0 t}dt$$

Haben wir es mit aperiodischen Funktionen zu tun, können wir diese beschreiben, indem wir ihre Periode gegen unendlich streben lassen: $T \to \infty$. Damit kommen wir zu Fourier-Transformationen:

Die folgenden Beziehungen gelten für $T \to \infty$ zwischen der sog. Zeitfunktion f und der sog. Spektralfunktion F:

$$F(\omega) = \int_{-\infty}^{\infty} f(t)e^{-i\omega t}dt \quad \text{und} \quad f(t) = \frac{1}{2\pi}\int_{-\infty}^{\infty} F(\omega)e^{i\omega t}d\omega$$

Die Transformation $f(t) \to F(\omega)$ wird Fourier-Transformation genannt.

Komplexe Funktionen können in Real- und Imaginärteil oder in Amplitude und Phase zerlegt werden: $F(\omega) = \mathbb{R}(\omega) + i\mathbb{I}(\omega) = |F(\omega)|e^{i\theta(\omega)}$

Es lassen sich bereits Aussagen über die Fouriertransformierte bzw. -reihe einer Funktion f anhand ihrer Symmetrieeigenschaften[15] treffen:

	Fourierreihe	**Fouriertransformierte**
f **ist gerade**	$\Rightarrow$ alle $b_n = 0$	$\Rightarrow$ F ist reell
f **ist ungerade**	$\Rightarrow$ alle $a_n = 0$	$\Rightarrow$ F ist rein imaginär
f **ist weder gerade noch ungerade**		$\Rightarrow$ F ist komplex.

Die Fouriertransformierte von periodischen Signalen enthält diskrete Frequenzen, während aperiodische Signale zu kontinuierlichen Spektren führen. Einige häufig vorkommen-

[15] Eine reelle Funktion $f : D \to \mathbb{R}$ heisst gerade, wenn für alle $x \in D$ gilt: $f(x) = f(-x)$ (Vgl. $\cos(x)$). Die heisst ungerade, wenn für alle $x \in D$ gilt: $f(x) = -f(-x)$ (Vgl. $\sin(x)$).

de Fouriertransformationen sind in den Abb. A22 und A23 dargestellt. Bemerkenswert ist:

- T-periodische Funktionen ergeben Delta-Funktionen[16] mit dem Abstand $\omega_0 = 2\pi/T$; Auch eine periodische Folge von Delta-Peaks ist fouriertransformiert wieder eine Folge von Delta-Peaks. (Siehe Abb. A22 zuoberst)
- Aperiodische Funktionen ergeben kontinuierliche Transformierte und umgekehrt. Zudem ergibt eine (einzelne!) Delta-Funktion fouriertransformiert eine Konstante und umgekehrt.

8.2.2. Abgetastetes Signal

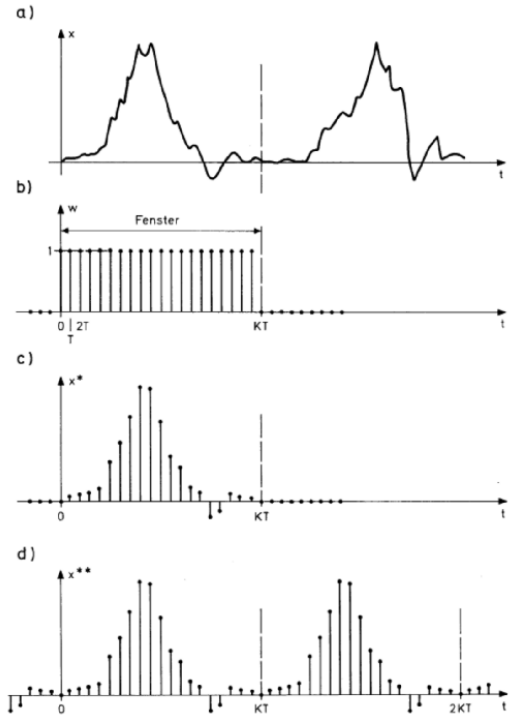


Abb. 35: Multiplikation des Signals $x(t)$ mit der Fensterfunktion $w(t)$.
a) Signal $x(t)$
b) $w(t)$ Multiplikation der Fensterfunktion (heir: Rechtecksfenster) mit der Abtastfunktion $\sum_j \delta(t - j \cdot T)$
c) Multiplikation des Signals $x(t)$ mit der Funktion $w(t)$
d) periodische Fortsetzung der Multiplikation des Signals mit der $w(t)$ (denn Fouriertransformation geht von einer unendlich ausgedehnten Funktion aus)

Bei der digitalen Datenerfassung wird das analoge Signal in gleichmässigen Abständen T abgetastet. Diese Abtastfunktion $s(t)$ können wir mathematisch als Summe von periodischen δ-Funktionen auffassen: $s(t) = \sum_j \delta(t - j \cdot T)$. Wir bezeichnen die letztendlich aufgezeichnete Messung mit $x^*(t)$ (siehe Abb. 35). Diese lässt sich als als Multiplikation dreier Funktionen ansehen: Die des Signals, einer Fensterfunktion (Rechtecksfunktion bzw. Heaviside-Funktion für Definitionsbereich $[0, \infty[$) und der Abtastfunktion $s(t)$, wobei wir die Multiplikation der Abtastfunktion $s(t)$ mit der Fensterfunktion in $w(t)$

[16] Kurze, nicht ganz strenge Erklärung der Delta-Funktion: Es soll gelten: $\delta(x) = \begin{cases} 1, \text{ falls } x = 0 \\ 0, \text{ sonst} \end{cases}$ so dass $\int_{\mathbb{R}} \delta(x) f(x) dx = f(0)$ bzw. $\int_{\mathbb{R}} \delta(x - x_0) f(x) dx = f(x_0)$

zusammenlegen:

$$x^*(t) = x(t)w(t) \qquad\qquad w(t) = \sum_{j=0}^{K-1} \delta(t - jT)$$

Die Multiplikation der Abtastfunktion $s(t)$ mit der Fensterfunktion manifestiert sich in $w(t)$ an der oberen Begrenzung für mögliche j in der Summe der δ-Funktionen.

Wir wollen nun also das Frequenzspektrum (= Fouriertransformierte) eines abgetasteten Signals aus dem Spektrum des kontinuierlichen Signals herleiten. Gesucht ist also $X^*(\omega) = F(x^*(t)) = F(x(t)w(t))$. Es zeigt sich, dass:

$$X^*(\omega) = \frac{1}{2\pi} \int_{-\infty}^{\infty} X(\eta)W(\omega - \eta)d\eta \equiv \frac{1}{2\pi} X(\omega) * W(\omega)$$

Wobei $X(\omega)$ und $W(\omega)$ die Fouriertransformierten von $x(t)$ und $w(t)$ sind und das Integral als Faltungsintegral bezeichnet wird. Kennt man also die Fouriertransformierte X des kontinuierlichen Signals x und die Fouriertransformierte W von w, so kann man die Fouriertransformierte des abgetasteten Signals durch Faltung bestimmen.

8.2.3. Graphische Erläuterung der Faltung

Wir wollen die Faltung anhand der Kosinus-Funktion erläutern.

Die Fouriertransformierte der Kosinusfunktion ist bekannt: Es sind zwei Delta-Funktionen (Abb. 36 a)). Die Fouriertransformierte der Fensterfunktion $w(t)$ ergibt sich zu $W(\omega) = 2K\frac{\sin(T\cdot\omega)}{\omega}$ (Abb. 36 b)). Hier ist $w(t)$ ein Rechtecksfenster um den Nullpunkt mit der Breite $2T$ und der Höhe K (also ist $w(t) = K$, falls $|t| < T$, $w(t) = K/2$, falls $|t| = T$ und ansonsten ist $w(t) = 0$).

Die Faltung "schiebt" nun W entlang der ω-Achse und die beiden Delta-Funktionen von X bilden W dabei gewissermassen doppelt ab (Abb. 36, Teilbilder c bis f).

Insgesamt passieren also zwei Dinge beim Abtasten: die Spektrallinien haben sich verbreitert (Breite = 2/KT) und die Spektrallinien sind periodisch geworden.

Die Fouriertransformierte des abgetasteten Signals ist immer noch eine kontinuierliche Funktion. Zur Verarbeitung im Computer brauchen wir aber eine diskrete Frequenzwerte. Dies erreichen wir durch periodische Fortsetzung des gefensterten Signals. Das Spektrum eines solchen periodisch fortgesetzten Signals besteht aus lauter diskreten Frequenzwerten mit dem ursprünglichen Spektrum als Hüllfunktion. Die entsprechende mathematische Operation nennt man eine diskrete Fouriertransformation (DFT). Eine besonders effiziente Implementierung des Algorithmus kann man erreichen, wenn die Anzahl der Abtastpunkte N eine Potenz von 2 ist. Ein solcher Algorithmus heisst Fast Fourier Transformation (FFT).

Probleme der diskreten Fouriertransformation sind Aliasfrequenzen und die Verbreitung der Spektrallinien. Aliasfrequenzen können durch die Erfüllung der Nyquistbedingung $\omega_S \geq 2\omega_0$ vermieden werden. Hierfür kann man entweder die Abtastfrequenz ω_S erhöhen oder mit vorgeschalteten analogen Tiefpassfiltern ω_0 verringern.

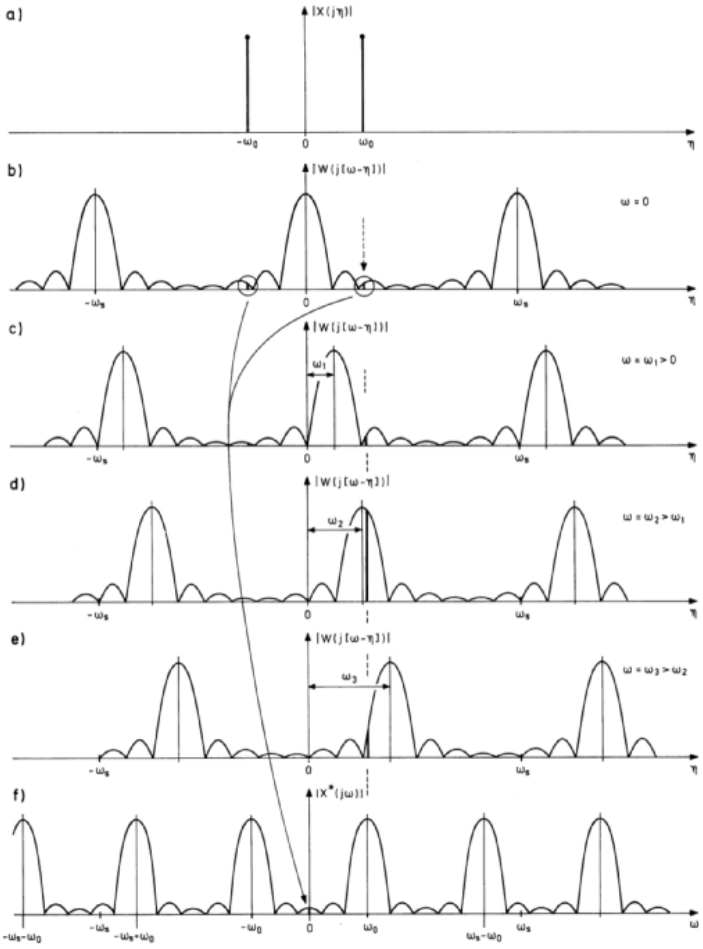


Abb. 36: Graphische Darstellung der Faltung. a) Fouriertransformierte der Kosinusfunktion; b) Fouriertransformierte der Fensterfunktion; c-f) Teilschritte der Faltung

Das Problem der Verbreitung der Spektrallinien lässt sich durch die Erhöhung der Anzahl der Abtastpunkte K vermeiden: $K > \frac{n_K \omega_S}{\Delta\omega_0}$, wobei $\Delta\omega_0$ die maximal erlaubte Verbreiterung der Spektrallinien und n_k der Index der Nebenlinie, bei der eine spezifische Dämpfung erreicht wird, sind. Der Effekt der Verbreiterung des Spektrallinien lässt sich aber auch durch die Wahl einer andern Fensterfunktion $w(t)$ verringern: Ursache der Verbreiterung ist ja das abrupte Abschneiden des Signals beim Rechteckfenster, das zu den nur langsam abfallenden Nebenlinien in $W(\omega)$ führt. Man kann stattdessen aber eine sanft abfallende Fensterfunktion verwenden. Beim sogenannten Hanning wird z.B. die Rechteckfunktion mit einer $(1+\cos)$-Funktion multipliziert. Das Spektrum einer solchen Fensterfunktion hat sehr viel schneller abfallende Nebenlinien. Es gibt eine ganze Reihe solcher Fensterfunktionen, z.B. Hanning-, Hamming-, Kaiser-Fenster, etc. (Siehe Abb. A24)

8.3. Beschreibung linearer Systeme

Ein lineares System ist dadurch definiert, dass eine am Eingang angelegte harmonische Schwingung auch am Ausgang eine harmonische Schwingung mit der gleichen Frequenz erzeugt. Dabei sind aber die Amplitude und die Phase der Ausgangsschwingung frequenzabhängig. Ein solches System kann daher eindeutig durch die Angabe der (komplexen) Übertragungsfunktion $Y(\omega)$ beschrieben werden. (Man nennt diese Funktion auch Frequenzgang des Systems.) Kennt man $Y(\omega)$, dann lässt sich zu jedem angelegten Eingangssignal das entsprechende Ausgangssignal vorhersagen. Im Frequenzraum braucht man nur eine Multiplikation durchzuführen.

Zur Bestimmung der Übertragungsfunktion kann man harmonische Schwingungen anlegen und misst am Ausgang jeweils die Amplitude A und Phase φ, während man die Frequenz schrittweise durchfährt.

Dieses Verfahren ist aufwändig, da Phasen und Amplituden als Funktion der Frequenzgemessen werden müssen (Signalgenerator nötig).

Eine andere Möglichkeit wäre es, die sogenannte Stossantwort (Impulsantwort) zu bestimmen: statt einer harmonischen Schwingung legt man einen δ-Stoss an den Eingang des Systems: $e(t) = K\delta(t)$. Am Ausgang misst man die Antwort $a(t)$. Die Übertragungsfunktion $Y(\omega)$ kann man jetzt einfach durch Fouriertransformation der Stossantwort $a(t)$ bestimmen:

$$e(t) = K\delta(t) \qquad a(t) = y(t) * e(t) = Ky(t)\delta(t) = Ky(t)$$

$$\text{Fouriertransformiert:} \qquad A(\omega) = K \cdot Y(\omega)$$

Der δ-Stoss kann nämlich als lineare Überlagerung von unendlich vielen harmonischen Schwingungen mit gleicher Phase und Frequenzen von $-\infty$ bis ∞ aufgefasst werden. Die Stossantwort ist das Analogon der Übertragungsfunktion im Zeitraum.

A. Bildanhang

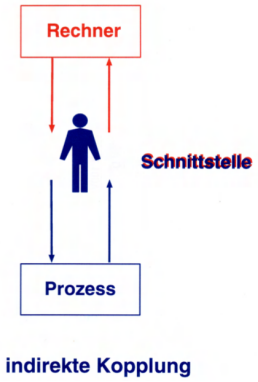


Abb. A1: Schema indirekte Prozesskopplung

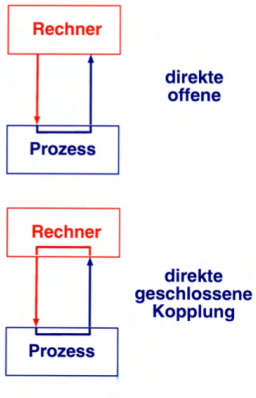


Abb. A2: Schema direkte Prozesskopplung

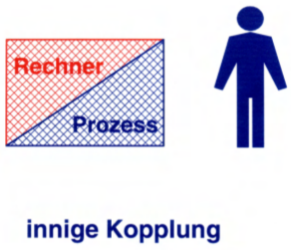


Abb. A3: Schema innige Prozesskopplung

x_1 / x_0	0011 / 0101	verbale Form	symbolische Darstellung	Bezeichnung
f_0	0000	konstant 0	0	Kontradiktion, Symbol: $\perp$ (unerfüllbar)
f_1	0001	x_1 und x_0	$x_1 \wedge x_0$	Konjunktion
f_2	0010	nicht x_0, aber x_1	$x_1 \wedge \overline{x_0}$	Inhibition
f_3	0011	identisch x_1	x_1	Identität
f_4	0100	nicht x_1, aber x_0	$\overline{x_1} \wedge x_0$	Inhibition
f_5	0101	identisch x_0	x_0	Identität
f_6	0110	x_1 ungleich x_0	$x_1 \not\leftrightarrow x_0$	Antivalenz, XOR
f_7	0111	x_1 oder x_0	$x_1 \vee x_0$	Disjunktion
f_8	1000	nicht (x_1 oder x_0)	$x_1 \overline{\vee} x_0$	NOR-Funktion, Peircescher Pfeil
f_9	1001	x_1 gleich x_0	$x_1 \leftrightarrow x_0$	Äquivalenz
f_{10}	1010	nicht x_0	$\overline{x_0}$	Negation
f_{11}	1011	wenn x_0, dann x_1	$x_0 \rightarrow x_1$	Implikation
f_{12}	1100	nicht x_1	$\overline{x_1}$	Negation
f_{13}	1101	wenn x_1, dann x_0	$x_1 \rightarrow x_0$	Implikation
f_{14}	1110	nicht (x_1 und x_0)	$x_1 \overline{\wedge} x_0$	NAND-Funktion, Shefferscher Strich
f_{15}	1111	konstant 1	1	Tautologie, Symbol: T (allgemeingültig)

Abb. A4: 16 mögliche zweistellige boolesche Funktionen

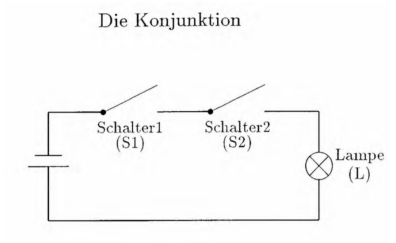


Abb. A5: Schema Konjunktion

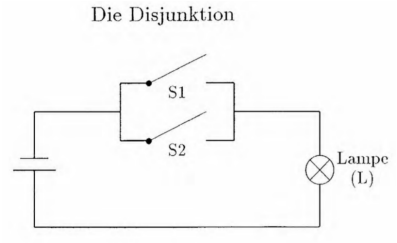


Abb. A6: Schema Disjunktion

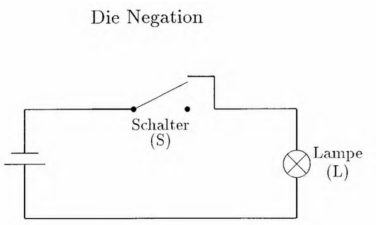


Abb. A7: Schema Negation

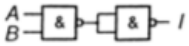
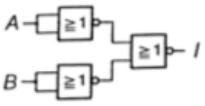
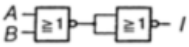
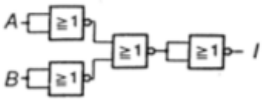
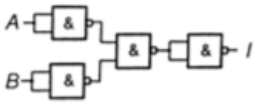
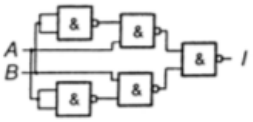
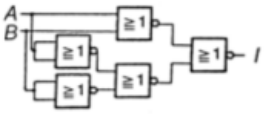
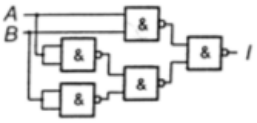
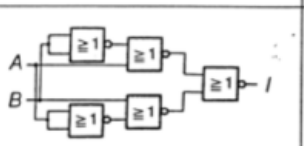
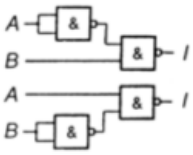
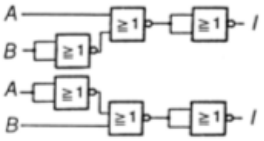
Funktion	mit NAND	mit NOR	Boolescher Ausdruck
UND			$I = A \wedge B$
ODER			$I = A \vee B$
NAND			$I = \overline{A \wedge B}$
NOR			$I = \overline{A \vee B}$
Antivalenz (exl. ODER)			$I = \overline{A}B \vee A\overline{B}$
Äquivalenz			$I = AB \vee \overline{A}\overline{B}$
Implikation Implikation			$I = A \vee \overline{B}$ $I = \overline{A} \vee B$
NOT			$I = \overline{A}$

Abb. A8: Zusammenfassung der logischen Funktionen und ihrer Realisierung mit NAND- oder NOR-Gattern

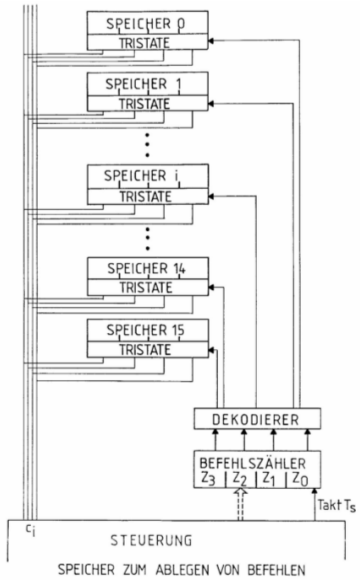


Abb. A9: Schema der Speicheraddressierung

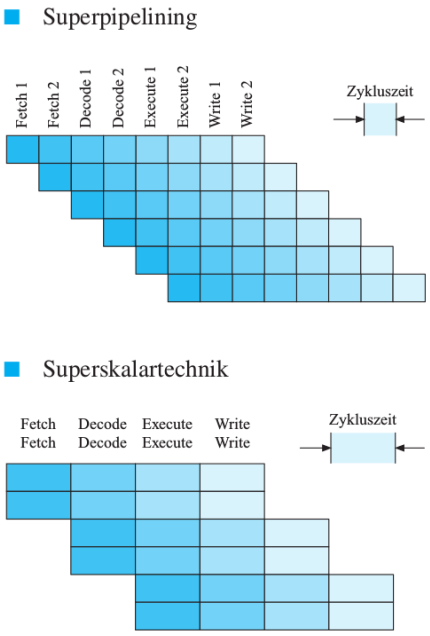


Abb. A10: Schema Superskalar und Superskalartechnik

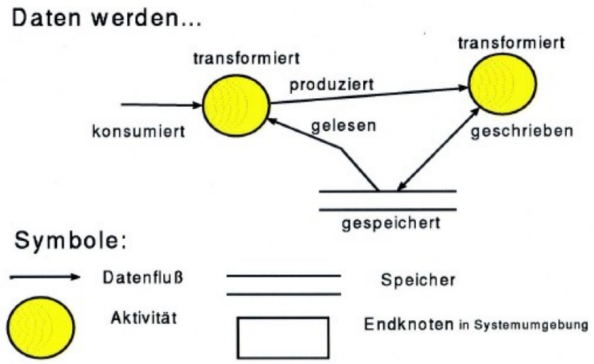


Abb. A11: Elemente eines Datenflussdiagramms

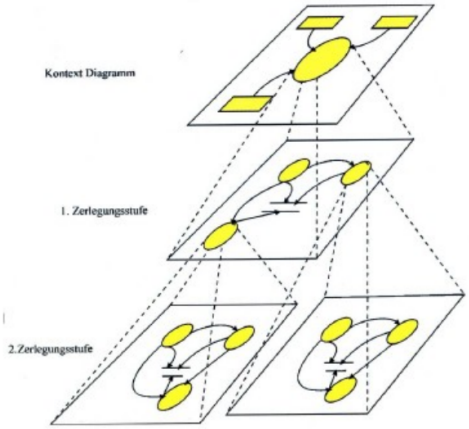


Abb. A12: Heirarchie der Datendiagramme

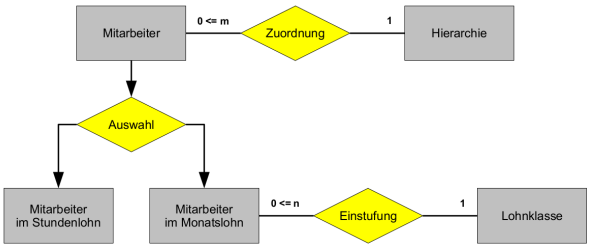


Abb. A13: Beispiel eines Entity Relationship Diagramms

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

Abb. A14: ASCII-Tabelle

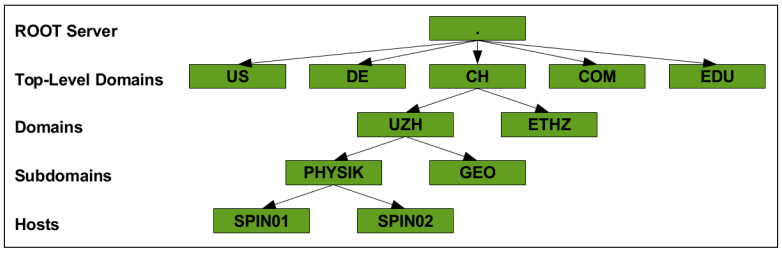


Abb. A15: Hierarchische Struktur des DNS

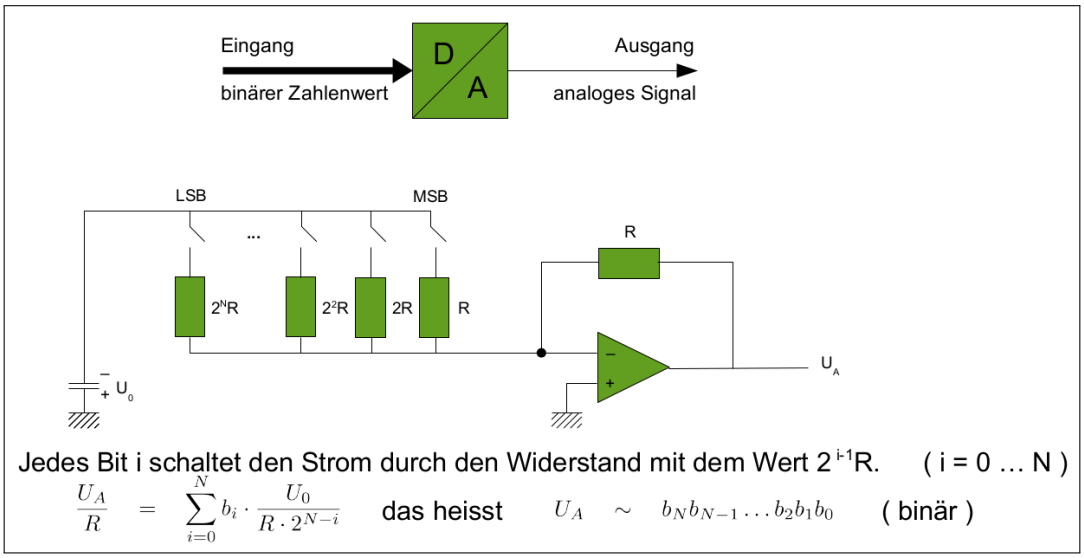


Abb. A16: Prinzip der technischen Realisierung eines DACs

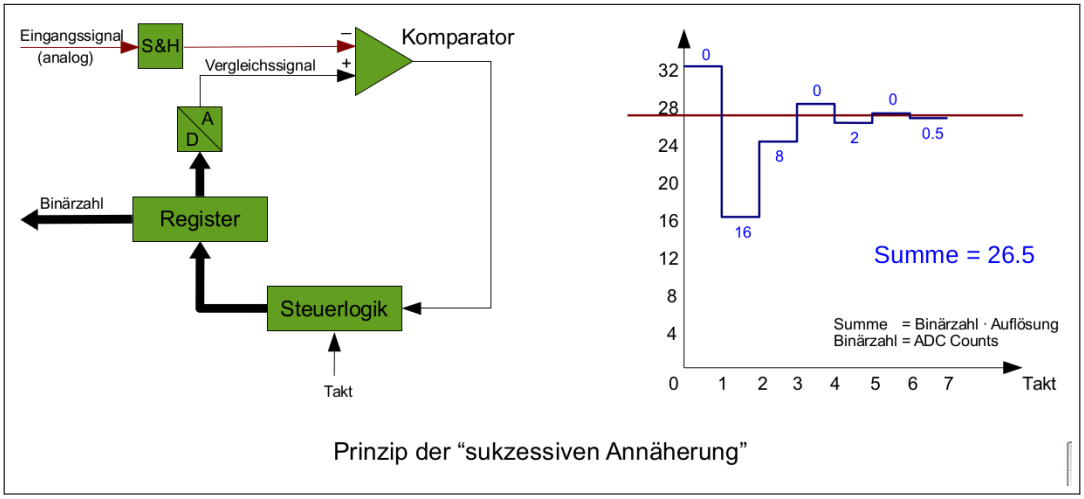


Abb. A17: Prinzip der technischen Realisierung eines ADCs. Sukzessive Annäherung/Approximation: der ADC baut einen Vergleichswert U_v jedes Mal neu auf. Das Eingangssignal wird mittels Intervallschachtelung eingegrenzt. Einfache sukzessive Approximation setzt dabei pro Schritt ein Bit um.

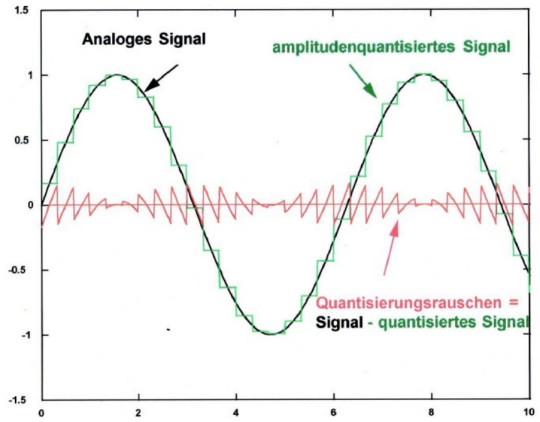


Abb. A18: Amplitudenquantisierung und Quantisierungsrauschen

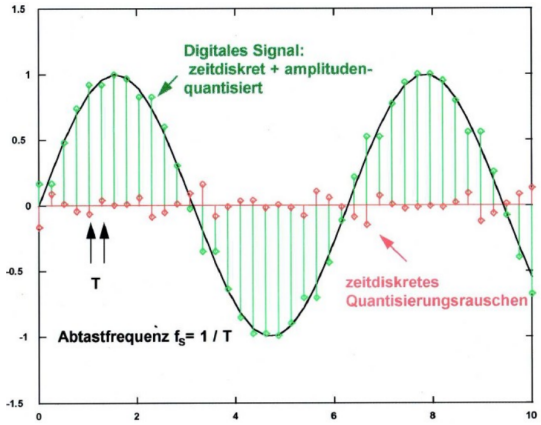


Abb. A19: Zeit- und Amplitudenquantisierung

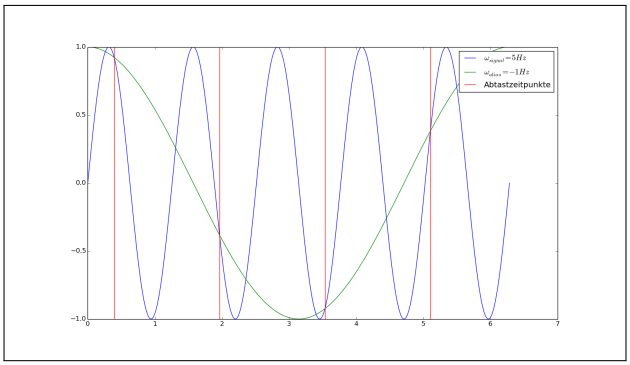


Abb. A20: Graphische Darstellung der Alias-Effektes mit Sinusfunktionen: Die Abtastfrequenz ω_S sei 4Hz und somit die Nyquist-Frequenz ω_{Ny} =2Hz. Die Signalfrequenz sei $\omega = 5$Hz. Somit sollten wir die Aliasfrequenz $\omega_{alias} = \omega_S - \omega = -1$Hz erhalten. $\omega_{alias} = -1$Hz heisst, dass die Frequenz zwar 1Hz ist, die Welle sich aber nicht mehr gleich verhält: $\sin(-x) = -\sin(x)$. Ausserdem ist die Alias-Welle Phasenverschoben, hier um $\frac{\pi}{2}$, weshalb die Welle einer Kosinuswelle entspricht.

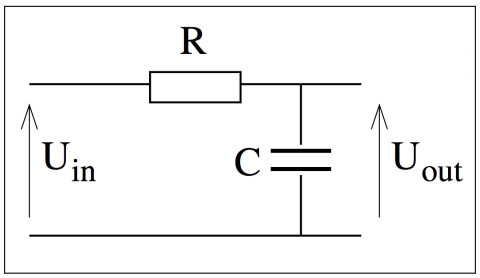


Abb. A21: Schaltplan Tiefpass

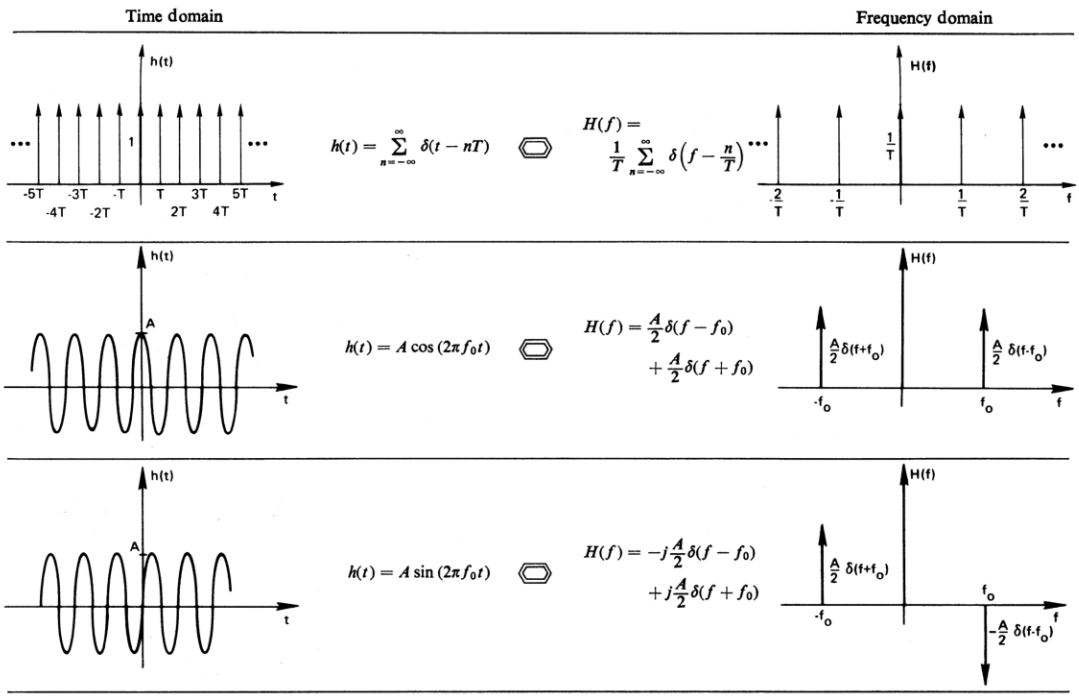


Abb. A22: Beispiele Fouriertransformation

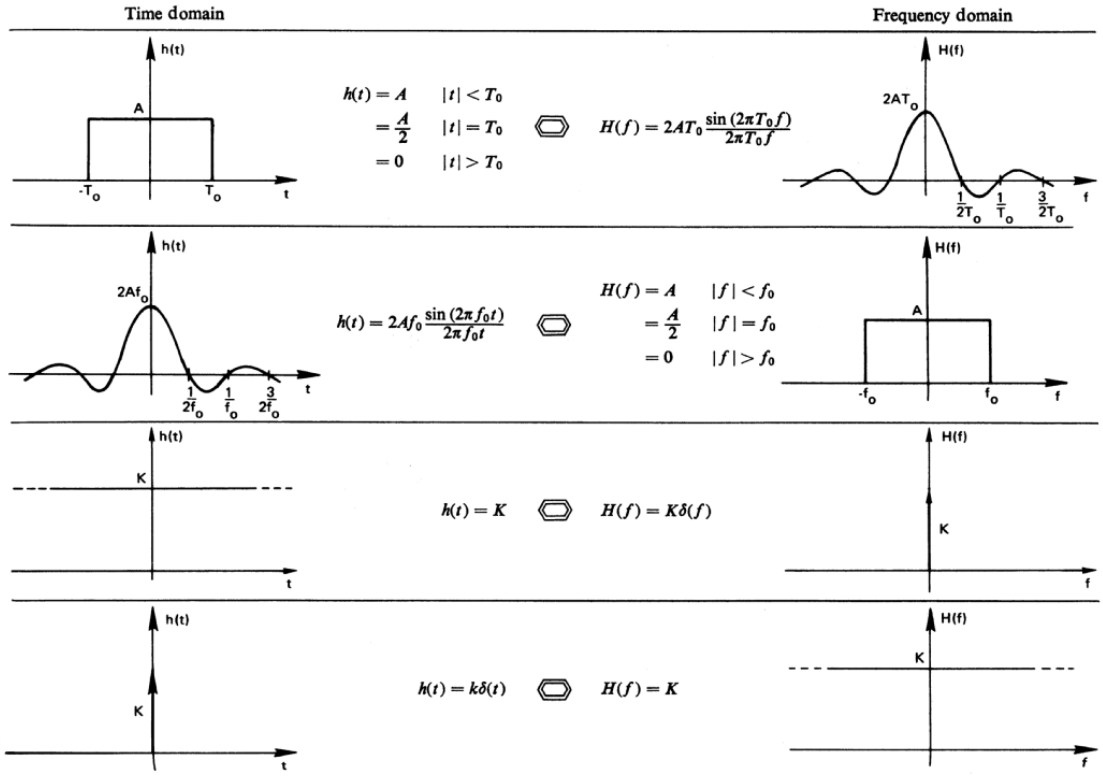


Abb. A23: Beispiele Fouriertransformation 2

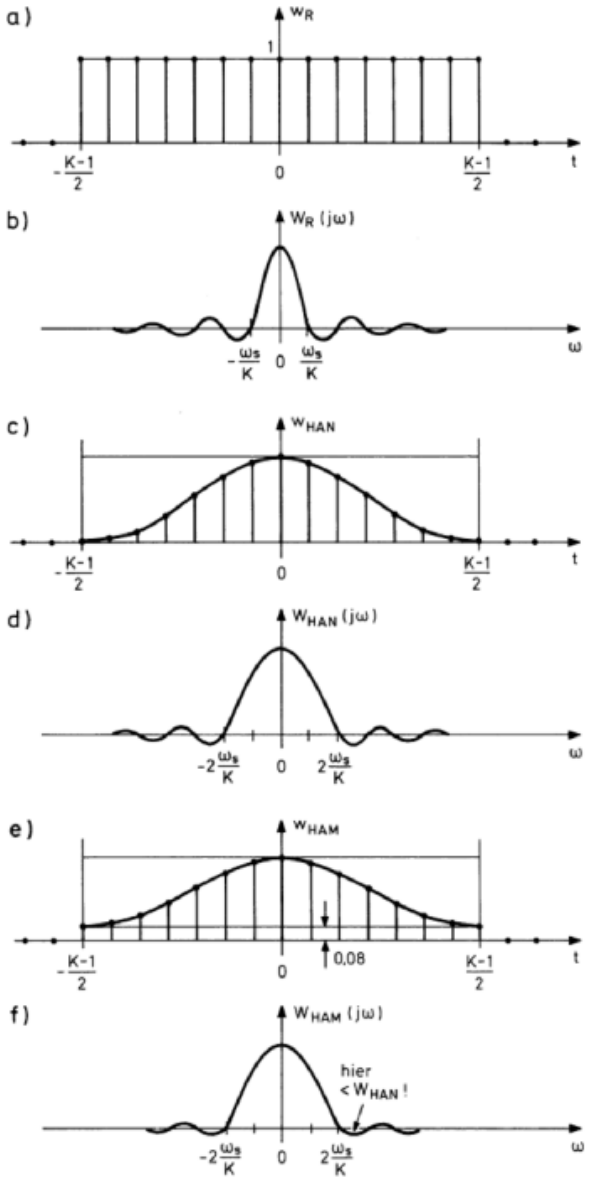


Abb. A24: a, b) Funktion und Spektrum fürs Rechteck-Fenster
c, d) Funktion und Spektrum fürs Hanning Fenster
e, f) Funktion und Spektrum fürs Hamming Fenster

B. Konstruktion eines 4-Bit Rechners

Die prinzipielle Funktionsweise eines Rechners soll anhand der Konstruktion eines 4-Bit Rechners demonstriert werden. Folgende Funktionseinheiten sollen über einen 4-Bit-Bus verbunden werden:

- Eine 4-Bit-Eingabe mit Kippschaltern (kann schreiben; Eingabewerk)
- Ein 4-Bit-Schieberegister A, das parallel geladen werden kann. Die Eingänge sollen und Ausgänge sollen mit dem Bus verbunden werden. (kann schreiben; Akkumulator)
- Ein 4-Bit-Ringschieberegister B, das über den Bus geladen werden kann (kann schreiben)
- Ein 4-Bit-Anzeigeregister, das es erlaubt, den Zustand der Busleitungen zu speichern und anzuzeigen (bei uns: LED-Anzeige; Ausgabewerk; kann schreiben, da er den Wert abspeichert)
- Ein serieller Addierer (vier parallel geschaltete Volladierer), der die Summe des Inhaltes der Register A und B berechnet und im Register A abspeichert. (kann schreiben)

(Siehe Abb. A25)Bei allen Komponenten, bei welchen 'kann schreiben' in Klammern beigefügt ist, müssen die Ausgaben über Tristate-Treiber mit dem Bus verbunden werden, um die Einfabe auf den Bus zu regulieren (es darf nur eine Eingabequelle auf den Bus haben). Die Register können mittels eines Rechenwerkes verknüpft werden. Auf die Realisierung eines Arbeitsspeichers wird verzichtet.

Steuerung: Die auszuführenden Operationen werden über drei Kippschalter (c_0, c_1, c_2) eingestellt: Also können wir maximal acht Befehle definieren. Damit wir logische Funktionen für die jeweiligen Befehle finden können, müssen wir uns zuerst anschauen, wie welche Komponenten angesteuert werden (Siehe Abb. A25):

- I_S, I_B, I_Σ, I_{Anz}: Steuerungen der Tristate-Treiber der Eingabe, des Registers B und des Addierers und der Anzeige. Ist $I = 1$, so kann die Komponente nicht auf den Bus schreiben.
- T_A, T_B: Taktsignale für die Register A und B
- L_A, L_B: Funktion der Register A und B bei einem Taktsignal. Ist $L = 1$, so laden die Register den momentanen Eingabewert. Ist $L = 0$, so fungieren sie als Schieberegister. Da Register A immer den anliegenden Wert annehmen soll und nie als Schieberegister benutzt wird, ist $L_A = 1$
- E_{konst}: (enable Konstante) Ansteuerung für den Multiplexer, welcher entweder eine 1 oder den Wert von B weitergibt. (Die Schalter ganz links unten in Abb. A25 wird nur dazu verwendet, um eine 1 zu erhalten.)

- E_{kompl}: (enable Komplement) Ansteuerung für den Multiplexer, welcher entweder den Wert oder dessen Komplement weitergibt.

In der folgenden Tabelle werden zuerst die acht Operationen einer Kombination der Kippschalter c_i zugewiesen. Dann folgt die Wahrheitstabelle für die ansteuerbaren Komponenten:

Operation	c_2	c_1	c_0	I_S	I_B	I_Σ	T_A	T_B	L_B	E_{konst}	E_{kompl}	I_{Anz}
$A + B \to A$	0	0	0	1	1	0	1	0	X	0	0	1
$A - B \to A$	0	0	1	1	1	0	1	0	X	0	1	1
$A + 1 \to A$	0	1	0	1	1	0	1	0	X	1	0	1
$A - 1 \to A$	0	1	1	1	1	0	1	0	X	1	1	1
$Eing. \to A$	1	0	0	0	1	1	1	0	X	X	X	1
$A \to Ausg.$	1	0	1	1	1	1	0	0	X	X	X	0
$A \leftrightarrow B$	1	1	0	1	0	1	1	1	1	X	X	1
$B_i \to B_{i-1}$	1	1	1	1	1	1	0	1	0	X	X	1

Wobei für ein X ein beliebiger Wert eingesetzt werden kann.
Daraus ergeben sich die folgenden Funktionen für die Komponenten:

$$I_S = \overline{c_2 \wedge \bar{c}_1 \wedge \bar{c}_0} = (c_2 \wedge \bar{c}_1)\overline{\wedge}\bar{c}_0 \qquad I_B = \overline{c_2 \wedge c_1 \wedge \bar{c}_0}$$
$$I_\Sigma = c_2 \qquad I_{Anz} = \overline{c_2 \wedge \bar{c}_1 \wedge \bar{c}_0}$$
$$T_A = \overline{(c_2 \wedge \bar{c}_1 \wedge c_0) \vee (c_2 \wedge c_1 \wedge c_0)} \qquad T_B = c_2 \wedge c_1$$
$$L_A = 1 \qquad L_B = \bar{c}_0$$
$$E_{konst} = c_1 \qquad E_{kompl} = \ddot{\text{U}} = c_0$$

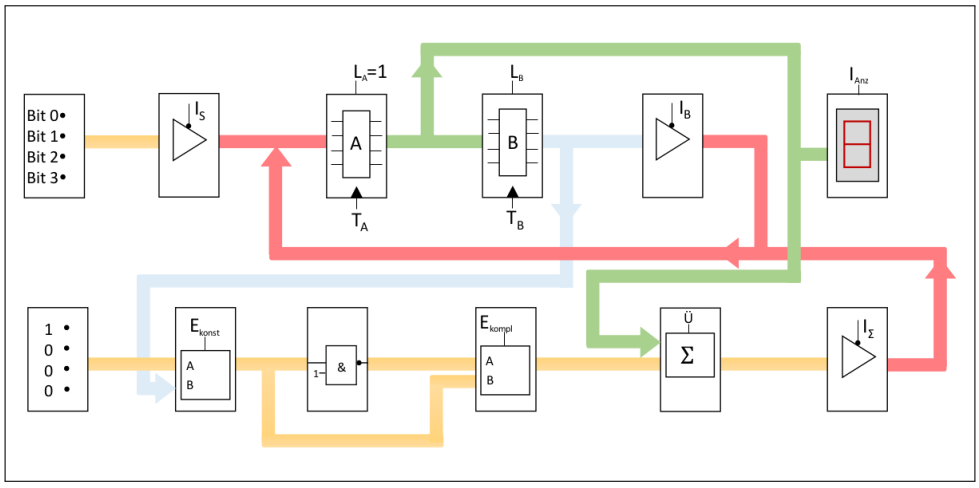


Abb. A25: Schaltplan des 4-Bit-Rechners. Die Verbindungen sind jeweils 4-Bit-Verbindungen.

C. Glossar

ALU	Arithmetisch-Logische Einheit (arithmetic logic unit)
Bit	Eine einzelne Binärstelle (0 oder 1), die ein Rechner speichert (Binary digit)
Cache	ein schneller Puffer-Speicher, der (wiederholte) Zugriffe auf ein langsames Hintergrundmedium oder aufwendige Neuberechnungen zu vermeiden hilft.
Gatter	Logische Verknüpfungsbausteine wie UND, ODER, Negation, NAND, NOR, XOR etc. Es wird auf das logische Verhalten beschränkt, die logischen Funktionen werden auf Schaltungen ohne tiefergehende elektrotechnische Kenntnisse abgebildet. Gatter haben entsprechende Schaltsymbole.
Hasard	Die durch das Schaltnetz gegebene logisch-strukturelle Vorbedingung für einen (Hasard)fehler
Latch	Zustands- bzw. pegelgesteuerte Elemente; Die Eingänge wirken sich nur auf den Zustand aus, wenn der Takt z.B. den Wert 1 hat. Im Gegensatz dazu: Flankensteuerung. Eingabewerte werden nur bei Taktflankenänderung, also von 1 auf 0 oder 0 auf 1, übernommen.
Operatorensystem	Ein System von Operatoren, mit dem alle booleschen Funktionen dargestellt werden können, heisst vollständiges Operatorensystem. Liste aller boolescher Funktionen mit zwei Variablen: Siehe Abb. A4. Es gibt mehrere vollständige Operatorensysteme, jedoch ist das System $(\vee, \wedge, ^-)$ das intuitivste.
Protokoll	Regeln, welche das Format, den Inhalt, die Bedeutung und die Reihenfolge gesendeter Nachrichten zwischen verschiedenen Instanzen (der gleichen Schicht) festlegen.
Schaltnetz	Kombinatorische Schaltkreise; Ausgabe hängt nur von den Werten der Eingangsvariablen zum gleichen Zeitpunkt ab.
Schaltwerk	Sequentielle Schaltkreise; die Ausgabewerte hängen auch von Belegungen der Eingangsvariablen zu vergangenen Zeitpunkten ab.
Wort/Datenwort	Grundverarbeitungsdatengrösse bei einem Computer. Die Grösse dieser Dateneinheit in Bit wird als Wortbreite oder Wortlänge, beziehungsweise als Busbreite bezeichnet.