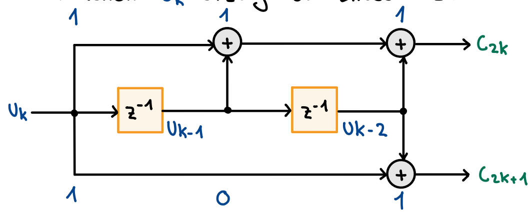
Faltungscodes

Blockcodes sind sehr effizient, um 1 Bitfehler zu korrigieren. Für Mehrbitfehler (Burst-Fehler) steigt aber der Aufwand der Blockcodes exponentiell.
Daher werden zur Korrektur von Mehrbitfehler Faltcodes eingesetzt. Diese sind lineare nicht aber systematische Codes.
Faltungscodes können beliebig lange Eingangsvektoren $\underline{u}$ (Streaming Code) verarbeiten.

Realisierung in Hardware

Schieberegister muss bei jedem Start des Encoders mit Nullen initialisiert werden.
Pro Datenbit u_k erzeugt der Encoder zwei Codebits c_{2k} und c_{2k+1}.



$c_{2k} = u_k \oplus u_{k-1} \oplus u_{k-2}$

$c_{2k-1} = u_k \oplus u_{k-2}$

$=$

Teilvektor $\underline{u}_k^* = (u_k \; u_{k-1} \; u_{k-2})$

$c_{2k} = (111) \cdot \underline{u}_k^{*T}$

$c_{2k-1} = (101) \cdot \underline{u}_k^{*T}$

↳ Vektoren : $\underline{a} = (a_1 a_2 a_3)$, $\underline{b}^* (b_3 \; b_2 \; b_1)$

Skalarprodukt: $a \cdot b^{*T} = a_1 \cdot b_3 + a_2 \, b_2 + a_3 \cdot b_1$

Gedächtnislänge m = Anzahl Flip-Flops (= Anzahl Tailbits) z.B. = 2

Einflusslänge $L = m + 1$ (= Länge Generatoren $\underline{g}$) z.B. auf $c_{2k} = m+1$ $(u_k \; u_{k-1} \; u_{k-2}) = 3$

Verlängerte Nutzdatenvektor $\underline{u}^+ = (\underline{u} \underbrace{0 \ldots 0}_{m \text{ Stück}})$ m angefügten Nullen = Tail-Bits, dadurch sind am Ende die Zustandbits $(u_{k-1} \; u_{k-2})$ auf Null zurückgesetzt

Coderate / maximale Coderate $R = \frac{K}{N} = \frac{K}{2 \cdot (K+m)}$ / R_{max}: $K \gg m \rightarrow R \approx 0.5$

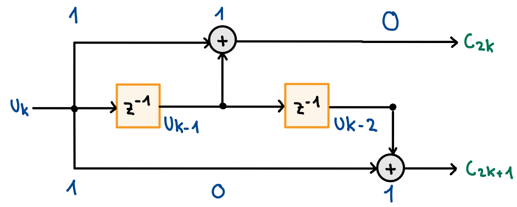
z.B. Encoderlauf mit $m=2$, $\underline{u} = (1011)$, $K=4$, $\underline{u}^+ = (101100)$ (K ist $\underline{u}^+$ ohne Tail-Bits)

Takt	Eingang	Zustand		Ausgang	
k	u_k	u_{k-1}	u_{k-2}	c_{2k}	c_{2k+1}
0	1	0	0	1	1
1	0	1	0	1	0
2	1	0	1	0	0
3	1	1	0	0	1
4	0	1	1	0	1
5	0	0	1	1	1
0	...	0	0	...	...

Initiale Werte

Tail-Bits

z.B $g_1 = [110]$, $g_2 = [101]$ Hardware



Ausgangscodewort $\underline{c} = (11 \; 10 \; 00 \; 01 \; 01 \; 11)$

Länge des Codeworts $N = 2 \cdot (K+m) = 2 \cdot (4+2) = 12$ Bit

Coderate $R = \frac{4}{2 \cdot (4+2)} \approx 0.333$

→ Dank Tail-Bits am Ende wieder initialisiert.

Generatorpolynome

Generatoren $\underline{g}$ können auch als Polynome dargestellt werden und die Ausgänge des Encoders durch eine Polynommultiplikation berechnet werden. (Tailbits ergeben sich bei der Multiplikation automatisch)

Generatoren $\underline{g}_1$ und $\underline{g}_2$ des Encoders : $\underline{g}_1 = (111) \rightarrow G_1(z) = z^2 + z + 1$

$\underline{g}_2 = (101) \rightarrow G_2(z) = z^2 + 1$

Nutzdatenvektor $\underline{u} = (u_0 \, u_1 \, u_2 \ldots u_{k-1})$: $\underline{u} = (1011) \rightarrow U(z) = z^3 + z + 1$

Generatoren $G_1(z)$, $G_2(z)$ ergeben Ausgangspolynome $C_1(z)$, $C_2(z)$: $C_1(z) = G_1(z) \cdot U(z) = (z^2+z+1)(z^3+z+1)$

$= z^5 + \cancel{z^3} + \cancel{z^2} + z^4 + \cancel{z^2} + \cancel{z} + \cancel{z^3} + \cancel{z} + 1 = z^5 + z^4 + 1 = 110001$

$C_2(z) = G_2(z) \cdot U(z) = (z^2+1) \cdot (z^3+z+1)$

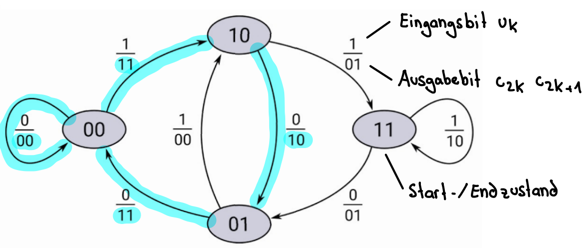
$= z^5 + \cancel{z^3} + z^2 + \cancel{z^3} + z + 1 = z^5 + z^2 + z + 1 = 100111$

Koeffizientenvektoren Kombination ergibt Codevektor $\underline{c}$: $\underline{c} = 11 \; 10 \; 00 \; 01 \; 01 \; 11$

Zustandsbeschreibung

Eingangsvektor $\underline{u} = (1011)$, $\underline{u}^+ (101100)$, Ausgangsvektor $\underline{c} = (11 \; 10 \; 00 \; 01 \; 01 \; 11)$

Takt	Eingang	Startzustand	Endzustand	Ausgang
k	u_k^+	$(u_{k-1}^+ \; u_{k-2}^+)$	$(u_k^+ \; u_{k-1}^+)$	$(c_{2k} \; c_{2k+1})$
0	1	(00)	(10)	(11)
1	0	(10)	(01)	(10)
2	1	(01)	(10)	(00)
3	1	(10)	(11)	(01)
4	0	(11)	(01)	(01)
5	0	(01)	(00)	(11)
0	...	(00)	...	...



dfree Codewort (5)

Freie Distanz

Man spricht nicht von minimaler Hamming-Distanz sondern einer freien Distanz d_{free} (faktisch gleich). Gesucht ist ein Codewort, welches so wenige Einsen wie möglich aufweist (aber mind. eine Eins). Im Zustandsdiagramm ein Codewort, welches immer im Zustand (00) verharrt und nur einmal verlässt, um auf dem kürzesten Weg (wenigsten Einsen am Ausgang) wieder dorthin zurück zu kehren.

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

z.B. Codewort $\underline{c} = (00\ 00\ 11\ 10\ 11\ 00\ 00)$

Freie Distanz $d_{free} = 5$

Erkennbare Fehler $= d_{free} - 1 \quad = 4$ Bit

Korrigierbare Fehler $= \left\lfloor \frac{d_{free} - 1}{2} \right\rfloor = 2$ Bit

Es gibt zu jedem Wertepaar $(\gamma ; m)$ eine maximal mögliche freie Distanz d_{free}. Diese nennt man Optimum Free Distance (OFD) und Codes welche diese freie Distanz erreichen OFD-Codes. ! γ = Anzahl Generatoren, m = Anzahl Tail-Bits z.B. $m = 2$, $\gamma = 2$ $(101_b, 111_b)$, $d_{free} = 5$

Decoder

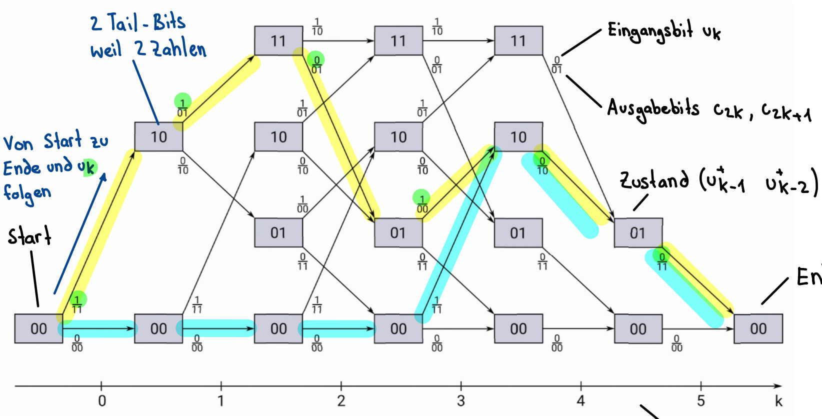
Soll zu möglicherweise fehlerhaften Bitmuster $\tilde{\underline{c}}$ das wahrscheinlichste Codewort $\hat{\underline{c}}$ finden und dies zu $\hat{\underline{u}}$ codieren. Die Schätzung $\hat{\underline{c}}$ und in Folge $\hat{\underline{u}}$ könnte falsch sein, daher mit Dach ^ markiert. Es wird mit dem Maximum-Likelihood-Algorithmus (Viterbi-Decoder) decodiert, basierend auf Trellis-Diagramm.

Trellis-Diagramm

Zeigt alle möglichen Zustandsabfolgen im zeitlichen Verlauf. $u_k = (110100)$, $\hat{\underline{c}} = (11\ 01\ 01\ 00\ 10\ 11)$

Current- / Next-State:

	Current State u_{k-1}	Current State u_{k-2}	Input u_k	Output c_{2k}	Output c_{2k+1}	Next State u'_{k-1}	Next State u'_{k-2}
7	1	1	1	1	0	1	1
6	1	1	0	0	1	0	1
5	1	0	1	0	1	1	1
4	1	0	0	1	0	0	1
3	0	1	1	0	0	1	0
2	0	1	0	1	1	0	0
1	0	0	1	1	1	1	0
0	0	0	0	0	0	0	0



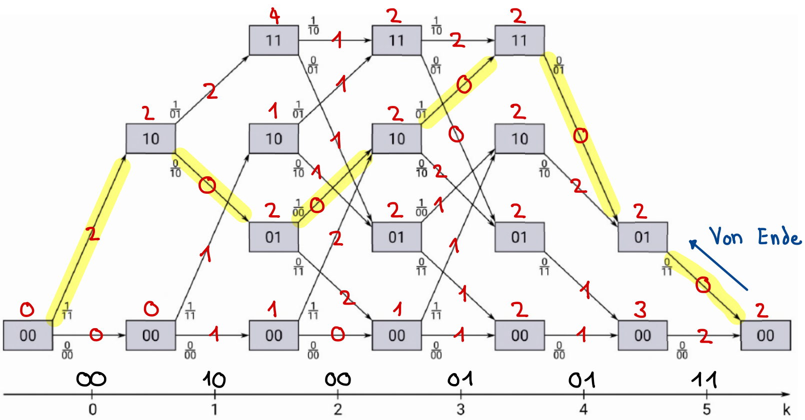
Wahrscheinlichster Pfad

Geringste Anzahl Fehler (Kostenmetrik)

d_{free} Codewort (5)

Viterbi-Decoder

$\underline{c} = (11\ 10\ 00\ 01\ 01\ 11)$, $\underline{e} = (11\ 00\ 00\ 00\ 00\ 00)$, $\tilde{\underline{c}} = \underline{c} + \underline{e} = (00\ 10\ 00\ 01\ 01\ 11)$, $m = 2$



1. Zustände mit Code vergleichen
2. Fehler eintragen (immer kleinster wählen)
3. Verbindung einzeichnen (mit kleinster Anzahl totaler Fehler)
4. Bits entsprechend anpassen: 00 10 00 01 01 11

   ↳ 11 10 00 01 01 11

$\hat{\underline{u}}^+ = (101100)$ ⟶ Anzahl Tail-Bits $m = 2$

$\hat{\underline{u}} = (1011)$