Kapitel 3: Kanalcodierung

Prof. Dr. Dirk Staehle

Vorlesung Kommunikationstechnik

Übersicht Kapitel 3: Kanalcodierung

Beispiel Kanalcodierung

Codewort

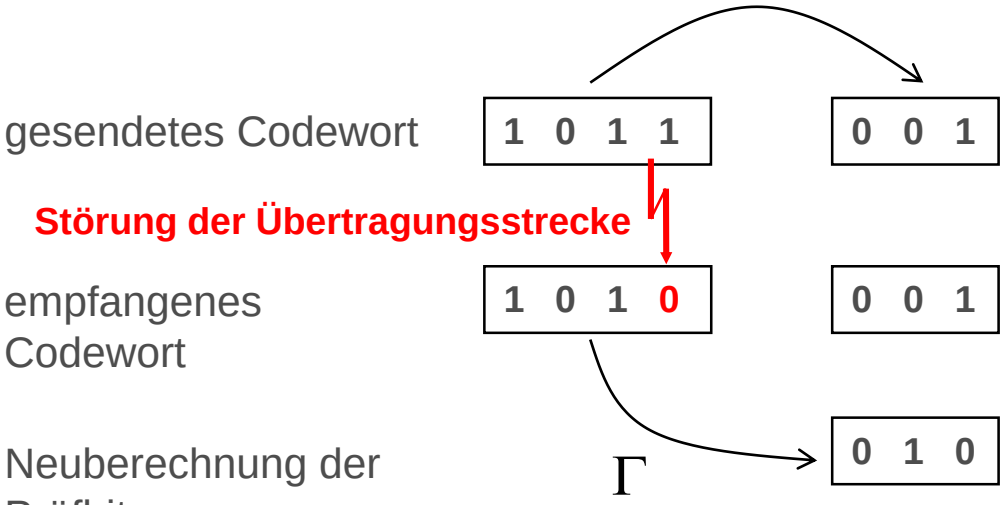
$\underbrace{x_1\ x_2\ x_3\ x_4}_{X}\ \underbrace{y_5\ y_6\ y_7}_{Y}$

Codierungsvorschrift

$$Y = \Gamma[X] \quad \begin{cases} y_5 = x_1 \oplus x_2 \oplus x_3 \\ y_6 = x_1 \oplus x_2 \oplus x_4 \\ y_7 = x_1 \oplus x_3 \oplus x_4 \end{cases}$$

Matrixdarstellung

x_1	x_2	x_3	x_4	y_5	y_6	y_7
1	1	1	0	1	0	0
1	1	0	1	0	1	0
1	0	1	1	0	0	1



Fehlerkorrektur durch stellenweise Addition von Y* und Y'

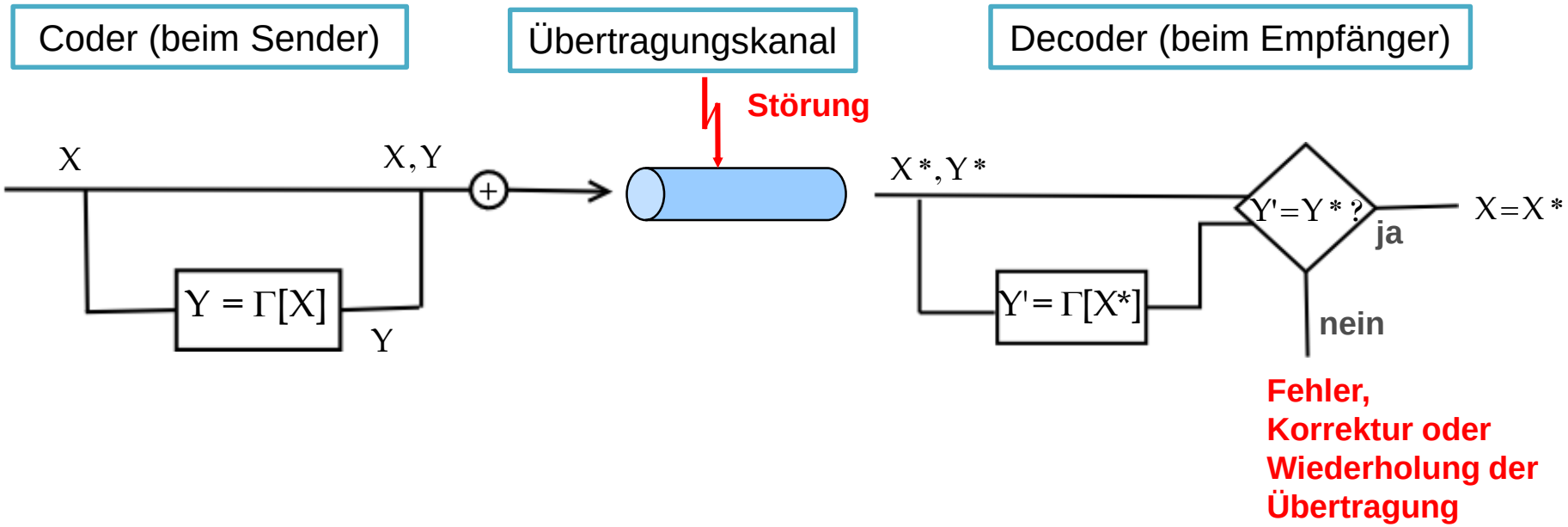
Y* **Y'** **Fehlersyndrom**

$$\begin{pmatrix}0\\0\\1\end{pmatrix} \oplus \begin{pmatrix}0\\1\\0\end{pmatrix} = \begin{pmatrix}0\\1\\1\end{pmatrix}$$

➡ **Spalte 4, 4. Stelle gestört !**

Übersicht Kapitel 3: Kanalcodierung

Prinzip der Kanalcodierung



X:	Quellcodewort (bestehend aus Nutzinformation)
$\Gamma(\,)$:	Vorschrift zur Berechnung der nützlichen Redundanz
Y:	nützliche Redundanz (in Form von Prüfbits oder Kontrollstellen)
X, Y:	gesendetes Codewort
X^*, Y^*:	empfangenes, evtl. störungsbehaftetes Codewort
Y':	vom Empfänger neu berechnete, nützliche Redundanz

Kanalcodierung

- Unter Kanalcodierung versteht man das Hinzufügen von zusätzlichen Bits (**nützlicher Redundanz**) zu einem **Nutzwort**, um eventuell auftretende **Bitfehler erkennen oder korrigieren** zu können.
- Das Nutzwort inklusive der zusätzlichen Bits wird als **Codewort** bezeichnet.
- Notation:
  - K: Länge eines Nutzwortes, Anzahl der Nutzbits
  - N: Länge eines Codeworts
  - N-K: Anzahl der zusätzlichen Bits (bei systematischen Codes: Prüfbits)
  - K/N: Coderate

Anwendungsgebiete

- Datenübertragung
  - Datensicherungsprotokolle in Rechnernetzen
    - fast alle Datenübertragungen in Rechnernetzen werden mit Prüfsummen (engl. Checksum) abgesichert, um das Weiterleiten von fehlerhaften Daten zu vermeiden
      - DSL, CAN-Bus, Ethernet, Bluetooth, Mobilfunk, TCP, IP, UDP, etc.
  - Fehlerreduzierung in mobilen Kommunikationssystemen
    - bei der drahtlosen Übertragung treten häufig Fehler auf
    - eine sehr niedrige Bitfehlerwahrscheinlichkeit kann nur auf Kosten von Datenrate und Reichweite erzielt werden
    - stattdessen werden relativ hohe Bitfehlerwahrscheinlichkeiten durch effiziente Kanalcodierungsverfahren akzeptabel
- Datenspeicherung
  - Fehlerreduzierung in Massenspeichern (Band, Platte, CD, DVD, Flash) und hochintegrierten Halbleiter-Speichern (RAM, ROM)

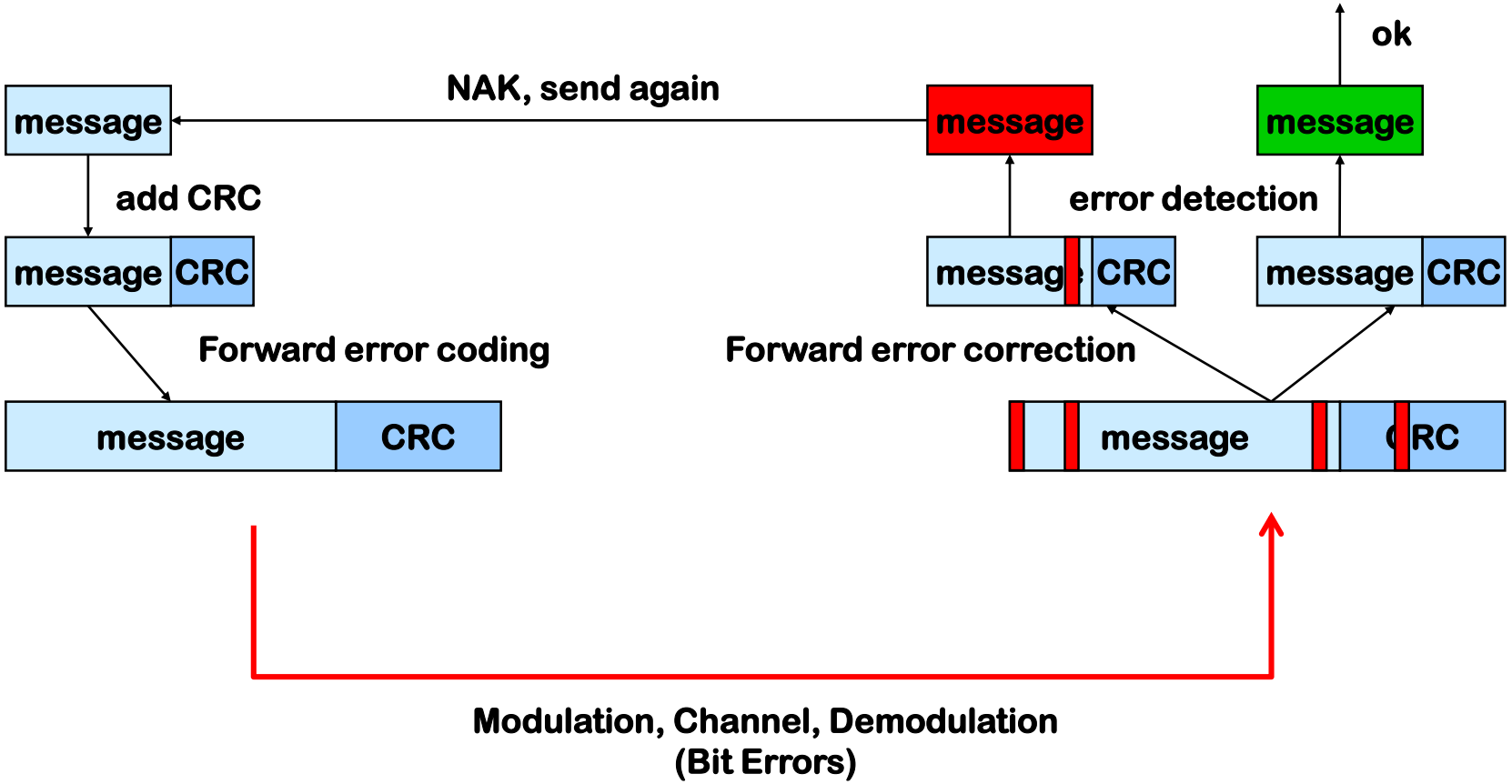
Codes zur Fehlererkennung und Fehlerkorrektur

- Fehlererkennung:
  - Empfänger erkennt Übertragungsfehler typischerweise durch
    - einfache Paritätsprüfung
    - Zyklische Redundanz Prüfung (Cyclic Redundancy Check, CRC)
- Fehlerkorrektur (Forward Error Correction, FEC):
  - Empfänger korrigiert Übertragungsfehler typischerweise durch
    - Block-Codes
      - einfache Codes: Hamming-Code, BCH-Code, etc.
      - Reed-Solomon Codes (z.B. DVB, CD)
      - Low Density Parity Check-Codes, LDPC (DVB-S2,IEEE 802.11n)
      - Repetition Coding, mehrfaches Übertragen von Bits (Mobilfunk)
    - basierend auf Faltungscodes (Convolutional Codes)
      - einfache Faltungscodes (Mobilfunk, IEEE 802.11)
      - iterative Faltungscodes, Turbo Codes (Mobilfunk)

Betriebsarten

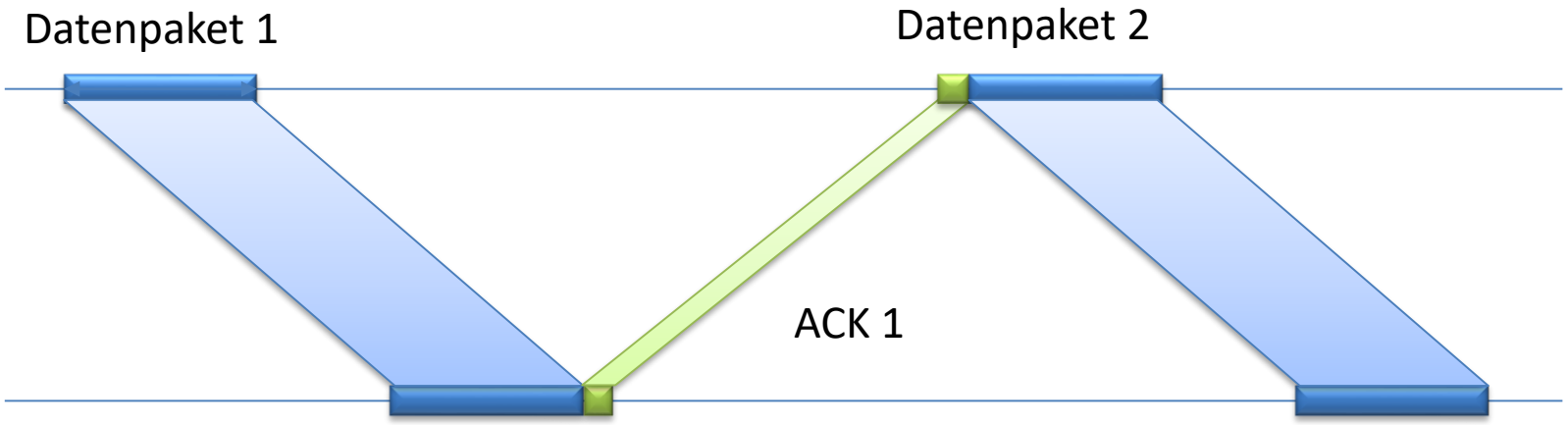
- „Reine“ Wiederholungsaufforderung
  - nach der Erkennung eines Fehlers wird eine Übertragungswiederholung angefordert
  - geringe Restfehlerwahrscheinlichkeit
    - Trade-Off zwischen Code-Overheads und Restfehlerwahrscheinlichkeit
  - aufwändigere Übertragungsprotokolle und zusätzlicher Delay
  - Einsatz in Systemen mit geringer Bitfehlerwahrscheinlichkeit
    - Bus-Systeme (CAN, Ethernet, etc.)
- Mischbetrieb (Fehlerkorrektur und Wiederholungsaufforderung)
  - zweistufiges Verfahren: Fehlererkennung findet nach der Fehlerkorrektur statt
    - Wiederholungsaufforderung (Acknowledged Mode)
    - Verwerfen/Akzeptanz des fehlerhaften Pakets (Unacknowledged Mode)
  - Einsatz in Systemen mit signifikanter Bitfehlerwahrscheinlichkeit
    - Funkübertragung, DSL
- Unüblich: automatische Fehlerkorrektur
  - gewisse Restfehlerwahrscheinlichkeit muss akzeptabel sein
  - möglich z.B. bei Übertragung von analogen Signalen (Sprache), wenn kleinere Verfälschungen akzeptabel sind

Beispiel: Funkübertragung



Wiederholungsaufforderung: Send-and-Wait

- Send-and-Wait Protokoll:
  - Sender überträgt ein Paket und wartet auf eine Bestätigung
  - nach korrektem Empfang eines Datenpakets sendet der Empfänger eine Bestätigung (Acknowledgement, ACK)
  - nach Erhalt der Bestätigung überträgt der Sender das nächste Datenpaket
  - falls nach einiger Zeit keine Bestätigung eintrifft (Timeout), wiederholt der Sender die Übertragung des Datenpakets



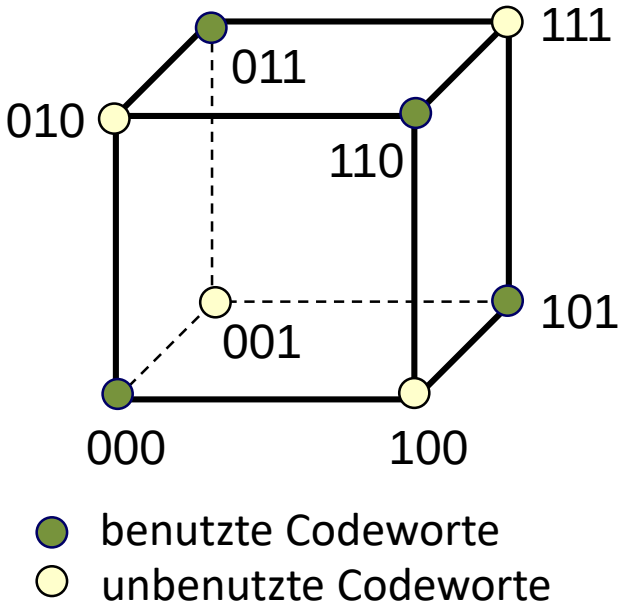
Übersicht Kapitel 3: Kanalcodierung

Beispiel: Einfache Paritätskontrolle

- Codierung:
  - Hinzufügen eines Paritätsbits
    - „0“: gerade Anzahl von „1“en
    - „1“: ungerade Anzahl von „1“en
  - Codierung für Nachricht mit 2 Bits

x_1	x_2	y_1
0	0	0
0	1	1
1	0	1
1	1	0

Codeworte im Raum



- Welche Fehler können erkannt werden? Welche Fehler können korrigiert werden?

Code und Korrekturbereich

- **Code:** Ein Code ist definiert als die Menge aller benutzten Codeworte.
  - benutztes Codewort: $y = \Gamma(x)$
  - Code: $C = \{y | \exists x: \ y = \Gamma(x)\}$
- **Korrekturbereich** eines benutzen Codeworts:
  - Menge der unbenutzten Codeworte, deren Distanz zu diesem benutzten Codewort am kleinsten ist
  - Codeworte innerhalb eines Korrekturbereichs können einem benutzten Codewort zugeordnet und korrigiert werden
  - **Korrekturkugel**: symmetrischer Korrekturbereich
  - **dichtgepackter Code**: es existieren keine Codewörter außerhalb der Korrekturbereichs

Hamming-Distanz

- **Hamming-Distanz** $h(x, y)$ von zwei Codeworten x, y:
  - Anzahl der verschiedenen Bits

- **Minimale Hamming-Distanz** h_C eines Codes C:
  - kleinste Distanz zwischen zwei benutzten Codeworten

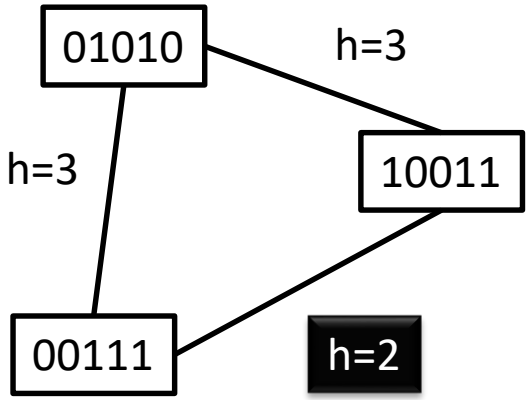
$$h_C = \min_{x,y \in C} h(x, y)$$

- Anzahl sicher **erkennbarer Fehler**: $\bar{\eta}_C = h_C - 1$

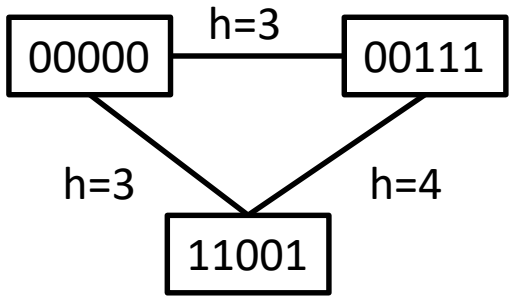
- Anzahl sicher **korrigierbarer Fehler**: $\eta_C = \left\lfloor \frac{h_C - 1}{2} \right\rfloor$

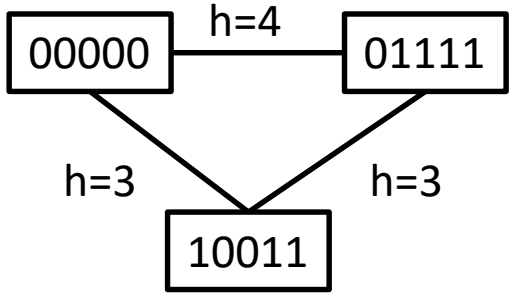
Beispiel Hamming-Distanz

- Was ist die minimale Hamming-Distanz dieser drei Codeworte?



- Was ist die maximale minimale Hamming-Distanz bei drei Codeworten mit 5 Bits?



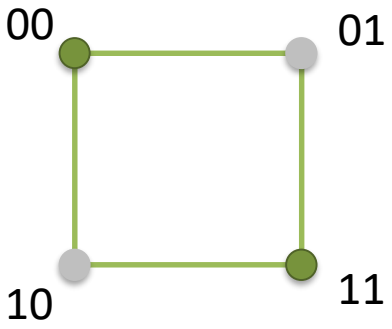


- Bei drei Codeworten mit 5 Bits können maximal 2 Bitfehler erkannt und ein Bitfehler sicher korrigiert werden

Beispiel: Repetition-Coding

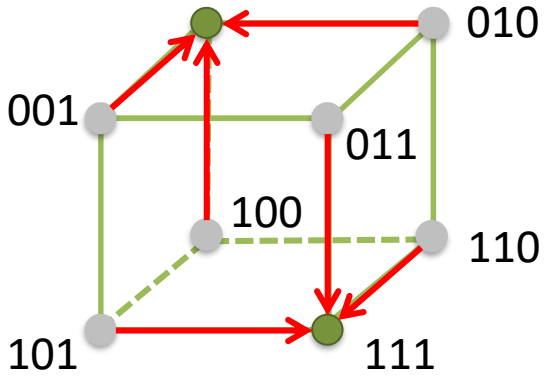
- Repetition-Coding (Codierung durch Bitwiederholung):
  - übertrage ein Bit nicht 1-mal sondern n-mal

Beispiel: n=2



Hamming-Distanz: 2
Anzahl erkennbarer Fehler: 1
Anzahl korrigierbarer Fehler: 0

Beispiel: n=3



Hamming-Distanz: 3
Anzahl erkennbarer Fehler: 2
Anzahl korrigierbarer Fehler: 1

Allgemein:

Hamming-Distanz: n
Anzahl erkennbarer Fehler: $n-1$
Anzahl korrigierbarer Fehler: $\lfloor (n-1)/2 \rfloor$

Beispiel: Korrekturbereiche und Hamming-Distanz

Codewort

$\underbrace{x_1\ x_2\ x_3\ x_4}_{X}\ \underbrace{y_5\ y_6\ y_7}_{Y}$

Codierungsvorschrift

$$Y = \Gamma[X] \quad \begin{cases} y_5 = x_1 \oplus x_2 \oplus x_3 \\ y_6 = x_1 \oplus x_2 \oplus x_4 \\ y_7 = x_1 \oplus x_3 \oplus x_4 \end{cases}$$

Matrixdarstellung

x_1	x_2	x_3	x_4	y_5	y_6	y_7
1	1	1	0	1	0	0
1	1	0	1	0	1	0
1	0	1	1	0	0	1

- Code:
  - Anzahl Nutz-Bits: $K = 4$; Anzahl benutzter Codeworte: $2^K = 16$
  - Codewortlänge: $N = 7$; Anzahl Codeworte: $2^N = 128$
  - Anzahl korrigierbarer Fehler: $\eta = 1$
- Größe des Korrekturbereich eines benutzten Codeworts:
  - Der Korrekturbereich umfasst alle Codeworte, die sich um höchstens $\eta = 1$ Bit von einem benutzten Codewort unterscheiden. Es gibt $N = 7$ benachbarte Codeworte, die sich um ein Bit unterscheiden. Das benutzte Codewort zählt auch zum Korrekturbereich.
    - Anzahl Codeworte pro Korrekturbereich: $1 + 7 = 8$
    - Anzahl Codewort in allen Korrekturbereichen: 16*(1+7)=128
- Es handelt sich um einen dichtgepackten Code, da alle Codeworte in einem Korrekturbereich liegen
  - zwei Fehler werden immer erkannt und immer falsch korrigiert
  - drei Fehler können nicht sicher erkannt werden
- Hamming-Distanz: 3
  - 1 Fehler korrigierbar, 2 Fehler erkennbar

Übersicht Kapitel 3: Kanalcodierung

3.1 Prinzip der Kanalcodierung

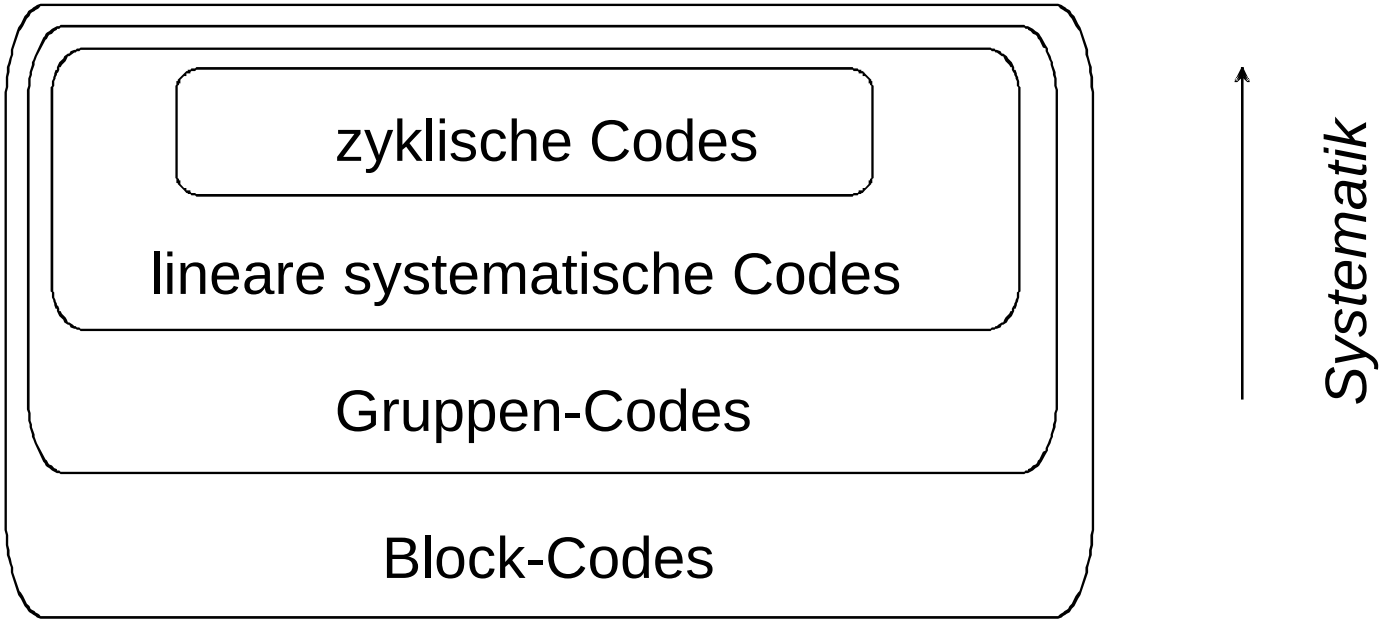
3.2 Blockcodes

3.2.1 Linear-systematische Blockcodes

3.2.2 Zyklische Blockcodes

3.3 Faltungscodes (Convolutional Codes)

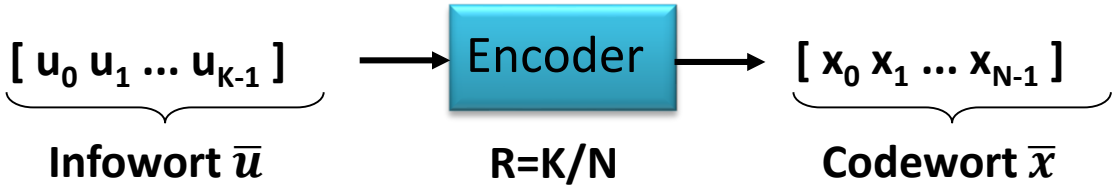
Übersicht Block-Codes



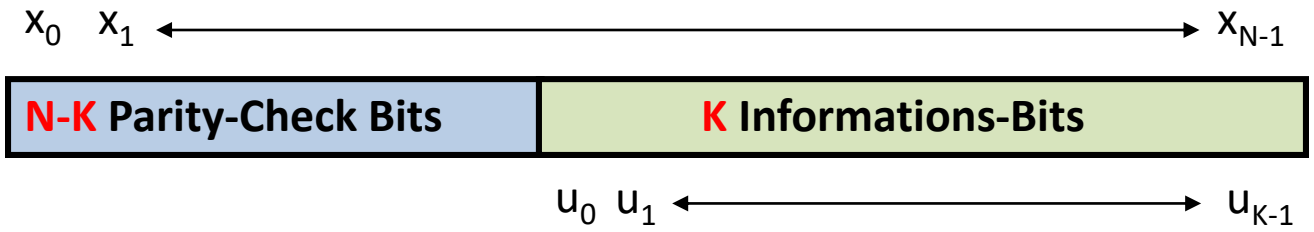
- Ein **Blockcode** ist ein Kanalcodierungsverfahren, bei aus einem Nutzwort mit einer festen Länge von K Bits (einem Block) eine Codewort mit N Bits berechnet wird. Ein **(N,K)-Code** funktioniert genau für diese eine feste Kombination aus N und K und weißt dann bestimmte Eigenschaften hinsichtlich Fehlererkennung und Fehlerkorrektur auf.
- **Faltungscodes**, die andere große Gruppe von Codes, sind flexibler und der Code (die Hardware) arbeitet unabhängig von der Anzahl Nutzbits. Faltungscodes werden daher nur über ihre **Coderate K/N** beschrieben. In der Praxis werden aber innerhalb einer Übertragungstechnologie auch Faltungscodes meist für wenige feste Blockgrößen eingesetzt.

Binäre Block-Codes

- Encoder für (N,K)-Block-Code $C = \{\bar{x}_0, \bar{x}_1, \cdots, \bar{x}_{2^K-1}\}$



- **Systematischer** (N,K) Block-Code C
  - K Info-Bits $\bar{u}$ erscheinen im Codewort $\bar{x}$
    - einfache Rückführung $\bar{x} \rightarrow \bar{u}$

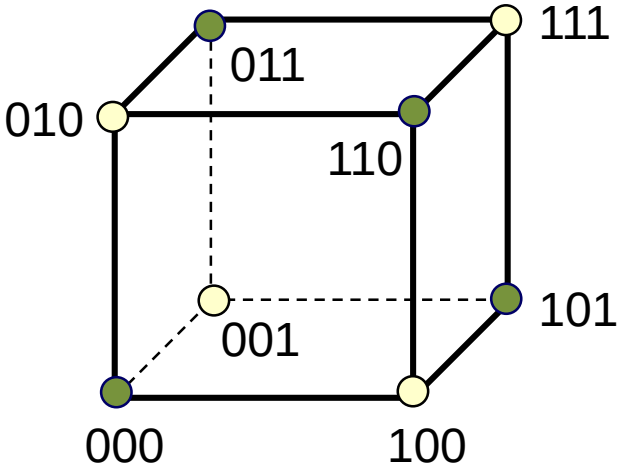


Binäre Block-Codes

- **Linearer** (N,K) Block-Code C
  - modulo-2 Summe von 2 Codewörtern ist wieder ein Codewort
- Linearer, **zyklischer** (N,K) Block-Code C
  - zyklische Verschiebung eines Codeworts ergibt wieder ein Codewort
- Eigenschaften linearer, zyklischer Block-Codes
  - einfache Beschreibung durch ein Generatorpolynom
  - einfach mit LFSR codierbar, günstige Eigenschaften für Decoder
    - LFSR: Linear Feedback Shift Register, linear-rückgekoppeltes Schieberegister
- Anwendung:
  - Fehlererkennung:
    - CRC (Cyclic Redundancy Check)
  - Fehlerkorrektur :
    - BCH-Codes
    - Reed-Solomon Codes

Beispiel: (3,2)-Block-Code

- (3,2)-Block-Code C = { [0 0 0], [1 1 0], [1 0 1], [0 1 1] }
- Encoder:
  - $K = 2$ Input-Bits
  - $2^K = 4$ Codeworte der Länge $N = 3$
  - Code-Rate $R = 2/3$
- Eigenschaften:
  - Hamming-Distanz $h_C = 2$
  - alle 1-Bit-Fehlermuster erkennbar
    - alle Fehlermuster mit ungerader Fehlerzahl erkennbar
    - alle Fehlermuster mit gerader Fehlerzahl nicht erkennbar
  - keine sinnvolle Fehlerkorrektur möglich



Linear? ✓ 110+101=011, 101+011=110, 110+011=101

Systematisch? ✓ letztes Bit ist Kontrollbit

Zyklisch? ✓ 000->000, 110->011, 011->101, 101->110

Übersicht Kapitel 3: Kanalcodierung

Codierungsvorschrift für lineare Codes: Generator Matrix

- Codeworte werden aus Nutzworten durch Multiplikation mit einer Generator-Matrix G erzeugt

$$(x_0, x_1, \ldots, x_{N-1}) = (u_0, u_1, \ldots, u_{K-1}) \cdot \begin{pmatrix} g_{0,0} & \cdots & g_{0,N-1} \\ \vdots & \ddots & \vdots \\ g_{K-1,0} & \cdots & g_{K-1,N-1} \end{pmatrix}$$

  - In Matrizenschreibweise: $\bar{x} = \bar{u} \cdot G$
- Linear-systematischer Code:
  - die Generator-Matrix G hat die Form $G = [P \quad I_K]$, wobei I_K die KxK-Einheitsmatrix ist
  - dadurch stehen die Nutzbits an den letzten K Stellen des Codeworts
- WICHTIG:
  - alle Berechnungen werden „modulo 2“ gerechnet
  - bei Additionen gibt es KEINEN Übertrag:

1101+1010=0111

Beispiel: Linearer (7,4) Hamming-Code

- Code ist durch die Generator-Matrix G beschrieben

$$G = \begin{pmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 \end{pmatrix}$$

- Codierung des Nutzwortes $\bar{u} = (0\ 1\ 1\ 0)$

$$\bar{x} = (0\ 1\ 1\ 0) \cdot \begin{pmatrix} 1\ 1\ 0\ 1\ 0\ 0\ 0 \\ 0\ 1\ 1\ 0\ 1\ 0\ 0 \\ 1\ 1\ 1\ 0\ 0\ 1\ 0 \\ 1\ 0\ 1\ 0\ 0\ 0\ 1 \end{pmatrix} = (1\ 0\ 0\ 0\ 1\ 1\ 0)$$

$$\begin{aligned} x_0 &= u_0 + u_2 + u_3 \\ x_1 &= u_0 + u_1 + u_2 \\ x_2 &= u_1 + u_2 + u_3 \\ x_3 &= u_0 \\ x_4 &= u_1 \\ x_5 &= u_2 \\ x_6 &= u_3 \end{aligned}$$

Parity-Check-Matrix

- Jeder lineare (N,K) Code hat eine (N-K) x N Parity-Check-Matrix H mit

$$(x_0, x_1, \ldots, x_{N-1}) \cdot H^T = (0, \ldots, 0) \text{ da } \bar{x} \cdot H^T = \bar{0}$$

  - ergibt also ein empfangenes Codewort bei Multiplikation mit H^T nicht 0, dann muss ein Bitfehler vorliegen

- Bei linear-systematischen Codes lässt sich die Parity-Check-Matrix direkt aus der Generator-Matrix konstruieren

$$G = [P\ I_K] \Longrightarrow H = [I_{N-K}\ P^T]$$

- Beispiel: **Hamming-Code**

$$G = \begin{pmatrix} 1&1&0&1&0&0&0 \\ 0&1&1&0&1&0&0 \\ 1&1&1&0&0&1&0 \\ 1&0&1&0&0&0&1 \end{pmatrix} \Rightarrow H = \begin{pmatrix} 1&0&0&1&0&1&1 \\ 0&1&0&1&1&1&0 \\ 0&0&1&0&1&1&1 \end{pmatrix}$$

Fehlererkennung und Fehlerkorrektur

- Beispiel:
  - Nutzwort: $\bar{u} = (0\ 1\ 1\ 0)$
  - Codewort: $\bar{x} = (1\ 0\ 0\ 0\ 1\ 1\ 0)$
  - Fehlervektor: $\bar{e} = (0\ 1\ 0\ 0\ 0\ 0\ 0)$
  - empfangenes Codewort: $\bar{y} = \bar{x} + \bar{e} = (1\ 1\ 0\ 0\ 1\ 1\ 0)$
    - 2. Bit durch Übertragung gekippt
  - Fehlersyndrom:

$$\bar{s} = \bar{y} \cdot H^T = (1\ 1\ 0\ 0\ 1\ 1\ 0) \cdot \begin{pmatrix} 1\ 0\ 0 \\ 0\ 1\ 0 \\ 0\ 0\ 1 \\ 1\ 1\ 0 \\ 0\ 1\ 1 \\ 1\ 1\ 1 \\ 1\ 0\ 1 \end{pmatrix} = (0\ 1\ 0)$$

  - Fehlersyndrom $\bar{s}$ ist identisch mit verfälschter Spalte der Parity-Check-Matrix

Mathematischer Background

- Fehlersyndrom:

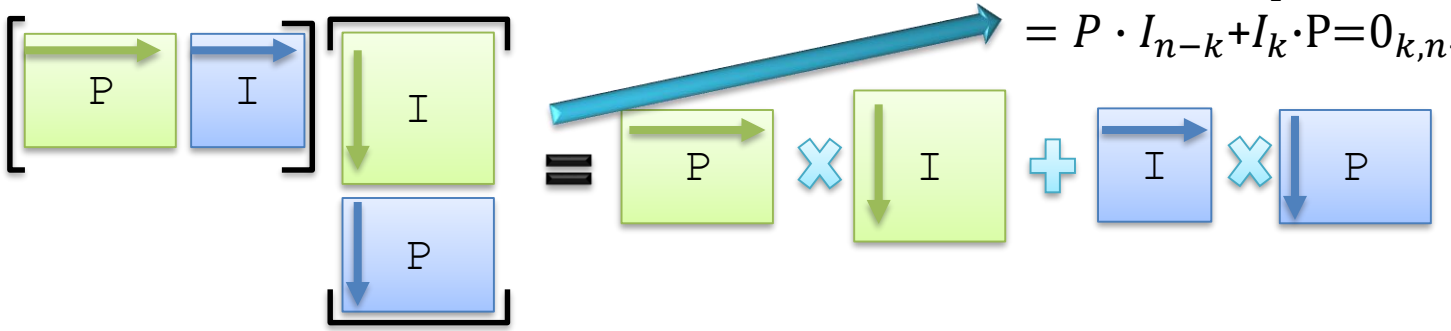
$$\bar{s} = \bar{y} \cdot H^T = (\bar{x} + \bar{e}) \cdot H^T = \bar{x} \cdot H^T + \bar{e} \cdot H^T = \bar{e} \cdot H^T$$

  - Fehlersyndrom entspricht der Summe der „fehlerbehafteten Spalten“ der Parity-Check-Matrix

  - Warum gilt $\bar{x} \cdot H^T = \overline{0}_n$?

$$\bar{x} \cdot H^T = (\overline{u} \cdot G) \cdot H^T = \overline{u} \cdot (G \cdot H^T) = \overline{u} \cdot 0_{k,n-k} = \overline{0}_n$$

$$G \cdot H^T = [P\ I_k] \cdot [I_{n-k}\ P^T]^T$$
$$= [P\ I_k] \cdot \begin{bmatrix} I_{n-k} \\ P \end{bmatrix}$$
$$= P \cdot I_{n-k} + I_k \cdot \mathrm{P} = 0_{k,n-k}$$



Eigenschaften

- Fehlererkennung:
  - Fehlersyndrom $\bar{s} \neq \bar{0}$
  - unerkannter Fehler: Summe der „fehlerbehafteten Spalten“ der Parity-Check-Matrix ergibt $\bar{0}$
    - Beispiel: $\bar{e} = (1\ 1\ 0\ 1\ 0\ 0\ 0)$
- Fehlerkorrektur:
  - Fehlersyndrom muss eindeutige Linearkombination aus Spalten der Parity-Check-Matrix sein
  - fehlerhafte Fehlerkorrektur bei 2 Fehlern
    - Beispiel: $\bar{e} = (1\ 1\ 0\ 0\ 0\ 0\ 0)$
- Hamming-Distanz h:
  - h-1 = Anzahl der **linear-unabhängigen Spalten der Prüfmatrix**
  - im Beispiel: Hamming-Distanz=3

Konstruktion von Codes

- Bei der Konstruktion von linear-systematischen Codes bietet es sich an, zunächst die Parity-Check-Matrix zu generieren.
- zur Korrektur von einem Fehler:
  - Hamming-Distanz: 3
  - alle Spalten der Prüfmatrix müssen paarweise linear unabhängig sein
  - einfach: alle Spalten der Parity-Check-Matrix müssen verschieden und ungleich dem Nullvektor sein
- zur Korrektur von zwei Fehlern:
  - Hamming-Distanz: 5
  - je vier beliebige Spalten der Prüfmatrix müssen linear unabhängig sein
  - alle Muster aus zwei Fehlern müssen erkennbar sein, d.h. alle Summen von je zwei beliebigen Spalten der Parity-Check-Matrix müssen verschieden sein
  - die Summe von vier (oder weniger) beliebigen Spalten muss ungleich dem Nullvektor sein

Konstruktion Hamming-Code

- Gewünschte Codeeigenschaften:
  - $\eta = 1$ Fehler korrigierbar
  - $N - K = 4$ Prüfbits
- Frage: Was ist die maximale Anzahl Nutzbits?
  - $2^{4+K} \geq 2^K(K + 4 + 1) \Leftrightarrow 2^4 \geq K + 4 + 1 \Leftrightarrow K \leq 11$
- Benötigen Parity-Check-Matrix $H = [I_{N-K}\ P^T]$ mit maximal $2^{N-K} - 1 = 15$ verschiedenen Spalten
  - 4 Spalten für Identitätsmatrix ➔ 11 weitere Prüfspalten ➔ 11 Nutzbits

$$H = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}[4x11] =$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 1 \end{bmatrix}$$

Übersicht Kapitel 3: Kanalcodierung

Generieren von Codes

- Eigenschaften von linear-systematischen Codes
  - Anzahl erkennbarer Fehler=Anzahl linear unabhängiger Spalten der Prüfmatrix
    - für h=3 einfach zu erzeugen
    - für größere h wird das schwierig
- Zyklische Codes
  - Generatorpolynom $g(D) = g_{N-1} \cdot D^{N-1} + \cdots + g_1 \cdot D + g_0$
  - Generieren von Codeworten

$$x(D) = D^{N-K} \cdot u(D) + p(D)$$

Nutzwort: $u(D) = u_{K-1} \cdot D^{K-1} + \cdots + u_1 \cdot D + u_0$

Parity-Bits: $p(D) = D^{N-K} \cdot u(D)\ mod\ g(D)$

- Zusammenhang von Codeworten und Polynomen
  - Bits der Codeworte werden als binäre Koeffizienten der Polynome interpretiert
  - Rechnung mit Koeffizienten immer „modulo 2“

Beispiel

- Generatorpolynom: $g(D) = D^3 + \text{D} + 1$ $\quad (1\ 0\ 1\ 1)$
- Nutzwort: $u(D) = D^2 + D$ $\quad (0\ 1\ 1\ 0)$
- Nutzwortteil des Codeworts:

$$u(D) \cdot D^{N-K} = D^5 + D^4 \qquad (0\ 1\ 1\ 0\ \ 0\ 0\ 0)$$

- Parity-Bits des Codeworts:

$$u(D) \cdot D^{N-K}\ mod\ g(D) = D^5 + D^4 \bmod D^3 + D + 1 = 1$$

Polynomdivision (Polynome)

$$D^5 + D^4 \ /\ \ D^3 + D + 1 = D^2 + D + 1$$
$$\underline{D^5 + D^3 + D^2}$$
$$D^4 + D^3 + D^2$$
$$\underline{D^4 + D^2 + \text{D}}$$
$$D^3 + D$$
$$\underline{D^3 + D + 1}$$
$$1 \quad \text{(Rest)}$$

Polynomdivision (Bits)

$$0110000/1011 = 0111$$
$$\underline{1011}$$
$$1110$$
$$\underline{1011}$$
$$1010$$
$$\underline{1011}$$
$$1 \text{ (Rest)}$$

- Codewort: $(0\ 1\ 1\ 0 \mid 0\ 0\ 1)$

Fehlererkennung und -korrektur

- Empfangenes Polynom: $y(D) = x(D) + e(D)$
- Fehlersyndrom: $s(D) = y(D)\ mod\ g(D)$
  - Herleitung:

$$s(D) = y(D)\ mod\ g(D) = x(D)\ mod\ g(D) + e(D)\ mod\ g(D)$$
$$= e(D)\ mod\ g(D)$$

  - Frage: Warum gilt $x(D)\ mod\ g(D) = 0$?
- Fehlererkennung:
  - Fehler, falls $s(D) \neq 0$
- Fehlerkorrektur:
  - bei Fehler an k-ter Stelle gilt $s(D) = D^k\ mod\ g(D)$
  - Identifikation des Fehlers über Vorbestimmung der Restklassen von $D^k\ mod\ g(D)$ für alle k

Beispiel:
$g(D) = D^3 + D + 1$

D^0	001
D^1	010
D^2	100
D^3	011
D^4	110
D^5	111
D^6	101

Beispiel 1

- Szenario
  - Nutzwort: $\bar{u} = (0\ 1\ 1\ 0)$
  - Generatorpolynom: $\bar{g} = (1\ 0\ 1\ 1)$
  - Codewort: $\bar{x} = (0\ 1\ 1\ 0\ 0\ 0\ 1)$
- 1. Fehlervektor: $\bar{e} = (0\ 0\ 1\ 0\ 0\ 0\ 0)$
  - empfangenes Codewort: $\bar{y} = \bar{x} + \bar{e} = (0\ 1\ 0\ 0\ 0\ 0\ 1)$
  - Fehlererkennung:

    $$(0\ 1\ 0\ 0\ 0\ 0\ 1)\ \mathrm{mod}\ (1\ 0\ 1\ 1) = (1\ 1\ 0)$$

    Restklasse von u^4 ➔3. Bit
- 2. Fehlervektor: $\bar{e} = (0\ 0\ 1\ 1\ 1\ 0\ 0)$
  - empfangenes Codewort: $\bar{y} = (0\ 1\ 0\ 1\ 1\ 0\ 1)$
  - Fehlererkennung:

    $$(0\ 1\ 0\ 1\ 1\ 0\ 1)\ \mathrm{mod}\ (1\ 0\ 1\ 1) = (0\ 0\ 1)$$

  - Restklasse von u^0 ➔7. Bit
  - Summe der Restklassen von u^4, u^3 und u^2

Restklassen:
$g(D) = D^3 + D + 1$

D^0	001
D^1	010
D^2	100
D^3	011
D^4	110
D^5	111
D^6	101

Beispiel 2: Codierung (Tafel)

- Generatorpolynom: $g(D) = D^3 + D^2 + 1 \qquad (1\ 1\ 0\ 1)$
- Codierung:

$\overline{u} = 0\ 0\ 1\ 1$

$\overline{x} = 0\ 0\ 1\ 1\ p_2\ p_1\ p_0 = 0011\ 010$

Polynomdivision mod 2:

$p(D) = u(D) \cdot D^{N-K}\ mod\ g(D)$

$(0011000)\ mod\ (1101)$

```
0011000 / 1101 = 1
  1101
   010=p2p1p0
```

$\overline{u} = 1\ 0\ 0\ 1$

$\overline{x} = 1\ 0\ 0\ 1\ p_2\ p_1\ p_0 = 1001\ 011$

Polynomdivision mod 2:

$p(D) = u(D) \cdot D^{N-K}\ mod\ g(D)$

$(0011000)\ mod\ (1101)$

```
1001000 / 1101 = 1111
1101
 1000
 1101
  1010
  1101
   1110
   1101
    011=p2p1p0
```

Beispiel 3: Fehlererkennung und -korrektur (Tafel)

- Generatorpolynom: $\mathrm{g}(D) = D^3 + D^2 + 1$ $\quad (1\,1\,0\,1)$
- Empfangen: $\overline{y} = 1010110$
- Gesendet?
- Fehlersyndrom: $s(D) = y(D)\ mod\ g(D)$

```
1010110 / 1101 = 1101
1101
 1111
 1101
  0101
   1010
   1101
    111=s2s1s0
```

- Fehlervektor: ?

Beispiel: Polynom-Restklassen (Tafel)

- Generatorpolynom: $G(D) = D^3 + D^2 + 1$ $\quad (1\,1\,0\,1)$

j	D^j (binär)	$D\cdot(D^{j-1} \bmod 2)$ (binär)	$\text{grad}(D\cdot(D^{j-1} \bmod 2))$	$D^j \bmod 2$ (binär)
0	1	-	-	001
1	10	10	1	010
2	100	100	2	100
3	1000	1000	3	1000 +1101 =101
4	10000	1010	4	1010 +1101 =111
5	100000	1110	4	1110 +1101 =011
6	1000000	0110	4	110
7	10000000	1100	2	1100 +1101 =0001

Beispiel 3: Fehlererkennung und -korrektur (Tafel)

- Generatorpolynom: $\mathrm{g}(D) = D^3 + D^2 + 1$
- Empfangen: $\overline{y} = 1010110$
- Gesendet?
- Fehlersyndrom:

  $s(D) = y(D)\ mod\ g(D) = 111$
- Fehlerkorrektur:
  - Fehlersyndrom entspricht $D^4\ mod\ g(D)$
  - Fehler an 5. Bit von hinten
  - Fehlervektor: $\overline{e} = 0010000$
  - Gesendet: $\overline{x} = \overline{y} + \overline{e} = 1000110$
  - Nutzbits: $\overline{u} = 1000$

Restklassen:
$g(D) = D^3 + D^2 + 1$

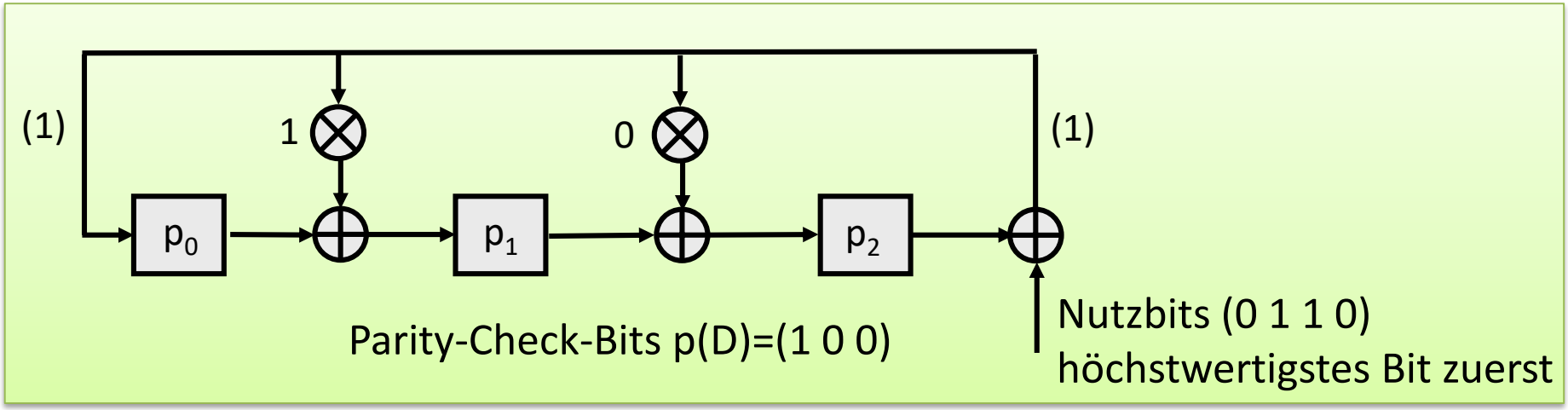
D^0	001
D^1	010
D^2	100
D^3	101
D^4	111
D^5	011
D^6	110

```
Kontrolle:
1000000 / 1101 = 111
1101
 1010
 1101
  1110
  1101
    110=p2p1p0
```

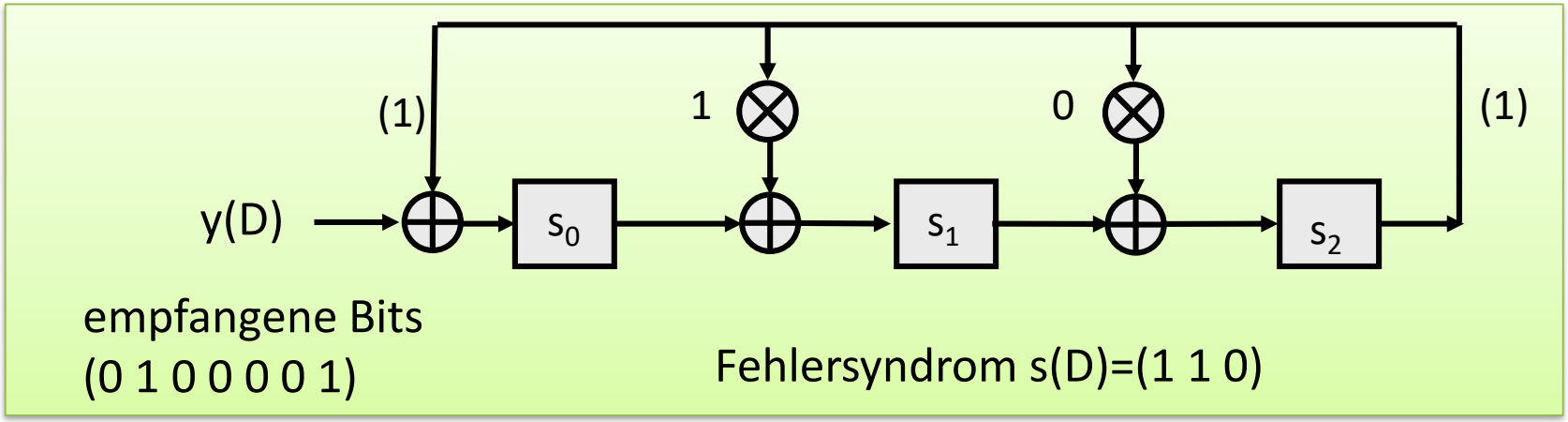
Übersicht Kapitel 3: Kanalcodierung

Hardwarerealisierung mit rückgekoppeltem Schieberegister

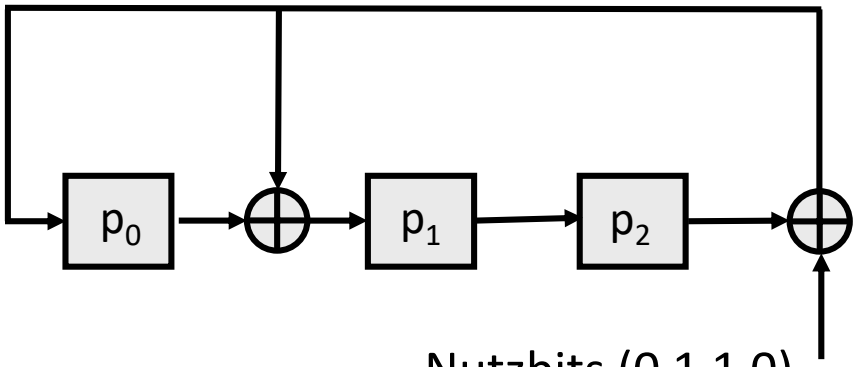
- Codierschaltung (Beispiel für $g(D) = D^3 + D + 1$)

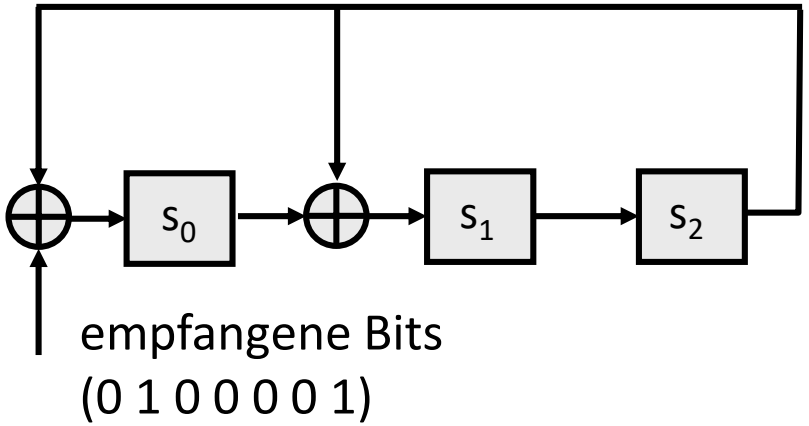


- Decodierschaltung (Beispiel für $g(D) = D^3 + D + 1$)



Codierung / Dekodierung (Tafel)

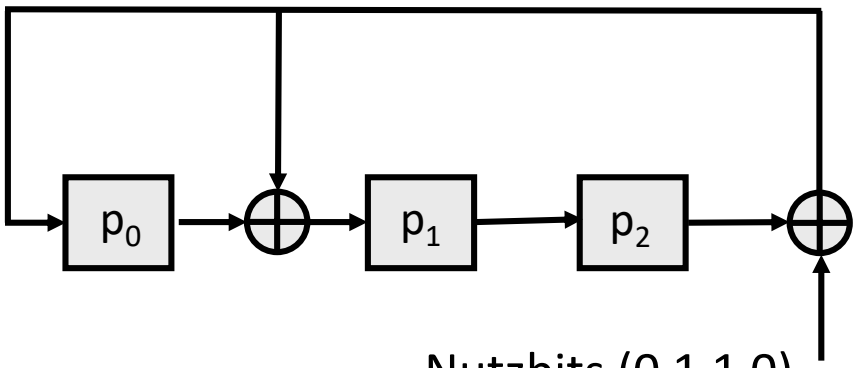


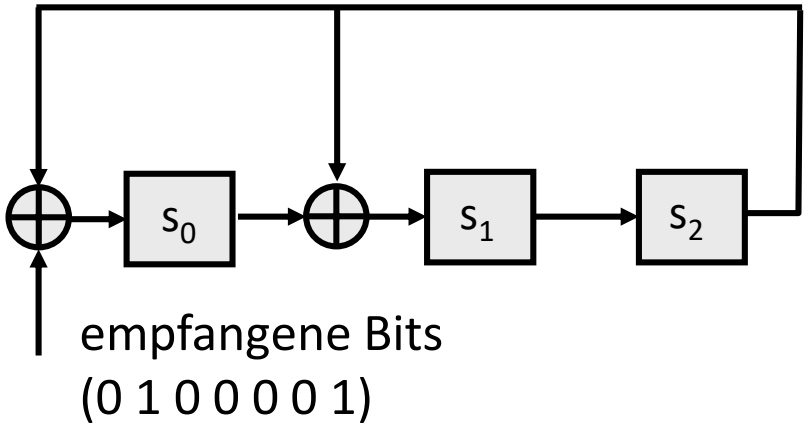


y_i	p_0	p_1	p_2
0			
1			
1			
0			

y_i	s_0	s_1	s_2
0			
1			
0			
0			
0			
0			
1			

Codierung / Dekodierung



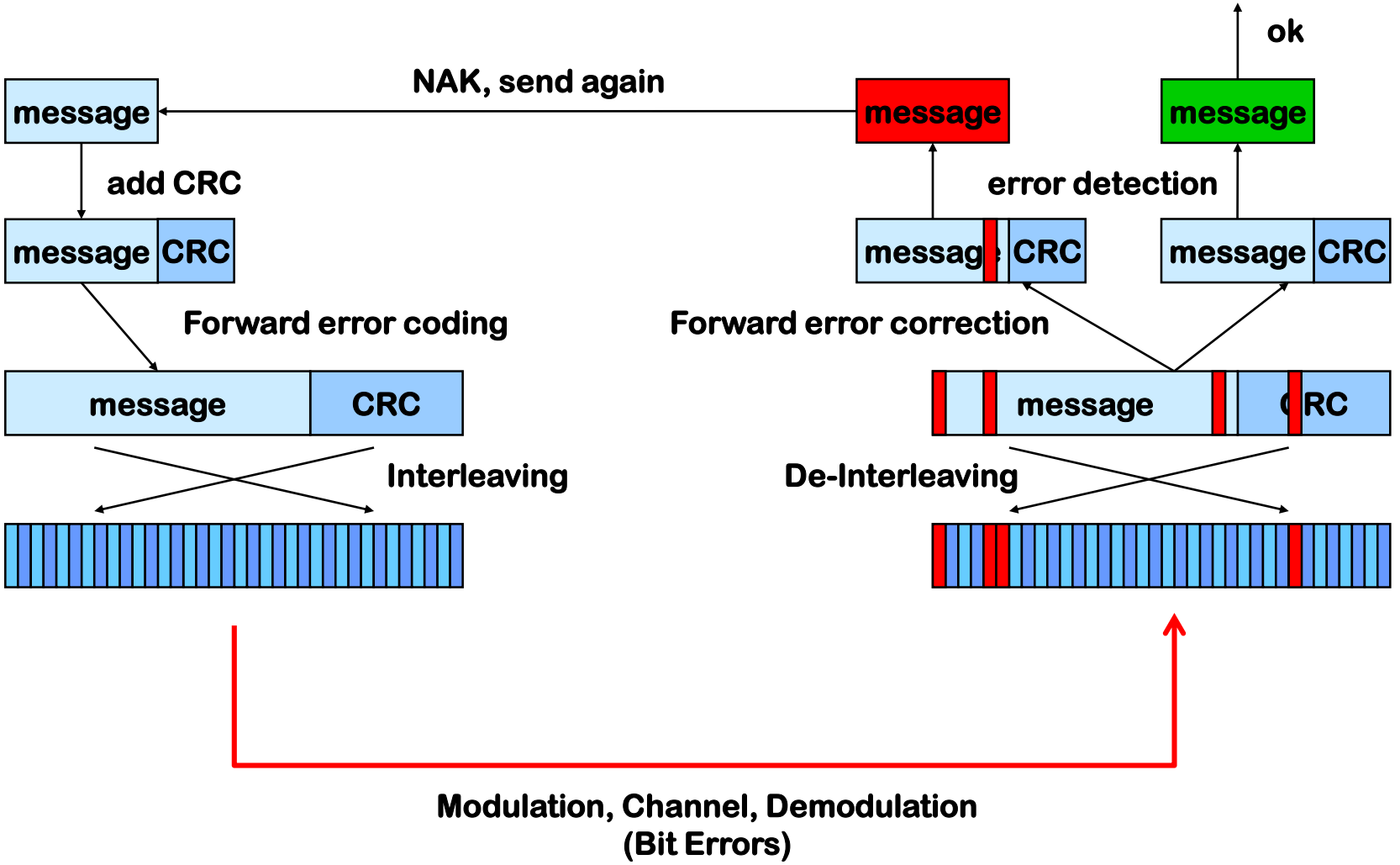


y_i	p_0	p_1	p_2
	0	0	0
0	0	0	0
1	1	1	0
1	1	0	1
0	1	0	0

y_i	s_0	s_1	s_2
	0	0	0
0	0	0	0
1	1	0	0
0	0	1	0
0	0	0	1
0	1	1	0
0	0	1	1
1	0	1	1

Übersicht Kapitel 3: Kanalcodierung

Cyclic Redundancy Check (Beispiel: Funkübertragung)



CRC

- Einsatz in fast allen Datenübertragungen zur Fehlererkennung
  - Unterschiedliche Generatorpolynome je nach Paketgröße und akzeptabler Restfehlerwahrscheinlichkeit
- Beispiele von Polynomen
  - Ethernet, WLAN, GZIP, PNG, MPEG-2, etc.:
    $g(D) = D^{32} + D^{26} + D^{23} + D^{22} + D^{16} + D^{12} + D^{11} + D^{10} + D^8 + D^7 + D^5 + D^4 + D^2 + D + 1$
  - CRC-16-CCITT (GPRS, UMTS, LTE, Bluetooth): $g(D) = D^{16} + D^{12} + D^5 + 1$
  - CRC-16-ANSI (USB): $g(D) = D^{16} + D^{15} + D^2 + 1$
  - UMTS, LTE:
    $g(D) = D^{24} + D^{23} + D^{18} + D^{17} + D^{14} + D^{11} + D^{10} + D^7 + D^6 + D^5 + D^4 + D^3 + D + 1$
    - nutzt Polynome vom Grad 24, 16, 12 und 8
  - CAN: $g(D) = D^{15} + D^{14} + D^{10} + D^8 + D^7 + D^4 + D^3 + 1$
  - FlexRay:
    $g(D) = D^{24} + D^{22} + D^{20} + D^{19} + D^{18} + D^{16} + D^{14} + D^{13} + D^{11} + D^{10} + D^8 + D^7 + D^6 + D^3 + D + 1$
- Einige Eigenschaften
  - erkannte Fehlermuster hängen von Polynom ab
    - Polynome, die zwei Fehler bis zu einer bestimmten Blocklänge erkennen, sind bekannt
  - Polynome mit „+1“ erkennen Fehler-Bursts der Länge N-K
  - Polynome mit Faktor „D+1“ erkennen alle Fehlermuster mit ungerader Anzahl von Fehlern

BCH-Codes

- große **Familie** von guten, linearen, zyklischen Codes
  - gut=gutes Verhältnis von minimaler Hamming-Distanz zu Redundanz N-K
  - 1959/60 entdeckt durch **B**ose, **C**haudhuri und **H**ocquenghem (**BCH**)
  - BCH-Codes sind gut tabelliert
- Eigenschaften
  - sie sind über einem Galois Feld GF(q) mit q Elementen definiert
  - für positive Zahlen $m \geq 3$ und $t \leq q^m - 1$ gibt es einen BCH-Code mit
    - Länge des Codeworts: $N = q^m - 1$
    - minimaler Hamming-Distanz (designated distance): $d_{min} \geq 2t + 1$

    Es gibt einen Code mit höchstens $m \cdot t$ Parity-Symbolen, so dass die Anzahl der Nutzsymbole mindestens $n - m \cdot t$ ist.
  - Binärer Code: q=2
    - Länge des Codeworts: $N = 2^m - 1$
    - minimaler Hamming-Distanz (designated distance): $d_{min} \geq 2t + 1$
    - Anzahl benötigter Parity-Bits ist höchstens $m \cdot t$
    - Anzahl der Nutzsymbole ist mindestens $n - m \cdot t$
  - Aufgabe: Binären Code für 100 Bytes Blockgröße, Bitfehlerwahrs‘ 1%, Restfehlerwahrs‘ < 0,001%

BCH-Codes

Problemstellung:

- In einer Übertragungsstrecke sollen Frames mit einer Payload von 100 Bytes übertragen werden. Aus Messungen ist bekannt, dass die Bitfehlerwahrscheinlichkeit für die Übertragungsstrecke bei 1% liegt. Wir suchen einen Code, der so viele Fehler korrigiert, dass die Restfehlerwahrscheinlichkeit, d.h. die Wahrscheinlichkeit nach der Fehlerkorrektur einen fehlerhaften Frame zu haben, kleiner als 0,001% ist.
- Gibt es dafür einen BCH Code?

BCH-Codes

- 100 Bytes Blockgröße, Bitfehlerwahrs‘ 1%, Restfehlerwahrs‘ < 0,001%
- Iterative Bestimmung der Anzahl der zu korrigierenden Bitfehler und des Code-Parameters m
  - Initiierung: wähle $m = \lceil \log_2 n \rceil = \lceil \log_2 800 \rceil = 10$ und $t = 0$
  - Iteration:
    1. Schritt: berechne die Anzahl t der zu korrigierenden Bitfehler, so dass die Restfehlerwahrscheinlichkeit bei einer Codewortlänge von $\mathrm{n} = k + m \cdot t$ kleiner als 0.001% ist.

       ```
       t>=binoinv(1-0.00001,800+m*t,0.01)
       ```

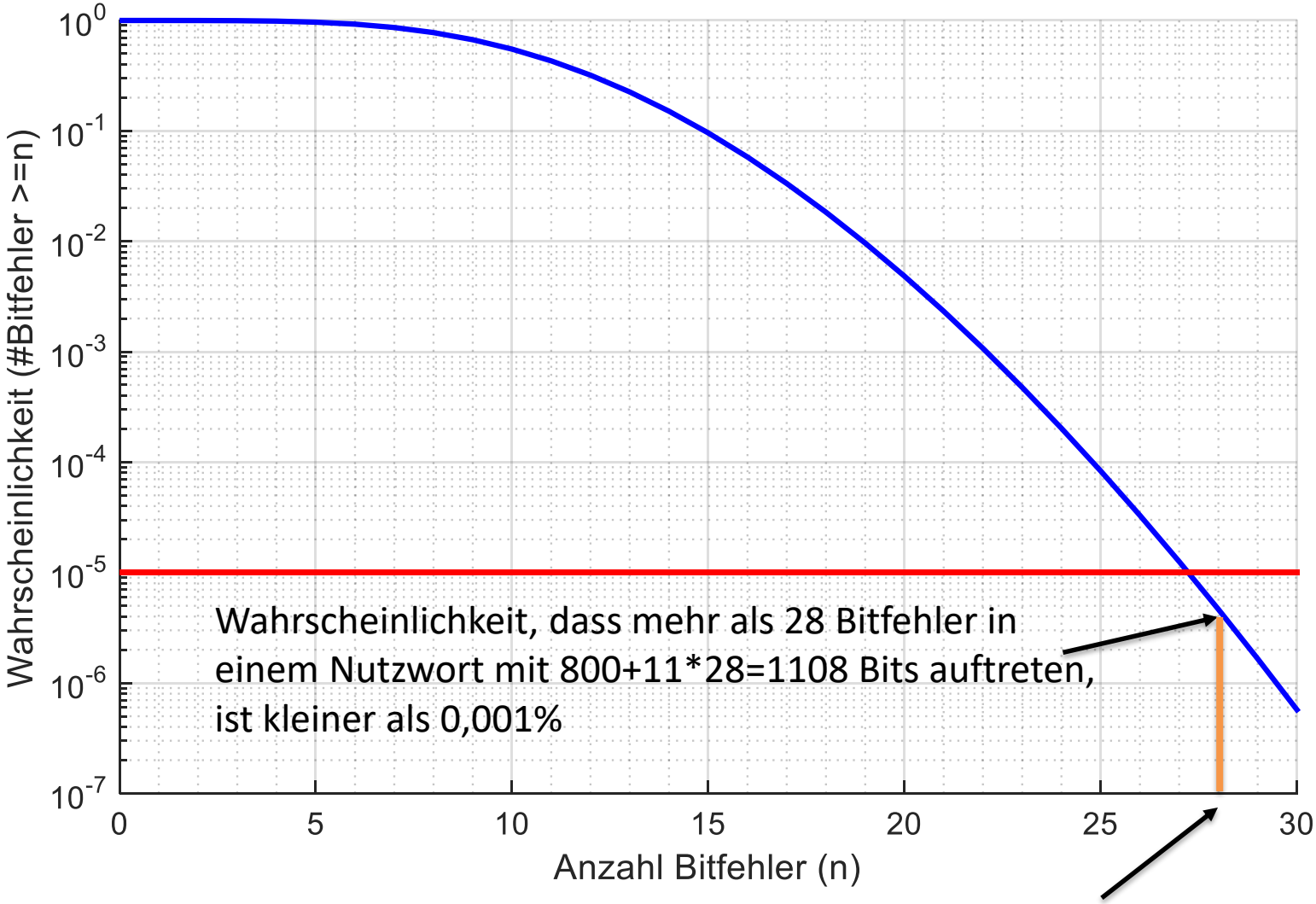
    2. Schritt: finde ist das kleinste m, das die folgende Bedingung erfüllt:

       $$2^m - 1 \geq k + m \cdot t$$

- Ergebnis:
  - 1. Iteration: $m = 10, t = 27, 1023 < 1070$
  - 2. Iteration: $m = 11, t = 28, 2047 \geq 1108$

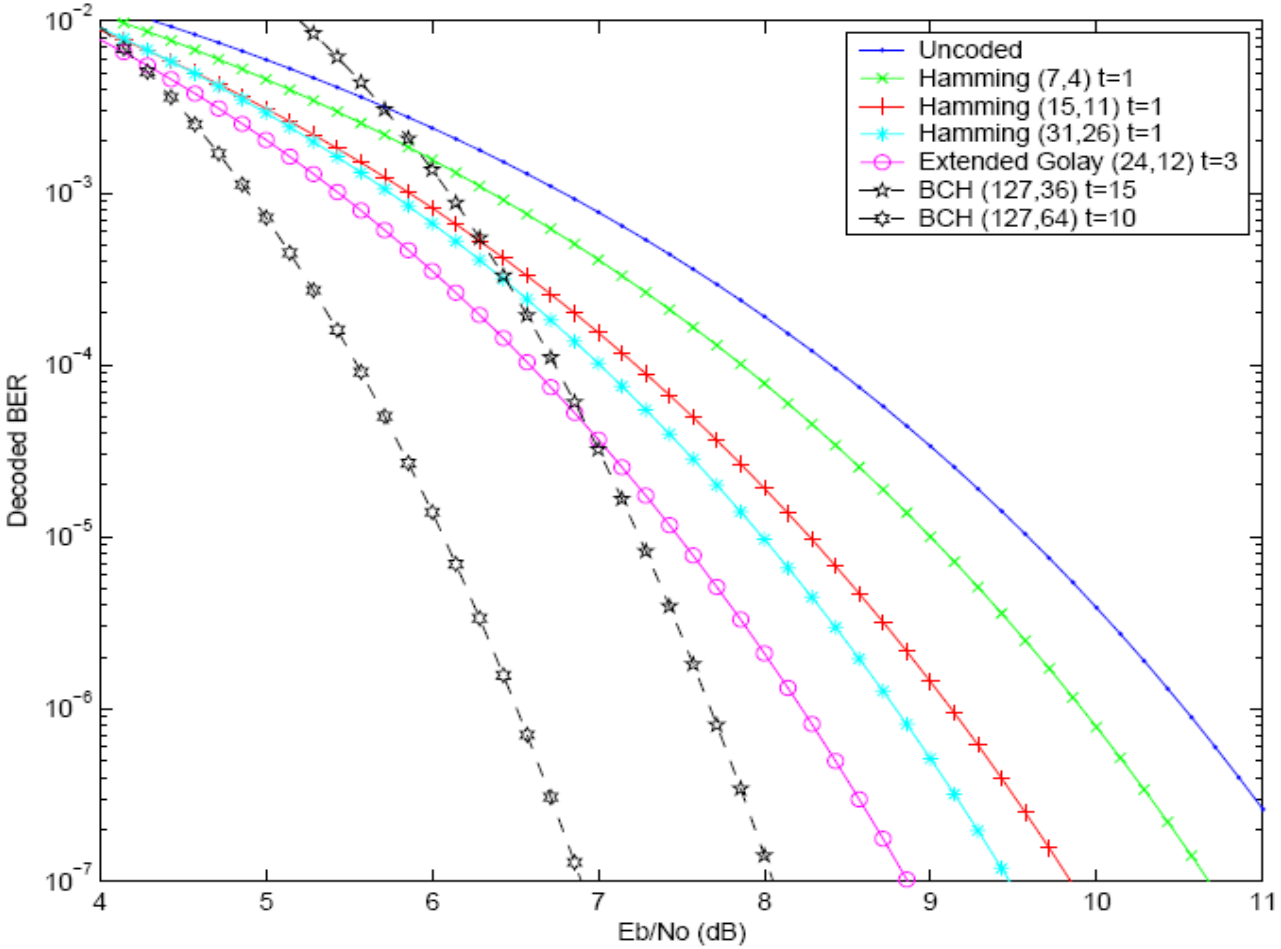
Anzahl zu korrigierender Bitfehler

Matlab: plot(0:30,1-binocdf(0:30,1108,0.01))



Matlab: t=binoinv(1-0.00001,1108,0.01)

Restbitfehlerwahrscheinlichkeiten einiger Block-Codes



Reed-Solomon-Codes

- Reed-Solomon Codes sind nicht-binäre BCH Codes
  - arbeiten z.B. mit Bytes als Symbolen
- Einsatzgebiete
  - Audio CDs, DVD, ...
  - Satellitenkommunikation
  - DVB, WiMAX
  - xDSL
- Reed-Solomon Codes: RS(N,K {,t}) mit m-Bit Symbolen
  - oft: m=8, also Bytes
  - K Nutzsymbole ($K \cdot q$) Nutzbits ergeben ein Codewort mit $\mathrm{N} \cdot q$ Bits
  - Arbeiten mit Gruppen von Symbolen macht den RS-Code geeignet um Burstfehler zu korrigieren
- Beispiel: (255,223)-Code mit Bytes
  - 223 Nutzbytes, 255 Bytes Codewort, 16 fehlerhafte Bytes können sicher korrigiert

Übersicht Kapitel 3: Kanalcodierung

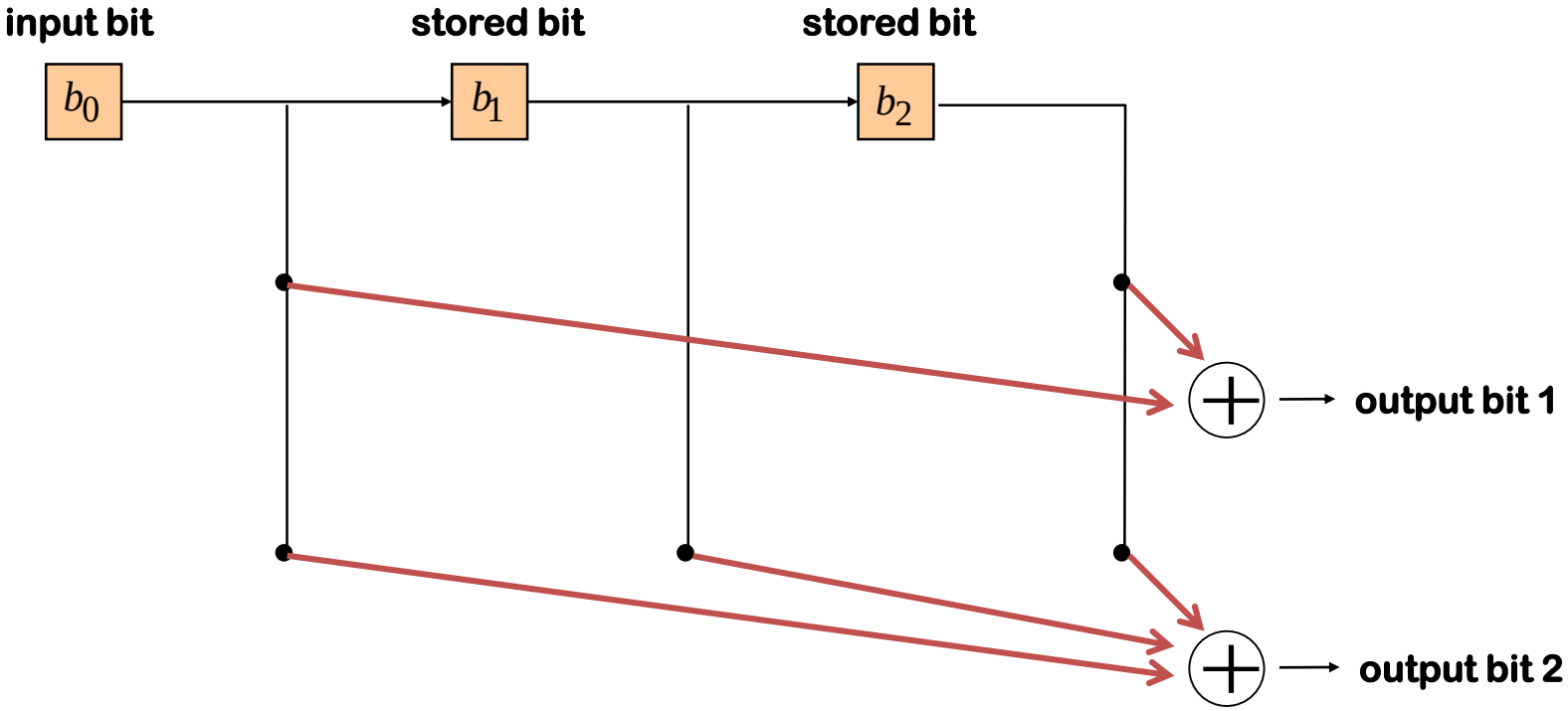
WiFi Modulation and Coding Schemes

- Mandatory forward error correction (FEC) in WiFi (IEEE 802.11) is done using a 1/2 convolutional code
  - coding rates 3/4 and 5/6 are achieved by puncturing (cf. 5.2.4)
- LDPC (Low Density Parity Check) codes are optional codes introduced with the IEEE802.11n standard
  - HT PHY (High Throughput Physical Layer)

Data Rate (Mbps)	Modulation	Coding Rate	Data bits per OFDM symbol
6	BPSK	1/2	24
9	BPSK	3/4	36
12	QPSK	1/2	48
18	QPSK	3/4	72
24	16-QAM	1/2	96
36	16-QAM	3/4	144
48	64-QAM	2/3	192
54	64-QAM	5/6	216

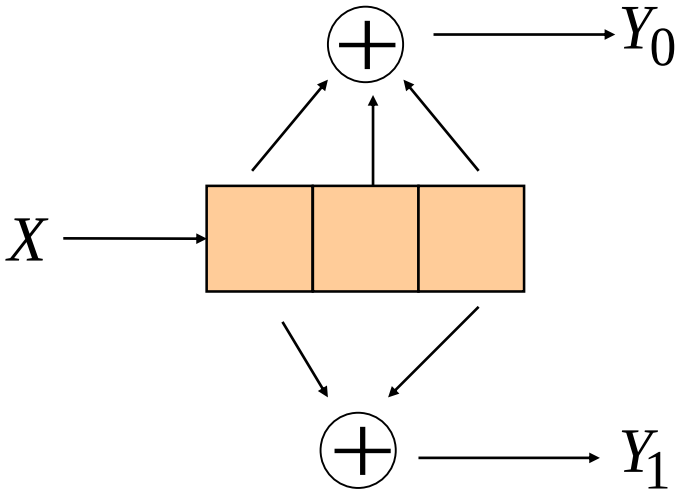
Note: The IEEE802.11 standard defines multiple physical layers. The table shows the data rates with modulation and coding schemes for the OFDM (Orthogonal Frequency Division Multiplex) PHY which is used in IEEE 802.11a/b/g

1/2 - Convolutional Encoder with Constraint Length 3

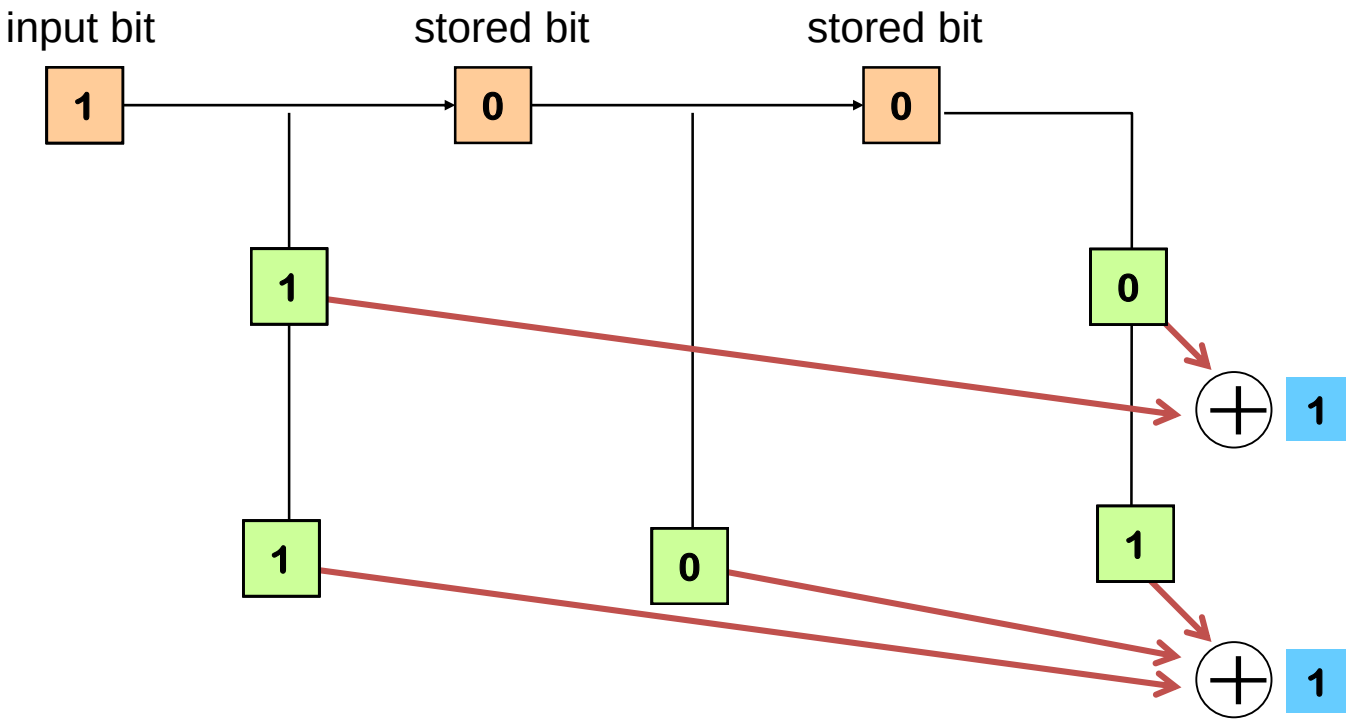


- The code rate defines the number of output bits produced per input bit
  - here the code rate is 1/2 which means two output bits for every input bit
- The constraint length specifies the number of consecutive input bits that are involved in producing an output bit
  - here the output bits depend on three consecutive input bits
- The convolutional coder works by producing two output bits for every 3-tupel of input bits
  - different output bit are produced by applying different binary functions of the input bits

Other Representation



1/2 - Convolutional Encoder with Constraint Length 3



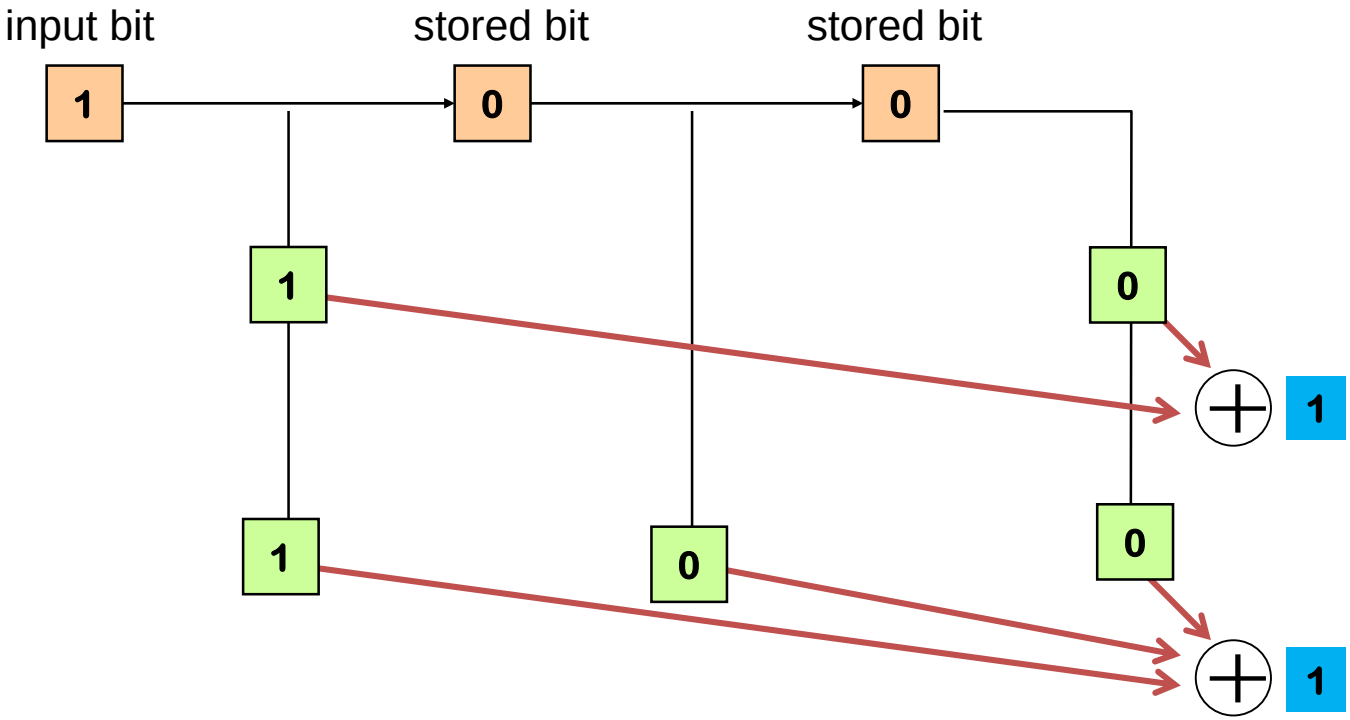
input: 1101(00)

output: **11 10 10 00 01 11**

Operation:

- input bits are fed one by one into the coder
- for every new input bit two output bits are generated by xor-ing the bit with some of the previous bits according to the definition of the coder
- the coder starts filled with zeros
- after the input bits two more zeros are fed to the coder

1/2 - Convolutional Encoder with Constraint Length 3

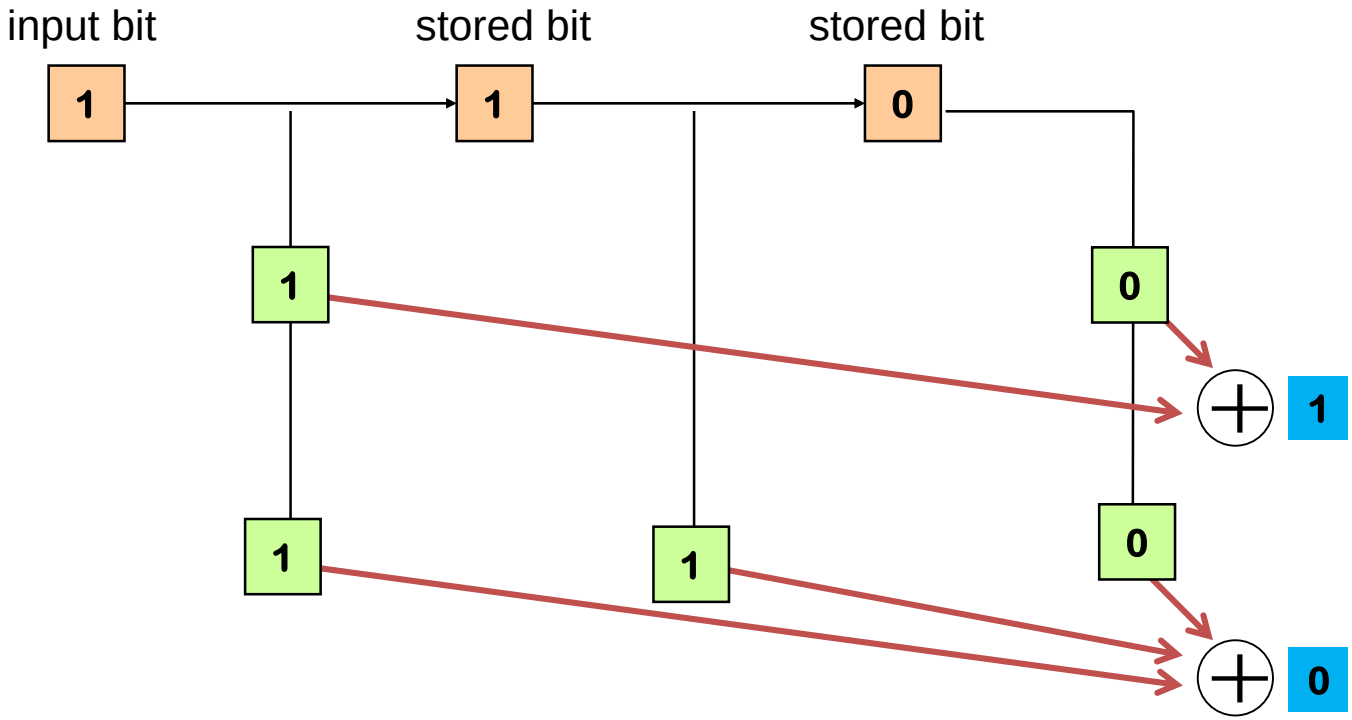


input: **1**101

output: 11

The example shows the encoding of bit sequence 1101. The encoding starts with the encoder in a well-defined state, the all-zero state. The input bits are fed one by one to the encoder.

1/2 - Convolutional Encoder with Constraint Length 3

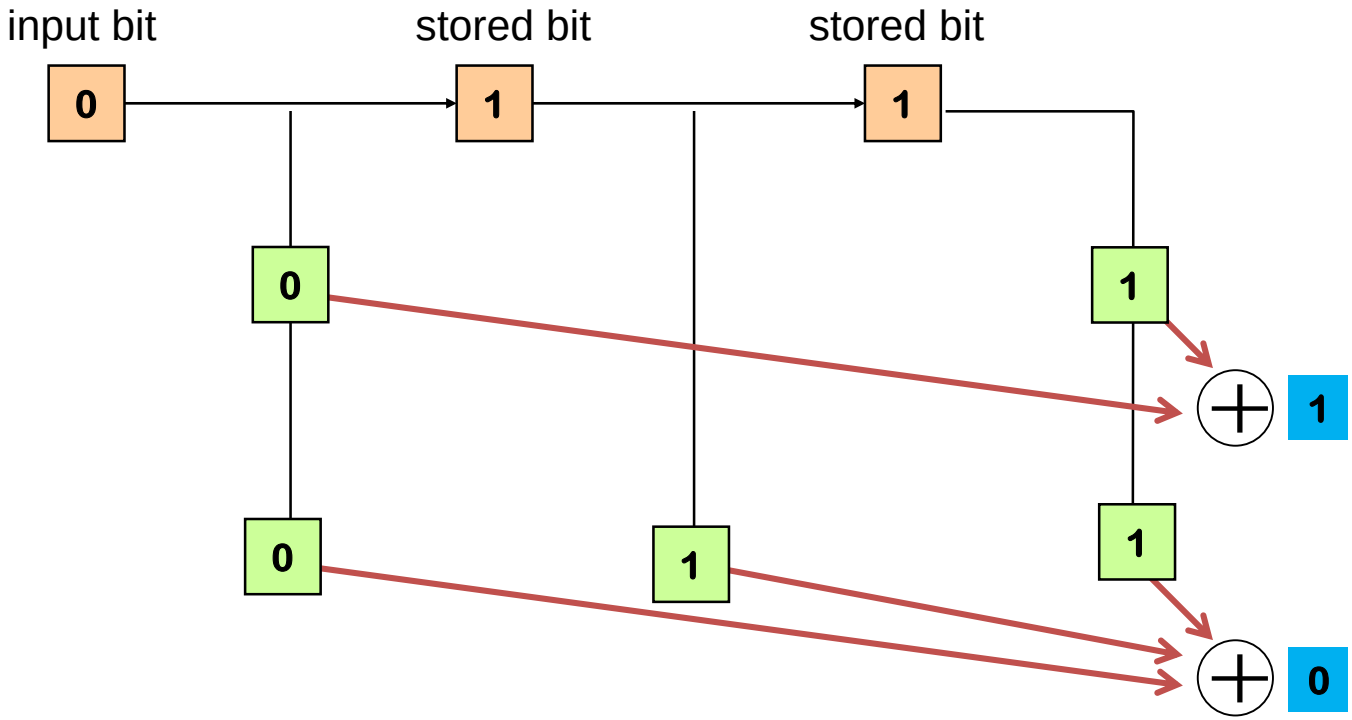


input: 1101

output: 11 10

The example shows the encoding of bit sequence 1101. The encoding starts with the encoder in a well-defined state, the all-zero state. The input bits are fed one by one to the encoder.

1/2 - Convolutional Encoder with Constraint Length 3

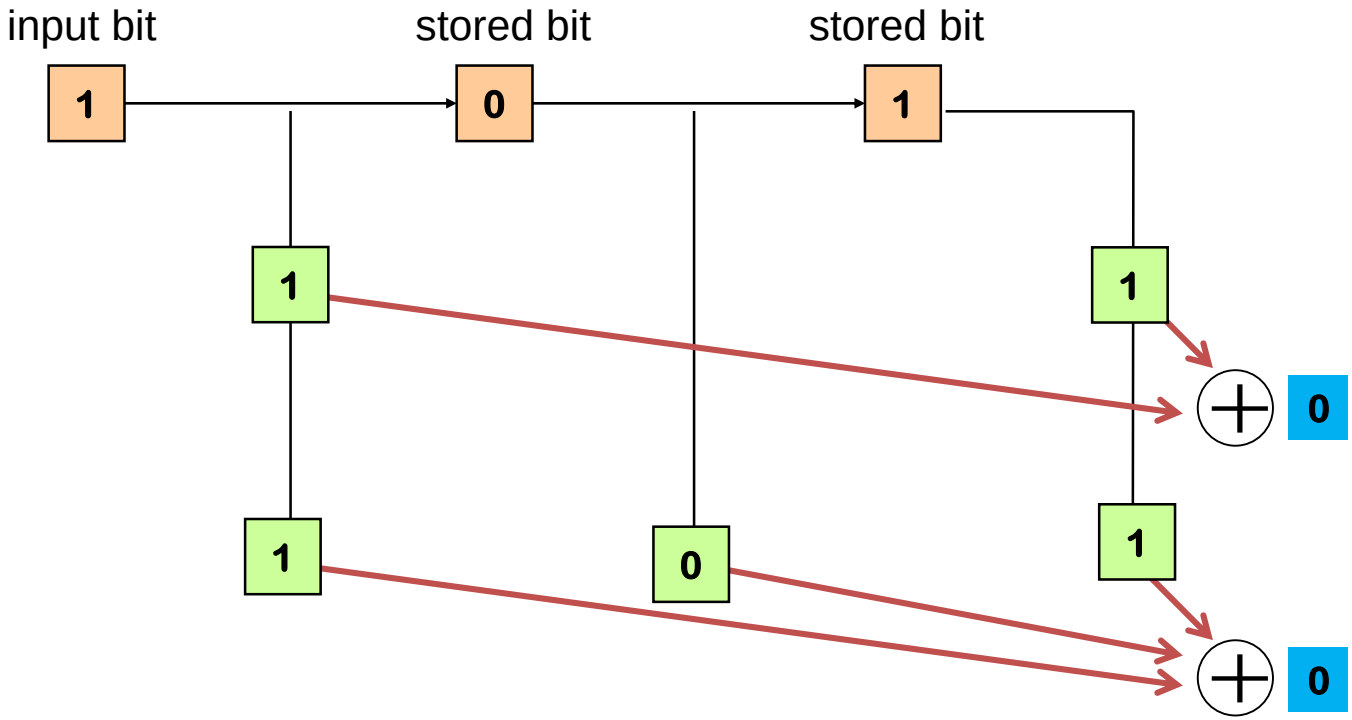


input: 1101

output: 11 10 10

The example shows the encoding of bit sequence 1101. The encoding starts with the encoder in a well-defined state, the all-zero state. The input bits are fed one by one to the encoder.

1/2 - Convolutional Encoder with Constraint Length 3

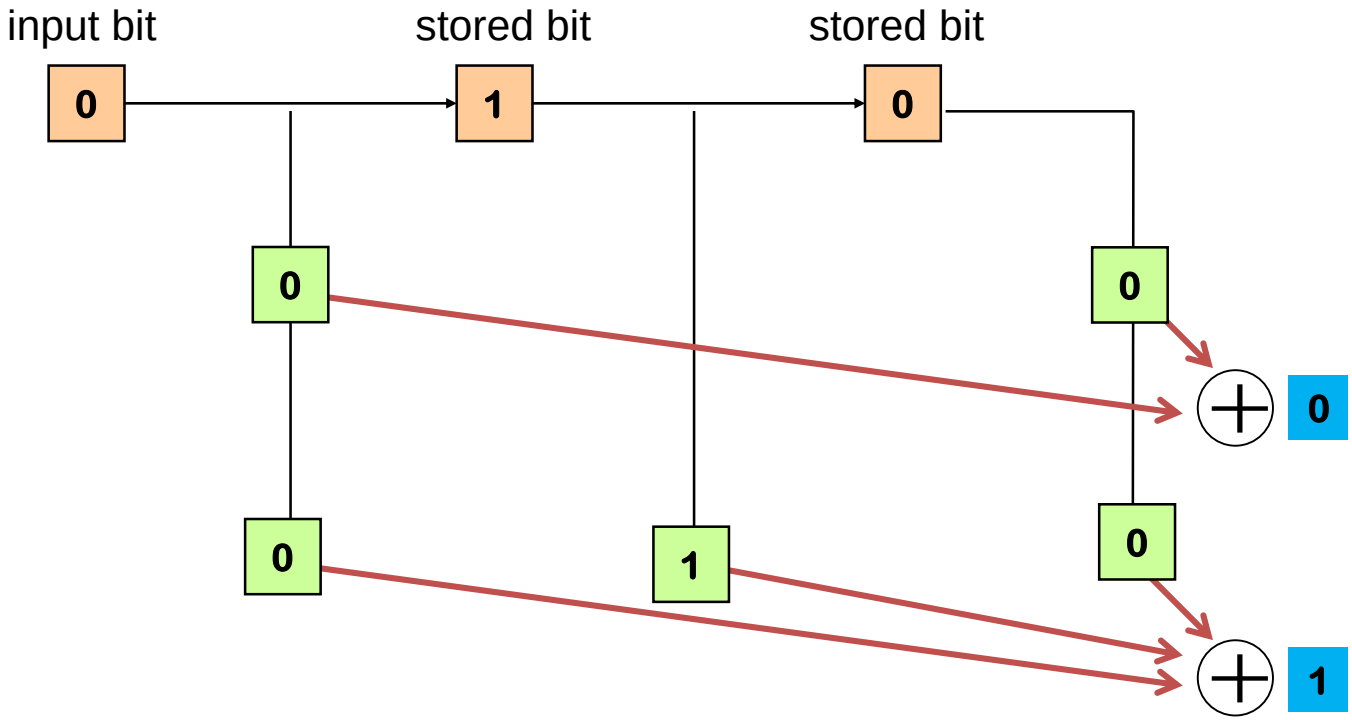


input: 1101

output: 11 10 10 00

The example shows the encoding of bit sequence 1101. The encoding starts with the encoder in a well-defined state, the all-zero state. The input bits are fed one by one to the encoder.

1/2 - Convolutional Encoder with Constraint Length 3

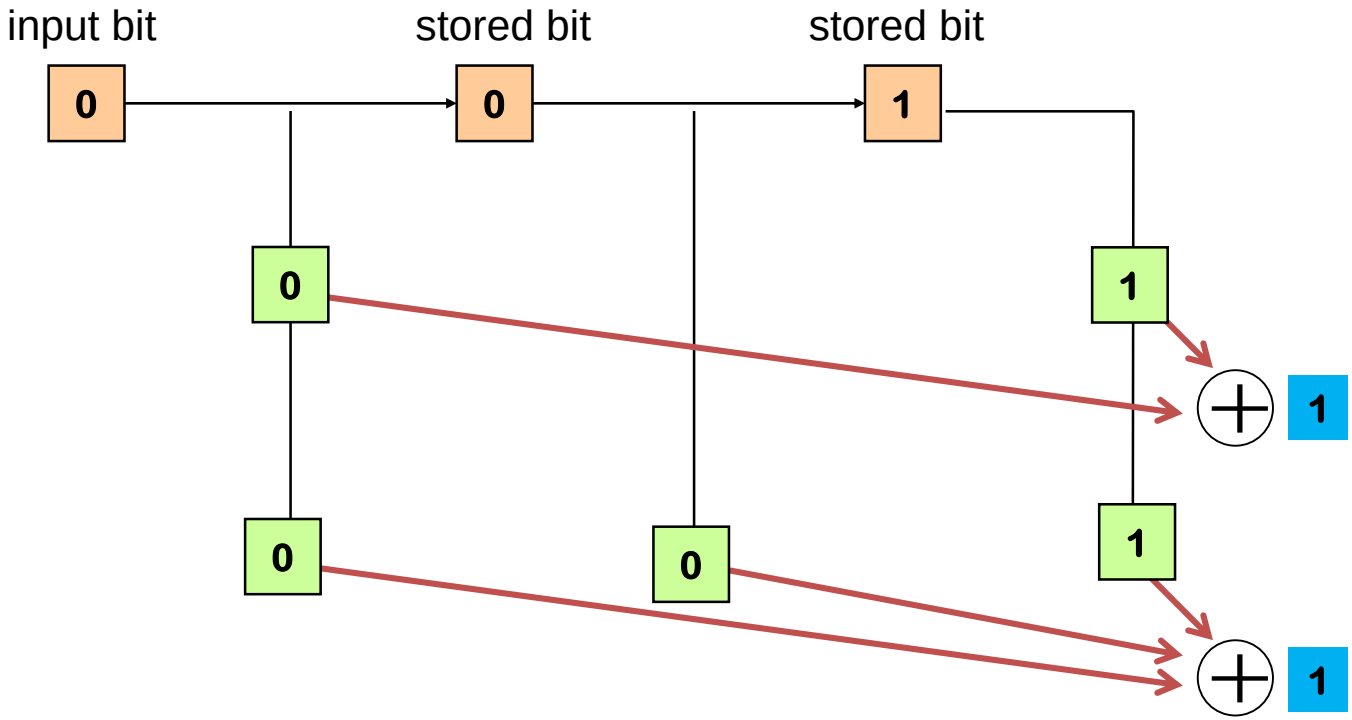


input: 1101(**0**0)

output: **11 10 10 00 01**

The encoding must end with the encoder in a well-defined state, the all-zero state. This is achieved by feeding two more zero bits into the coder after the bit sequence.

1/2 - Convolutional Encoder with Constraint Length 3



input: 1101(0**0**)

output: **11 10 10 00 01 11**

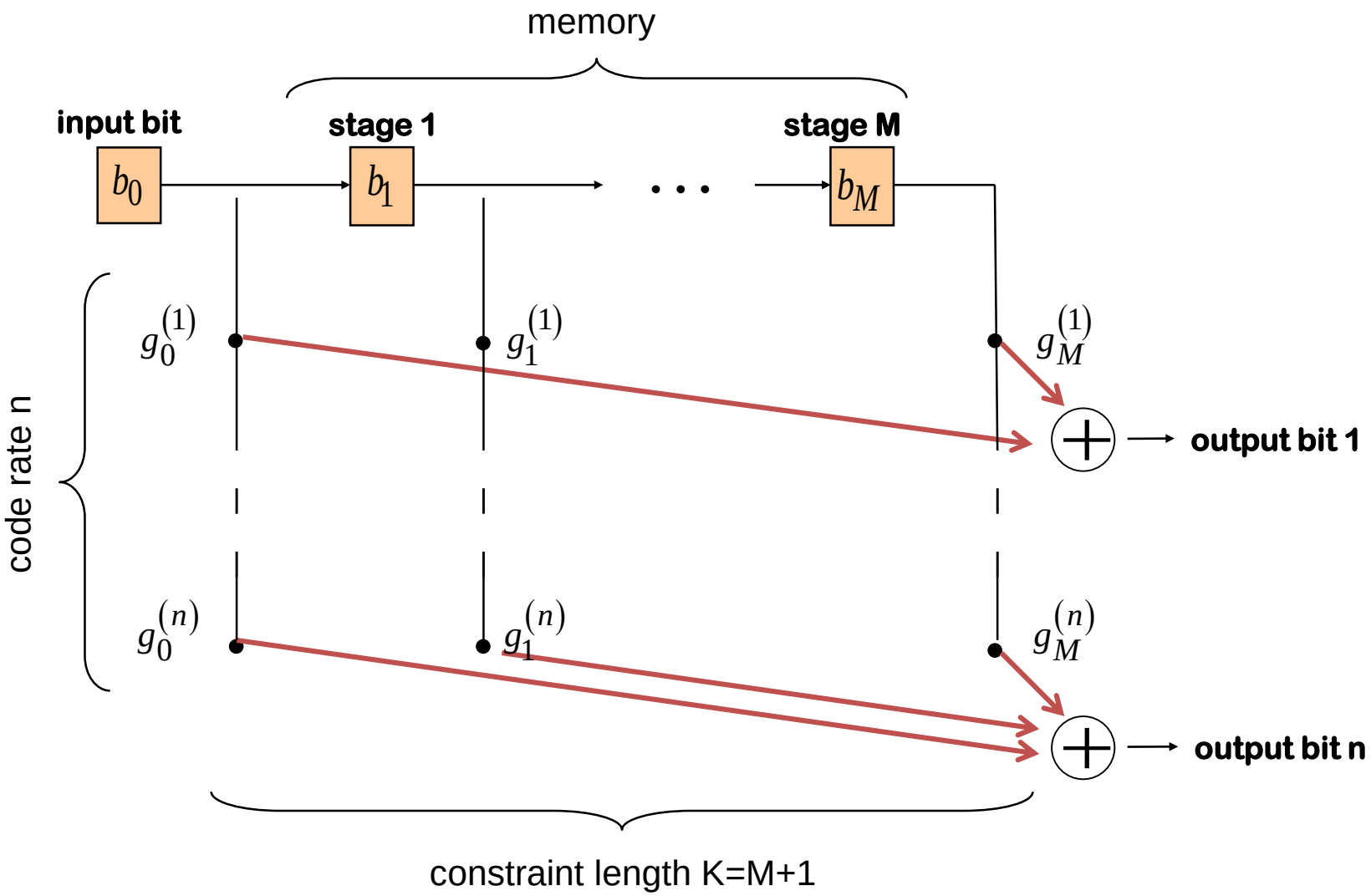
At the end of the encoding operation the encoder is in the all-zero state. The number of bits to transmit is $n=2\cdot(k+M)$

Convolutional Code

- encoding is done bit by bit
- convolutional coder has a coding rate of one-to-n
- n is the number of output bits per time shift
- every output bit is produced by a path
- a path can be represented by a generator polynom
- n is the number of paths/generator polynoms
- the encoder stores M bits
- the constraint length of a code is K=M+1, i.e. an output bit depends on the last M+1 input bits or an input bit influences the next M+1 output bit n-tuples

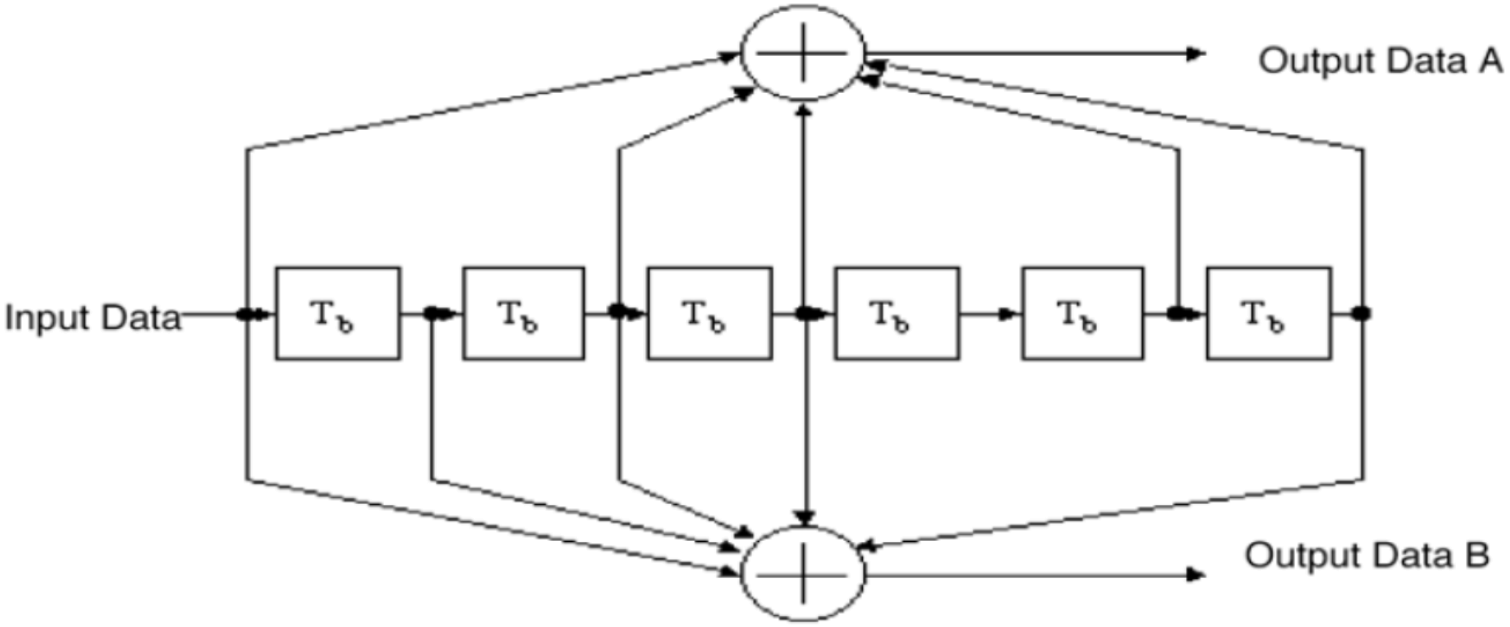
- Convolutional codes are the perhaps most prominent FEC codes in wireless communications. They are used e.g. for
  - GSM/GPRS/EDGE/UMTS
  - WiMAX
  - DVB
  - ...

General convolutional encoder



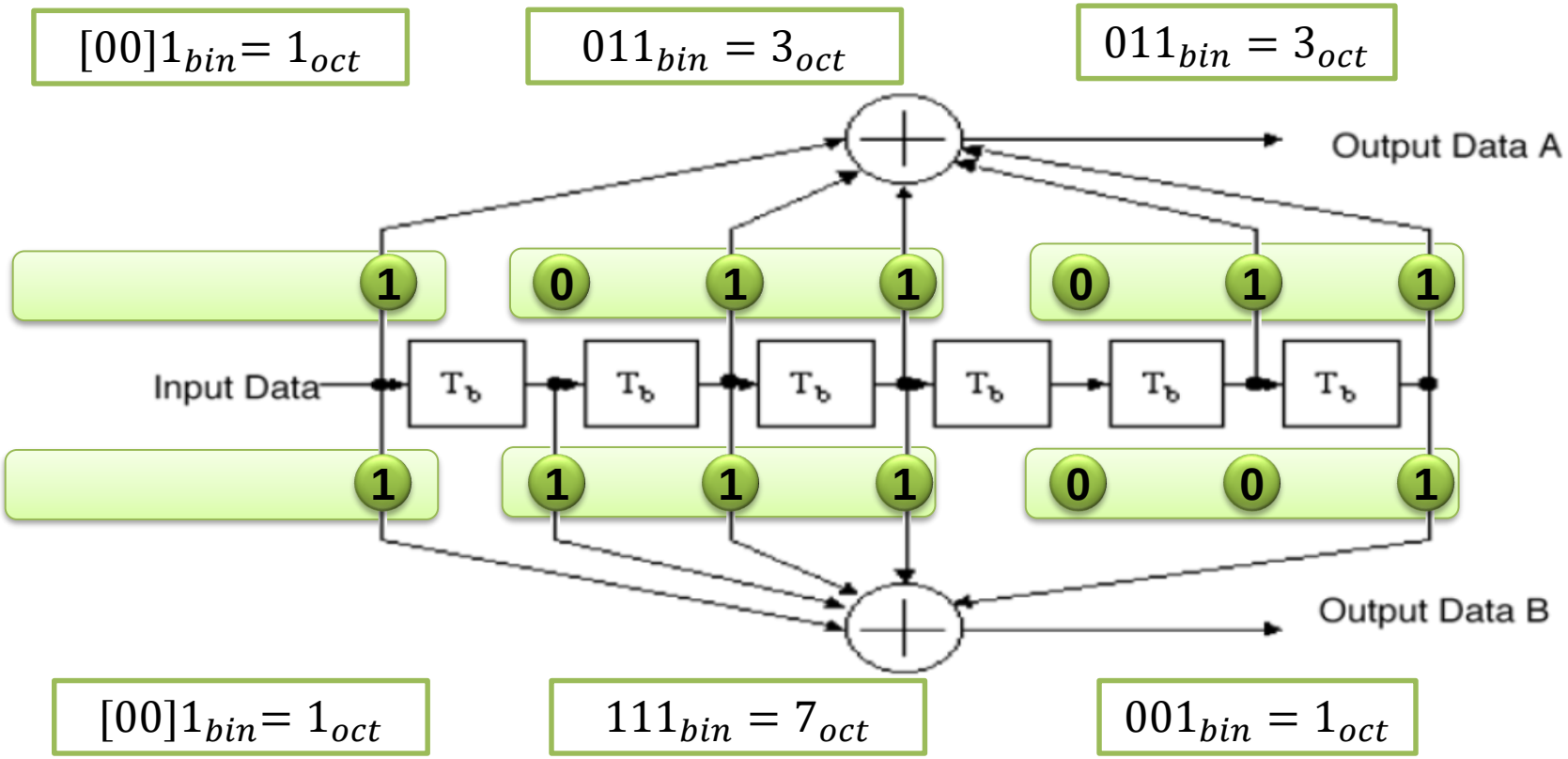
WiFi Convolutional Coder

- constraint length K=7
- code rate: 1/2
- octal notation: [133 171]



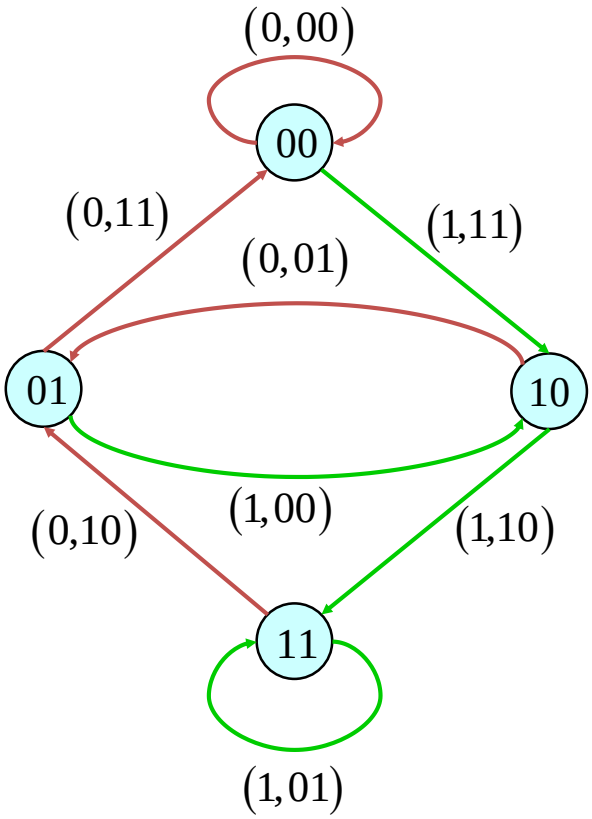
WiFi Convolutional Coder

- constraint length K=7
- code rate: 1/2
- octal notation: [133 171]

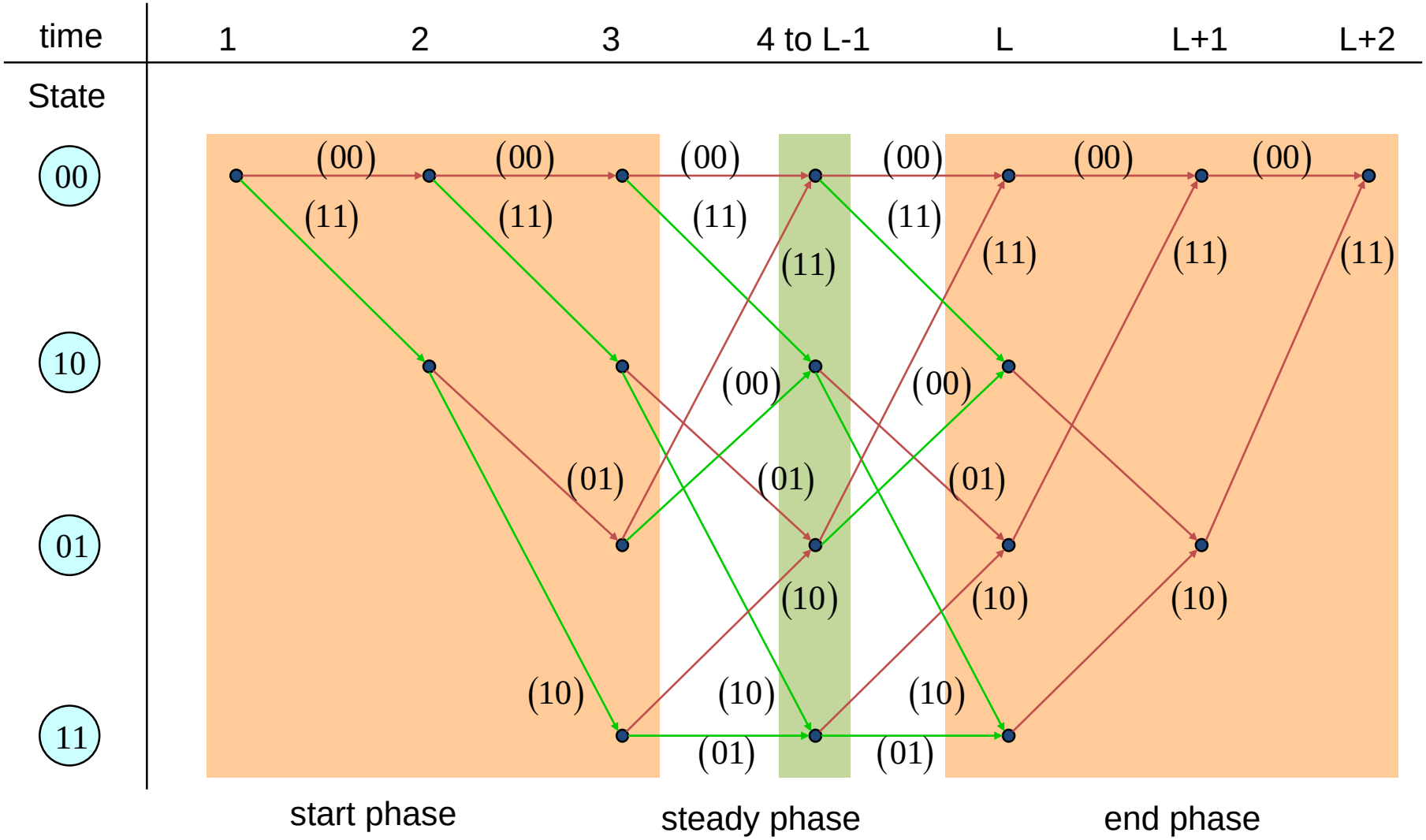


Representation as Finite State Machine

- state: content of memory
- number of states: 2^M
- state transition: input bit, output bits

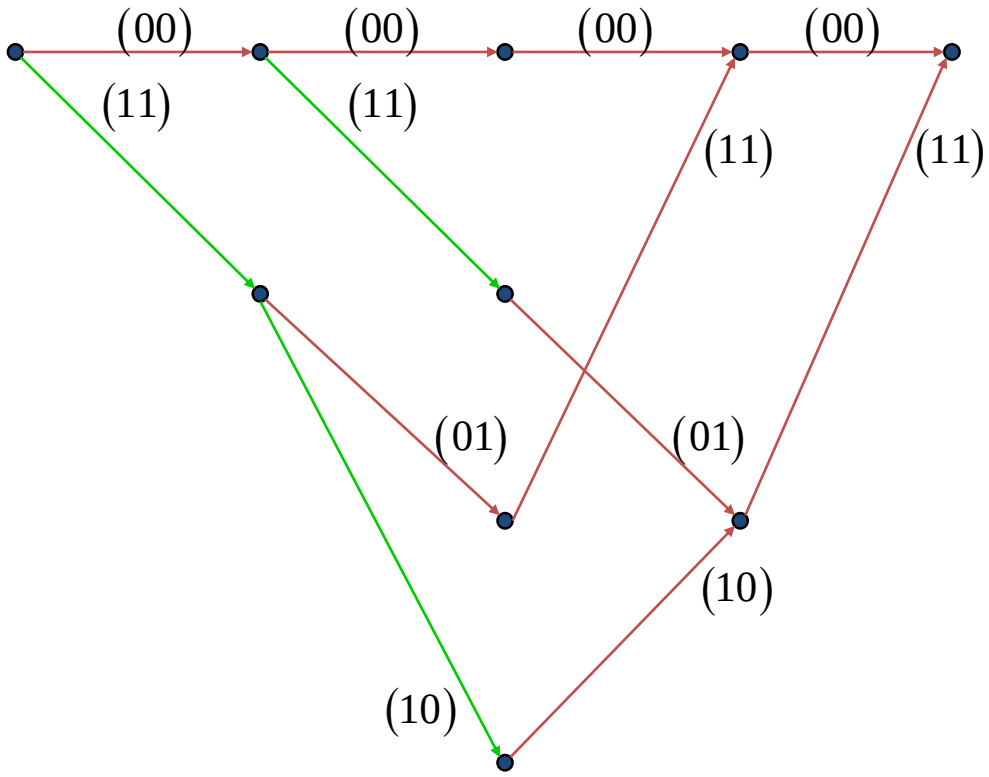


Representation as Trellis



Correctable Errors

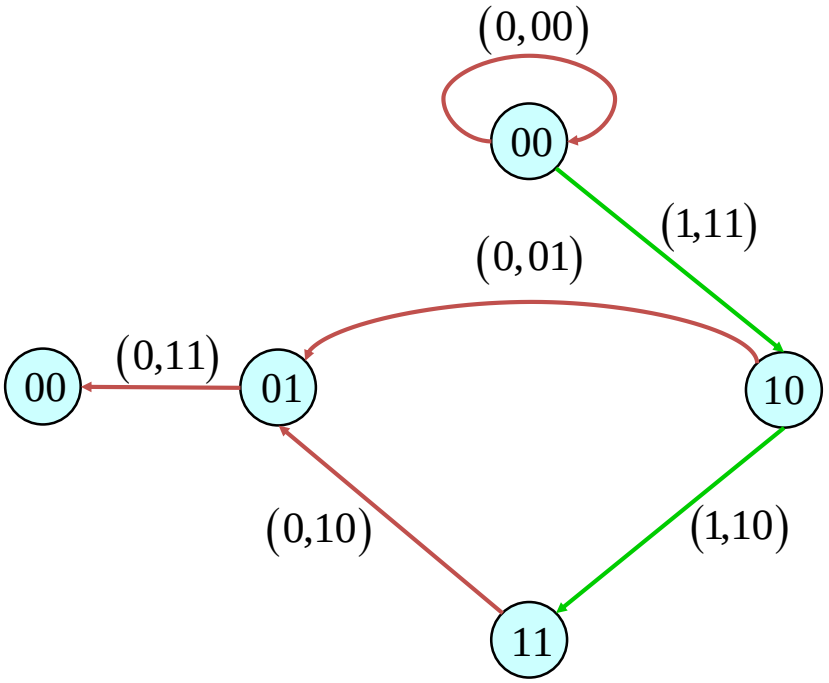
- Hamming distance: number of different bits in two (existing) code words
- Free distance: minimum Hamming distance between any two code words
- Correctable bit errors: less than half the free distance
- Determine free distance by trellis: path with minimum distance than diverges and remerges with all-zero path
- Example with trellis:



00 00 00 00
11 01 11 00 -> 5
00 11 01 11 -> 5
11 10 10 11 -> 6

free distance=5
correctable error burst=2

Free Distance with State Diagram



00 00 00 00
11 01 11 -> 5
11 10 10 11 -> 6

free distance=5
correctable error burst=2

Übersicht Kapitel 3: Kanalcodierung

Viterbi Decoding

- Objective:
  - find the code word that is closest to the received bit sequence
  - find the path through the trellis that is closest to the received bit sequence
- Algorithm:
  - states: $s = 0, \ldots, 2^M - 1$
  - path metric of state s at step j: $J(j,s)$
  - initialization of path metric: $J(0,s) = \begin{cases} 0 & \text{, for } s = 0 \\ \infty & \text{, for } s \neq 0 \end{cases}$
  - at step j for every state s:
    - best predecessor to state s: $q(s) = \arg\min_q \left\{ J(j-1,q) + H(r_j,q,s) \right\}$

$$H(r_j,q,s) = \begin{cases} \text{Hamming Distance}\left(r_j, output(q \to s)\right) & \text{if } \exists q \to s \\ \infty & else \end{cases}$$

    - path metric for state s: $J(j,s) = J(j-1,q(s)) + H(r_j,q(s),s)$
    - best path (code word) to state s: $c(j,s) = \left[c(j-1,s)\, input(q \to s) \right]$

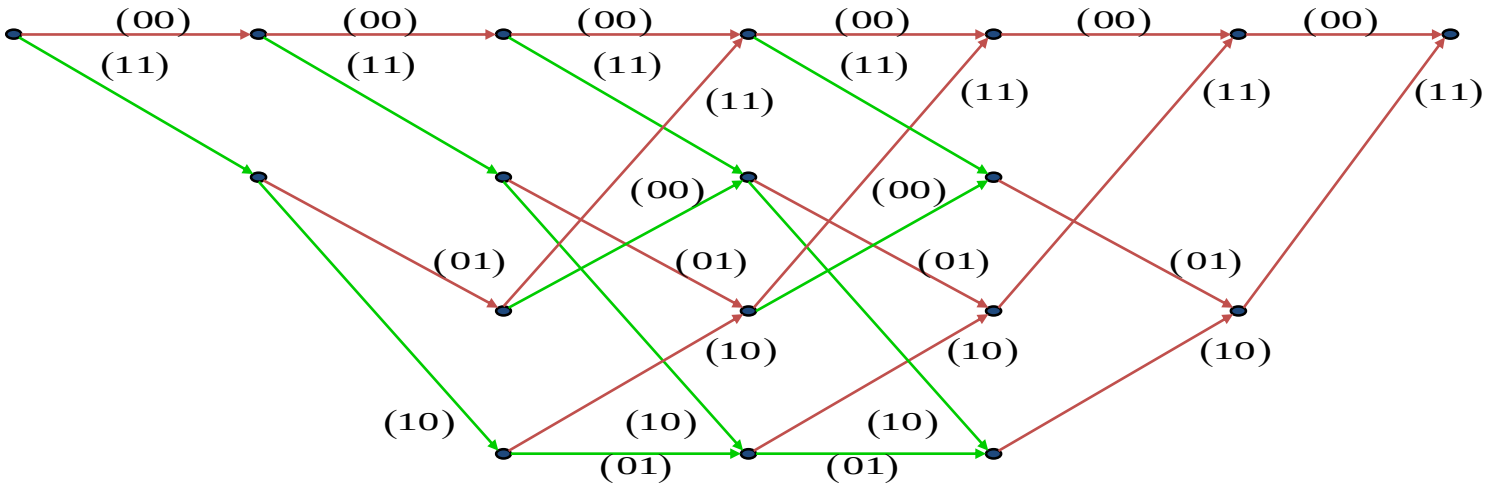
Andrew J. Viterbi

Decoding Example

- Input: 1101

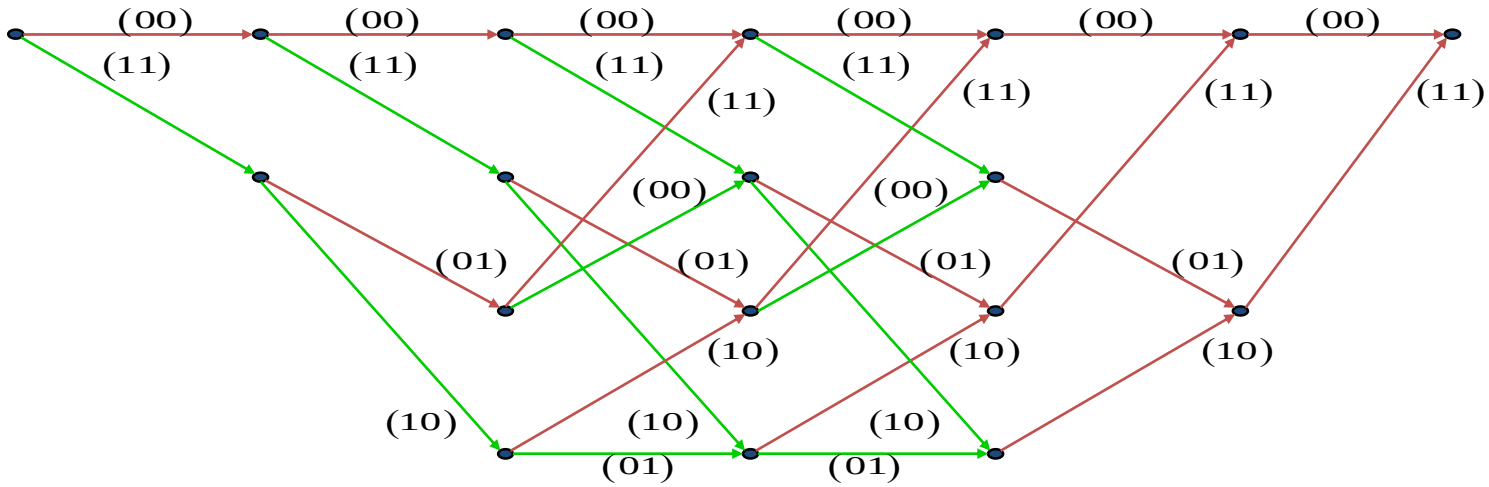
Output: 11 10 10 00 01 11

Received bit sequence: 11 11 10 01 01 11



Decoding Example

- Input: 1101 Output: 11 10 10 00 01 11
  Received bit sequence: 11 11 10 01 01 11



J(j,s)/c(j,s)		11	11	10	01	01	11
(00)	0	2/0	4/00	2/100	2/1100	3/11000	2/110100
(10)	∞	0/1	2/01	2/101	2/1101	∞	∞
(01)	∞	∞	1/10	1/110	2/1010	2/11010	∞
(11)	∞	∞	1/11	2/011	2/0111	∞	∞

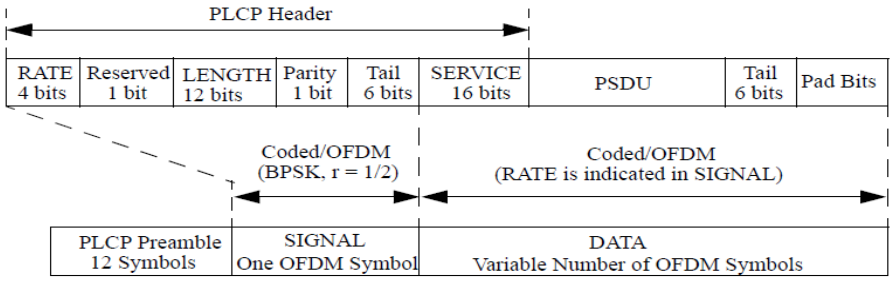
Stage by stage the low-cost path to every state is found. In each stage every state can be reached by two states of the previous stage. The costs for the path via a previous state are the costs to reach the previous state plus the transition costs to the next state. The costs for the cheapest path are the cost for the state in the current stage. Due to the appended zeros there is only a single state, the all-zeros state, in the last stage and the path with minimal costs from the all-zeros state at the beginning to the all-zeros state at the end corresponds to the decoded bit sequence.

Übersicht Kapitel 3: Kanalcodierung

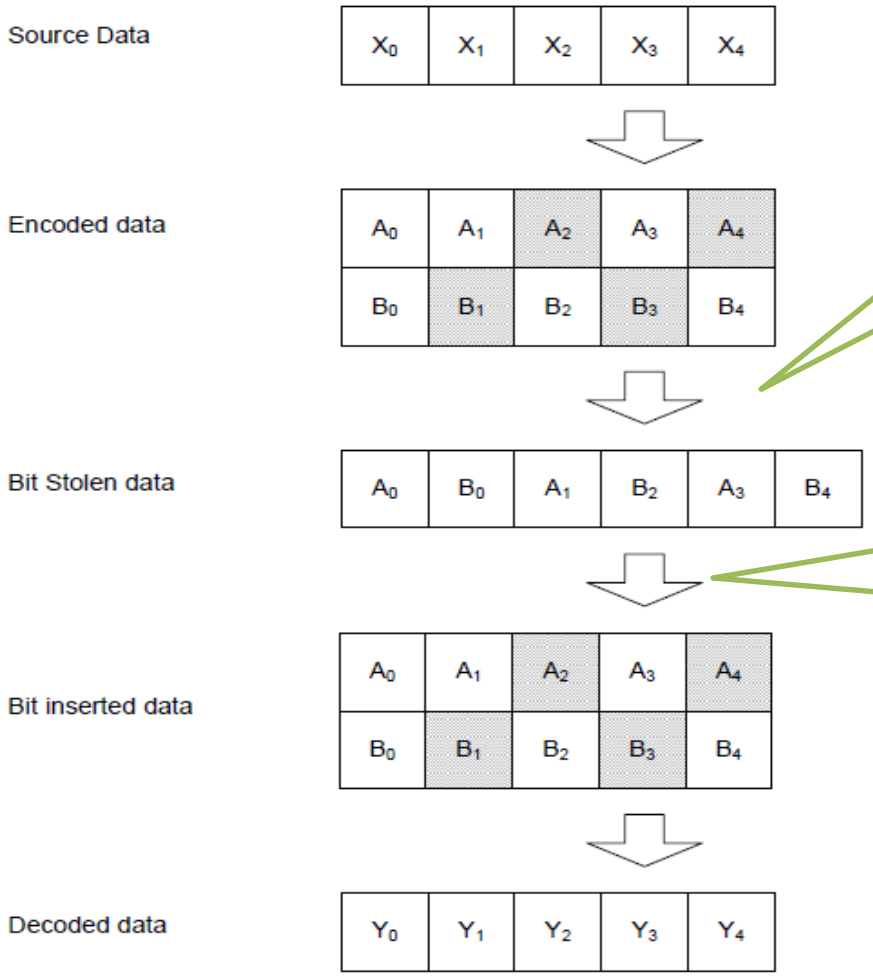
Puncturing

- Convolutional codes have integer code rates that result in a coarse granularity of “resulting bit probabilities”
- Puncturing is the technique to achieve arbitrary code rates, e.g. 2/3, 3/4, 5/6,...
- This is done by simply discarding (not transmitting) selected bits that are known at the receiver
- The decoder
  - assigns no weight to these bits (value 0.5)
  - all paths are equally possible
  - we have a “don’t care transition”
- Example: 54 Mbps in WiFi
  - symbols per OFDM symbol: 48
  - bit per OFDM symbol: 288 with 64-QAM
  - data bits per OFDM symbol: 216
  - data+header bits per ODFM symbol: 240
  - output bits with 5/6 coding: 288
  - output of 1/2 coder: 480 bits
  - puncturing 4 out of 10 bits: 192 bits
  - after puncturing: 288 bits

Data Rate (Mbps)	Modulation	Coding Rate	Data bits per OFDM symbol
6	BPSK	1/2	24
9	BPSK	3/4	36
12	QPSK	1/2	48
18	QPSK	3/4	72
24	16-QAM	1/2	96
36	16-QAM	3/4	144
48	64-QAM	2/3	192
54	64-QAM	5/6	216



Puncturing for 5/6 Coding in WiFi

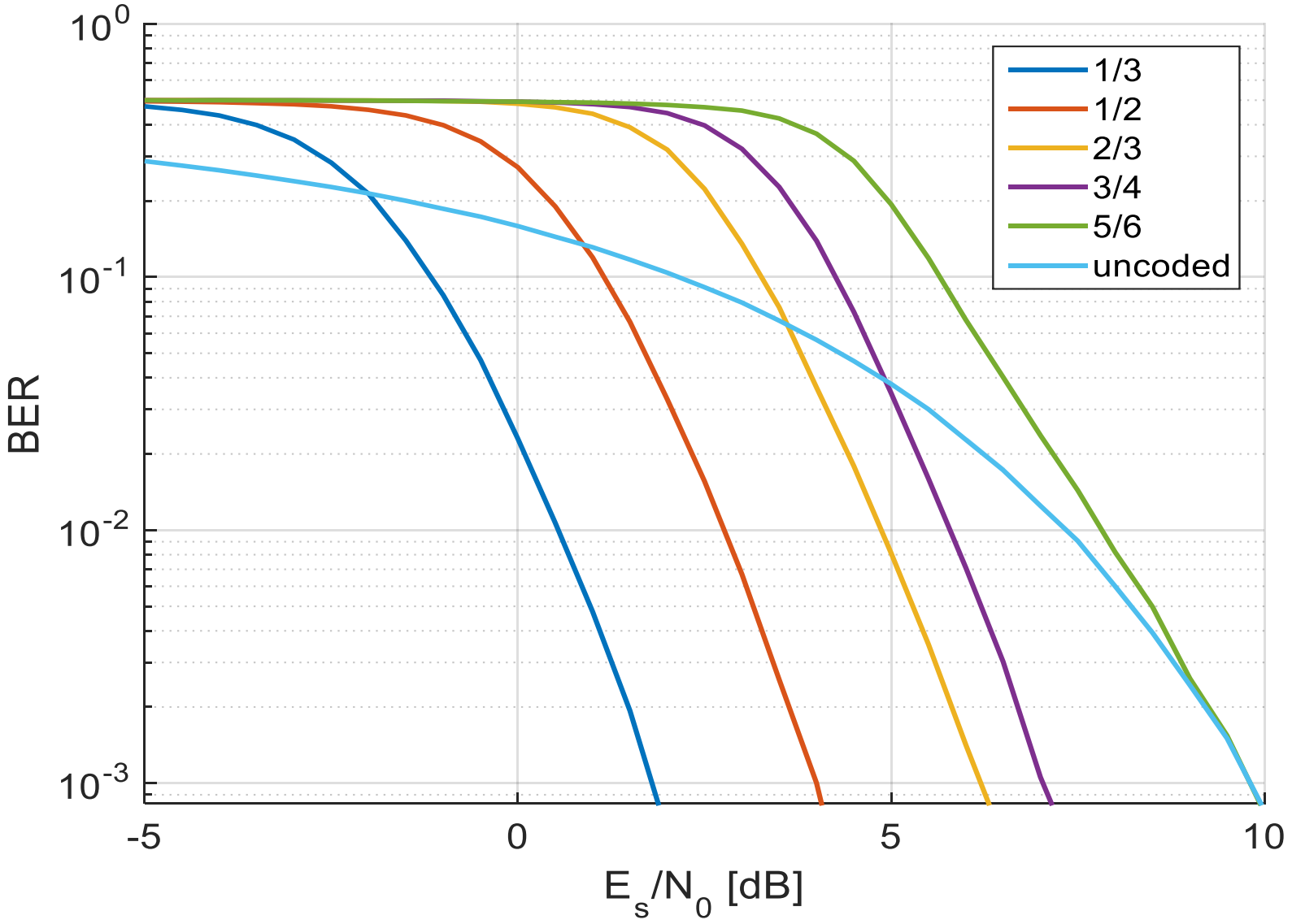


after encoding by the ½ convolutional coder out of every group of 10 bits the 4th, 5th, 8th, and 9th bits are deleted

after reception and before decoding by the ½ convolutional coder to every group of 6 bits a two “0” bits are added after the 3th and 5th bit.

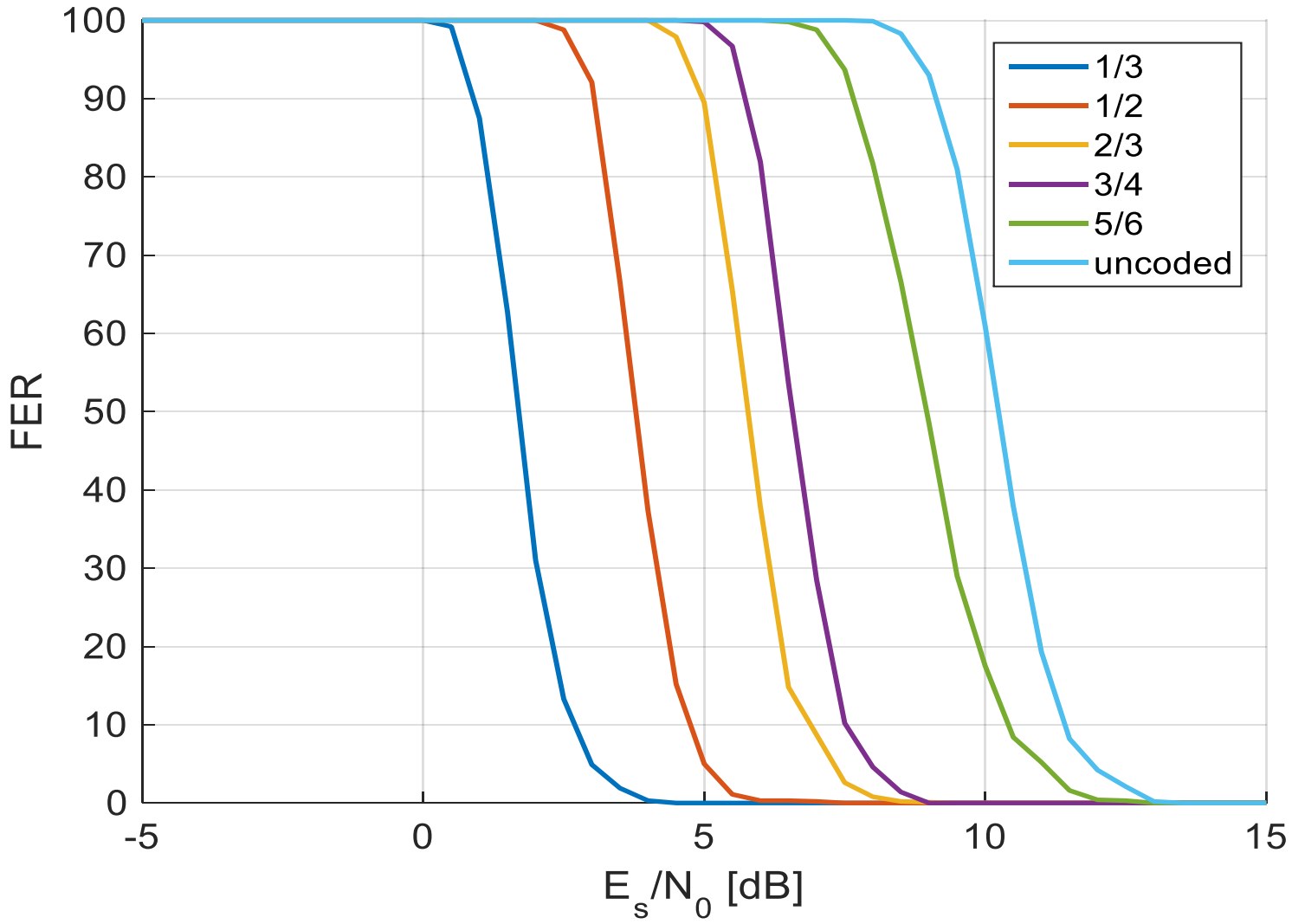
Effects of Puncturing - BER

Simulation Parameters: QPSK, single frame with 120000 bits payload



Effects of Puncturing - FER

Simulation Parameters: QPSK, 1000 frames with 1200 bits payload



Zusammenfassung

- Kanalcodierung dient zur Erkennung und Korrektur von Bitfehlern, die bei der Übertragung oder Speicherung von Daten auftreten
- Zur Fehlererkennung und Fehlerkorrektur werden unterschiedliche Codes verwendet, die für den jeweiligen Zweck am Besten geeignet sind
  - Fehlererkennung (CRC): Zyklische Blockcodes unterschiedlicher Länge
  - Blockcodes zur Fehlerkorrektur: Hamming, BCH, Reed-Solomon, LDPC
- Faltungscodes und deren Erweiterung Turbo-Codes werden vor allem in Mobilfunknetzen zur Fehlerkorrektur eingesetzt
- Codierung und Fehlerkorrektur:
  - Linear-systematische Blockcodes mit Generator- und Parity-Check-Matrix
  - Zyklische Blockcodes mit Generatorpolynom und Restklassen
  - Faltungscodes mit hard-decision Viterbi Decoding