INCO Zusammenfassung Fortlaufend

2. Oktober 2022; rev. 15. Januar 2023

Linda Riesen, (rieselin)

Inhaltsverzeichnis

1 Vorlesung 1 Kombinatorische Logik

Kombinatorische vs. Sequentielle Logik

Kombinatorische Logik	Sequentielle Logik
kein Gedächtnis	Gedächtnis
Einfache Logische Operationen	D-Flip-Flop: Speicher, Takt Eingang (Clock)
Symbole/Logische Gleichungen/ Wahrheitstabellen	Timing Diagram

1.1 Kombinatorische Logik

- Ausgänge ändern sich nur in Abhängigkeit von Eingängen
- Jeder Ausgang y_i lässt sich durch seine boolsche Funktion der Eingänge beschreiben $y_i = f_i(x_0, x_1, ...x_n)$
- Für N Eingänge gibt es 2^{N-1} mögliche Eingangskombinationen die ein unterschiedliches Ausgangsresultat ergeben.

Um Das Schema zu Zeichnen, die Wahrheitstabelle ausrechnen und für alle Pfade, für welche das Resultat 1 ist die Minterme (Ausgang = 1) oder Maxterme (Ausgang = 0) aufstellen und ins Schema einbauen. Dann werden jeweils alle Min/Maxterme aneinander gehängt (0 wird dargestellt als !Eingang, 1 als Eingang) mit # bzw. & + !() Verknüpfungen = tada. ⇒ Vereinfachung darf mit Tool gemacht werden.

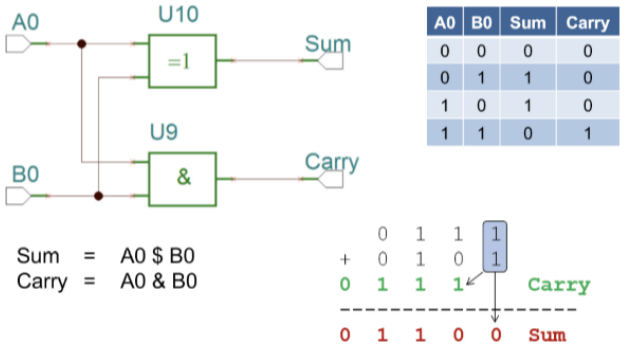
Wahrheitstabelle Ein x in der Wahrheitstabelle steht für "don't care", ein Wert der entweder 1 oder 0 sein kann und demnach die Vereinfachung erleichtert. ($\neq$ das gleiche wie die Bedingung einfach weglassen)

Function	Boolean Algebra (1)	IEC 60617-12 since 1997	US ANSI 91 1984	DIN 40700 until 1976
AND	A & B	&		
OR	A # B	≥ 1		
Buffer	A	1		
XOR	A $ B	= 1		
NOT	!A	1		
NAND	!(A & B)	&		
NOR	!(A # B)	≥ 1		
XNOR	!(A $ B)	= 1		

NOT		AND			NAND			OR			NOR			XOR			XNOR		
A	X	B	A	X	B	A	X	B	A	X	B	A	X	B	A	X	B	A	X
0	1	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	1
1	0	0	1	0	0	1	1	0	1	1	0	1	0	0	1	1	0	1	0
		1	0	0	1	0	1	1	0	1	1	0	0	1	0	1	1	0	0
		1	1	1	1	1	0	1	1	1	1	1	0	1	1	0	1	1	1

Abbildung 1: Alle Symbole der Kombinatorischen Logik

1.2 Bitaddierer



A0	B0	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Abbildung 2: 1-Bit Halb Addierer

Für Volle Addition wird dies einfach wiederholt, für weitere Bits dann fortgeführt.

2 Vorlesung 2 Sequentielle Logik

2.1 D-Flip-Flop

Basis Element für sequentielle Logik, Daten kommen, werden gespeichert bis Clock kommt, dann an den Ausgang übertragen (Q=D), ansonsten ändert sich Q nie. Ein Flip-Flop kann daher immer 2 Zustände annehmen (0, 1) = 2^N Zustände.

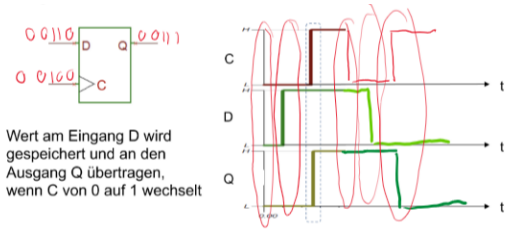


Abbildung 3: D-Flip-Flop

2.2 3 Typische Schaltungen

Alle enthalten Speicher und Kombinatorische Logik, sie werden bei jedem Clock aktualisiert und oft ist der Speicher Ausgang mit dem Dateneingang rückgekoppelt.

2.2.1 Zähler (Counter)

- Endliche Zustände
- Reihenfolge wird nicht von aussen beeinflusst
- Ausgänge hängen nur von innerem Zustand ab

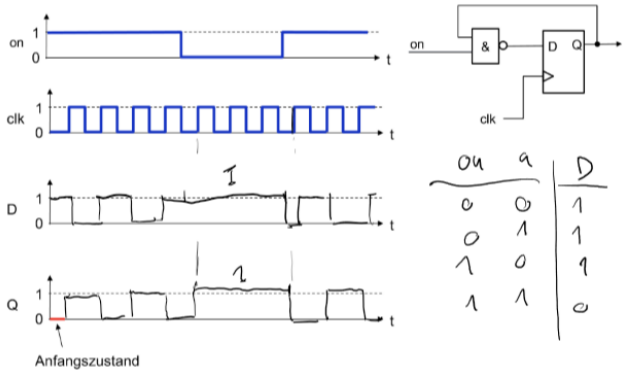


Abbildung 4: Beispiel einer Schaltung (D ist immer eins hinter Q)
Mit: Wahrheitstabelle, Zustandsdiagramm (Bubbles Diagramm mit Pfeilen) und Zeitdiagramm (Bei einem Vektor werden die Werte in das Diagramm geschrieben + alle Graphen gleichzeitig gezeichnet)

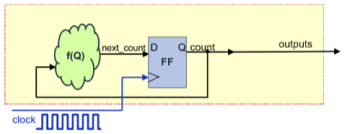


Abbildung 5: Counter

2.2.2 Zustandsautomaten (Finite State Machines, FSM)

- Der Ausgangswert des FF's (Flip-Flop) entspricht dem Input und dem Speicher
- Ausgänge abgängig von Input und Status des Speichers

2.2.3 Schieberegister (Shiftregister) = Speicher

- Enthält mehrere in Reihe geschaltete FF's

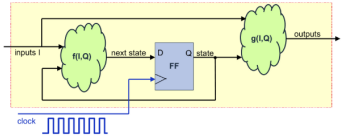


Abbildung 6: Finite State Machine

- Register: Ein Ausgänge sind Parallel, Schieberegister: Ein Ausgänge sind Seriell
- Bsp: Flankendetektor: wenn Unterschiedliche Resultate bei FF's wird detektiert. (Schieberegister).

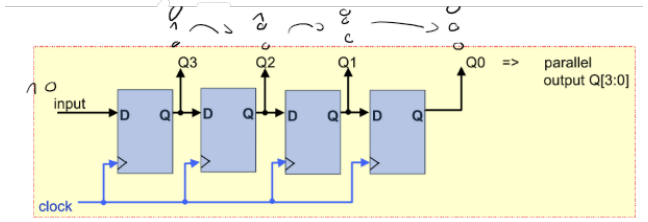


Abbildung 7: Schieberegister

3 Vorlesung 3 Zahlensysteme

Bit (binary digit): Speicher für 0,1 / True, False

Byte (octet): 8 Bit / 2 *(Nibble = 4 Bit)

Uneinheitliche Definitionen: Word 16 Bit, Doubleword: 32 Bit, Quadword 64 Bit, Octaword 128 Bit

Das Binärsystem / 2-er System / Dualsystem: wird mit b am Ende oder 0B am Anfang bezeichnet

Das Hexiadezimalsystem / 16-er System wird mit h am Ende oder 0x am Anfang bezeichnet.

Hexadezimal: Bits in 4er Gruppen, Kompakte Darstellung

Stellenwertsystem: Je nach Stelle haben die Zeichen unterschiedliche Wertigkeit

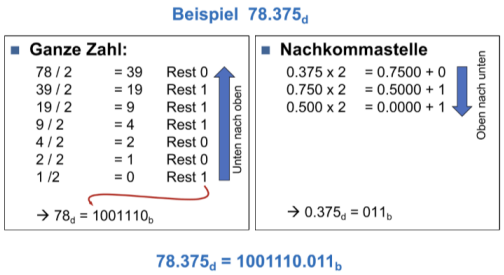
Unicode: Beschreibt wie ASCII einen Zeichensatz, umfasst aber (fast) alle bekannten Zeichen (bisher zu 10% belegt ⇒ Platz zur Erweiterung), Inkompatibilität zw. Dokumentent entfällt (theoretisch)

Encoding (UFT-8/ UFT-16, UFT-32): Nicht kompatibel, muss jeweils erkannt werden vom System oder angegeben werden.

Gray-Code: Vermindert Ablesefehler, kann durch einfache Umformung aus Bitcodierung hergestellt werden.

10-er System	2-er System	16-er System
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

(a) Werte 1-15 Vergleich



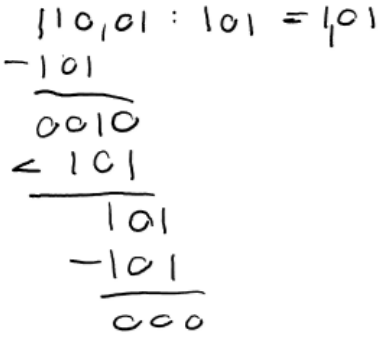
(b) **Hornerschema**
Hornerschema an Bsp, für Hexadez einf :16 und Resultate in Hexadez umformen

3.1 Addition / Subtraktion / Multiplikation / Division

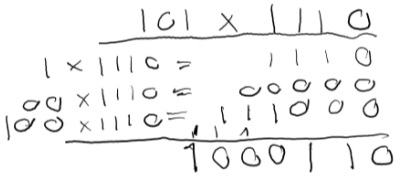
3.1.1 Addition + Subtraktion (Binär und Hexa)

Analog zu 10er System Schriftlich, Mit Übertrag ab 2 bzw 16

3.1.2 Multiplikation + Division (Binär)



(c) Bsp Division



(d) Bsp Mulitplikation

3.2 Negative Zahlen

Wunschvorstellung für Definition Neg. Zahlen Binär

1. Gleichviele Negative wie Positive Zahlen
2. Pos. Zahl = Vorzeichenlose Zahl (nichts spezielles)
3. Einfacher Vorzeichenwechsel
4. Gleiche Verfahren (Grundrechenarte, Vergleiche, Überlauferkennung)
5. Kontinuität der Werte -1, 0, 1 (nicht -0, +0)

⇒ Normalerweise Entscheidung für **Zweierkomplement** Dies wird gebildet durch Zahl invertieren (0 => 1 und 1 => 0) und 1 addieren. Wenn dabei Überlauf entsteht, wird dieser abgeschnitten (ignoriert), dadurch wird 5. erfüllt. (Kritischer Übergang ist bei kleinster negativer und grösster Positiver Zahl, dort muss Übertrag beachtet werden)

- Je nach dem ob vorzeichenlos/ vorzeichenbehaftet heisst das Bitmuster etwas anderes

- Vorzeichenlos: Überläufe zw. 0 und grösster darstellbarer Zahl
- Vorzeichenbehaftet: Überläufe zw. grössten Positiven und kl. negativen Zahl
- Überläufe können zu falschem Resultat führen wenn jeweils das betreffende **Überlaufsflag** nicht betrachtet wird (Carry/Overflow) = Normalfall!

4 Vorlesung 4 Informationstheorie

Information: Abstrakte Grösse, kann gemessen werden durch wie viel Information muss man Angeben um etw. auszuwählen, wie gross ist Überraschung bei bekannter Sammlung ⇒ kleinste techn. Informationseinheit = 1 Bit, kleinste Menge ist kleiner. (keine Entscheidung mehr)

DMS (Discrete Memoryless Source): liefert zeitl. einzelne Ereignisse, erinnert sich nicht an Vorgeschichte

BMS (Binary Memoryless Source): DMS mit nur 2 versch. Ereignissen

- Entropie BMS: $H_b = p \cdot log_2(\frac{1}{p}) + (1-p) \cdot log_2(\frac{1}{1-p})$
- ⇒ Je näher p bei 0.5 desto höher die Entropie

Ja/Nein - Frage: = 1 Bit Info = Eingrenzbare Fälle *2, F Fragen = 2^F Fälle

Enthaltene Information: $I(x_n) = log_2(N)$ in Bit, aufgerundet für Anz. benötigten Fragen

Enthaltene Information / Wahrscheinlichkeit: $I(x_n) = log_2(\frac{1}{P(x_n)}) = -log_2(P(x_n))$ in Bit

Auftetenswahrscheinlichkeit: Alle gleichen Symbole aufsummieren, durch alle Symbole teilen

Entropie (Mittlerer Info. Gehalt v. Quellen): $\sum_{n=0}^{N-1} P(x_n) \cdot log_2(\frac{1}{P(x_n)})$

- Minimum Entropie: Wenn ein Symbol die Wahrscheinlichkeit 1 hat ⇒ Entropie = 0
- Maximum Entropie: Wenn alle Symbole gleiche Wahrscheinlichkeit haben ($\frac{1}{N}$) ⇒ Entropie = $log_2(N)$
- Kann auf bisschen mehr als dieser Wert Zip komprimiert werden (muss noch Info zu Encodierung, Textart, etc enthalten) falls kleiner evtl Lauflängencodierung

5 Vorlesung 5 Quellencodierung 1

5.1 Ziele der Datenkompression

- Speicher sparen, Bandbreite reduzieren ⇒ Kosten minimieren
- Übertragungszeit reduzieren reduzieren, Energie Sparen ⇒ Optimierung zw. Verarbeitung und Übertragung

5.2 Reduktion

Redundanz = Weitschweifigkeit, Überflüssigkeit

Verlustbehaftete Datenkompression Reduktion richtet sich nach Bedürfnissen des Empfängers (Irrelevanz) (Redundanz <= 0)

Verlustfreie Kompression Reduktion richtet sich nach Eigenschaften der Quelle, Anteil der Codierung der keine Information trägt (Redundanz R > 0)

$$R = L - H(Bit/Symbol) \tag{1}$$

Equation 1: Redundanz

⇒ Bsp BCD (Binary Coded Decimal (4 Stellen pro Decimal Stelle)), H_{BBCD}= 3.32 Bit/Symbol, Code-Länger = 4Bit/Symbol, Redundanz = 4-3.32 = 0.68 Bit/Sym.

5.3 Codes mit Unterschiedlicher Länge

Je höher der Informationsgehalt desto länger der Code (= Weniger Redundanz), bei Binären Codes muss dann *Präfixfreiheit* gelten (nicht 2 Codes beginnen gleich), sonst kann man sie nicht mehr trennen.

$$L(x) = \sum_{n=0}^{N-1} P(x_n) \cdot l_n \tag{2}$$

Equation 2: Mittlere Länge

5.4 Lauflängen Codierung (Run Length Encoding)

Codiert so: Marker, Anzahl, Zeichen Dabei wird als Marker ein selten genutzer Code verwendet Die Zählbreite so das Runs in Typischer Länge abgebildet werden können Falls der Marker M selbst kommt wird dieser halt sehr umständlich aufgezeichnet (M01M)

5.5 Huffman Code

Statistisches Kompressionsverfahren, automatisch Präfixfrei, und Optimale Codes (keine besseren Codes mit diesen Voraussertzungen möglich) ⇒ Wahrscheinlichkeiten müssen bekannt sein. Um dies noch zu verbessern können 2-fach, 3-fach Symbole verwendet werden, diese müssen dann auch bei der Redundanz Berechnung als solche berücksichtigt werden

1. Ordne alle Symbole nach aufsteigenen Auftretenswahrscheinlichkeiten auf einer Zeile. Dies sind die Blätter des Huffman-Baums.
2. Notiere unter jedes Blatt seine Wahrscheinlichkeit.
3. Schliesse die beiden Blätter mit der kleinsten Wahrscheinlichkeit an einer gemeinsamen Astgabel an und ordne dem Ast die Summe der Wahrscheinlichkeiten der beiden Blätter zu.
4. Wiederhole Punkt 3 mit Blättern und Ästen so lange, bis nur noch der Stamm des Baums übrig bleibt.
5. Nun wird bei jeder Astgabel dem einen Zweig eine 0 und dem anderen eine 1 zugeordnet. (Die Zuordnung ist frei wählbar, muss aber über den ganzen Baum einheitlich sein).
6. Nun werden auf dem Pfad vom Stamm zu jedem Blatt die Nullen und Einsen ausgelesen und von links nach rechts nebeneinander geschrieben. Dies sind die Huffman-Codeworte.

Abbildung 8: Huffmanverfahren Anleitung

6 Vorlesung 6 Quellencodierung 2

6.1 Kompression mittels Substitution

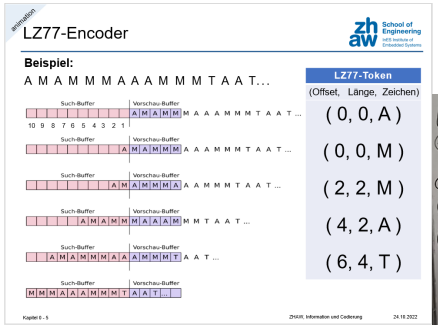
$$R = \frac{CodierteBits}{OriginaleBits} \tag{3}$$

Equation 3: Kompressionsrate (*Achtung* $\neq$ *Redundanz R*)

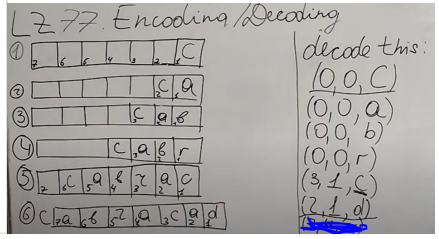
6.1.1 LZ77 - Verfahren

Embedded Systems (bsp Game Boy Advance)

Idee: Alle Zeichen werden durch Token fixer Länge ersetzt (Token = Offset, Länge, Zeichen). Es gibt einen Such und Vorschau Buffer, dann wird die längste Übereinstimmung im Such-Buffer im Vorschau-Buffer gesucht und als Token (zsm mit nächsten Zeichen übergeben)
Damit es am Ende aufgeht mit (0,0,X) und es wird der erste gefundene Buchstabe von Hinten im Searchbuffer (bei mehreren matches) übergeben, Wenn der Vorschaubuffer zu Lang/der Suchbuffer zu Kurz ist, muss die letzte Übereinstimmung jeweils im Token übermittelt werden



(a) LZ77 Encoder Beispiel



(b) LZ77 Decoder Beispiel

6.1.2 LZW - Verfahren

Verlustloses Kompressionsverfahren für Grafikformate (GIF, TIFF)

Idee: Wörterbuch (statt Sliding Window), mit Nummeriertem Index, Wörterbuch wird anfangs Initialisiert mit allen Möglichen einzelnen Zeichen (bsp ASCII $= 0 - 255$ (Ascii hat immer 8 Bit pro Zeichen)) Token enthählt nur den Index des schon Bestehenden Zeichen, das Neue Zeichen wird mit dem nächsten Eintrag übermittelt (=Überlappung)
Jedes Zeichen ist neu und jedes Zeichen kommt 2x im Wörterbuch vor wenn schon bekannt dann wird neues längeres Zeichen erstellt.

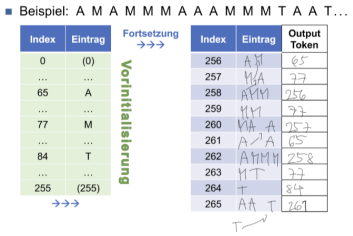


Abbildung 9: LZW Encoder Beispiel

Für Decodierung muss ein eigenes Wörterbuch aufgebaut werden und gleich erstellt werden...

7 Vorlesung 7 JPEG Codierung

Bildaufbau Digitales Bild: Pixel-Array mit $M \times N$ Matrix und $M \times N$ Auflösung oder dpi (dots per inch) Auflösunng

Bildtypen: schwarz-weiss (2-Wertig), Graustufen $(0...2^W$ typ. W =8 Bit), Natürlich: Farbpixel mit 3 Komponenten (zbsp RGB, benachbarte Pixelwerte fast Gleich gross), Grafik oder synthetisches Bild (scharfe Kanten, discrete-tone Bereiche mit identischen Pixeln)

- Bilder wichtig aber gross... dh. grosse Speicher und Übertragungskosten
- benachbarte Pixel haben oft dieselbe Farbe (Helligkeit) => Kompression muss Korrelation benachbarter Pixel ausnützen.
- Wörterbuch Kompression ungeeignet da benachbarte Pixel selten identisch/repetitiv ebenso wenig wie Zeilenweises Scanning
- Meist von Menschen betrachtet daher geweisser Verlust akzeptabel

7.1 7 Schritte der JPEG Kompression + Prinzipien

1. Transformation Farbbilder RGB => Luminanz / Chrominanz:
   Auge viel empfindlicher auf kl. Helligkeitsunterschiede als auf Farbunterschiede (Irrelevanz)

2. Downsampling der beiden Chrominanz-Komponenten:

   - Downsampling = Die Chrominanz-Ebenen werden in der Horizontalen/Vertikalen mehrere Pixel zsmgefasst.
   - Schema Indikator gibt an wie: J:a:b [J=Pixel Breite Ref Block, immer 2 Pixel Höhe Ref Block, a und b geben an wie viele Pixel nach dem Downsampling auf der **ersten Zeile (a)** und auf der **zweiten Zeile (b)** vorliegen, (a) wird zuerst verarbeitet.

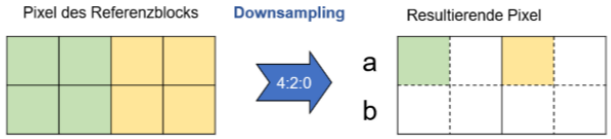


Abbildung 10: Downsampling Beispiel (4:2:0) (**Achtung, kann nur bei Cb+ Cr gemacht werden, Y beleibt gleich gross)**

3. Pixel-Gruppierung Farbkomponenten in 8x8 Blöcke
   Ausnützung der Horizontalen und Vertikalen Korrelation => Blöcke werden separat komprimiert (dies gibt Fehler: Schwachstelle)

4. Diskrete Cosinus Transformation (DCT (8x8))
   Transformation in den Frequenzbereich: Vorbereitung für Datenkompression, DC und tieffrequente AC-Werte enthalten nun Bildinformation.

5. Individuelle Quantisierung einzelner Frequenzkomponenten
   Prinzip: Frequenzkomponenten mit viel/wenig Bildinformation werden fein/grob quantisiert

   - Je mehr ein Bild Quantisiert wird desto mehr wird es vereinfacht (die Kompressionsrate wird durch (Pixel nachher/Pixel vorher) dargestellt)
   - Wenn ein Bild 0.0 Quantisiert wird ist die Kompressionsrate > 1 durch die Entropy Codierung
   - Bei intensiver Quantisierung entstehen Artefakte (Bildfehler = mehrere gleiche Pixel die dann sichtbar sind)
   - Bei Grafiken werden Artefakte sehr schnell sichtbar, daher nur wenig oder gar nicht Quantisieren.

6. Entropy-Coding der quantisierten Frequenzkomponenten
   verlustlos, Kombination von RLE mit EOB (End Of Block = bis zum Ende nur noch 0 en) und Huffman (jeweils im Zickzackmuster bis zu den Werten die nicht 0 sind)

7. Hinzufügen von Header mit JPEG-Parametern

7.2 Blockverarbeitung

Jeder 8x8 Pixel-Block lässt sich exakt als gewichtete Summe der folgenden 64 Ortsfunktionen darstellen: Basisfunktion... (Mathematisch definiert, statt Pixel-Werte werden Koeffizienten der Ortsfunktionen codiert.
Kann man sich auch Graphisch Vorstellen als immer genauere Verbesserung durch Cosinus Addition nach Addition etc.
⇒ Dabei entstehen neue Werte. Alle Werte unter 0.5 werden auf 0 gesetzt. (hier entsteht zum ersten mal verlust.)
Es entsteht ein DCT Koeffizient (links oben, mittlere Helligkeit) und 63 ACT Koeffizienten (Struktur)

Abbildung 11: Basisfunktion

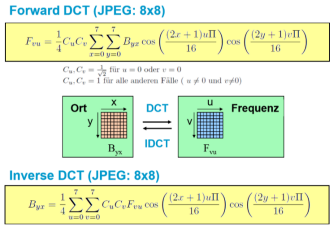


Abbildung 12: Definition zweidimensionale DCT

Bei glatter Fläche entstehen weniger hohe AC Werte (mehr Vereinfachung)
Schlangenmuster dabei von DC zuerst nach oben

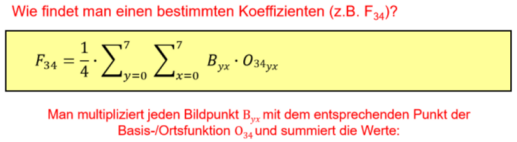


Abbildung 13: Finden der Koeffizienten

7.3 Anwendungsbereich von JPEG

Vorallem natürliche Bilder (Halbwertbilder)

8 Vorlesung 9 Audiocodierung

8.1 Abtasttheorem

Um ein analoges Audio-Signal in ein digitales umzuwandeln, wird der Pegel periodisch mit einer **Tastfrequenz** gemessen (= abgetastet) und einem

Wert zugewiesen (=**quantisiert**).
$F_{abtast} > 2 \cdot f_{max}$: [Wird dies nicht erfüllt, können Fehler entstehen = **Spiegelung** (Eine Sinuskurve wird viel weniger hoch Periodisch angezeigt als sie ist)]

8.2 Quantisierung

Quantisierungsrauschen = Die Differenz zw. Quantisierung und Signal. wird kleiner je mehr Bits verwendet wurden.

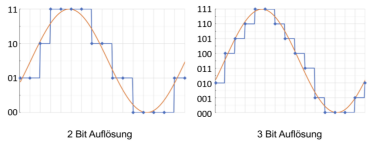
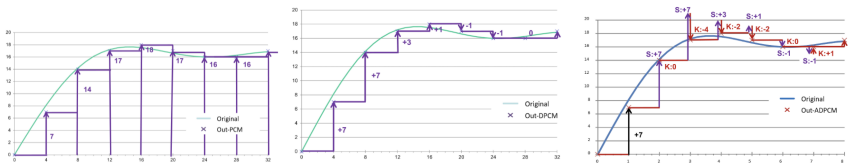


Abbildung 14: Quantisierung (blau)

8.3 Pulse Code Modulation (PMC) und Anwendungen

Anwendungsbeispiele: Sprachcodierung für Telefonie, Musikcodierung (Audio-CD), DVD Audio
Gibt dann jeweils andere Bit-Übertragungsbereiche.



(a) Lineare PMC (b) Differentiale PMC (c) Adaptive PMC

Abbildung 15: Different PMC Methods

8.3.1 Linear Prediction Coder (LPC)

Modellierung des Sprachorganes (Vocoder): Bilden ein vereinfachtes Modell des menschlichen Stimmtrakts nach, mit dem die Sprache synthetisiert werden kann. => Nur Koeffizienten werden übertragen. => Group Calls und Music wird dann je nach LPC schwierig.

8.4 Wave File Format

- Bestimmen bei Audacity: 10 kHz à 10 Abtastungen zwischen 0..1 Millisekunden ablesen (obere Skala)
- Frequenz für 1 Kanal: Linke Skala (Höhe) 1.0 = 1kHz
- Dezibel: gute frage
- File grösse (Länge Header 44 Bytes, Auflösung 16 Bit, Ende bei 4s): (10'000 Sample/s * 4 s + 1) * 2 Byte/Sample * 2 Kanäle + 44 = 160'048 Bytes

Wave-File Format: unkompromiertes Audio.

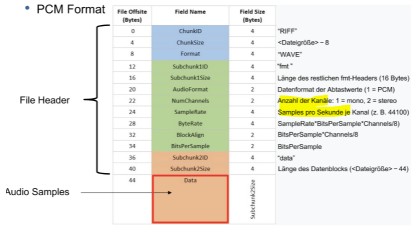


Abbildung 16: Wave-File Format

Ein Einzelnes Sample S_i : $S_i = K \cdot sin(\frac{i \cdot 2\pi f}{R})$ [f = Frequenz, R = Abtastrate, K = Skalaierungsfaktor]
Amplitude = K
Intervall = $\frac{2\pi f}{R}$

8.5 Verlustfreie Audio-Codierung

Bsp (meist genutzt) FLAC (Free Lossless Audio Codec) ähnlich wie Laufzeit Verfahren, Codierung durch lineare Vorhersage.

8.6 Verlustbehaftete Audiocodierung

(Schalldruckpegel (SPL) = logarithmische Grösse zur Beschreibung der Stärke eines Schallereignisses.

- Ausnutzung Menschlicher Hörschwelle (individuell und altersabhängig)
- Ausnutzung des Maskierungseffekts (ein lauter Ton maskiert andere Töne mit leicht unterschiedlicher Frequenz und leise Töne vor und nach lautem Ereignis sind nicht hörbar)

⇒ Verwendung von weniger Bits, aber so dass Quantisierungsrauschen gerade noch unter der Hörschwelle bleibt.

8.7 Clipping

Clipping (Hörbare Schwebung) ist wenn die oberen Teile einer Welle abgeschnitten werden da sie den Wertebereich überschreiten. Der Skalierungsfaktor K muss dann angepasst werden

9 Vorlesung 10 Kanalcodierung

9.1 Forward / Backward Error Correction

9.1.1 Backward Error Correction

Redundanz erlaubt nur **Fehler zu erkennen** => Neuübertragung der Daten kann angefordert werden.
Bsp: Blockcodes, CRC Nachteile:

- Rückkanal erforderlich
- Sender wartet auf Quitung (im Normalfall, im Fehlerfall auf Timeout)
- Nicht anwendbar bei Datenauslöschungen (Defekten)
- Nicht Anwendbar bei einmaligen Ereignissen
- Nicht Anwendbar bei zu hoher Fehlerwahrscheinlichkeit

9.1.2 Forward Error Correction

Redundanz der Kanalcodierung erlaubt beim Empfänger **Fehler zu korrigieren**.
Bsp: Blockcodes, Minimum-Distance-Decoding, Faltungscodes fehlerwahrscheinlichkeit von mehrbit fehlern einer Sequenz

9.2 Binäre Kanäle

Können nur 0 und 1 übertragen => Fehler wirkt sich aus dadurch dass einzelne Werte in das jeweilg andere umgewandelt werden.
ε = Bitfehlerwahrscheinlichkeit (BER): Störquelle die mit Wahrscheinlichkeit ε ein Bit des Kanalcodes verfälscht.
Bitfelherrate = Bitfehler pro Zeit

9.2.1 Binäre Symmetrische Kanäle (BSC)

Fehlerwahrscheinlichkeit ist unabhängig vom Eingangssymbol. Die Wahrscheinlichkeiten sind so wie im Bild.

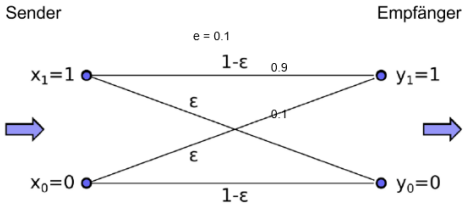


Abbildung 17: Wahrscheinlichkeiten im Symmetrischen Binären Kanal

9.2.2 Fehlerwahrscheinlichkeit eines Datenblocks

Erfolgswahrscheinlichkeit (Fehlerfrei): $P_{0,N} = (1-\varepsilon)^N$

Fehlerwahrscheinlichkeit auf N Datenbits: $= 1 - P_{0,N} = 1 - (1-\varepsilon)^N$
wobei für $N \cdot \varepsilon \ll 1$ gilt Näherung: $(1-\varepsilon)^N \cong (1 - N \cdot \varepsilon)$

9.3 Mehr-Bit Fehlerwahrscheinlichkeiten

$$P_{F,N} = \binom{N}{F} \cdot \varepsilon^F \cdot (1-\varepsilon)^{N-F} \tag{4}$$

Equation 4: Wahrscheinlichkeit dass in einer Sequenz von N Datenbits genau F Bitfehler auftreten (use nCr)

Für Maximal F Fehler: $P_{\leq F,N} = \sum_{t=0}^{F} P_{F=t,N}$
Restwahrscheinlichkeit (mehr als F Fehler): $= P_{>F,N} = 1 - P_{\leq F,N}$
[schwierig zu berechnen], besser: $= P_{>F,N} = \sum_{t=F+1}^{N} \binom{N}{t} \cdot \varepsilon^t \cdot (1-\varepsilon)^{N-t}$

9.4 Grundlagen zur Fehlererkennung

9.4.1 Kanalkapazität

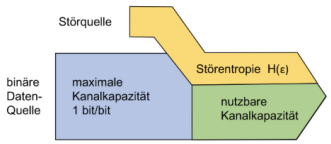


Abbildung 18: Kanalkapazität

$$C_{BSC}(\varepsilon) = 1 - H_b(\varepsilon) = 1 - (\varepsilon \cdot log_2 \frac{1}{\varepsilon} + (1-\varepsilon) \cdot log_2 \frac{1}{1-\varepsilon}) \tag{5}$$

Equation 5: Nutzbare Kanalkapazität

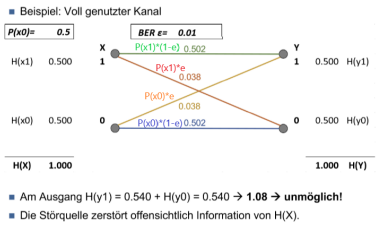


Abbildung 19: Entropie des BSC wird durch Störungsquelle beeinträchtigt (Die Übertragene Information ist diejenige die nicht durch die Störungsquelle verloren ging)

9.5 Binäre Kanalcodes

Binäre Kanalcodes sind eine Sammlung von Codewörtern die mit Vorkehrungen versehen sein können damit Übertragungsfehler erkannt/korrigiert werden können.
(evtl auch siehe oben Binäre Kanäle)

9.6 Hamming Distanz

Anzahl wechselnde Bits von einem Code zum Nächsten. **Minimale Hammingdistanz eines Codes**: $d_{min}(C)$ = kleinste Distanz zw. beliebigen Codewörtern. **Perfekter Code:** Heisst ein Code wenn jedes empfangene Wort genau ein Wort hat zu dem es einen geringsten Hamming-Abstand hat und damit eindeutig zugeordnet werden kann.

- Hammingdistanz ≥ 2: Erkennung von Ein- oder Mehr-Bitfehlern
- Hammingdistanz ≥ 3: Korrektur von Fehlern

=>Anwendung: XOR Bilden und Hamming Gewicht von XOR = Hamming Distanz

9.7 Hamming Gewicht

Gibt an wieviele 1en das Codewort enthält. => Einfach zählen.

9.8 Systematik, Linearität, Zyklität

Coderate: $R = \frac{K}{N}$ [N = Codebits, K = Informationsbits am Stück]
K = log_2(Anz. versch. Codes)
$\Rightarrow$ Damit alle Information in nutzbaren Bits Platz hat muss R < C (**Kanalcodierungstheorem**) !

9.8.1 Systematischer (N,K) Blockcode

N-K Fehlerschutzbits und K Informationsbits (In beliebiger Reihenfolge)

9.8.2 Linearität

XOR verknüpfung mit einem 2 beliebiger Codewörter gibt wieder ein beliebiges Gültiges Codewort
Null-Codewort muss zwingend enthalten sein damit Linearität gilt.

9.8.3 Zyklizität

Linearer, Zyklischer N,K-Blockcode: Zyklische Verschiebung eines Codewortes gibt wieder ein (beliebiges) Codewort.

10 Vorlesung 11 Fehlererkennung

Es gibt: $P \geq log_2(N+1)$ [P = Prüfbits = Paritätsbits = Parity-Bit] die notwändig sind zur Fehlererkennung von 2^K gültigen Codeworten in 2^N Kombinationen.

- Even Parity (linear) / Odd Parity (nicht linear) (50% Fehlererkennung): Jeweils Hammingdistanz -1 Fehler können erkannt werden. Parity wird von Sender definiert und berechnet jeweils Modulo so dass even/odd erfüllt ist.
- Prüfsumme: $\sum_{i=0}^{n-1}$Element[i]: Hat Hamming Distanz 2, Kann Fehler besser als Parity erkennen wenn viele Fehler in der gleichen Spalte passiert wurden.

10.1 1-Bit Arithmetik

Multiplikation r = $a \cdot b$ (AND)

a	b	r
0	0	0
0	1	0
1	0	0
1	1	1

(a) Multiplikation

Subtraktion r = $a \oplus b$ (XOR)

a	b	r
0	0	0
0	1	1
1	0	1
1	1	0

Addition r = $a \oplus b$ (XOR)

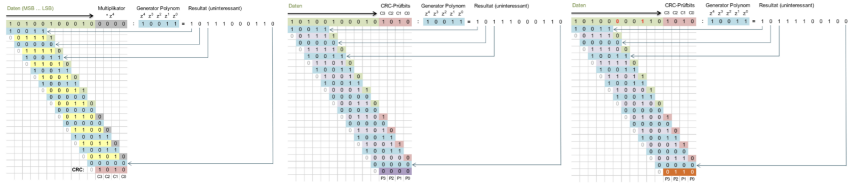
a	b	r
0	0	0
0	1	1
1	0	1
1	1	0

(b) Subtraktion = Addition

10.2 Fehlererkennung mit CRC (Cyclic Redundandcy Check)

Binäre Zahlen werden zu Koeffizienten eines Polynoms
(10011 > $1 * z^4 + 0 * z^3 + 0 * z^2 + 1 * z^1 + 1 * z^0 = z^4 + z + 1$)

- Generatorpolynom g (siehe oben) vom Grad m
- Encoder: Nachricht (= Polynom p) plus anhängen von m Nullen, Codewort durch g teilen und Rest r bilden. => r = CRC
- p + CRC (anstatt Nullen) ist nun mit r=0 teilbar durch g
- Decoder: Empfangene Nachricht h + CRC wenn teilbar mit r = 0 => Fehlerfrei, teilbar mit Rest => Fehler, Falsch Fehlerfrei wenn h Fehlervektor (falsche Bits aneinandergereit mit 0 wo nicht verfälscht) teilbar durch g



(c) Encodierung mit Polynomdivision (d) Decodierung mit Polynomdivision (e) Decodierung mit Fehler

Name	Polynom G(z)	Bemerkungen
CRC-1	$z+1$	Parity-Bit (gerade)
CRC-4	z^4+z+1	Identisch zu (15,11)-Hamming-Code
CRC-5-USB	z^5+z^2+1	Identisch zu (31,26)-Hamming-Code
CRC-5-ITU	$z^5+z^4+z^2+1$	Verwendet bei Bluetooth.
CRC-7	z^7+z^3+1	Identisch zu (127,120)-Hamming-Code
CRC-8-CCITT	z^8+z^2+z+1	Verwendet für ATM, ISDN
CRC-12	$z^{12}+z^{11}+z^3+z^2+z+1$	Verwendet für Telecom
CRC-16-CCITT	$z^{16}+z^{12}+z^5+1$	Verwendet für X.25
CRC-16-IBM	$z^{16}+z^{15}+z^2+1$	Verwendet z.B. für Modbus, USB
CRC-32	$z^{32}+z^{26}+z^{23}+z^{22}+z^{16}+z^{12}+z^{11}$ $+z^{10}+z^8+z^7+z^5+z^4+z^2+z+1$	Verwendet für Ethernet, ZIP, PNG, SATA, MPEG-2, ZMODEM
CRC-64-ISO	$z^{64}+z^4+z^3+z+1$	

Abbildung 20: Generator Polynome

10.2.1 Hardware Implementierung

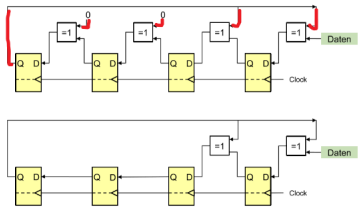


Abbildung 21: Hardware CRC Implementierung und Vereinfachng am Beispiel von 10011

11 Vorlesung 12 Fehlerkorrektur

11.1 Forward Error Correction

Beispiele

- Repetitions-Code (zBsp 3x) => Infobit wird 3x gesendet (110 -> 111 111 000)
- Hamming-Distanz von 3 erlaubt Zweibitfehlererkennung oder 1 - Bit Fehlerkorrektur zum wahrscheinlichsten Codewort.

Regel Fehlererkennung von k Bitfehler: $k \leq (d_{min}-1)$ Fehlerkorrektur von k Bitfehler (abgerundet): $k \leq (d_{min}-1)/2$ Anz. Prüfbits um 1 Einbitfehler in K Datenbits zu korrigieren können: $I(p) \approx I(K+1) = log_2(N+1)$ => Diese wären optimal und zu finden mit linearen Blockcodes

11.2 Lineare Blockcodes

- Definiert durch Generatormatrix (Paritätsmatrix + Einheitsmatrix)
  $\Rightarrow$ Alle Zeilen sind linear unabhängig und haben mind. d_{min} 1en inkl der einen 1 in der Generatormatrix-Zeile.
- Für Eingabe in Generatormatrix anz. Zeilen der Generatormatrix und alle Möglichen Kombinationen von Binärcodes eingeben... gibt alle möglichen Codewörter 2^n
- Codewort wird erzeugt durch Multiplikation mit Generatormatrix
- Decoder: Multiplikation des Empfangenen Bitmusters mit der Prüfmatrix => Syndrome jeweils 000 wenn fehlerfrei und nicht 000 wenn nicht
  $\Rightarrow$ Prüfmatrix = Generatormatrix oben oder unten an Paritätsmatrix angehängt und auf Grösse der Paritätsmatrix verkleinert.

$$\begin{pmatrix} 3 & 2 & 1 \\ 1 & 0 & 2 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 0 \\ 4 \end{pmatrix} = \begin{pmatrix} 3\cdot 1+2\cdot 0+1\cdot 4 \\ * \end{pmatrix} = \begin{pmatrix} 7 \\ * \end{pmatrix}$$

Abbildung 22: Matrix Vektor Produkt für Lineare Blockcodierung Abekippe und multipliziere + addiere

12 Vorlesung 13 + 14 Faltungscodes

Korrektur v. mehr als 1 Bit Fehler.
Statt fixen Bit-Blöcken verwendet man ein Fenster das über den Datenstrom schiebt (= Schieberegister), Codebits werden aus dem Bitsmuster im Fenster ermittelt = Faltung

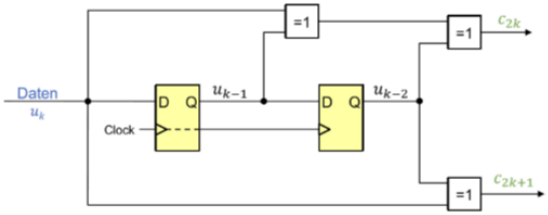


Abbildung 23: Encoder Implementierung Hardware (linear, 2 Gedächtnislänge) m = Anz. Bits = Tail-Bits (Tailbits) 2^m = Anz Reihen im Tellis Diagramm

- Faltungscodes nicht systematisch aber linear
- könen mit Hilfe von Schieberegistern & binärer Logik erzeugt werden.
- keine Systematische Methode zur Auffindung von Faltungscodes
- Encoder einfach, Decoder schwieriger zu Realisieren (bsp Viterbi (Kostenprinzip))
- Faltungscodes haben nicht d_{min} als Hammingdistanz sondern Freie Distanz (ist dasselbe und $d_{free} = w_{min}$ aber die freie Distanz ist gleich **der minimalen Hamming–Distanz zwischen zwei beliebigen Pfaden durch das Trellis** (Suche min Distanz). Je länger das Gedächtnis desto grösser d_{free}
- aktueller Ausgangszustand eines Faltungscodierers hängt ab von aktuellem Eingangszustand und m früheren Eingangszuständen.
- Die Generatoren γ und die Länge des Schieberegisters m bestimmen die Fehlerwahrscheinlichkeit und die d_{free}
- Coderate Faltungscode: $R = \frac{K}{2 \cdot (K+m)}$ [mit K = Codebits (von u nicht Codiert), m = Tailbits (nicht Codiert)] (optimal bei 0.5)

m	$\gamma = 2$ Generatoren	d_{free}
2	$(101_b, 111_b)$	5
3	$(1101_b, 1111_b)$	6
4	$(10011_b, 11101_b)$	7
5	$(101011_b, 111101_b)$	8
6	$(1011011_b, 1111001_b)$	10
7	$(10100111_b, 11111001_b)$	10
8	$(101110001_b, 111101011_b)$	12

Gewichtungsvektoren

Abbildung 24: Optimum Free Distance (OFD = Codes die d_{free} erreichen)

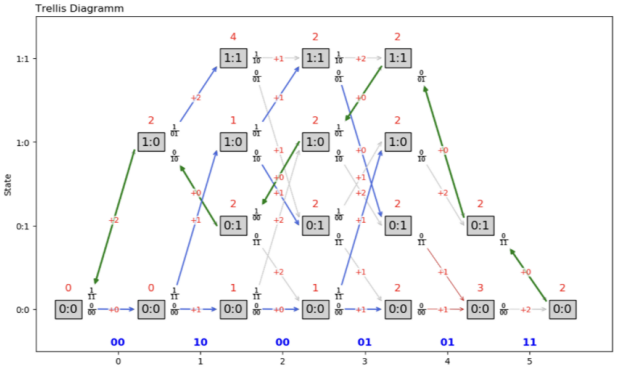


Abbildung 25: Trellis Diagramm mit 2 Fehlern, Decodierung (der einzige mögliche Weg rückwärts), der Pfad mit dem kleinsten Kosten ist der Wahrscheinlichste. Dieser basiert auf Pfadfehlern (anz. benötigter Fehler um zum Zustand zu gelangen)