

## Samlet udgave af slides for fejlkorrigerende koder

1. Fejlkorrigerende og fejldetekterende kodning Slides KOD-1 – KOD-21	Side 4
2. Lineære koder Slides LIN-1 – LIN-18	Side 25
3. Eksempler på lineære koder. Sandsynlighedsregning og koder Slides EKS-1 – EKS-13	Side 43
4. Foldningskoder Slides FOL-1 – FOL-15	Side 56
5. Foldningskoder. Viterbi dekodning Slides VIT-1 – VIT-16	Side 71
6. Cykliske koder Slides CYC-1 – CYC-17	Side 87
7. Endelige legemer. Reed-Solomon koder Slides RSC-1 – RSC-16	Side 104
8. Dekodning af Reed-Solomon koder, Slides RSD-1 – RSD-15	Side 120
9. Simulering af fejlkorrektion med foldnings- og blokkoder Slides SIM-1 – SIM-16	Side 135
10. Binære BCH koder Slides BCH-1 – BCH-10	Side 151
11. Sammensætning af koder. Produktkoder med iterativ dekodning. Lidt om turbo-koder Slides SAM-1 – SAM-39	Side 161

## Indeks for fejlkorrigerende koder

(N,K)-kode	KOD-9
Afstand (Hamming)	KOD-12
AWGN	SIM-5
BCH kode	EKS-6, BCH-8
BCH kode dekodning	BCH-8
BDD-dekodning	KOD-16
Berlekamp-Massey algoritme	RSD-11, BCH-8
Binomialkoefficient	EKS-8
Binær symmetrisk kanal, BSC	EKS-7
Bitfejlsandsynlighed	EKS-7

Blokkode	KOD-9
Bløde beslutninger	VIT-10 - VIT-12, SAM-23
CRC	CYC-14 - CYC-15
Cyklist kode	CYC-3, CYC-6, RSC-12
Cyklist kode - dekodning	CYC-9
Cyklist kode - kodning	CYC-7
Dekodning	KOD-10
DVB-T	VIT-14
Euklids algoritme	RSD-11, BCH-8
Fejldetektion	KOD-15, KOD-18
Fejlkorrektion	KOD-15, KOD-17, LIN-12
Fejlkorrigerende kode	KOD-8 - KOD-10
Fejllokeringspolynomium	RSD-7
Fejsandsynlighed	EKS-7, EKS-9
Fejsandsynlighed fejldetektion	EKS-10 - EKS-11
Fejsandsynlighed fejlkorrektion	EKS-9
Foldningskode	FOL-4
Foldningskode - kodning	FOL-4, FOL-10
Foldningskode - tilstand	FOL-6 - FOL-7
Fri afstand	FOL-12
Gaussisk støj	SIM-5
Generatormatrix, G	LIN-13, LIN-16
Hamming-kode	KOD-5, KOD-11, EKS-3
Hamming-kode dekodning	KOD-7, LIN-5 - LIN-6
Hamming-kode indkodning	KOD-6, LIN-4
Hastighed for kode	KOD-9, FOL-5
Hukommelse, M	FOL-6
Indkodning	KOD-6, LIN-4, LIN-13, RSC-11
Indre kode	SAM-4
Informationsbit	KOD-9, LIN-14
Interleaving	SAM-4
Iterativ dekodning	SAM-6, SAM-10, SAM-22, SAM-33
K (antal informationsbit)	KOD-9
Kodningsgevinst	SIM-8, SAM-5
Legeme - endeligt	RSC-3, RSC-5
LFSR	SIM-11
Lineær kode	LIN-9, RSC-10
Maksimallængdesekvens	SIM-11
MAP-dekodning	KOD-16
Minimalpolynomium	BCH-6
ML-dekodning	KOD-16
Mobiltelefon	VIT-14
Modulo 2	LIN-3
m-sekvens	SIM-11

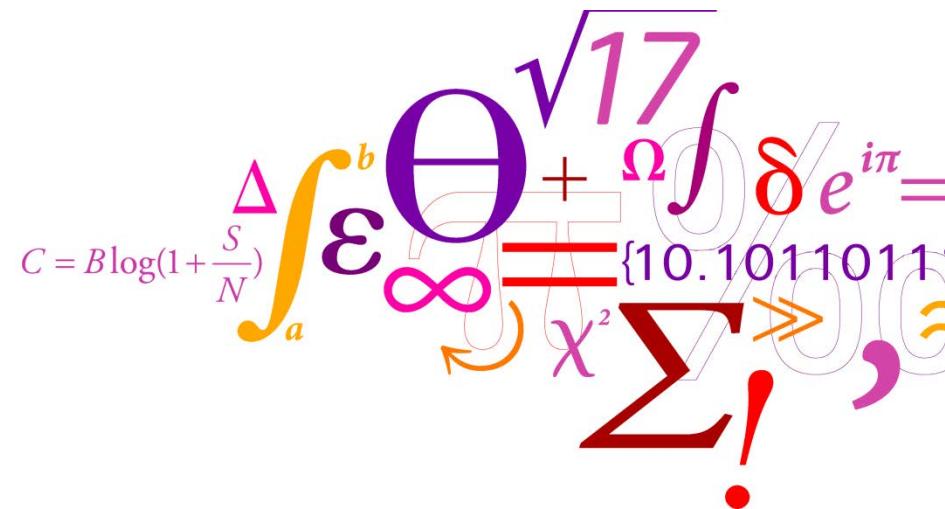
N (længde af kodeord)	KOD-9
Paritetscheck	KOD-4
Paritetscheckmatrix, H	LIN-8 - LIN-9, LIN-16
Permutation	EKS-8
Polynomiumsdivision	CYC-6
PRBS	SIM-11
Primitivt element	RSC-4
Primitivt polynomium	CYC-12, RSC-5
Produktkode	SAM-9
Pseudotilfældig sekvens	SIM-10 - SIM-12
Punktering af foldningskode	FOL-13, VIT-13
Redundansbit	KOD-9, LIN-14
Reed-Solomon kode	RSC-11
Restklasseregning	LIN-3, RSC-3
RS kode	EKS-6, RSC-11
RS kode dekodning	RSD-4 - RSD-11
RS kode fejsandsynlighed	RSD-13
RS kode kodning	RSC-11, RSC-14
RS kode syndrom	RSD-9
Sammensat kode	SAM-3
Satellitkommunikation	VIT-14, SAM-7 - SAM-8
Scrambling	SIM-14
Signal/støjforhold	SIM-6
Simulering af transmissionskanal	SIM-9
Symbolfejsandsynlighed	RSD-13
Syndrom	LIN-5, LIN-11, CYC-9, RSD-9, BCH-8
Systematisk kodning	KOD-18, LIN-14
Tilbagesøgningslængde	VIT-9
Tilfældig sekvens	SIM-10
Tilstandsdiagram	FOL-7
Trellisgraf	FOL-3, FOL-9
Turbo kode	SAM-31 - SAM-33
Udtyndet Hammingkode	EKS-5, CYC-14
Udvidet Hammingkode	EKS-4
Varians for støj	SIM-6
Viterbi-algoritmen	VIT-6 - VIT-12
Vægt (Hamming)	KOD-12
Ydre kode	SAM-4

# Fejlkorrigerende og fejldetekterende kodning

34220

Knud J. Larsen

Slides: KOD

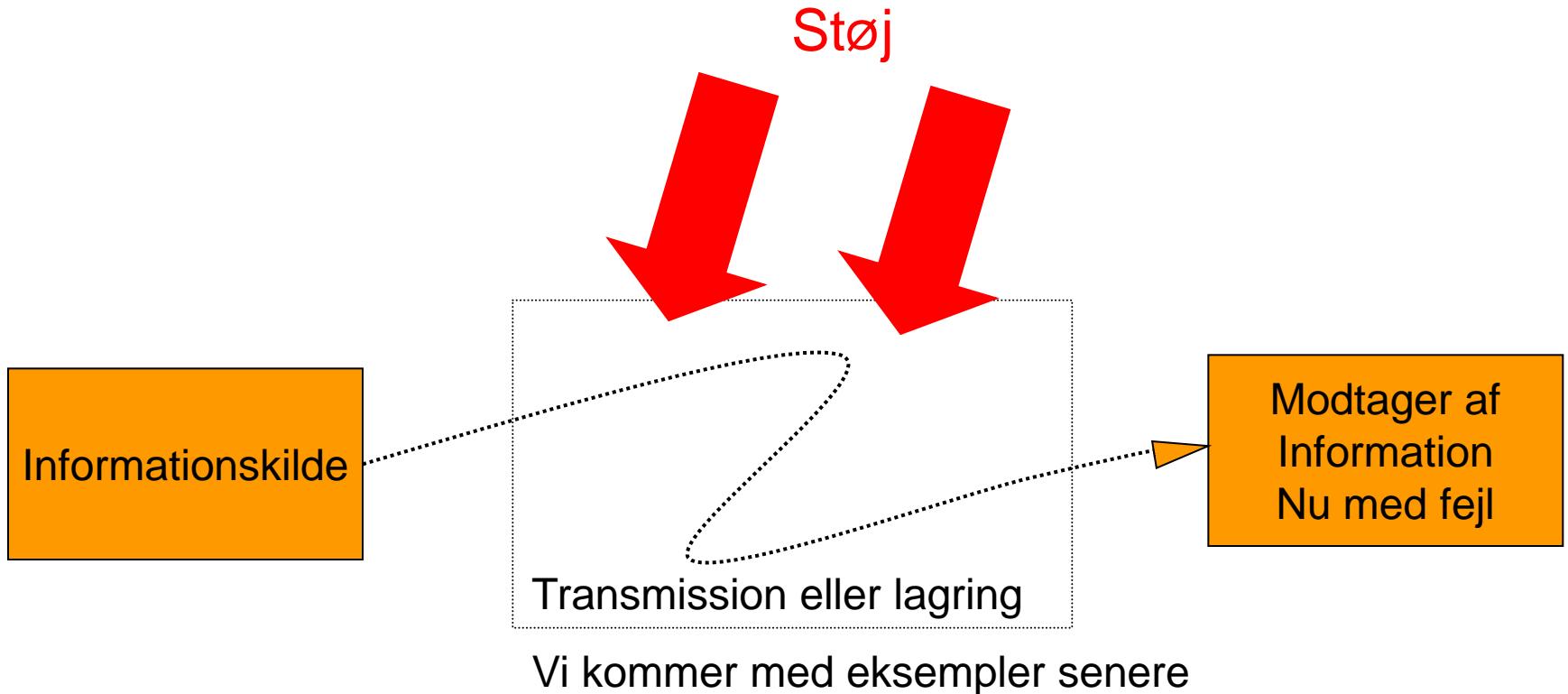
$$C = B \log\left(1 + \frac{S}{N}\right)$$

$$\int_a^b \Theta_+^{\sqrt{17}} \delta e^{i\pi} = \{10.1011011\}$$

# Fejldkorrigende og fejldetekterende kodning

I denne del af kurset vil vi se på

- Fejldetection og -korrektion ved hjælp af koder
- Introducere begreber
- Hamming-kode eksempel
- Lineære koder – paritetscheckmatrix, syndromer, dekodning, minimumsvægt, generatormatrix, systematisk kodning
- Fejlsandsynligheder ved anvendelse af blokkoder
- Foldningskoder og deres dekodning (Viterbi-algoritmen)
- Cykliske koder – polynomiumsbeskrivelse
- Endelige legemer og koder over disse
- Reed-Solomon koder og deres dekodning
- Simulering af koder på virkelige kanaler
- BCH-koder
- Sammensætning af simplere koder til stærke systemer

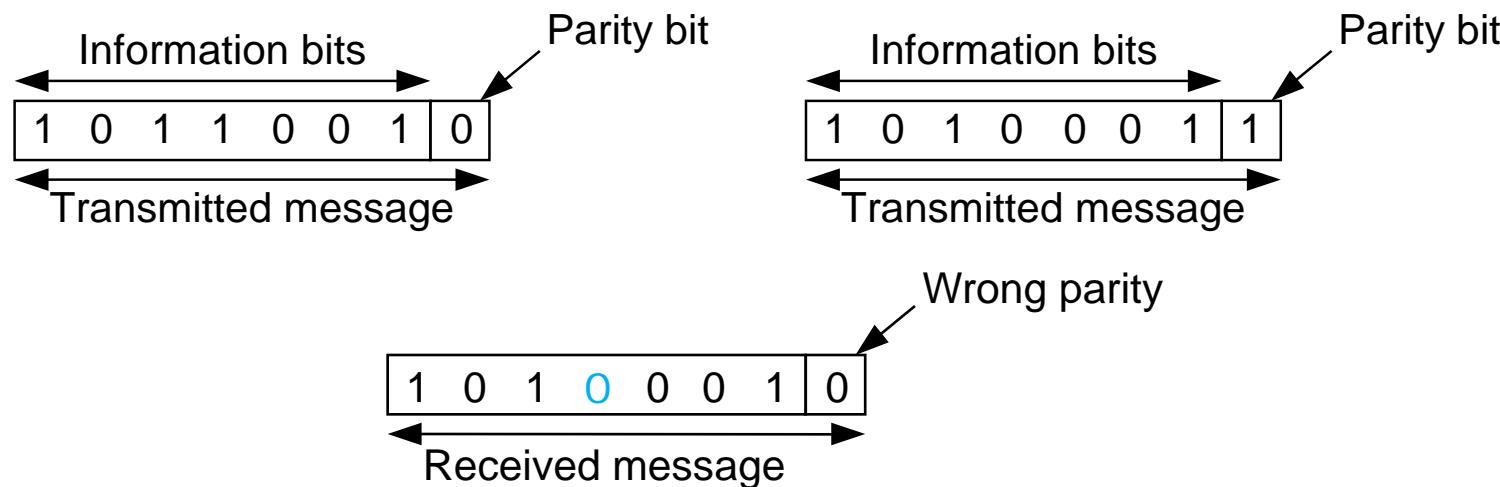
# Fejlkorrigerende og fejldetekterende kodning



Information kan være  
0, 1, a, B, \$, tale, video osv.

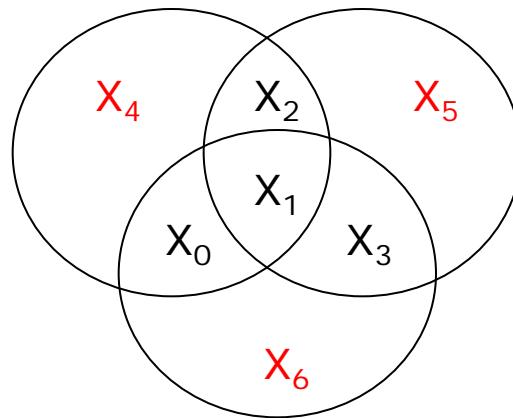
# Fejlkorrigerende og fejldetekterende koder

- En kode kan opdage hvis en modtaget meddeelse har fejl
- Simpelt eksempel: Paritetscheck
  - Antallet af 1'er skal være lige
  - F.eks. brugt i seriel port på PC
- Hvad er karakteristisk? En matematisk struktur!!
- Med mere komplikerede strukturer kan man også rette fejl!



# Eksempel: Hamming kode - struktur

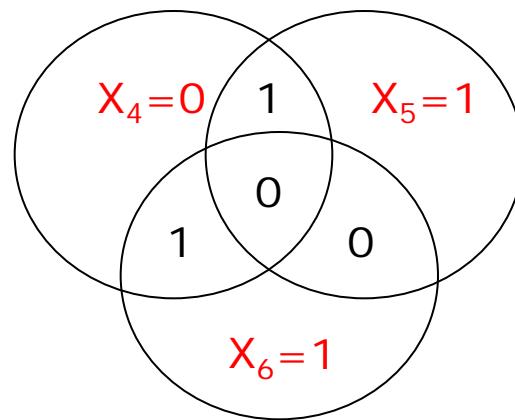
- Eksempel: Hamming kode - struktur
  - Informationsbit:  $x_0, x_1, x_2, x_3$
  - Nu vil vi kræve et lige antal 1'ere i hver af de 3 cirkler og sender i stedet for en hel vektor af 7 bits:



- For at få det har vi tilføjet 3 redundansbit, de røde
  - Transmitteret kodeord er  $(x_0, x_1, x_2, x_3, x_4, x_5, x_6)$

# Hamming kode

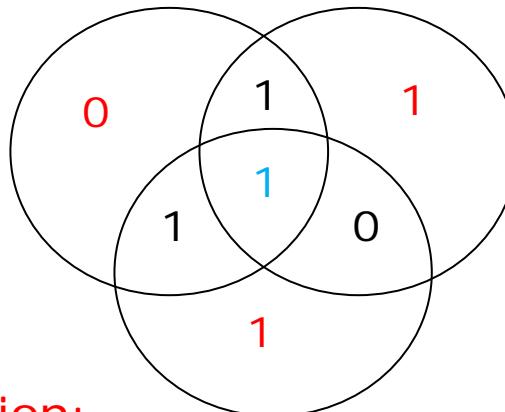
- Eksempel: Hamming kode – (ind)kodning
  - Informationsbit:  $(x_0, x_1, x_2, x_3) = (1, 0, 1, 0)$
  - Et lige antal 1'ere



Transmitteret kodeord bliver altså  $(1, 0, 1, 0, 0, 1, 1)$

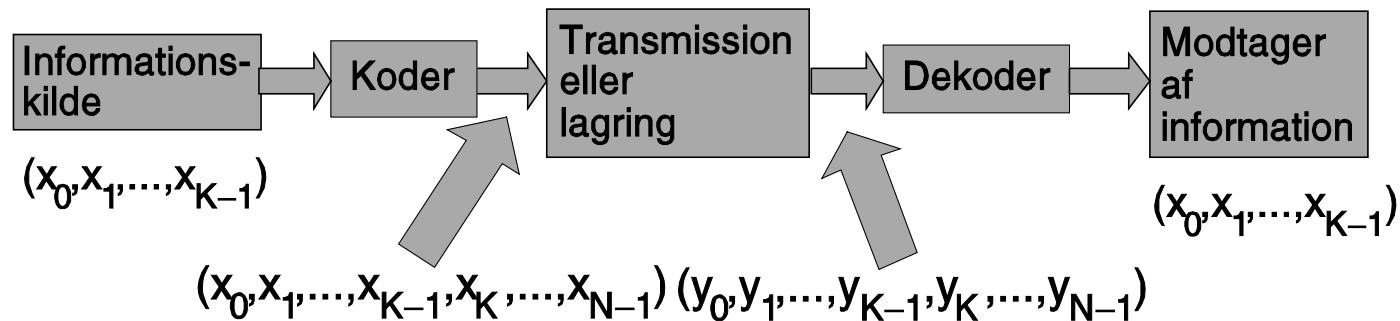
# Hamming kode - dekodning

- Eksempel: Hamming kode – dekodning (afkodning)
  - En fejl indtræffer:  $x_1 = 1$
  - Et lige antal 1'ere i alle 3 cirkler kræves - fejldetektion



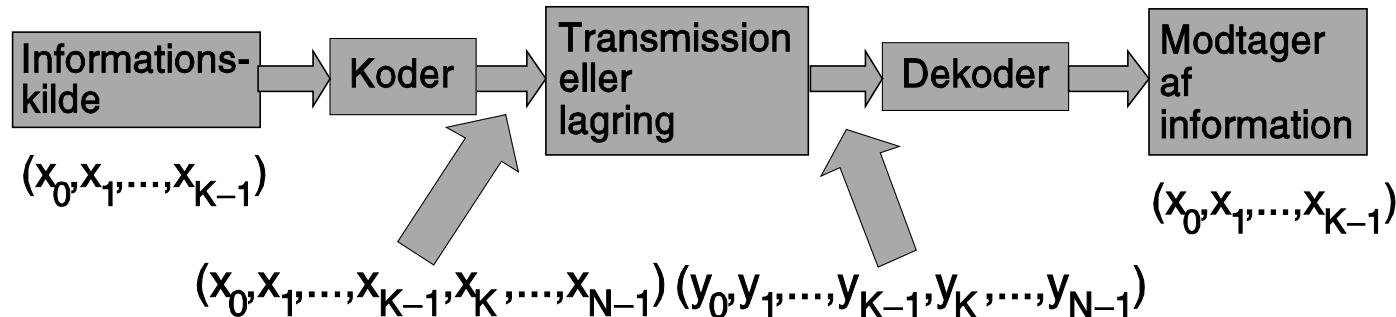
- Fejlkorrektion:
  - Ved at prøve at vende hver bit kan man finde at  $x_1$  må være 0
  - Man kan også prøve med 2 eller 3 fejl, men så viser der sig fejlslutninger
- Fejldetektion:
  - Man kan også lade være med at rette fejlen – blot konstatere at der er noget galt

# Generel model med anvendelse af fejlkorrigerende kode



En fejlkorrigerende kode skaber en matematisk sammenhæng i de transmitterede sekvenser og det udnyttes til at rette fejl der opstår ( $y_j \neq x_j$ )

# Parametre for fejlkorrigerende koder



- N antal bit i **kodeord** ( $x_0, x_1, \dots, x_{N-1}$ )
- K antal **informationsbit** ( $x_0, x_1, \dots, x_{K-1}$ )
- $(N, K)$  – kode – (7,4) i vores eksempel
- **Kodens hastighed** er  $K/N$  – lille hastighed kan rette mange fejl, men koster også mest da  $N-K$  ekstra **redundansbit** skal sendes
- En sådan kode kaldes en **blokkode** da kodeordene findes i blokke af længde N – vi skal senere se koder med en mere flydende struktur

# Fejlkorrigerende princip

En fejlkorrigerende kode skaber en matematisk sammenhæng i de transmitterede sekvenser

Den storslæde idé er så at finde en struktur, der spreder de  $2^k$  kodeord så godt som muligt ud over samtlige  $2^n$  sekvenser, således at hvis der er (få) fejl kan man genkende det nærmest liggende kodeord og derved få rettet fejlene (**dekodning**).

Det svarer meget til vores sprog:

Hvis man ikke får fat i alle detaljer eller der er stavfejl kan man som regel rekonstruere det sagte/skrevne

# (7,4) Hamming kode

- Eksempel: (7,4)-Hamming kode

Hvis vi gennemløber alle 16 muligheder for at vælge informationsbit  $x_0, x_1, x_2, x_3$  fås ved at udfylde cirklerne rigtigt:

0 0 0 0 0 0 0	1 0 0 0 1 0 1
0 0 0 1 0 1 1	1 0 0 1 1 1 0
0 0 1 0 1 1 0	1 0 1 0 0 1 1
0 0 1 1 1 0 1	1 0 1 1 0 0 0
0 1 0 0 1 1 1	1 1 0 0 0 1 0
0 1 0 1 1 0 0	1 1 0 1 0 0 1
0 1 1 0 0 0 1	1 1 1 0 1 0 0
0 1 1 1 0 1 0	1 1 1 1 1 1 1

Sorte bit informationsbit, røde bit er paritetsbit

Bemærk at der altid skal ændres mindst 3 bit for at komme fra et kodeord til et andet. Dette kaldes (Hamming-)afstanden mellem kodeord.

# Afstand mellem vektorer

## (Hamming-)afstand

Hamming-afstanden mellem 2 vektorer,  $\mathbf{a} = (a_0, a_1, \dots, a_{N-1})$  og  $\mathbf{b} = (b_0, b_1, \dots, b_{N-1})$ ,

$d(\mathbf{a}, \mathbf{b})$  = antal positioner der er forskellige,  $a_i \neq b_i$

$d(\mathbf{a}, \mathbf{b})$  opfylder de matematiske krav til en afstand:

$d(\mathbf{a}, \mathbf{a}) = 0$ ,  $d(\mathbf{a}, \mathbf{b}) = d(\mathbf{b}, \mathbf{a})$  og  $d(\mathbf{a}, \mathbf{c}) \leq d(\mathbf{a}, \mathbf{b}) + d(\mathbf{b}, \mathbf{c})$

Den sidste ulighed kaldes trekantsuligheden og kan bevises:

Del de  $d(\mathbf{a}, \mathbf{c})$  positioner, hvor  $a_i \neq c_i$  i to disjunkte mængder X hvor  $a_i = b_i$  og Y hvor  $a_i \neq b_i$ . Så er  $d(\mathbf{a}, \mathbf{c}) = |X| + |Y|$ . For Y gælder  $|Y| \leq d(\mathbf{a}, \mathbf{b})$  og for positionerne i X gælder  $a_i = b_i \neq c_i$ , så  $|X| \leq d(\mathbf{b}, \mathbf{c})$ .

Da det er en afstand dropper vi Hamming i navnet og taler nu kun om afstande

## (Hamming-)vægt

Hamming-vægten af en vektor,  $\mathbf{a} = (a_0, a_1, \dots, a_{N-1})$ ,

$w(\mathbf{a}) = \text{antal positioner der er forskellig fra } 0, a_i \neq 0$ .

Det er klart at afstanden mellem 2 vektorer,  $\mathbf{a} = (a_0, a_1, \dots, a_{N-1})$  og  $\mathbf{b} = (b_0, b_1, \dots, b_{N-1})$ ,  $d(\mathbf{a}, \mathbf{b}) = w(\mathbf{a} - \mathbf{b})$  (hvis man kan regne på værdierne)

Som for afstande dropper vi Hamming i navnet og taler nu kun om  
**vægte**

## Egenskaber for lige og ulige vægt

- Hvis man har kodeord med lige vægte, er det klart at afstanden mellem to kodeord med lige vægt også må være et lige tal da
$$d(\mathbf{a}, \mathbf{b}) = w(\mathbf{a}) + w(\mathbf{b})$$
  - $2 \cdot (\text{antal overensstemmende 1-positioner})$ .

- Er der også ulige vægte viser samme ligning at afstanden mellem to kodeord med ulige vægt er et lige tal mens afstanden mellem et ord med lige vægt og et med ulige må være ulige.

# Geometrisk fortolkning

Geometrisk fortolkning af en kode og dekodning med det indførte afstandsbegreb

Hvis koden anvendes til  
**fejlkorrektion** kan man i  
hvert fald rette

$$T = \frac{d - 1}{2} \text{ fejl}$$

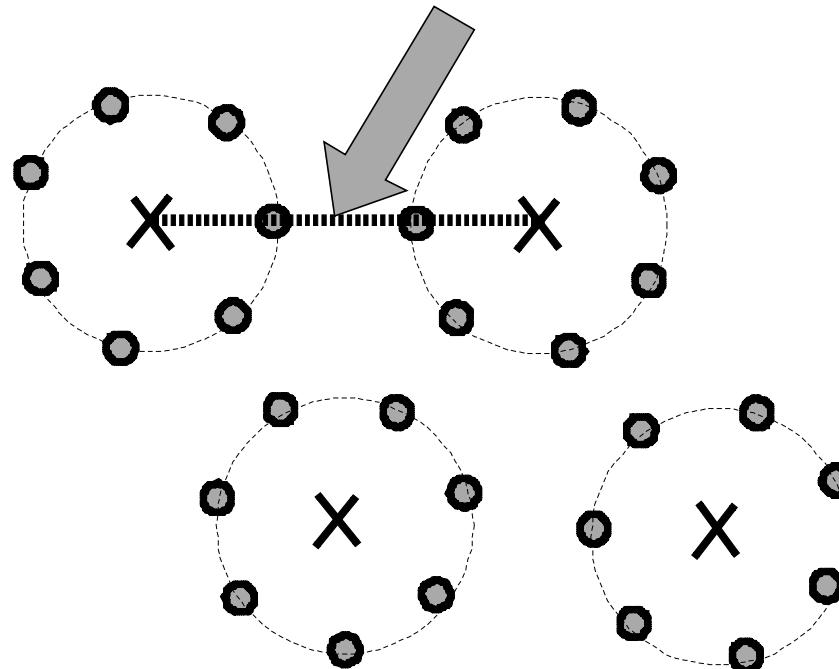
men måske også flere i  
visse situationer

Hvis koden anvendes til  
**fejldetektion** kan man i  
hvert fald detektere

$$d - 1 \text{ fejl}$$

men ofte langt flere – de  
eneste fejmønstre man  
ikke kan detektere er  
dem der er kodeord i sig  
selv

**Minimumsafstand  $d$  (=3 her)**

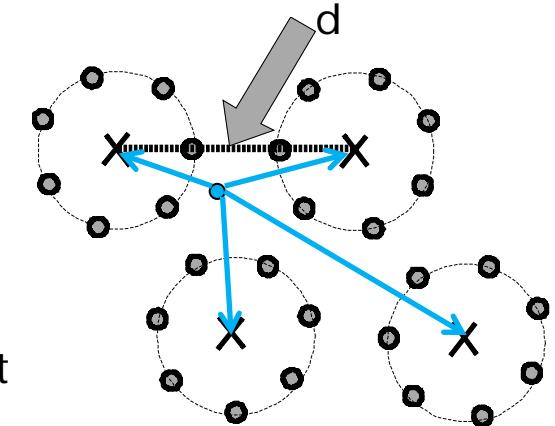


**X Kodeord**      **● Ord med fejl**

(Hamming (7,4) som eksempel)

# Forskellige dekodingsprincipper

- Man er interesseret i at skabe så gode koder som muligt givet N og K, dvs. kunne rette så mange fejl som muligt, men dekodingsmetoden spiller også en rolle
- En dekodingsmetode der kun kan rette op til et fast antal fejl betegnes oftest BDD (*bounded distance decoding*). Et eksempel er vist på figuren hvor man kan rette fejl ud til cirklen – som regel ved  $(d-1)/2$ , hvor d er kodens minimumsafstand
- Hvis alle kodeord er lige sandsynlige kunne man forbedre det ved altid at vælge det kodeord der er nærmest ved det modtagne. Illustreret med blåt. En sådan metode kaldes ML (*maximum likelihood*)
- Hvis der også er forskellige sandsynheder bliver det mere kompliceret. Sådanne metoder betegnes MAP (*maximum a posteriori probability*)
- I første omgang beskæftiger vi os kun med BDD-metoder, men senere kommer også en ML-metode



# Koder og deres dekodning

- Man er interesseret i at skabe så gode koder som muligt givet N og K, dvs. kunne rette så mange fejl som muligt
- Det bliver svært med rigtigt lange koder, fx med  $N = 1023$  kan man opbygge en (BCH-)kode med  $K = 973$  med minimumsafstand 11
- Det vil være svært at sammenligne en modtaget sekvens med alle mulige  $2^K = 8 \cdot 10^{292}$  sekvenser (ML-dekodning)
- Arbejdet kan reduceres for en vis type koder, **lineære koder**, ved – som vi skal se – at udregne nogle såkaldte syndromer som kun afhænger af fejlmønstret – af dem er der såmænd kun  $2^{50}$  mulige og så skal man "bare" finde hvilke der svarer til de  $9,3 \cdot 10^{12}$  fejlmønstre med op til 5 fejl (BDD-dekodning)
- Man skal være snedig!

# Fejldetektion

- Fejldetektion er noget enklere
- Det kan fx gøres ved at kode de  $K$  informationsbit igen og se om det passer med det modtagne
- Det kræver jo at man kender dem, så som regel er fejldetekterende koder **systematiske**, dvs. at de  $K$  informationsbit viser sig i kodeordet (som de første  $K$  positioner som vi også antydede på tidligere slides)
- Det er klart at detektionen fejler hvis fejlene lige præcis falder på de positioner hvor to kodeord adskiller sig

# Litteratur

- Artikel fra bladet Teleteknik i 1992 – uploaded
- J. Justesen & Tom Høholdt: "A Course in Error-Correcting Codes, Second edition". European Mathematical Society Publishing, 2017. ISBN 978-3-03719-179-8. Anvendes i DTU kursus 01405
- Shu Lin & Daniel J. Costello, Jr.: "Error Control Coding: Fundamentals and Applications, 2nd Edition". Prentice-Hall 2004. ISBN 0-13-283796

## Kort oversigt over dette afsnit

- Fejl detektion ved simpelt paritetscheck
- Hamming-kode som eksempel på fejlkorrektion
- Introducere begreber
  - Kodning, fejl og dekodning
  - Blokkode, kodeord, N
  - Informationsbit, K, og redundansbit
  - Hastighed
  - Afstand, vægt og minimumsafstand
- Dekodningsmetoder
  - BDD (*bounded distance decoding*)
  - ML (*maximum likelihood*)
  - MAP (*maximum a posteriori*)
- Dekodning er komplettest, mens fejl detektion er ret enkel

## ØVELSE #1: KODER

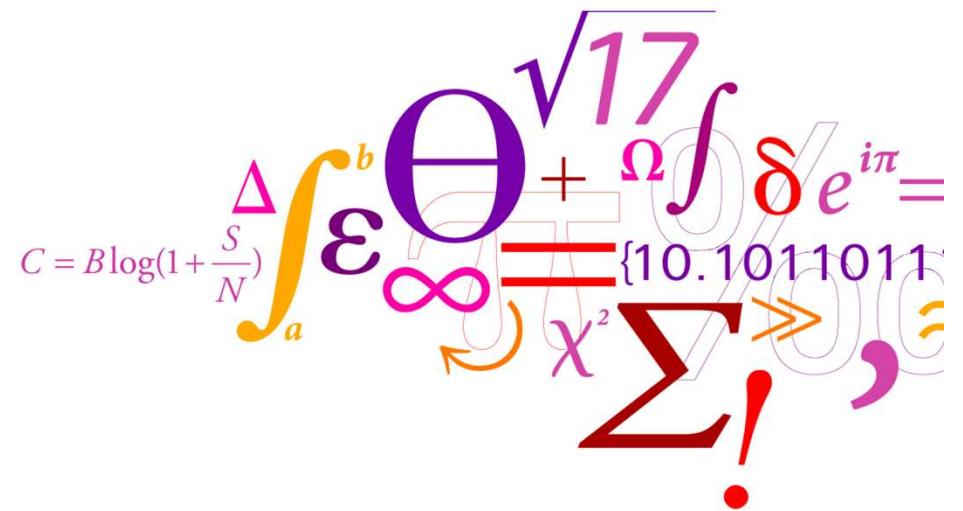
1. 4 sekvenser skal kodes ved hjælp af cirklerne for Hamming-koden
2. Dekod en modtaget sekvens ved at prøve alle 7 muligheder for enkeltfejl (og også ingen fejl)
3. Vis at kodeordene udgøre et vektorrum (med 4 basisvektorer)
4. Vis at kodeordene også er kodeord hvis de skiftes cyklisk
5. Hvor mange indbyrdes afstande er der blandt de 16 kodeord?
6. Og hvilke afstande er mulige?
7. MATLAB:
  1. Lav en  $16 \times 7$  matrix med kodeord
  2. Lav et dekodingsprogram der kan modtage en vektor med 7 reelle tal, og som ved at udregne afstande til alle de 16 kodeord kan beslutte hvilket ord der er nærmest (i almindeligt afstandsmål, såkaldt euklidisk afstand) – en ML-dekoder for Hamming-koden

# Lineære koder

**34220**

Knud J. Larsen

Slides: LIN

$$C = B \log\left(1 + \frac{S}{N}\right)$$
$$\int_a^b \Theta^\sqrt{17} + \Omega \int \delta e^{i\pi} =$$
$$\sum_{\chi^2} \gg,$$


# Fejlkorrigerende og fejldetekterende kodning

I denne del af kurset vil vi se på

- Fejldetection og -korrektion ved hjælp af koder
- Introducere begreber
- Hamming-kode eksempel
- Lineære koder – paritetscheckmatrix, syndromer, dekodning, minimumsvægt, generatormatrix, systematisk kodning
- Fejsandsynligheder ved anvendelse af blokkoder
- Foldningskoder og deres dekodning (Viterbi-algoritmen)
- Endelige legemer og koder over disse
- Cykliske koder – polynomiumsbeskrivelse
- Reed-Solomon koder og deres dekodning
- BCH-koder
- Simulering af koder på virkelige kanaler
- Sammensætning af simplere koder til stærke systemer

# Modulo 2 regning

- At kræve et lige antal 1'er i en vektor af 0 og 1 (eller inden for en cirkel) er det samme som
  - Summen af alle elementer er lige, da 0-rne ikke bidrager
  - Resten ved division af summen med 2 bliver 0
  - At regne udelukkende med rester kaldes modulo-regning eller **restklasseregning**, her **modulo (mod) 2**
- Ved modulo 2 er der kun 2 mulige rester: 0 og 1
- Vi kan opstille en additions-( $\oplus$ ) og en multiplikationstabel ( $\cdot$ ):

$\oplus$	0	1
0	0	1
1	1	0

$\cdot$	0	1
0	0	0
1	0	1

- $1 \oplus 1 = 0$  så der er ikke forskel på + og – når man regner mod 2

# Lineære fejlkorrigende koder

- Eksempel: (7,4)-Hamming koden

- Informationsbit:  $x_0, x_1, x_2, x_3$

indkodes uforandret som

$$c_0, c_1, c_2, c_3$$

(systematisk kodning)

- Paritetsbit i redundansen skal findes ved et lige antal 1'er i hver cirkel, men når der regnes modulo 2 er det det samme som at kræve at summen af alle elementer inden for en cirkel skal være 0,
  - fx  $x_0 + x_1 + x_2 + c_4 = 0$ , og for de tre cirkler får vi

$$c_4 = x_0 + x_1 + x_2, \quad c_5 = x_1 + x_2 + x_3, \quad c_6 = x_0 + x_1 + x_3$$

- Transmitteret kodeord bliver  $(c_0, c_1, c_2, c_3, c_4, c_5, c_6)$

## Lineære fejlkorrigerende koder

- Eksempel: (7,4)-Hamming koden fortsat
  - Modtageren skal checke om der er overensstemmelse i de tre cirkler og tidligere skabte vi overensstemmelse ved at “vippe” bittene én ad gangen
  - men nu er vi smartere:
  - Modtageren udregner *syndromer*  $s_0, s_1, s_2$  som viser sig **kun at afhænge af de fejl** der er, ikke det transmittede ord:

$y_i$  = modtagne bit,  $y_i = c_i + e_i$ , hvor  $e_i$  er (mulige) fejl,  
 $e_i = 1$  eller  $e_i = 0$  for ingen fejl i position  $i$

$$s_0 = y_0 + y_1 + y_2 + y_4 = e_0 + e_1 + e_2 + e_4$$

$$s_1 = y_1 + y_2 + y_3 + y_5 = e_1 + e_2 + e_3 + e_5$$

$$s_2 = y_0 + y_1 + y_3 + y_6 = e_0 + e_1 + e_3 + e_6$$

# Lineære fejlkorrigerende koder

- Eksempel: (7,4)-Hamming koden fortsat
  - Fejlfri transmission, hvis  $s_0 = s_1 = s_2 = 0$  (kunne dog være et ondt fejlmønster)
  - Ellers, antager man 1 fejl som bestemmes af tabellen

$s_0$	$s_1$	$s_2$	$e_0$	$e_1$	$e_2$	$e_3$	$e_4$	$e_5$	$e_6$
0	0	1	0	0	0	0	0	0	1
0	1	0	0	0	0	0	0	1	0
0	1	1	0	0	0	1	0	0	0
1	0	0	0	0	0	0	1	0	0
1	0	1	1	0	0	0	0	0	0
1	1	0	0	0	1	0	0	0	0
1	1	1	0	1	0	0	0	0	0

- Det kan checkes at det giver præcis samme resultat som da vi prøvede os frem med at vippe en bit ad gangen. Syndromerne fortæller hvilke cirkler der ikke passer for de enkelte fejl

# Lineære fejlkorrigende koder

- Alle udregninger i det følgende sker modulo 2 (De gælder dog også i ethvert såkaldt legeme,  $F(q)$ , men mere herom senere. Modulo 2 regning er  $F(2)$ )
- Som tidligere
  - En kode består af et antal kodeord,  $\mathbf{c} = (c_0 c_1 \dots c_{N-1})$ ,  $c_i \in F(q)$
  - $N$  = længde af kodeord
  - $K$  = antal informationssymboler,  $\mathbf{x} = (x_0 x_1 \dots x_{K-1})$ ,  $x_j \in F(q)$
- De 3 paritetsligninger for (7,4) Hammingkoden  
(med  $c_i = x_i$  for  $0 \leq i \leq 3$ )

$$c_4 = c_0 + c_1 + c_2 , \quad c_5 = c_1 + c_2 + c_3 , \quad c_6 = c_0 + c_1 + c_3$$

kan også skrives som

$$c_0 + c_1 + c_2 + c_4 = 0$$

$$c_1 + c_2 + c_3 + c_5 = 0$$

$$c_0 + c_1 + c_3 + c_6 = 0$$

Alle  $\mathbf{c}$  der opfylder disse betingelser er altså kodeord

# Lineære fejlkorrigerende koder

Alle  $\mathbf{c}$  der opfylder disse er kodeord

$$c_0 + c_1 + c_2 + c_4 = 0$$

$$c_1 + c_2 + c_3 + c_5 = 0$$

$$c_0 + c_1 + c_3 + c_6 = 0$$

Det kan udtrykkes lidt mere elegant med en matrix:

$$[c_0, c_1, c_2, c_3, c_4, c_5, c_6] \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = [0, 0, 0]$$

Altså

$$\mathbf{cH} = \mathbf{0}$$

$\mathbf{H}$  kaldes en **paritetscheckmatrix** for koden

# Lineære fejlkorrigerende koder

Generel definition, paritetscheckmatrix

- Vi kan nu nemt definere lineære fejlkorrigerende koder:  
For en (lineær) blokkode findes der en  **$N \times (N-K)$  fast matrix  $\mathbf{H}$**  for hvilke

$$\mathbf{c}\mathbf{H} = \mathbf{0}$$

gælder for ethvert kodeord  **$\mathbf{c}$**  i koden.  **$\mathbf{H}$**  kaldes en **paritetscheckmatrice** for koden. (Der er flere mulige for den samme kode)

- At sådanne koder bliver **lineære** er nemt at vise:

$$\mathbf{a}, \mathbf{b} \in \text{kode}, f, g \text{ konstanter} \Rightarrow$$

$$\mathbf{0} = f\mathbf{a}\mathbf{H} + g\mathbf{b}\mathbf{H} = (f\mathbf{a} + g\mathbf{b})\mathbf{H} \Rightarrow (f\mathbf{a} + g\mathbf{b}) \in \text{kode}$$

# Lineære fejlkorrigerende koder



## Afstande og vægte

- En meget vigtig egenskab ved lineære koder er at afstande mellem kodeord ”nemt” kan findes:
- Som i sidste del er **afstanden** mellem to kodeord, **a** og **b**,  
 $d(\mathbf{a}, \mathbf{b})$  = antallet af symboler der er forskellige og antallet af ikke-0 symboler i et kodeord som **vægten** af kodeordet,  $w(\mathbf{a})$
- $d(\mathbf{a}, \mathbf{b}) = w(\mathbf{a} - \mathbf{b})$  og da  $\mathbf{a} - \mathbf{b} \in \text{koden}$  (da den er lineær), kan alle afstande **findes som vægte** for et eller andet kodeord, så det er meget simplere at finde fx minimumsafstanden d
- For Hamming (7,4) kan man se at der findes 0, 3, 4 og 7 som vægte så det er nemt at se at  $d = 3$  som tidligere skrevet.

## Syndromer

- $\mathbf{H}$  kan bruges til udregne **syndromer** for en modtaget vektor  $\mathbf{y} = \mathbf{c} + \mathbf{e}$  hvor  $\mathbf{e}$  er fejlmønsteret

$$\mathbf{s} = \mathbf{yH} = (\mathbf{c}+\mathbf{e})\mathbf{H} = \mathbf{cH} + \mathbf{eH} = \mathbf{0} + \mathbf{eH} = \mathbf{eH}$$

- dvs. syndromer **afhænger kun af fejlene** for en lineær kode (som vi så for Hamming (7,4)-koden)

# Lineære fejlkorrigerende koder

## Fejlkorrektion på basis af syndromer

- En **fejlkorrektionsalgoritme** finder den mest sandsynlige løsning **e** for

$$\mathbf{s} = \mathbf{eH}$$

- Det giver  $N-K$  ligninger for de ukendte fejlpositioner, dvs.  $N$  ubekendte:  $e_i = 1$  for fejl og  $e_i = 0$  for ingen fejl i position  $i$ .
- Generelt er der altså  $2^K$  løsninger, men en fornuftig algoritme finder den **løsning med færrest fejl**
- Som regel er algoritmerne BDD som retter op til

$$\frac{d-1}{2} \quad \text{fejl}$$

- fx 1 for Hamming

# Lineære fejlkorrigerende koder

## Generatormatrix

- For en lineær  $(N,K)$  kode er der også en  $K \times N$  matrix  $\mathbf{G}$  som kan bruges direkte til at finde kodeordet  $\mathbf{c}$  for de  $K$  informationssymboler  $\mathbf{x}$

$$\mathbf{c} = \mathbf{x}\mathbf{G}$$

$\mathbf{G}$  kaldes en **generatormatrix** for koden.

- En **kode** skal forstås som det **vektorrum** de mulige  $\mathbf{c}$  udgør
- Der vil således være mange mulige generatormatricer for en given kode, idet rækkerne i en sådan blot skal være basis for vektorrummet og rækkeoperationer vil kunne frembringe nye baser. Hver mulig  $\mathbf{G}$  specificerer en afbildning af informationsvektorer  $\mathbf{x}$  på kodeord  $\mathbf{c}$ .

# Lineære fejlkorrigerende koder

## Systematisk kodning

- En speciel interessant version af  $\mathbf{G}$  indeholder en  $K \times K$  enhedsmatrix,  $\mathbf{I}_K$  i de første  $K$  søjler. I det tilfælde kan  $\mathbf{x}$  ses som de første  $K$  positioner i  $\mathbf{c}$  (som i Hammingkoden i introduktionen). De sidste  $N - K$  positioner er **redundansen** og kaldes også paritetscheck.
- Kodning med  $\mathbf{G}$  på denne form kaldes **systematisk kodning** da man kan se  $\mathbf{x}$  direkte.
- $\mathbf{G}$  kan altid bringes på en systematisk form, men det er ikke sikkert at de "systematiske søjler" står som den  $K$  første.

# Lineære fejlkorrigerende koder

Eksempel Hamming (7,4)

Alle **c** der opfylder disse er kodeord

$$c_0 + c_1 + c_2 + c_4 = 0$$

$$c_1 + c_2 + c_3 + c_5 = 0$$

$$c_0 + c_1 + c_3 + c_6 = 0$$

Udregning med en systematisk generatormatrix  $\mathbf{G} = [\mathbf{I}_4 \ \mathbf{P}]$ :

$$\begin{bmatrix} [c_0, c_1, c_2, c_3] & \left[ \begin{array}{cccc} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{array} \right] & \left[ \begin{array}{ccc} 1 & 0 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \end{array} \right] \end{bmatrix} = \begin{bmatrix} [c_0, c_1, c_2, c_3, c_0 + c_1 + c_2, c_1 + c_2 + c_3, c_0 + c_1 + c_3] \end{bmatrix}$$

Helt som vi så før!

# Lineære fejlkorrigende koder

Hvordan er sammenhængen mellem  $\mathbf{G}$  og  $\mathbf{H}$ ?

$$\mathbf{0} = \mathbf{cH} = \mathbf{xGH} \Rightarrow \mathbf{GH} = \mathbf{0}$$

Hvis  $\mathbf{G}$  er på sin systematiske form er det nemt at finde en  $\mathbf{H}$  der opfylder dette:

$$\mathbf{G} = [\mathbf{I}_K \quad \mathbf{P}] \Rightarrow \mathbf{H} = \begin{bmatrix} -\mathbf{P} \\ \mathbf{I}_{N-K} \end{bmatrix} \text{ da } [\mathbf{I}_K \quad \mathbf{P}] \begin{bmatrix} -\mathbf{P} \\ \mathbf{I}_{N-K} \end{bmatrix} = -\mathbf{I}_K \mathbf{P} + \mathbf{P} \mathbf{I}_{N-K} = -\mathbf{P} + \mathbf{P} = \mathbf{0}$$

I (7,4)-eksemplet havde vi netop denne sammenhæng:

$$\mathbf{G} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix} \quad \mathbf{H} = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Da der regnes i  $F(2)$  betyder fortegnet ikke noget. Det er tilfældet for alle  $F(2^m)$  legemer.

Hvis de systematiske sjøller ikke star først I  $\mathbf{G}$  må man bytte om og byte tilbage for at få den rigtige  $\mathbf{H}$ .

# Fejlkorrigerende og fejldetekterende kodning

Kort oversigt over dette afsnit

- Modulo 2 regning
- Hamming-koden er lineær
- Syndromer afhænger kun af fejl for en lineær kode
- Paritetscheckmatrix,  $\mathbf{H}$
- Afstand,  $d(\mathbf{a}, \mathbf{b})$ , og vægt,  $w(\mathbf{a})$ , for lineære koder
- Syndromdekodning
- Dekodningsmetoder, som regel BDD (*bounded distance decoding*)
- Dekodning er stadig vanskeligt
- Generatormatrix,  $\mathbf{G}$ , systematisk og ikke systematisk kodning
- Sammenhæng mellem  $\mathbf{G}$  og  $\mathbf{H}$

## ØVELSE #2: LINEÆRE KODER

1. Vis hvordan Hamming-koderne (15,11), (31,26) osv. defineres v.hj.a. paritetscheckmatricen
2. Foretag dekodning af 3 modtagne sekvenser ved syndromberegning
3. Undersøg den udvidede Hamming-kode (8,4)
4. MATLAB (et temmeligt stort program til slut):
  1. En lidt indviklet generatormatrix for en (32,16)-kode med  $d = 8$  er givet
  2. Find paritetscheckmatricen
  3. Lav et dekodingsprogram der kan modtage en vektor med 32 bit og ved hjælp af en tabel over syndromer kan dekode op til 3 fejl samt indikere hvis der er flere fejl (en BDD-dekoder)
  4. Frembring et tilfældigt fejlmønster for 100000 anvendelser af denne kode, dvs. en vektor af længde 3200000 med 1% fejl. Tæl antallet af 32 bits blokke der ikke kan dekodes og hvor mange der dekodes forkert.
  5. Modificer programmet til en ML-dekoder ved at fyldе fejlmønstre i de pladser hvor der ikke kunne dekodes før. Start med 4 fejl osv.
  6. Anvend denne dekoder på fejlmønsteret fra punkt 4

# Eksempler på lineære koder

## Sandsynlighedsregning og koder

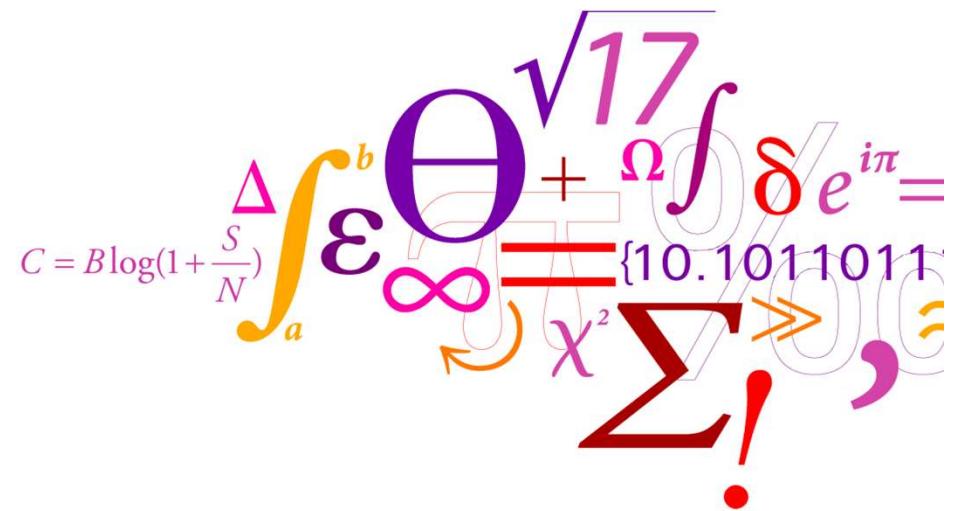
34220

Knud J. Larsen

Slides: EKS

DTU Fotonik  
Department of Photonics Engineering

---



# Fejlkorrigerende og fejldetekterende kodning

I denne del af kurset vil vi se på

- Fejldetection og -korrektion ved hjælp af koder
- Introducere begreber
- Hamming-kode eksempel
- Lineære koder – eksempler
- Fejsandsynligheder ved anvendelse af blokkoder
- Foldningskoder og deres dekodning (Viterbi-algoritmen)
- Endelige legemer og koder over disse
- Cykliske koder – polynomiumsbeskrivelse
- Reed-Solomon koder og deres dekodning
- BCH-koder
- Simulering af koder på virkelige kanaler
- Sammensætning af simplere koder til stærke systemer

## Simple lineære blokkoder

- **Hamming-koder:** Hvis vi skal kunne rette 1 fejl kræver det at syndromer for 1 fejl er forskellige:  $\mathbf{s} = \mathbf{eH}$  og da e så udpeger én række i H skal rækkerne være forskellige
- Ved at vælge alle  $2^m - 1$  mønstre  $\neq \mathbf{0}$  med m bit kan man danne en paritetscheckmatrix for en  $(N, K) = (2^m - 1, 2^m - m - 1)$ -kode. Den kan så rette 1 fejl og har minimumsafstand  $d = 3$ . Anbringes rækker med én 1'er nederst som en enhedsmatrix fås en systematisk kodning
- Eksempler  $(3,1)$  (trivial),  $(7,4)$ ,  $(15,11)$ ,  $(31,26)$  ....

## Simple lineære blokkoder

- **Udvidede Hamming**-koder: Tilføjes en paritetsbit til Hamming-koderne får de  $d = 4$  (da der tilføjes 1 til alle vægt 3 kodeord. Man får så  $(N,K) = (2^m, 2^m-m-1)$ )
- **G** fås umiddelbart ved at tilføje paritetsbit i hver række. En **H** kan dannes ud fra denne eller man danne en **H** ud fra Hammingkodens **H** ved først at tilføje en række med  $m$  0'er og så en søjle med  $2^m$  1'er
- Eksempel (8,4) udvidet Hamming kode

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

## Simple lineære blokkoder

- Man kan også fjerne alle kodeord med ulige vægt og få  $(N, K) = (2^m - 1, 2^m - m - 2)$  som stadig bliver lineær (**a** med lige vægt + **b** med lige vægt giver altid **a+b** med lige vægt). Dette kaldes den **udtyndede Hamming-kode (expurgated)**
- De udvidede og udtyndede Hamming-koder kan altså rette 1 fejl, men kan samtidigt bruges til at detektere 2 fejl da sådanne ligger midt i mellem kodeordene
- Den udtyndede  $(7, 3)$  Hamming kode:

$$\mathbf{G} = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 & 1 \end{bmatrix}$$

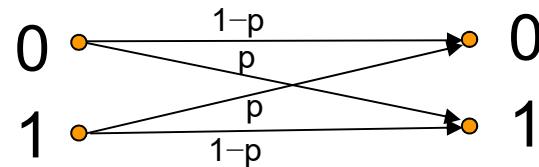
$$\mathbf{H} = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

## Andre lineære blokkoder

- **BCH** koder: Fra Hamming-kode kunne man lære at det koster  $m$  bit i redundans at rette 1 fejl. Man kunne så få den gode idé at det vil koste  $T \cdot m$  bit at rette  $T$  fejl. BCH-koder er faktisk sådanne koder med  $(N, K) = (2^m - 1, 2^m - T \cdot m - 1)$  som retter  $T$  fejl og har  $d = 2T + 1$ , men som vi skal se senere er det ikke helt let at opstille en **H** for disse, da de ekstra  $m$  søjler der skal tilføjes for hver ekstra fejl skal være uafhængige af de foregående (K passer ikke helt for lidt større  $T$ , da man får nogle muligheder for at rette fejl "gratis")
- **Reed-Solomon** koder (**RS**) er koder med symboler fra det såkaldte legemer  $F(q)$  med  $q$  elementer (i praktiske anvendelser  $F(2^m)$ ). De har  $N = q - 1$ ,  $K = N - 2T$  og  $d = N - K + 1 = 2T + 1$ . Bemærk at der nu er  $q - 1$  muligheder for en fejlværdi, ikke bare 1.
- Vi ser nærmere på BCH og RS koder senere

# Fejsandsynlighed – fejlkorrektion

- Vi vil gerne være i stand til at udregne sandsynligheden for at en BDD-algoritme ikke kan dekode en modtaget sekvens
- En BDD-algoritme kan som tidligere sagt rette op til et vist antal,  $T$ , fejl, som regel  $(d-1)/2$
- Vi antager at kommunikationen foregår via en såkaldt **binær symmetrisk kanal, BSC**, som vist



- Det skal forstås således, at for hver transmitteret bit er der sandsynligheden  $p$  for at et 0 bliver til 1 eller omvendt. Vi kalder **p bitfejsandsynligheden**
- Vi antager også at de enkelte fejl er uafhængige af hinanden så sandsynlighederne kan multipliceres, fx er der sandsynligheden  $(1-p)^N$  for at der ikke er nogen fejl i  $N$  bit.

# Fejsandsynlighed - kombinatorik

- Det er lidt mere indviklet at finde chancen for præcis  $j$  fejl blandt de  $N$  mulige positioner
- Her må vi gøre til nogle kombinatoriske overvejelser:
- På hvor mange mulige måder,  $P_N$ , kan  $N$  forskellige tal ordnes?

$$P_N = N(N-1)(N-2) \cdot \dots \cdot 2 \cdot 1 = N!$$

- Dette kaldes permutationer
- Hvis de  $j$  første positioner i permutationerne betyder fejl, findes der  $P_j$  permutationer som starter med disse, og da fejlene er uafhængige af hinanden kan rækkefølgen være ligegyldig. Det gælder også om de  $N-j$  positioner uden fejl som findes i  $P_{N-j}$ , permutationer. Antallet af forskellige fejlmønstre er da

$$\frac{P_N}{P_j P_{N-j}} = \frac{N!}{j!(N-j)!} = \binom{N}{j}$$

den såkaldte binomialkoefficient. I MATLAB kan den udregnes med `nchoosek(N, j)`.

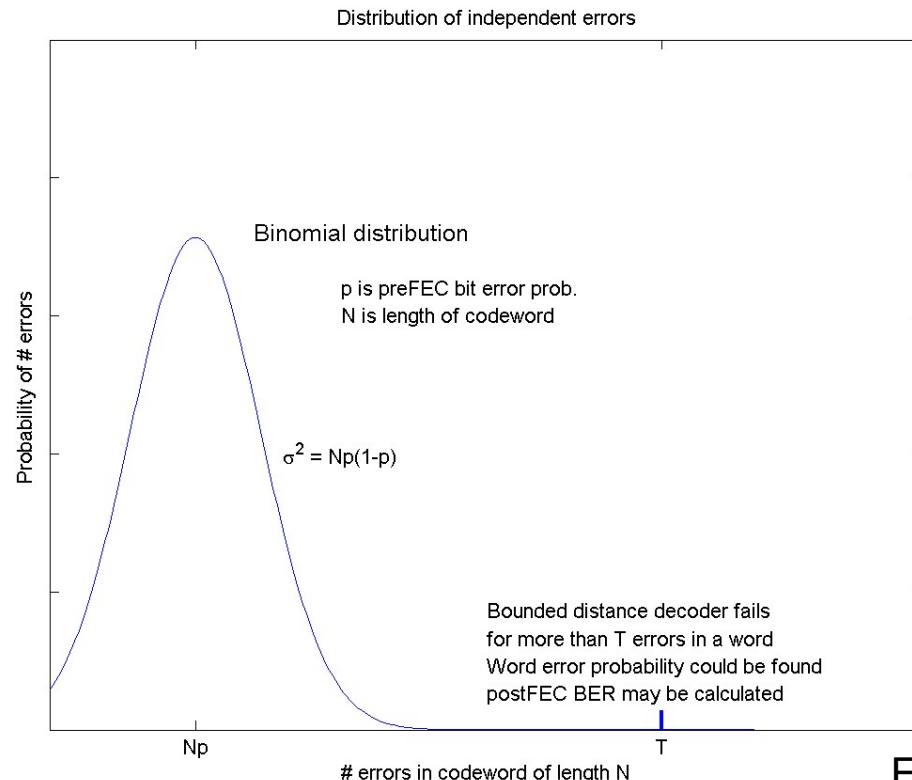
# Fejlsandsynlighed – fejlkorrektion

Hvis koden kan rette  $T$  fejl bliver sandsynligheden for at der ikke kan rettes eller at der rettes noget forkert

$$P_e = 1 - \sum_{j=0}^T \binom{N}{j} p^j (1-p)^{N-j} \approx \binom{N}{T+1} p^{T+1} (1-p)^{N-T-1} \quad \text{hvor } \binom{N}{j} = \frac{N!}{j!(N-j)!}$$

I gennemsnit vil der være omkring  $pN$  fejl blandt  $N$  bit, men der er en betydelig risiko for at der er en del flere, som kurven viser.

Det må man tage hensyn til hvis man skal have en bestemt fejlsandsynlighed



# Fejsandsynlighed - fejldetektion

Når koder anvendes til **fejldetektion**, markeres der fejl så snart der ikke modtages et kodeord. Vi definerer  $P_d$  som sandsynligheden for at der alligevel er fejl.

Umiddelbart vil alle fejlmønstre med mindre end  $d - 1$  fejl blive detekterede, så sandsynligheden for udetekterede fejl er højst

$$P_d = 1 - \sum_{j=0}^{d_{\min}-1} \binom{N}{j} p^j (1-p)^{N-j}$$

Men mange flere kan detekteres, da kun de fejlmønstre der helt præcist er kodeord ikke detekteres, for lineariteten betyder at  $\mathbf{y} = \mathbf{c} + \mathbf{e} = \mathbf{c} + \mathbf{c}' = \mathbf{c}''$  hvor  $\mathbf{c}$ ,  $\mathbf{c}'$  og  $\mathbf{c}''$  er kodeord. Vi kan så finde et mere præcist udtryk for  $P_d$ :

## Fejsandsynlighed - fejldetektion

Hvis  $A_w$  angiver antallet af kodeord med vægt  $w$  (den såkaldte vægtfordeling som kendes for en del koder), finder man

$$P_d = \sum_{w=1}^N A_w p^w (1-p)^{N-w} = (1-p)^N \sum_{w=1}^N A_w \left(\frac{p}{1-p}\right)^w$$

Ofte kendes vægtfordelingen som et polynomium  $A(z)$ :

$$A(z) = \sum_{w=0}^N A_w z^w$$

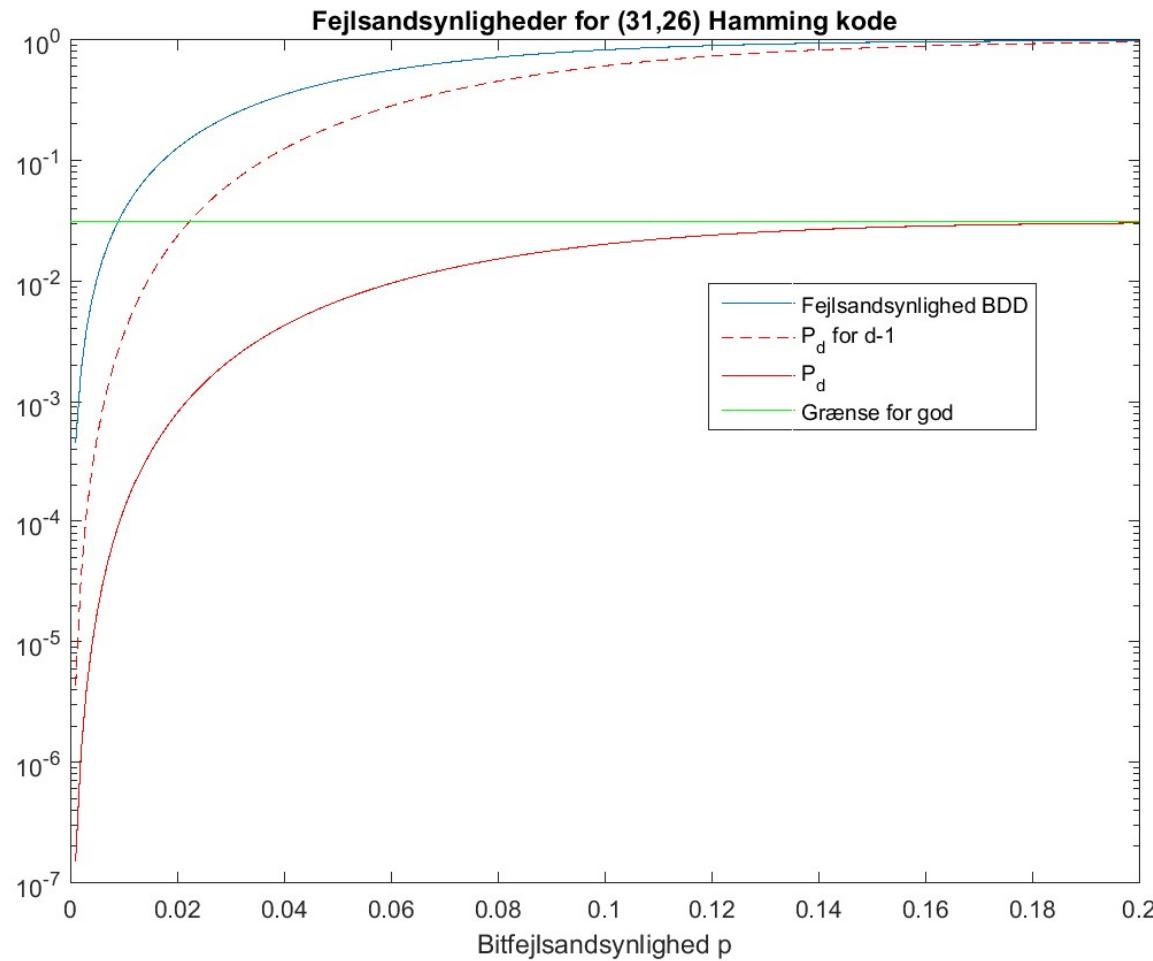
hvor  $A_0 = 1$  (0-ordet) og  $A_w = 0$  for  $w < d$ . Fx.  $A(z)=1+7z^3+7z^4+z^7$  for vores (7,4) Hammingkode. Vi får da:

$$P_d = (1-p)^N \left( A\left(\frac{p}{1-p}\right) - 1 \right)$$

Man siger at en fejldetekterende kode er **god**, hvis  $P_d \leq 2^{-(N-K)}$  for  $0 \leq p \leq \frac{1}{2}$ , hvor  $2^{-(N-K)}$  er sandsynligheden for at et helt tilfældigt fejlmønster med sandsynlighed  $\frac{1}{2}$  for 1 accepteres af en tilfældig kode med redundans  $N-K$ .

# Fejsandsynlighed - eksempel

En (31,26) Hamming kode kan rette 1 fejl og detektere alle ikke kodeord:



# Fejlkorrigerende og fejldetekterende kodning

Kort oversigt over dette afsnit

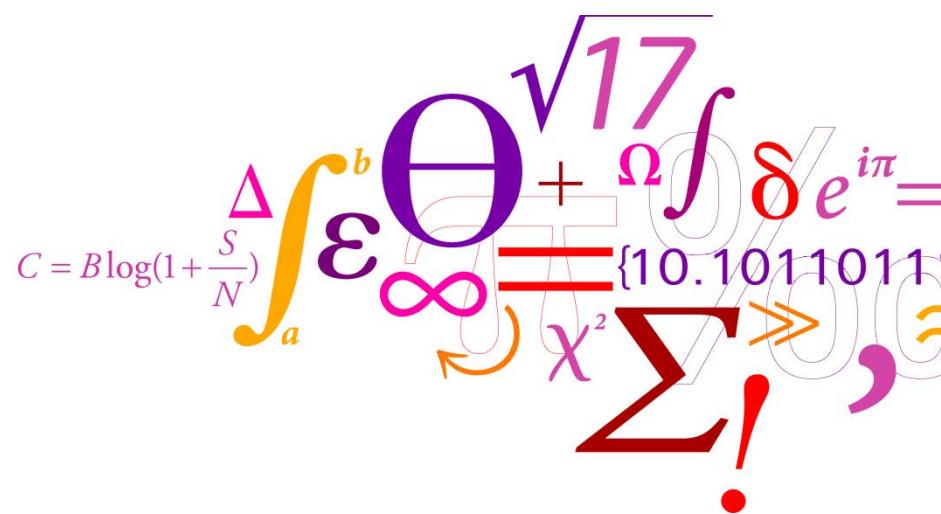
- Eksempler på lineære koder
- Hamming, Udvidet Hamming, Uddyndet Hamming
- BCH, RS
- Binær symmetrisk kanal
- Lidt kombinatorik
- Sandsynlighed for fejl ved BDD dekodning af blokkode
- Sandsynlighed for udetekterede fejl i en blokkode anvendt til fejldetektion

# Foldningskoder

34220

Knud J. Larsen

Slides: FOL

$$C = B \log\left(1 + \frac{S}{N}\right)$$

$$\int_a^b \Theta^{\sqrt{17}} + \Omega \int \delta e^{i\pi} =$$
$$\sum_{!}$$

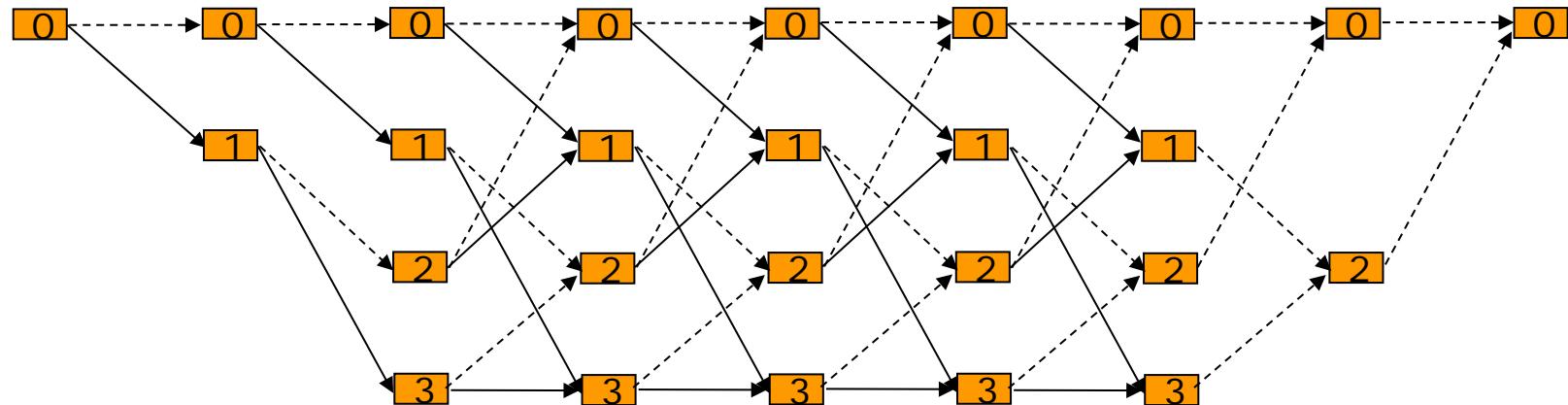
# Fejldkorrigende og fejlddetekterende kodning

I denne del af kurset vil vi se på

- Fejlddetektion og -korrektion ved hjælp af koder
- Introducere begreber
- Hamming-kode eksempel
- Lineære koder – eksempler
- Fejlsandsynligheder ved anvendelse af blokkoder
- **Foldningskoder** og deres dekodning (Viterbi-algoritmen)
- Endelige legemer og koder over disse
- Cykliske koder – polynomiumsbeskrivelse
- Reed-Solomon koder og deres dekodning
- BCH-koder
- Simulering af koder på virkelige kanaler
- Sammensætning af simplere koder til stærke systemer

# Foldningskoder – koder ved hjælp af en graf

En alternativ metode til at danne kodeord hvor forskelligheden opnås ved at kodeordene beskrives som veje gennem en graf som nedenstående



Trellisgraf

# Definition af foldningskode

Lad den binære information være  $\mathbf{u} = (u_0, u_1, u_2, \dots)$

(i princippet uendelig, men i praksis er der en eller anden afslutning)

$\mathbf{u}$  foldes med n **generatorer** (også binære):

$$\mathbf{g}^{(0)} = (g_0^{(0)}, g_1^{(0)}, \dots, g_M^{(0)})$$

⋮

$$\mathbf{g}^{(n-1)} = (g_0^{(n-1)}, g_1^{(n-1)}, \dots, g_M^{(n-1)})$$

hvor mindst én af  $g_M^{(i)}$  er 1. Disse n foldninger udregnes modulo 2 (dvs. i  $F(2)$ ) og giver n sekvenser  $\mathbf{v}^{(0)}, \mathbf{v}^{(1)}, \dots, \mathbf{v}^{(n-1)}$ :

$$v_j^{(0)} = \sum_{i=0}^M g_i^{(0)} u_{j-i} = g_0^{(0)} u_j + g_1^{(0)} u_{j-1} + \dots + g_M^{(0)} u_{j-M}$$

⋮

$$v_j^{(n-1)} = \sum_{i=0}^M g_i^{(n-1)} u_{j-i} = g_0^{(n-1)} u_j + g_1^{(n-1)} u_{j-1} + \dots + g_M^{(n-1)} u_{j-M}$$

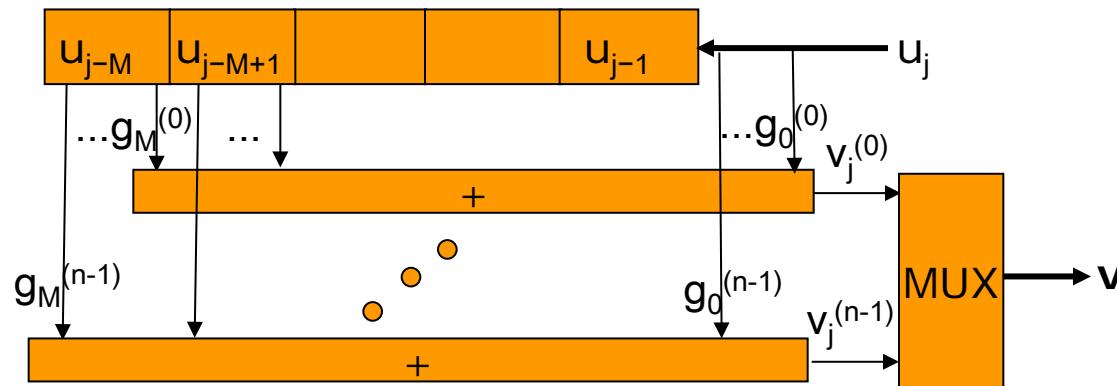
# Definition af foldningskode

De  $n$  sekvenser sættes sammen til én bitstrøm:

$$\mathbf{v} = (v_0^{(0)} v_0^{(1)} \dots v_0^{(n-1)}, v_1^{(0)} v_1^{(1)} \dots v_1^{(n-1)}, v_2^{(0)} v_2^{(1)} \dots v_2^{(n-1)}, \dots)$$

Vi har brugt et punktum (.) til at skille  $n$ -blokkene.

Figuren viser et kredsløb med skifteredistre der giver en simpel implementering i hardware. (Registrene skifter mod venstre! g-koefficienterne angiver en forbindelse og der regnes modulo 2)



Da der er  $n$  udgangsbit,  $v_j^{(0)}, v_j^{(1)}, \dots, v_j^{(n-1)}$  for hver indgangsbit,  $x_j$ , er kodens hastighed  $R=1/n$

# Definition af foldningskode

M kaldes kodens **hukommelse** (mindst én af  $g_M^{(i)} = 1$ ) og det er klart at M+1 informationsbit påvirker den nuværende blok (og heraf er M "gamle" i hukommelsen).

Dette giver umiddelbart en beskrivelse som en sekvensmaskine (*finite state machine*):

Tilstand:  $(u_{j-M} u_{j-M+1} \dots u_{j-1}) = s$  kan fortolkes som heltal:

$$s = u_{j-M} 2^{M-1} + u_{j-M+1} 2^{M-2} + \dots + u_{j-2} 2 + u_{j-1}$$

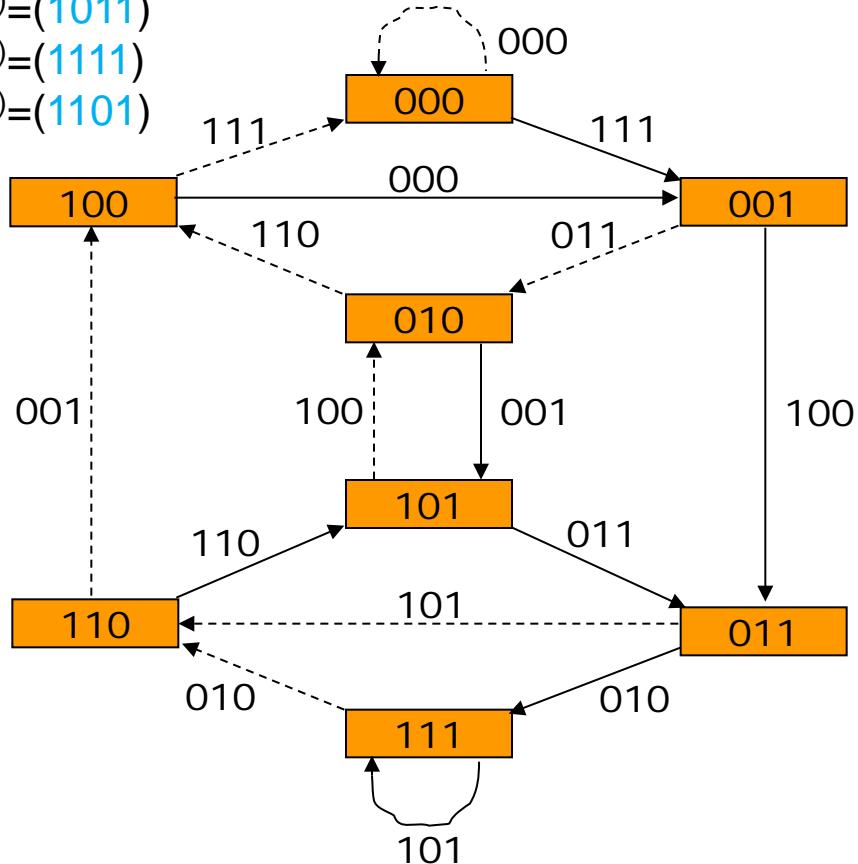
Næste tilstand for  $u_j = 0$ :  $(u_{j-M+1} \dots u_{j-1} 0) = 2s \bmod 2^M$

Næste tilstand for  $u_j = 1$ :  $(u_{j-M+1} \dots u_{j-1} 1) = 2s+1 \bmod 2^M$

Udgangsbit for overgangen til næste tilstand bestemmes af foldningerne

# Eksempel på foldningskode

$$\begin{aligned}g^{(0)} &= (1011) \\g^{(1)} &= (1111) \\g^{(2)} &= (1101)\end{aligned}$$



Tilstandsdiagram

Stiplet er overgangen for  $u = 0$   
Fuld optrukket er for  $u = 1$

Eksempel på udgangsbit

Tilstand=011

$$u=0: v^{(0)} = 1 \cdot 0 + 0 \cdot 1 + 1 \cdot 1 + 1 \cdot 0 = 1$$

$$v^{(1)} = 1 \cdot 0 + 1 \cdot 1 + 1 \cdot 1 + 1 \cdot 0 = 0$$

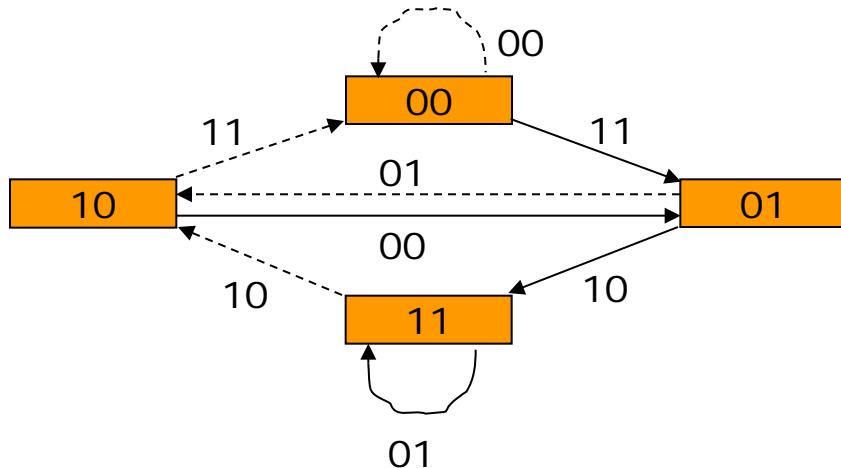
$$v^{(2)} = 1 \cdot 0 + 1 \cdot 1 + 0 \cdot 1 + 1 \cdot 0 = 1$$

Næste tilstand=110

Kodens hastighed er 1/3  
da tre bit dannes for  
hver informationsbit

# Simpere eksempel på foldningskode

$$\begin{aligned}g^{(0)} &= (101) \\g^{(1)} &= (111)\end{aligned}$$



Tilstandsdiagram

Stiplet er overgangen for  $u = 0$

Fuldt optrukket er for  $u = 1$

Eksempel på udgangsbit

Tilstand=**01**

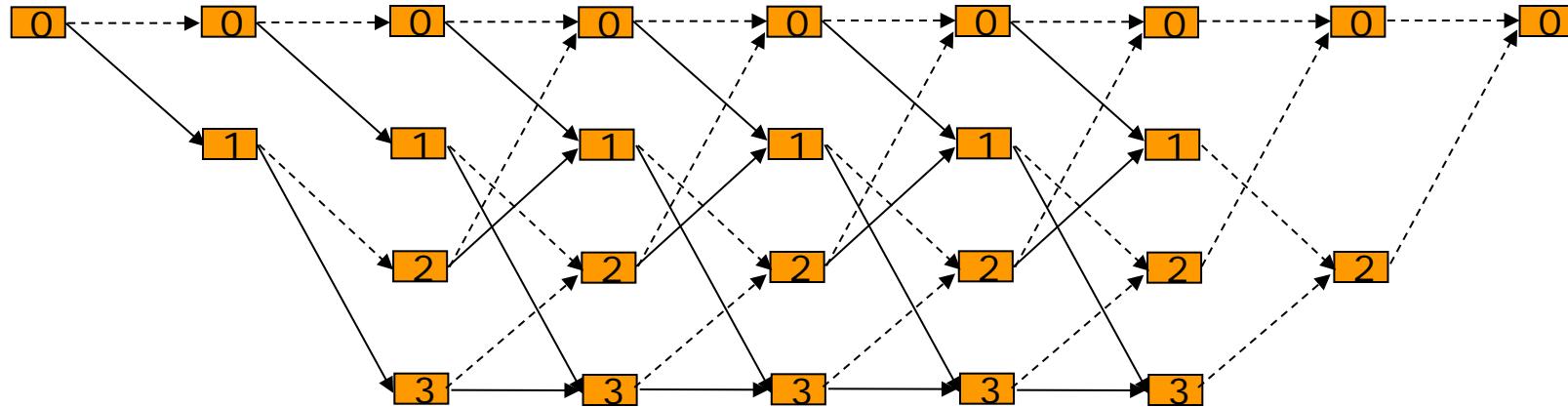
$$u=0: v^{(0)} = 1 \cdot 0 + 0 \cdot 1 + 1 \cdot 0 = 0$$

$$v^{(1)} = 1 \cdot 0 + 1 \cdot 1 + 1 \cdot 0 = 1$$

Næste tilstand=**10**

## Beskrivelse af foldningskode ved hjælp af en graf

Det viser sig smart at beskrive foldningskoden ved hjælp af en graf der viser hvordan tilstanden skifter med tiden og hvad de kodede blokke bliver. Denne graf kaldes et trellis (amerikansk for en type havehegn)

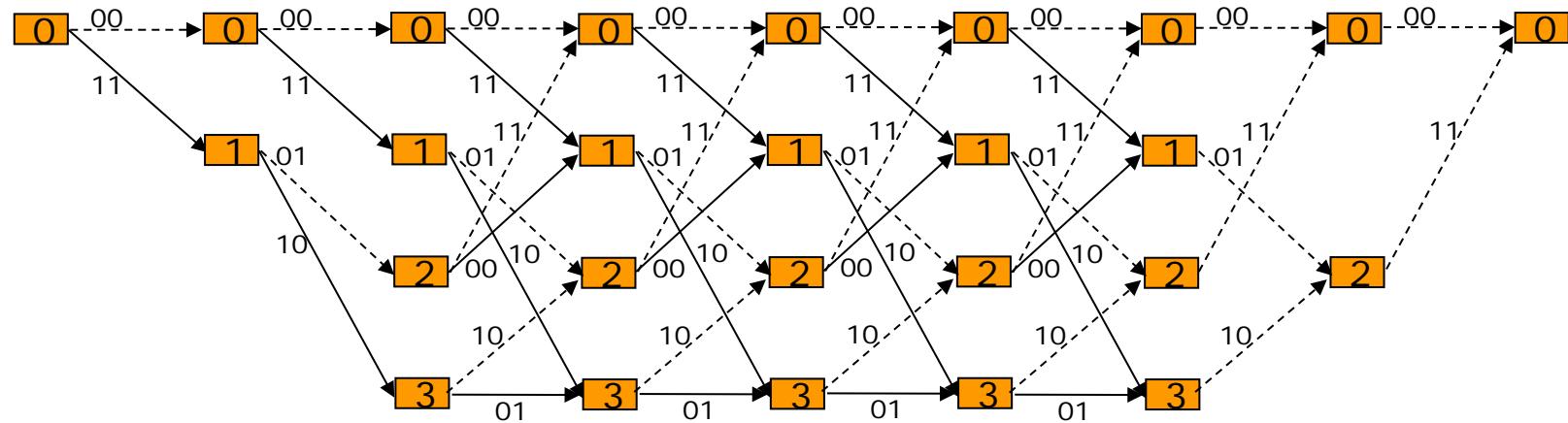


Opbygning af trellisgraf – tilstandsovergange

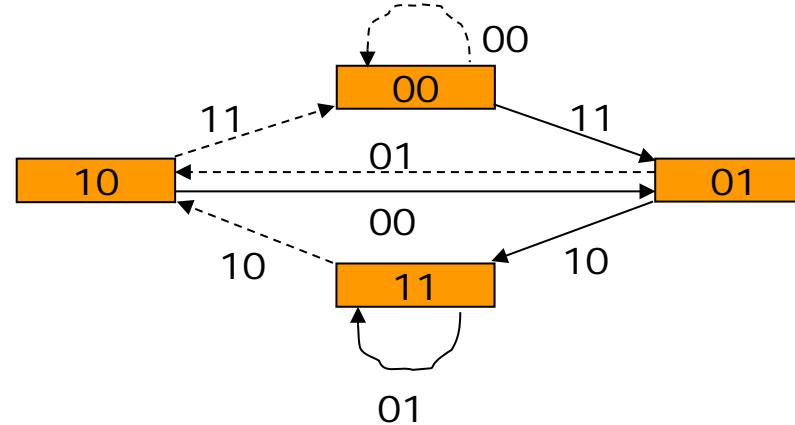
$$\left. \begin{array}{l} \text{Knude nr } p \mapsto 2p \bmod 2^M \quad (0-\text{gren}) \\ \text{Knude nr } p \mapsto (2p+1) \bmod 2^M \quad (1-\text{gren}) \end{array} \right\} \text{Her er } M=2, 2^M=4$$

Grafen er i princippet uendelig, men vi har valgt at afslutte den ved kun at kode 0'er til slut ( $M$  0'er). Dette er for at lægge op til en dekodingsmetode

## Kodning af foldningskoden vist på grafen

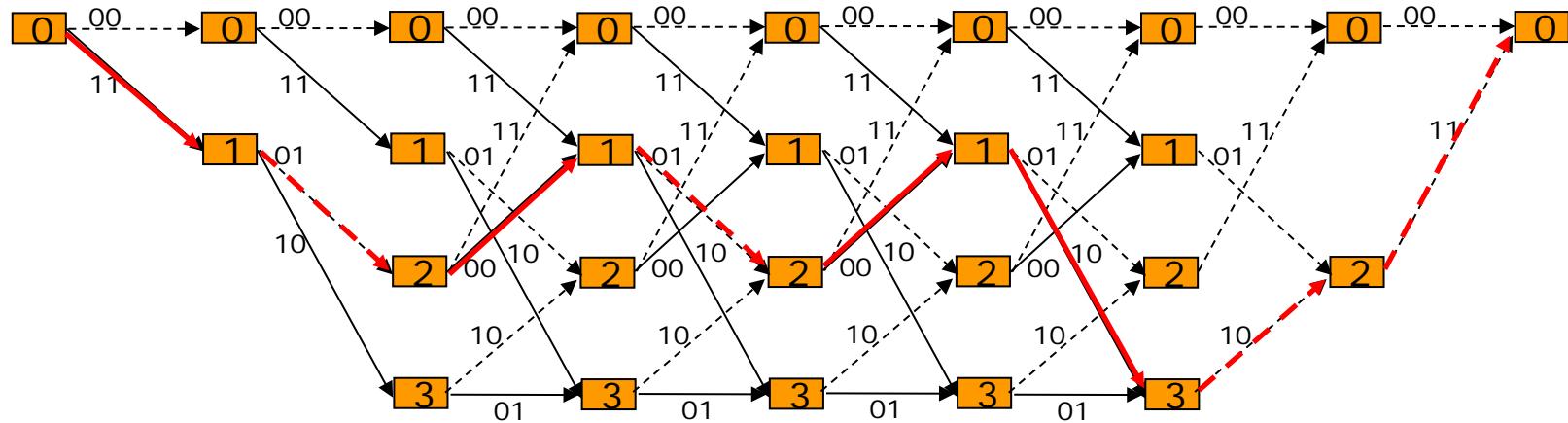


Kodens hastighed er  $\frac{1}{2}$   
da to bit dannes for  
hver informationsbit



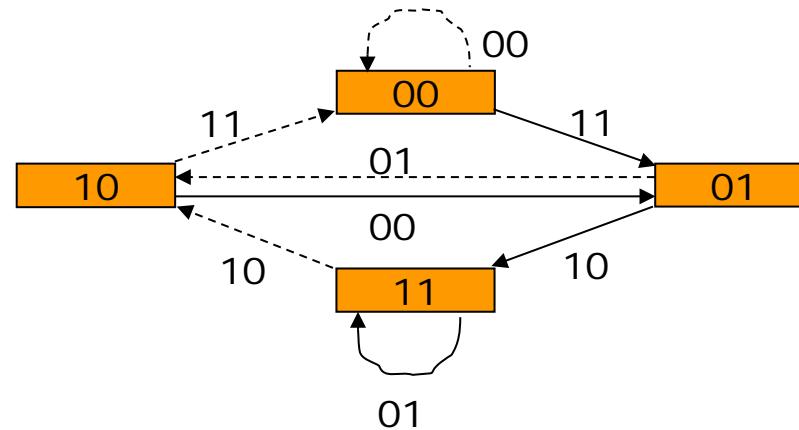
Fra	Til	Inf	Gren
0	0	0	00
0	1	1	11
1	2	0	01
1	3	1	10
2	0	0	11
2	1	1	00
3	2	0	10
3	3	1	01

## Kodning af foldningskoden vist på grafen - eksempel



Kode  $v^{(0)}v^{(1)}$ : 11 01 00 01 00 10 10 11

Inf u: 1 0 1 0 1 1 0 0



Fra	Til	Inf	Gren
0	0	0	00
0	1	1	11
1	2	0	01
1	3	1	10
2	0	0	11
2	1	1	00
3	2	0	10
3	3	1	01

## Foldningskoder - afstande

- Som for blokkoder bliver egenskaberne bestemt af afstande mellem de kodede sekvenser, men det er mere indviklet her da disse kan være af forskellig længde
- Da koden er lineær er det som for blokkoder at afstande og vægte er de samme så vi taler i principippet om vægte
- Den mest betydningsfulde parameter for en foldningskode er den såkaldte  $d_f = \text{fri afstand}$  som defineres som den mindste vægt en vej, der forlader 0-tilstanden ad 1-grenen (dvs.  $u_0 = 1$ ) og afsluttes når den vender tilbage til 0, kan have
- For de to viste eksempler er  $d_f = 10$  ( $n=3$  koden) og 5

## Foldningskoder – hastighed $\neq 1/n$ – punktering

- Med hastigheder  $1/n$  kan de viste foldningskoder korrigere ret mange fejl og det har man måske ikke lige brug for. Der er også temmelig meget redundans
- Man kan definere koder med hastighed  $k/n$  ret generelt, men i de fleste anvendelser frembringes disse ud fra  $1/n$  koder ved anvendelse af såkaldt **punktering**
- Punktering betyder at man fjerner nogle bit fra kodeordene i en regelmæssig rytme. Dekodningen, som vi ser på senere, indsætter bit igen, men ved, at den ikke skal tage hensyn til dem. Vi kommer tilbage til dette.
- Lad punkteringsmønsteret have en længde på  $nL$  og  $P$  bits i kodens fernes. Kodens hastighed bliver da

$$\frac{k'}{n'} = \frac{L}{nL - P}$$

- Et eksempel for den viste hastighed  $\frac{1}{2}$ -kode:  $L = 2$  og  $P = 1$  med punkteringsmønster 11 10, hvor 0 angiver punkteringen:

$$\frac{L}{nL - P} = \frac{2}{2 \cdot 2 - 1} = \frac{2}{3} = \frac{k'}{n'}$$

# Fejlkorrigerende og fejldetekterende kodning

Kort oversigt over dette afsnit

- Kodning ved hjælp af en graf
- Definition af foldningskode med hastighed  $1/n$
- Definition af hukommelse,  $M$
- Foldningskoder beskrevet som sekvensmaskine
- To eksempler
- Trellisgraf og kodning
- Definition af fri afstand
- Punktering af koder til hastighed  $k/n$

## ØVELSE #3: KODNING AF FOLDNINGSKODE

1. For to forskellige foldningskoder:
  1. Hastighed ( $1/n$ ) og hukommelse (M)
  2. Tegn et tilstandsdiagram
  3. Foretag kodning af en sekvens
2. MATLAB:
  1. Lav et program til kodning af en vilkårlig  $1/n$ -foldningskode (flere implementerings tip er givet)
  2. Afprøv programmet med sekvens fra punkt 1

# Foldningskoder Viterbi dekoding

**34220**

Knud J. Larsen

Slides: VIT

**DTU Fotonik**  
Department of Photonics Engineering

---

$$C = B \log\left(1 + \frac{S}{N}\right)$$
$$\int_a^b \mathcal{E} \Theta^{\sqrt{17}} + \Omega \int \delta e^{in} =$$
$$\infty = \{10.1011011\ldots\}$$
$$\chi^2 \sum \gg,$$
$$!$$

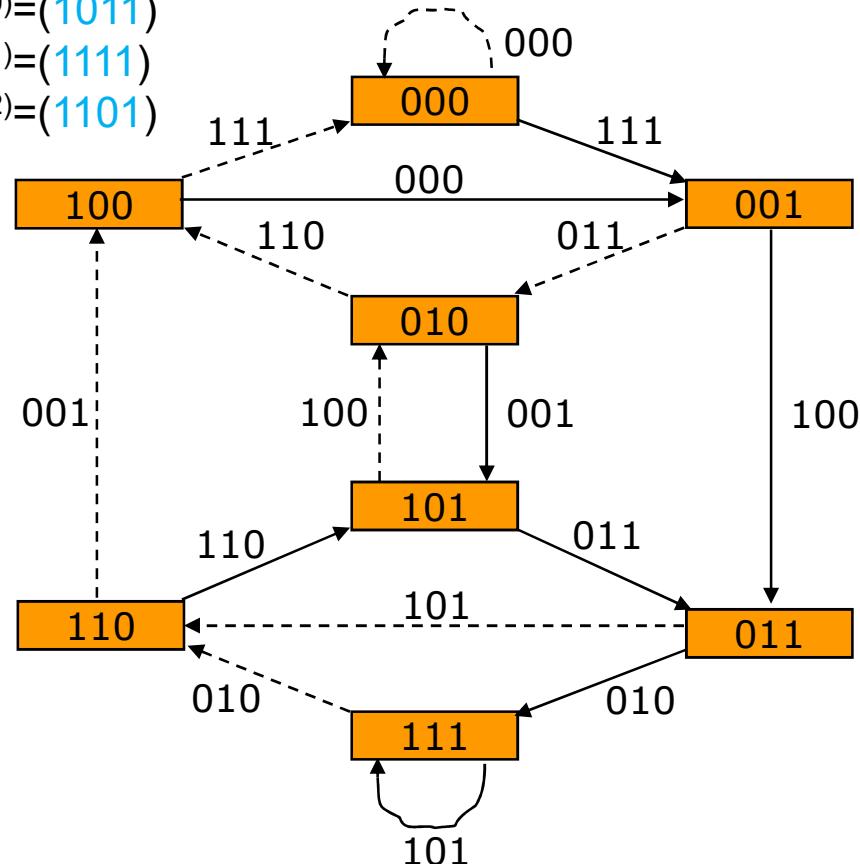
# Fejlkorrigerende og fejldetekterende kodning

I denne del af kurset vil vi se på

- Fejldetection og -korrektion ved hjælp af koder
- Introducere begreber
- Hamming-kode eksempel
- Lineære koder – paritetscheckmatrix, syndromer, dekodning, minimumsvægt, generatormatrix, systematisk kodning
- Fejlsandsynligheder ved anvendelse af blokkoder
- Foldningskoder og **deres dekodning (Viterbi-algoritmen)**
- Cykliske koder – polynomiumsbeskrivelse
- Endelige legemer og koder over disse
- Reed-Solomon koder og deres dekodning
- Simulering af koder på virkelige kanaler
- BCH-koder
- Sammensætning af simple koder til stærke systemer

## Eksempel på foldningskode

$$\begin{aligned}g^{(0)} &= (1011) \\g^{(1)} &= (1111) \\g^{(2)} &= (1101)\end{aligned}$$



Tilstandsdiagram  
Stiplet er overgangen for  $u = 0$   
Fuldt optrukket er for  $u = 1$

Repetition

Eksempel på udgangsbit

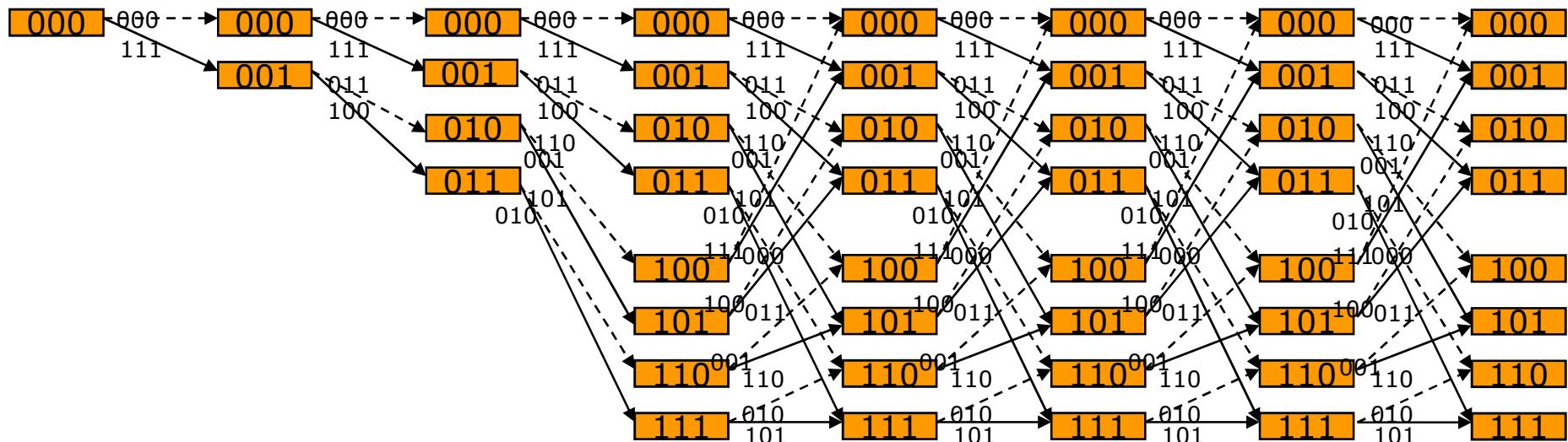
Tilstand=**011**

$$\begin{aligned}u=0: v^{(0)} &= 1 \cdot 0 + 0 \cdot 1 + 1 \cdot 1 + 1 \cdot 0 = 1 \\v^{(1)} &= 1 \cdot 0 + 1 \cdot 1 + 1 \cdot 1 + 1 \cdot 0 = 0 \\v^{(2)} &= 1 \cdot 0 + 1 \cdot 1 + 0 \cdot 1 + 1 \cdot 0 = 1\end{aligned}$$

Næste tilstand=**110**

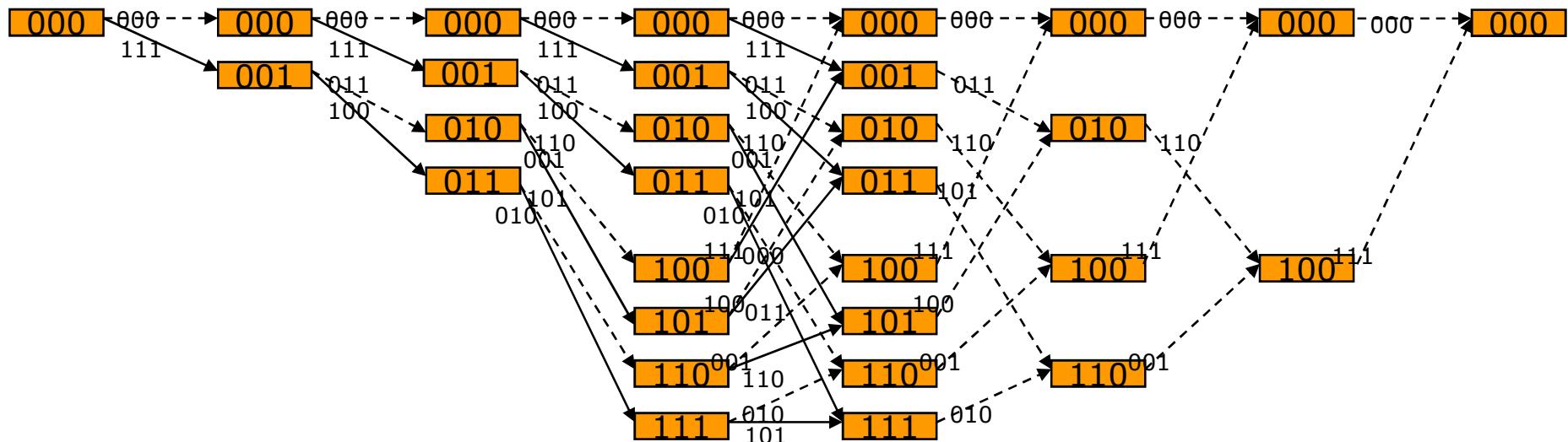
Kodens hastighed er  $1/3$   
da tre bit dannes for  
hver informationsbit

## Trellis for eksemplet



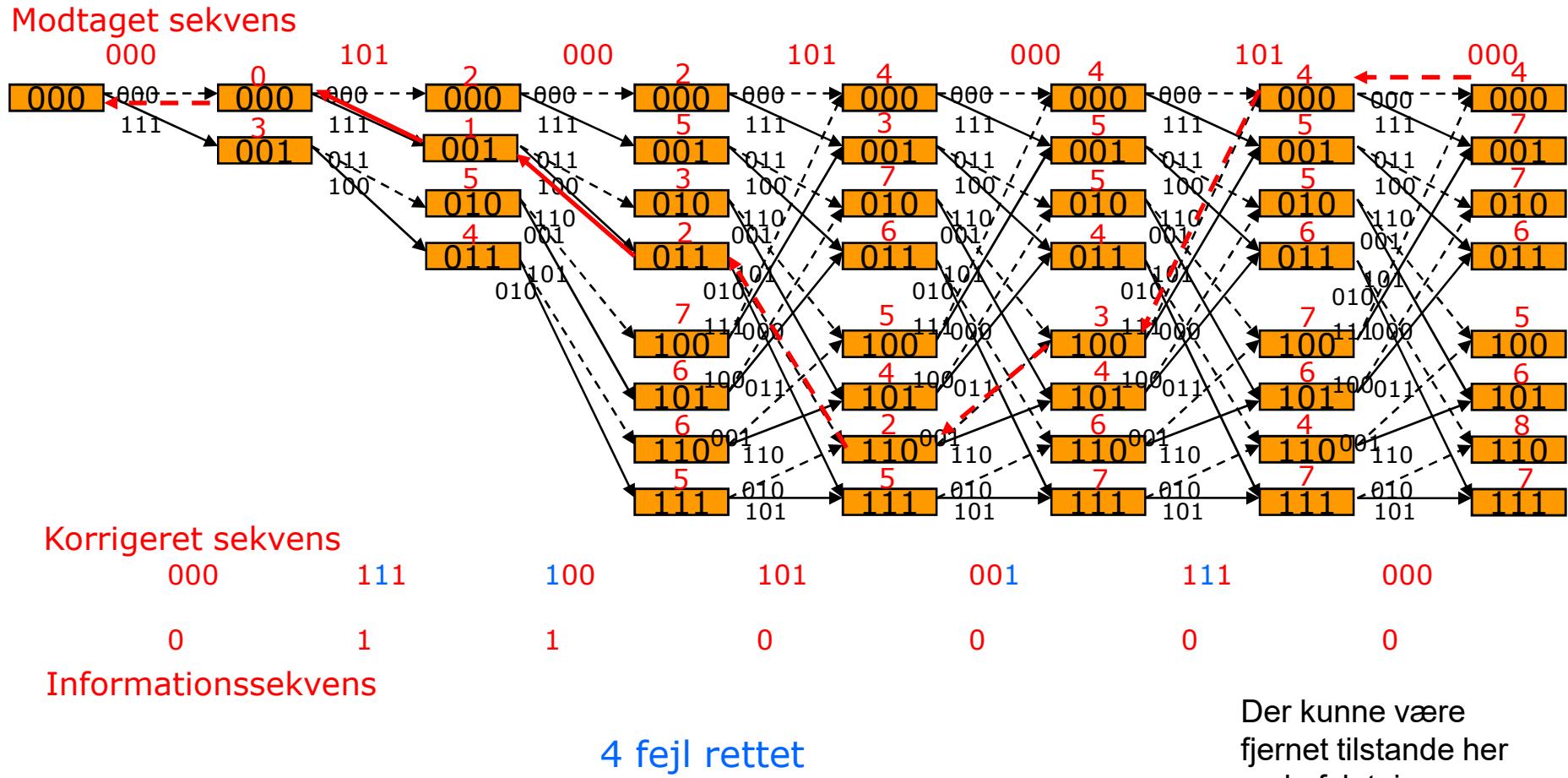
- **Trellis** viser hvordan tilstande og kodeordene udvikler sig i tid startende i en bestemt tilstand (00...0)
- Stippled linjer er informationsbit = 0, fuldt optrukne er for 1 bits

## Trellis for dekodning



- Vi afslutter den ellers uendeligt lange sekvens ved kun at beholde 0-tilstanden til sidst og fjerne alle 1-overgange, dvs. vi afslutter informationen med mindst  $M$  0'er. Dette kaldes terminering og kodeordene danner en blokkode, her med  $N = 7n = 21$ . Den tidligere indførte fri afstand er minimumsvægten for en blokkode, når denne er lang nok (her er den 10).

# Viterbi-algoritmen



Viterbi dekodning for eksemplet

Der kunne være fjernet tilstande her ved afslutningen – men det viser alligevel at 0 er den med færrest fejl

## Viterbi-algoritmen



Denne algoritme er en **maximum likelihood** dekodingsalgoritme, dvs. den finder altid det kodeord der er mest sandsynligt i modsætning til de tidligere viste BDD-algoritmer for blokkoder, der må give op med flere fejl end den faste værdi, den er designet til, som regel den halve minimumsafstand.

Nedenfor viser vi at det **nærmeste kodeord også er det mest sandsynlige**:

Sandsynligheden for at modtage en blok,  $r(j)$  (med  $n$  bit) over en BSC med fejsandsynlighed  $p$ , givet at det sendte stammer fra overgangen fra  $s$  til  $t$ , og at der indtraf  $e_{ts}$  fejl er

$$P[r(j)|s,t] = p^{e_{ts}(j)}(1-p)^{n-e_{ts}(j)} = (1-p)^n \left(\frac{p}{1-p}\right)^{e_{ts}(j)}$$

og for en hel kæde,  $r$ , af sådanne blokke, bliver sandsynligheden produktet. Det er nemmere at bruge logaritmer:

$$\log(P[r]) = \sum_j \log \left( (1-p)^n \left(\frac{p}{1-p}\right)^{e_{ts}(j)} \right) = C + \log\left(\frac{p}{1-p}\right) \cdot \sum_j e_{ts}(j)$$

hvor  $C$  er en konstant og faktoren foran summationen er negativ ( $p < \frac{1}{2}$ ). Således maksimerer man sandsynligheden ved at minimere antallet af fejl som ventet og implicit antaget under diskussionen om blokkoder.

Da Viterbi algoritmen altid afskærer den gren der har flest fejl (eller en gang imellem det samme antal), vil den overlevende vej altid være en med det **mindste antal fejl** og dermed den **nesten sandsynlige**

## Hvordan laver man en Viterbi dekoder? Del I

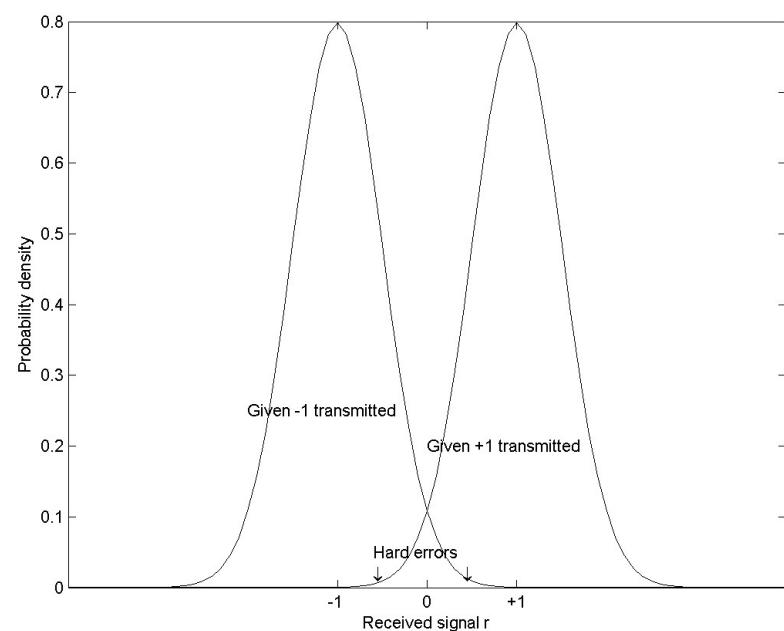
- Man må have et sted at lagre antallet af fejl for hver tilstand, dvs. en vektor af længde  $2^M$  i programmer eller så mange registre i hardware.
- I hvert trin skal man beregne det nye fejlantal i hver tilstand  $s$ . Det gøres ved at tælle antallet af fejl i forhold til den modtagne blok for de to grene der fører ind til  $s$  og addere dette til værdierne i de foregående tilstænde,  $\lfloor s/2 \rfloor$  og  $\lfloor s/2 \rfloor + 2^{M-1}$
- Man kan nu vælge hvilken gren der bevares og gemme denne beslutning samtidig med at der nu er en ny værdi af antallet af fejl i tilstanden. Man må nok have et nyt sted at lagre resultatet for ikke at overskrive de værdier der skal bruges i ovenstående.
- Hvis koden er termineret starter man tilbagesøgningen fra 0-tilstanden gennem beslutningerne for hver tilstand,  $s$ . De tilhørende informationsbit findes som den mindst betydende bit i  $s$ .

## Hvordan laver man en Viterbi dekoder? Del 2

- Oplagringen af beslutninger fylder 1 bit for hver tilstand for hvert af de  $L$  trin, så der skal lagres  $L \cdot 2^M$  bit som kunne blive ganske meget.
- Nu viser det sig heldigvis at hvis man går mere end  $5M - 10M$  tilbage, mødes alle overlevende veje i én tilstand så man kan være sikker på den bit som så kan udleveres til brugerne.
- Det kan benyttes til kun at oplagre et **vindue** af beslutninger af størrelsesorden  $5M - 10M$  i stedet for en meget lang (i princippet måske uendelig) række af beslutninger. For implementeringer – som den i MATLAB – kendes dette som **tilbagesøgningslængde**.
- Når man bruger sådan et vindue, er der ikke en 0-tilstand at starte fra, men da alt alligevel mødes kan man bare starte tilbagesøgningen i en vilkårlig tilstand
- MATLABs `vitdec` kan implementere dette ved passende valg af parametre.

# Viterbi-algoritmen på reelle kanaler

- Vi har demonstreret Viterbi-algoritmen med "hårde beslutninger" som indgang, dvs. det modtagne signal havde værdier 0 og 1 som det sendte.
- I mange tilfælde vil modtageren være i stand til at give mere information end blot en hård beslutning, såkaldte "bløde beslutninger".
- For kanalen med additiv hvid Gaussisk støj, AWGN, som kendes fra kursus 34210, vil det modtagne signal efter det tilpassede filter have en fordeling som vist:



Lad det transmitterede signal i blok j være  $c_i(j) \in \{-1, +1\}$ , dvs. det binære signal fra koderen ændres til et bipolært signal og  $r_i(j)$  bliver modtaget,  $i = 1 \dots n$ .

Man kan så udskifte antallet af fejl,  $e_{ts}(j)$ , ved en given overgang med et bidrag til den samlede værdi for en vej med

$$\ln(P[r(j) | s, t]) = \sum_{i=1}^n \ln(P[r_i(j) | c_i(j), s, t])$$

(logaritmer. fordi de kan summeres i stedet for at regne på sandsynligheder der skal multipliceres)

## Viterbi-algoritmen på reelle kanaler



Det samlede udtryk kan forenkles hvis vi i stedet for

$$\ln(P[r(j) | s, t]) = \sum_{i=1}^n \ln(P[r_i(j) | c_i(j), s, t])$$

fratrækker logaritmen til sandsynligheden for symbolet modsat af det transmitterede så vi for overgangen fra s til t får den såkaldte **grenmetrik**, GM:

$$GM(j | s, t) = \sum_{i=1}^n \ln(P[r_i(j) | c_i(j), s, t]) - \ln(P[r_i(j) | -c_i(j), s, t]) = \sum_{i=1}^n \ln\left(\frac{P[r_i(j) | c_i(j), s, t]}{P[r_i(j) | -c_i(j), s, t]}\right)$$

Beslutningerne påvirkes ikke af dette

For AWGN med varians  $\sigma^2$  (introduceres næste gang, tro bare på det!) får vi

$$\begin{aligned} GM(j | s, t) &= \sum_{i=1}^n \ln(P[r_i(j) | c_i(j), s, t]) - \ln(P[r_i(j) | -c_i(j), s, t]) \\ &= \frac{1}{2\sigma^2} \sum_{i=1}^n -(r_i(j) - c_i(j))^2 - (r_i(j) + c_i(j))^2 = \frac{2}{\sigma^2} \sum_{i=1}^n r_i(j)c_i(j) \end{aligned}$$

dvs. en korrelation mellem det modtagne signal og den kodede blok for overgangen.

## Viterbi-algoritmen på reelle kanaler

I implementeringer bliver det modtagne signal kvantiseret , dvs. variationsområdet inddeltes i et antal, Q, intervaller, som regel af ens størrelse. Disse nummereres normalt  $0, \dots, Q-1$  og disse værdier kan bruges direkte i korrelationen da multiplikation af  $r(j)$  med en konstant (så der bliver afstand 1 mellem niveauerne) og addition af en konstant (så der ikke er negative værdier) ikke forandrer beslutningerne i Viterbi algoritmen og alt kan nu udregnes med heltal som ved optælling af fejl. Dette gælder selvfølgelig for alle  $j$ , så (de modificerede) korrelationer kan adderes til en samlet værdi for en vej i Viterbi algoritmen i stedet for optælling af fejl:

$$\begin{aligned} L(\mathbf{r}) &= \sum_j \sum_{i=1}^n \ln(P[r_i(j) | c_i(j)]) - \ln(P[r_i(j) | -c_i(j)]) \\ &= C_1 \sum_j \sum_{i=1}^n r_i(j)c_i(j) + C_2 \end{aligned}$$

Man skal dog nu vælge den gren der har størst værdi ved beslutningerne så  $L(\mathbf{r})$  bliver maksimeret. Mange implementeringer af Viterbi algoritmer arbejder med minimering, så vi kan i stedet minimere

$$\sum_j \sum_{i=1}^n (c_i(j) - r_i(j))c_i(j)$$

# Viterbi-algoritmen og punkteringer



- Som omtalt kan man frembringe foldningskoder med hastighed  $k/n$  ud fra en  $1/n$  grundkode ved punktering
- Punktering betyder at man fjerner nogle bit fra den kodede  $1/n$  sekvens i et regelmæssigt mønster
- Viterbi dekodning med hårde beslutninger kender punkteringsmønsteret og indsætter de fjernede symboler i sekvensen som slettede symboler, og overspringer disse i optællingen af antallet af fejl.
- Viterbi dekodning med bløde beslutninger indsætter  $r_i(j) = 0$  – dvs. midt mellem  $-1$  og  $+1$  – og de tæller derfor heller ikke med i udregningen af grenmetrikker.
- Modtageren skal altså vide hvor det regelmæssige mønster starter og hvis dette ikke sker ud fra en rammestruktur eller lignende, kan det klares ved at prøve alle muligheder. De forkerte positioner giver hurtigt mange fejl og må forkastes. En lignende metode kan anvendes hvis man heller ikke ved hvor  $n$ -blokkene skiller i det modtagne.

## Eksempler på anvendelse af fejlkorrigerende foldningskoder

- I det ret nye system for TV (**DVB-T**) foregår transmissionen digitalt. Støj påvirker det modtagne signal, og det vil kunne give mange ubehagelige visuelle effekter. Foruden blokkoden nævnt før er transmissionen også beskyttet af en foldningskode med  $M = 6$  som kan vælges med forskellige evner til korrektion: Kodehastighed  $1/2$ ,  $2/3$ ,  $3/4$ ,  $5/6$  og  $7/8$ , hvor de sidste fremkommer ved punktering af grundkoden med hastighed  $1/2$ .
- I **mobiltelefoner** foregår transmissionen som digitale signaler. F.eks. sendes der 50 gange i sekundet 260 bit for at gengive talen i en GSM-telefon. Dette er meget utsat for støj, så det beskyttes med en  $(N,K) = (456, 260)$ -kode. Dette opnås ved at de fleste bit kodes med en foldningskode med kodehastighed  $\frac{1}{2}$  og  $M = 4$  (og nogle efterlades ubeskyttede)
- Kommunikation med videnskabelige satellitter. Her er i mange år benyttet en foldningskode med kodehastighed  $\frac{1}{2}$  (den samme som i DVB-T) og  $M = 6$  sammen med en blokkode (Reed-Solomon) med  $(N,K) = (2040,1784)$  (målt i bit).
- Kommunikation med videnskabelige satellitter. For endnu bedre egenskaber – meget fjerne satellitter – kan man benytte en såkaldt turbo-kode hvor flere foldningskoder arbejder sammen om at beskytte data.

# Fejlkorrigerende og fejldetekterende kodning

Vi har set på

- Eksempel på foldningskoder beskrevet som sekvensmaskine
- Trellisgraf og kodning
- Afsluttet trellis for dekodning
- Viterbi-algoritmen
- Viterbi-algoritmen for reelle kanaler
- Viterbi-algoritmen og punkterede koder
- Eksempler på anvendelse af foldningskoder

## ØVELSE #4: VITERBI DEKODNING AF FOLDNINGSKODE

1. På basis af et trellis for  $\mathbf{g}^{(0)} = (1\ 0\ 1)$ ,  $\mathbf{g}^{(1)} = (1\ 1\ 1)$  kodes en sekvens
2. Dernæst foretages Viterbi-dekodng af en modtaget sekvens
3. Flere fejl introduceres så dekodningen går galt
4. MATLAB:
  1. Anvend MATLABs `vitdec` til dekodningen

# Cyklistiske koder

34220

Knud J. Larsen

Slides: CYC

DTU Fotonik  
Department of Photonics Engineering

---

$$C = B \log\left(1 + \frac{S}{N}\right)$$
$$\int_a^b \mathcal{E} \Theta_{\infty}^{\sqrt{17}} + \Omega \int \delta e^{i\pi} = \sum \chi^2 \gg \{10.1011011, \dots\}$$

# Fejlkorrigerende og fejldetekterende kodning

I denne del af kurset vil vi se på

- Fejldetection og -korrektion ved hjælp af koder
- Introducere begreber
- Hamming-kode eksempel
- Lineære koder – paritetscheckmatrix, syndromer, dekodning, minimumsvægt, generatormatrix, systematisk kodning
- Fejsandsynligheder ved anvendelse af blokkoder
- Foldningskoder og deres dekodning (Viterbi-algoritmen)
- **Cykiske koder – polynomiumsbeskrivelse**
- Endelige legemer og koder over disse
- Reed-Solomon koder og deres dekodning
- BCH-koder
- Simulering af koder på virkelige kanaler
- Sammensætning af simplere koder til stærke systemer

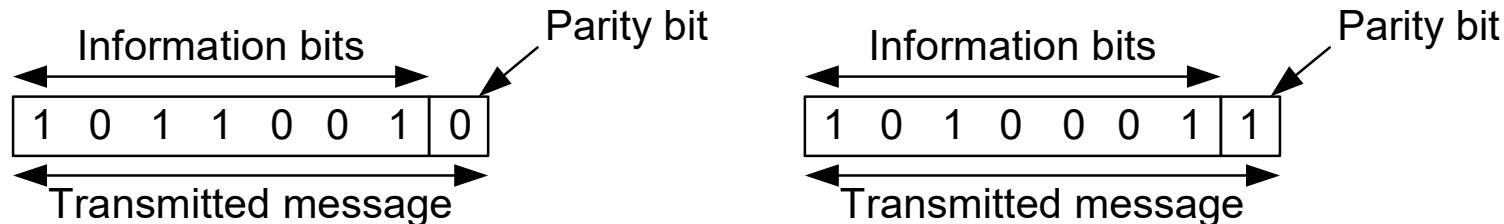
## Cyklistiske koder

- Nogle koder er **cyklistiske**, dvs. et cyklisk skift (rotation) af et kodeord er også et kodeord
- Ved efterprøvning ses (7,4) Hammingkoden i eksemplet at have den egenskab
- Men for at se nærmere på den cyklistiske egenskab formulerer vi kodeord som polynomier i stedet for vektorer:

$$C(x) = c_{N-1}x^{N-1} + c_{N-2}x^{N-2} + c_{N-3}x^{N-3} + \dots + c_2x^2 + c_1x + c_0$$

- Oftest transmitteres de fra højeste potens og nedefter, så vi har vendt rækkefølgen om i forhold til de tidligere eksempler. Det ændrer selvfølgelig ikke ved kodens egenskaber.

# Cyklistiske koder - paritetscheck



- Som vi så tidligere kan vi detektere fejl i sekvenser ved at tilføje en paritetsbit
- Det er en cyklistisk kode da paritetten passer lige meget hvordan vi roterer
- Men hvad betyder det når vi formulerer det med polynomier?

$$C(x) = c_{N-1}x^{N-1} + c_{N-2}x^{N-2} + c_{N-3}x^{N-3} + \dots + c_2x^2 + c_1x + c_0$$

- Når antallet af 1'er er lige, har  $C(x)$  en faktor som er  $x + 1$  (og andre, men det er uvedkommende). Hvorfor?

# Cyklistiske koder – polynomiers division

- Division og faktorisering af polynomier er nemmest vist ved et lille eksempel

$$\begin{array}{r}
 & x^5 + x^4 + & x & + 1 \\
 \hline
 x+1 & \overline{x^6 + x^4 + x^2 + 1} \\
 & x^6 + x^5 \\
 \hline
 & x^5 + x^4 + x^2 + 1 \\
 & x^5 + x^4 \\
 \hline
 & x^2 + 1 \\
 & x^2 + x \\
 \hline
 & x+1 \\
 & x+1 \\
 \hline
 & 0
 \end{array}$$

$$\begin{aligned}
 x^6 + x^4 + x^2 + 1 &= \\
 (x+1)(x^5 + x^4 + x+1) &
 \end{aligned}$$

Hver gang  $x+1$  har "spist" et lige antal potenser er den betydende rest 0 og vi springer ned til næste ikke-0 led og til sidst går det op hvis der er et lige antal 1'ere

# Cyklistiske koder – mere generelt

Generelt er det en større divisor,  $G(x)$  vi skal have for at kunne rette fejl

$$G(x) = g_{N-K}x^{N-K} + c_{N-K+1}x^{N-K+1} + \dots + g_2x^2 + g_1x + g_0$$

dvs. kodeord  $C(x) = G(x)K(x)$ , hvor  $K(x)$  er et  $K-1$ 'te grads polynomium (der fx kan have de  $K$  informationssymboler som koefficienter – mere senere)

En sådan kode er selvfølgelig lineær da  $G(x)$  er faktor i en linearkombination af to kodepolynomier

Men hvornår bliver den cyklistisk?

$G(x)$  skal gå op i  $x^N - 1$ :

$$\text{Res}_{G(x)}(\text{cyklistiskshift } C(x)) = \text{Res}_{G(x)}(xC(x) - c_{N-1}x^N + c_{N-1}) =$$

$$0 - c_{N-1} \text{Res}_{G(x)}(x^N - 1) = 0 \text{ hvis og kun hvis } G(x) | (x^N - 1)$$

# Cyklistiske koder – kodning

Hvordan koder vi de  $K$  informationsbit  $K(x)$ ?

- En simpel, men usystematisk løsning er  $C(x) = K(x)G(x)$ , men så er det sværere at se  $K(x)$  i  $C(x)$
- Hvis vi i stedet udregner

$$C(x) = K(x)x^{N-K} + \underset{G(x)}{\text{Rest}}(-K(x)x^{N-K}) = K(x)x^{N-K} + R(x)$$

kan vi se  $K(x)$  direkte på de første  $K$  pladser (**systematisk kodning**) og  $R(x)$  er de  $N-K$  **redundanssymboler** som tilføjes for at beskytte informationen så fejl kan detekteres eller rettes.

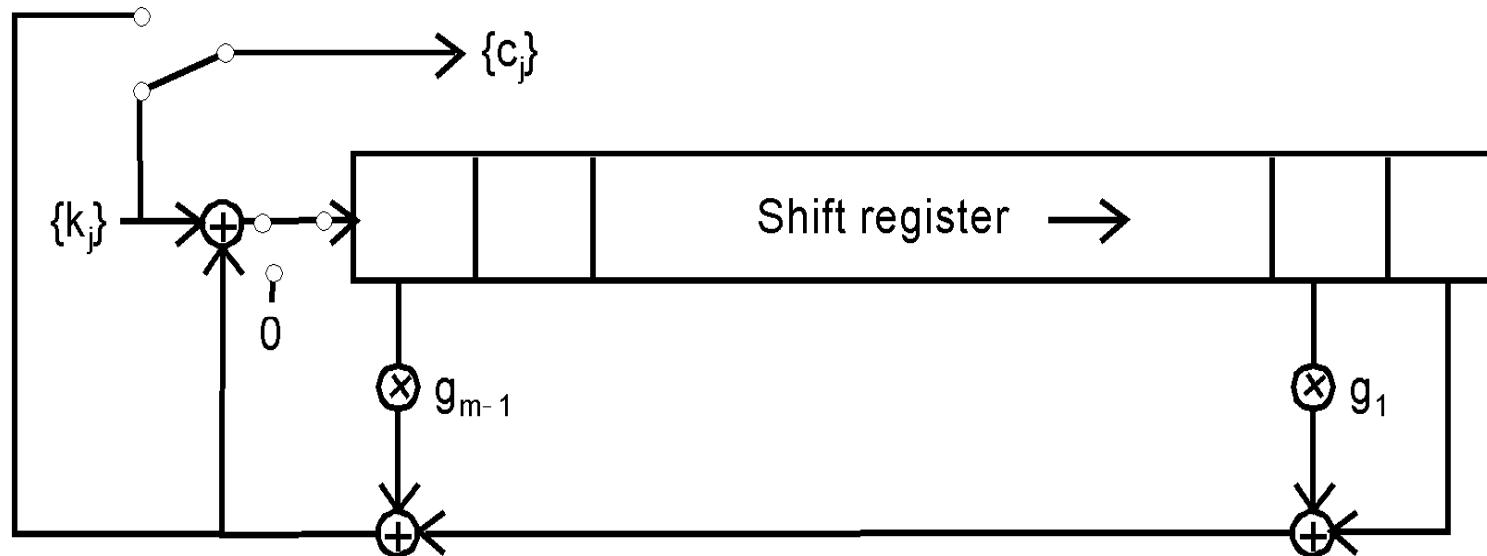
At det fungerer bevises nemt:

$$\begin{aligned}\underset{G(x)}{\text{Rest}}(C(x)) &= \underset{G(x)}{\text{Rest}}(K(x)x^{N-K}) + \underset{G(x)}{\text{Rest}}\left(\underset{G(x)}{\text{Rest}}(-K(x)x^{N-K})\right) = \\ \underset{G(x)}{\text{Rest}}(K(x)x^{N-K}) - \underset{G(x)}{\text{Rest}}(K(x)x^{N-K}) &= 0\end{aligned}$$

I MATLAB kan binære polynomiers division `pol/G` udføres med `[Q, R] = deconv (pol, G)` efterfulgt af `mod(R, 2)` for udregning af resten eller man kan bruge datatypen `gf` som introduceres i Øvelse 5 og undgå den efterfølgende modulo 2 operation `mod(R, 2)`.

# Cyklistiske koder – kodning

En hardwareimplementering af en systematisk koder for en binær cyklistisk kode – addition modulo 2



Kodere i hardware og software kan gøres hurtigere ved at udregne flere bit ad gangen, men så ser det ikke simpelt ud længere!

Initialiseres med 0...0 og omskifttere som vist.  
Omskifttere flyttes efter de K informationsbit og  $R(x)$  fremkommer med  $N-K$  yderligere skift.

## Cyklistiske koder – dekodning

Ved modtagelse af fejlbehæftet sekvens har vi

$$R(x) = C(x) + E(x)$$

Hvor  $E(x)$  har fejlværdierne ved potenser med fejl. Som ved matrixmetoden kan vi udregne syndromværdier. Vi prøver med

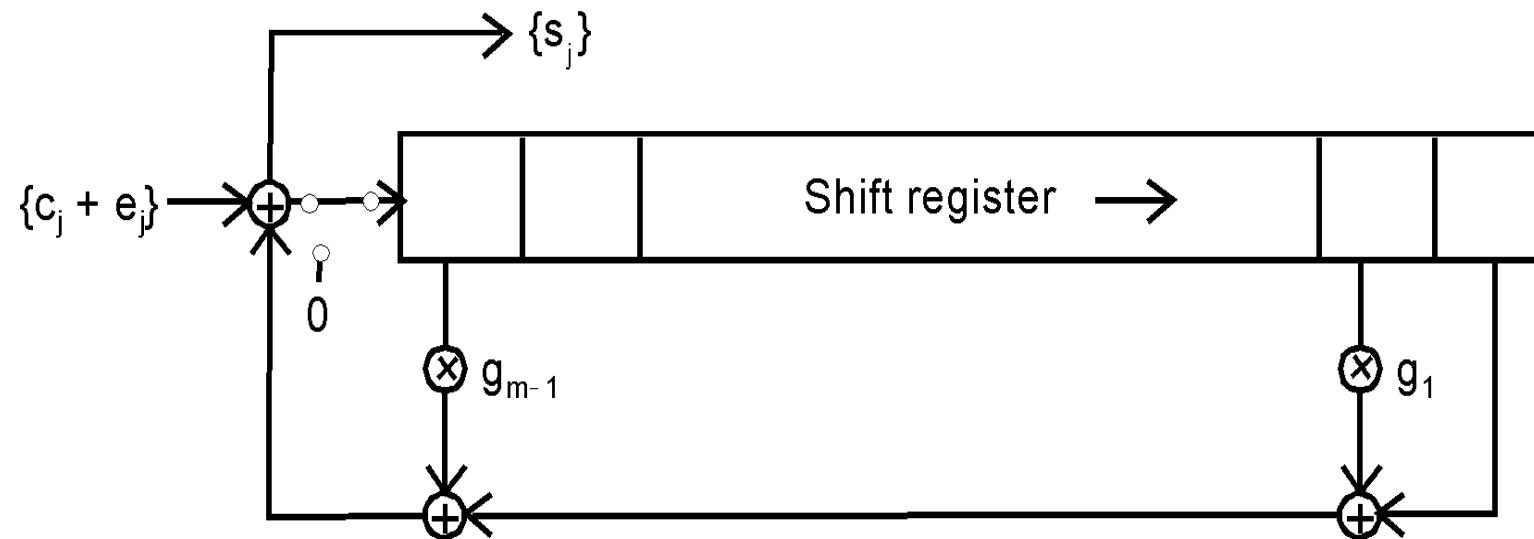
$$\text{Syndrom}(R(x)) = \text{Rest}_{G(x)}(R(x)) = \text{Rest}_{G(x)}(C(x) + E(x)) = 0 + \text{Rest}_{G(x)}(E(x))$$

Dette **syndrom** afhænger også kun af fejlene, ikke den transmitterede information (og viser sig i øvrigt at være helt det samme som vi fandt med paritetscheckmatricen)

- syndrom = 0 hvis  $E(x) = 0$  (eller  $G(x)|E(x)$  som så har en del fejl)
- syndrom  $\neq 0$  viser fejl i  $R(x)$  (fejldetektion, vil også kunne anvendes til at rette fejl hvis syndromerne adskiller sig)

# Cyklistiske koder – dekodning

En hardwareimplementering af en syndromdanner for en binær cyklistisk kode – addition modulo 2



Initialiseres til 0...0 og omskifttere som vist  
Omskifter flyttes efter K skift og producere syndromet efter N-K skift

# Cykliske koder – dekodning



Men hvordan vælger vi  $G(x)$ ?

- Det skal gå op i  $x^N - 1$ , dvs.  $G(x)$  skal være et produkt af faktorer i  $x^N - 1$
- Det skal give forskellige rester for forskellige fejlmønstre
- Hvis der ikke er alt for mange faktorer vil det være forholdsvis nemt at prøve de relevante  $G(x)$  og se hvordan syndromerne opfører sig for 1 fejl, 2 fejl osv.
- Som vi skal se senere i kurset kan noget mere matematik føre en til fornuftige konstruktioner

# Hamming koden som cyklisk kode

Lad  $N = 2^m - 1$  for heltallige  $m > 0$ .

- Enkeltfejl viser sig altså som  $E(x) = x^0 \dots x^{N-1}$
- Som vi skal se senere vil disse  $N = 2^m - 1$  rester blive forskellige hvis vi anvender  $G(x) = P(x)$ , hvor  $P(x)$  er såkaldt **primitivt polynomium**. Der er altid mindst ét sådant for ethvert  $m$ . På denne måde har vi altså lavet en cyklisk kode der kan rette 1 fejl ( $d = 3$ ), en Hamming kode.
- De primitive polynomier er nogle af faktorerne i  $x^N - 1$  for  $N = 2^m - 1$ , så

$$G(x) | (x^N - 1)$$

# Lineære fejlkorrigerende koder

## Repetition

Betingelserne for  $\mathbf{c}$  som et kodeord i en Hamming kode var:

$$[c_0, c_1, c_2, c_3, c_4, c_5, c_6] \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = [0, 0, 0]$$

Altså

$$\mathbf{cH} = \mathbf{0}$$

$\mathbf{H}$  kaldes en paritetscheckmatrix for koden

Rækkerne i  $\mathbf{H}$  er syndromerne for enkeltfejl men de kan nu ses at fremkomme som

$$\text{Rest}_{G(x)}(E(x)) = \text{Rest}_{x^3+x+1}(x^j) \quad \text{for } j=0 \dots 6 \text{ regnet nedefra}$$

# Hamming koden som cyklisk kode

- Hvis vi multiplicerer  $G(x)$  med  $x + 1$  vil alle kodeord få lige vægt, 4, og vi kan dermed frembringe den tidligere omtalte **udtyndede Hamming-kode** med  $(N, K) = (2^m - 1, 2^m - m - 2)$  der kan rette 1 fejl og detektere når der er to fejl, idet syndromerne for 1 fejl alle er unikke og forskellige fra dem for to fejl.
- Hamming koder og deres udtyndning anvendes ofte til fejlcheck, såkaldt CRC-check (*cyclic redundancy check*). Vi giver nogle eksempler på næste slide. De angivne N og K er det maksimale – checket kan anvendes på kortere sekvenser der kan betragtes som startende med et passende antal 0'er der ikke influerer på de udregnede rester.
- Vi skal senere se på andre konstruktive eksempler på cykliske koder

## Eksempler på CRC-check

N	K	Polynomium G(x)	d-1	Navn
Alle	N-1	$x+1$	1	Paritetscheck
7	4	$x^3 + x + 1$	2	Hamming kode
7	3	$x^4 + x^3 + x^2 + 1$	3	Udtyndet Hamming
15	8	$x^7 + x^6 + x^4 + 1$	3	CRC-7
93	85	$x^8 + x^7 + x^6 + x^4 + x^2 + 1$	3	CRC-8
2047	2035	$x^{12} + x^{11} + x^3 + x^2 + x + 1$	3	CRC-12
32767	32751	$x^{16} + x^{12} + x^5 + 1$	3	CRC-ITU-T
32767	32751	$x^{16} + x^{15} + x^2 + 1$	3	CRC-16, - ANSI
$2^{32}-1$	$2^{32}-33$	$x^{32} + x^{26} + x^{23} + \dots + x^2 + x + 1$	2	CRC-32, Hamming

Med minimumsafstand d kan d-1 fejl altid detekteres (sammen med mange andre, mindre sandsynlige fejlmønstre)

## Nogle korte binære cykliske koder

N	K	Polynomium G(x)	d	Navn
15	11	$x^4 + x + 1$	3	Hamming
21	12	$x^9 + x^8 + x^5 + x^4 + x^2 + x + 1$	5	
23	12	$x^{11} + x^9 + x^7 + x^6 + x^5 + x + 1$	7	Golay
31	26	$x^5 + x^2 + 1$	3	Hamming
31	21	$x^{10} + x^9 + x^8 + x^5 + x^4 + x^3 + 1$	5	BCH
31	16	$x^{15} + x^{11} + x^{10} + x^9 + x^8 + x^7 + x^5 + x^3 + x^2 + x + 1$	7	BCH

# Fejlkorrigerende og fejldetekterende kodning

Kort oversigt over dette afsnit

- Kodeord beskrevet ved polynomier
- Polynomiers division
- Generatorpolynomium for cyklisk kode
- Systematisk og ikke systematisk kodning for cyklisk kode
- Syndromberegning for cyklisk kode
- Fejldetection med cyklisk kode (CRC)
- Eksempler på CRC og fejlkorrigerende cykliske koder

# Endelige legemer Reed-Solomon koder

34220

Knud J. Larsen

Slides: RSC

$$C = B \log\left(1 + \frac{S}{N}\right)$$
$$\int_a^b \mathcal{E} \Theta^{\sqrt{17}} + \Omega \int \delta e^{i\pi} =$$
$$\infty - \{10.1011011\}$$
$$\chi^2 \sum!$$

# Fejlkorrigerende og fejldetekterende kodning

I denne del af kurset vil vi se på

- Fejldetection og -korrektion ved hjælp af koder
- Introducere begreber
- Hamming-kode eksempel
- Lineære koder – paritetscheckmatrix, syndromer, dekodning, minimumsvægt, generatormatrix, systematisk kodning
- Fejsandsynligheder ved anvendelse af blokkoder
- Foldningskoder og deres dekodning (Viterbi-algoritmen)
- Cykliske koder – polynomiumsbeskrivelse
- Endelige legemer og koder over disse
- Reed-Solomon koder og deres dekodning
- BCH-koder
- Simulering af koder på virkelige kanaler
- Sammensætning af simplere koder til stærke systemer

# Modulo-regning og endelige legemer

Et **legeme** er en algebraisk struktur med

- to regningsarter (kompositioner), addition og multiplikation,  $+, \cdot$
- et neutralelement for hvert, dvs. henholdsvis 0 og 1
- inverse elementer, dvs.  $-a$  for addition,  $a^{-1}$  ( $a \neq 0$ ) for multiplikation
- alle normale regneregler kan bruges og normale sætninger gælder
- Eksempler: rationale, reelle, eller komplekse tal, men ikke fx heltal
- Der findes også legemer med et **endeligt** antal,  $q$ , elementer,  $F(q)$
- Eksempel: Restklasseregning: heltal modulo et primtal  $p$ , med  $p$  elementer

$F(11)$ :

elementer	0	1	2	3	4	5	6	7	8	9	10
+ invers	0	10	9	8	7	6	5	4	3	2	1
* invers		1	6	4	3	9	2	8	7	5	10

- $F(pq)$  hvor  $p$  og  $q$  er to forskellige primtal eksisterer ikke, da de multiplikative inverse ikke findes: antag fx at  $p$  har en invers:  $pq=0 \pmod{pq} \rightarrow p^{-1}pq=0 \rightarrow q=0$  og det skulle jo være et primtal
- Man kan dog godt have andet end primtal som antal elementer:
- De eneste endelige legemer der findes  $F(p^m)$ ,  $p$  primtal,  $m > 0$ .

# Endelige legemer

Alle endelige legemer har et **primitivt element**,  $\alpha$ :

- Så alle elementer  $\neq 0$  kan frembringes som potenser af  $\alpha$ , dvs.  $\alpha^i$
- $\alpha^{q-1} = 1$

$F(11)$ :  $\alpha = 2$

$i$	0	1	2	3	4	5	6	7	8	9	10
elementer	1	2	4	8	5	10	9	7	3	6	1

Multiplikation er ret simpel når eksponenterne ("logaritmer") anvendes:

$$8 \cdot 6 = 48 = 4 \text{ mod } 11$$

$$\alpha^3 \alpha^9 = \alpha^{12} = \alpha^2 \alpha^{10} = \alpha^2$$

Alle elementer  $\neq 0$  er rødder i  $x^{q-1} - 1$  da  $(\alpha^i)^{q-1} = (\alpha^{q-1})^i = 1$

# Endelige legemer

Der skulle eksistere et endeligt legeme med  $2^m$  elementer,  $F(2^m)$ , men hvordan?

- modulo  $2^m$  ikke muligt
- Elementerne kan beskrives som alle polynomier,  $g(z)$ , med grad  $< m$  med koefficienter,  $g_k$ , fra  $F(2)$ :  $g(z) = g_{m-1}z^{m-1} + g_{m-2}z^{m-2} + \dots + g_2z^2 + g_1z + g_0$
- addition er blot addition af koefficienterne i  $F(2)$  for hver potens  $z^k$
- Multiplikation er mere indviklet:
  - Polynomierne skal multipliceres modulo et specielt polynomium  $P(z)$
  - $P(z)$  har koefficienter i  $F(2)$ , har grad  $m$ , og er irreducibelt dvs. det kan ikke splittes i faktorer
  - Eksempel
    - $(z^2+z+1)(z+1) \text{ mod } z^3+z+1 =$
    - $z^3+z^2+z+z^2+z+1 \text{ mod } z^3+z+1 =$
    - $z^3+1 \text{ mod } z^3+z+1 = (z^3+1)+(z^3+z+1) = z$  (husk – og + er ens i  $F(2)$ )
- Ved rigtigt valg af  $P(z)$ , bliver  $z^1$  det primitive element, a. Polynomiet kaldes så primitivt.
- Multiplikationer bliver så meget simplere da alle elementer  $\neq 0$  kan repræsenteres som en potens,  $a^i$ , og eksponenterne kan så blot adderes (modulo  $q-1 = 2^m-1$ )

# Endelige legemer

Eksempel  $F(2^4) = F(16)$ ,  $P(z) = z^4 + z + 1$ , dvs  $z^4 = z + 1$

		$z^3$	$z^2$	$z^1$	$z^0$	
0	0	0	0	0	0	
$\alpha^0$	$z^0$	0	0	0	1	
$\alpha^1$	$z^1$	0	0	1	0	
$\alpha^2$	$z^2$	0	1	0	0	
$\alpha^3$	$z^3$	1	0	0	0	
$\alpha^4$	$z^4$	0	0	1	1	$z^4 \bmod z^4 + z + 1 = z + 1$
$\alpha^5$	$z^5$	0	1	1	0	
$\alpha^6$	$z^6$	1	1	0	0	
$\alpha^7$	$z^7$	1	0	1	1	$z^7 = z^6z = z^3z + z^2z = z^3 + z + 1$
$\alpha^8$	$z^8$	0	1	0	1	$z^4 = z + 1$
$\alpha^9$	$z^9$	1	0	1	0	
$\alpha^{10}$	$z^{10}$	0	1	1	1	$z^4 = z + 1$
$\alpha^{11}$	$z^{11}$	1	1	1	0	
$\alpha^{12}$	$z^{12}$	1	1	1	1	$z^4 = z + 1$
$\alpha^{13}$	$z^{13}$	1	1	0	1	$z^4 = z + 1$
$\alpha^{14}$	$z^{14}$	1	0	0	1	$z^4 = z + 1$

med  $\alpha = z$  som  
primitivt element:

$$\begin{aligned}
 \alpha^7\alpha^{10} &= \alpha^{17} = \alpha^2\alpha^{15} = \alpha^2 \\
 (\alpha^3 + \alpha + 1)(\alpha^2 + \alpha + 1) &= \\
 &= \alpha^5 + \alpha^4 + 1 \\
 &= (\alpha + 1)(\alpha^4 + \alpha + 1) + \alpha^2 \\
 &= \alpha^2 \bmod P(\alpha)
 \end{aligned}$$

MATLAB indeholder en datatype for endelige legemer  $F(2^m)$  og kan regne direkte på sådanne data. Det ser vi på i øvelsen

# Endelige legemer

Et specielt eksempel  $F(2^4) = F(16)$ ,  $P(z) = z^4 + z^3 + z^2 + z + 1$ , dvs  $z^4 \equiv z^3 + z^2 + z + 1$

		$z^3$	$z^2$	$z^1$	$z^0$	
0	0	0	0	0	0	
$\alpha^0$	$(1+z)^0$	0	0	0	1	
$\alpha^1$	$(1+z)^1$	0	0	1	1	
$\alpha^2$	$(1+z)^2$	0	1	0	1	
$\alpha^3$	$(1+z)^3$	1	1	1	1	
$\alpha^4$	$(1+z)^4$	1	1	1	0	$z^4 \equiv z^3 + z^2 + z + 1$
$\alpha^5$	$(1+z)^5$	1	1	0	1	$z^4 \equiv z^3 + z^2 + z + 1$
$\alpha^6$	$(1+z)^6$	1	0	0	0	$z^4 \equiv z^3 + z^2 + z + 1$
$\alpha^7$	$(1+z)^7$	0	1	1	1	$z^4 \equiv z^3 + z^2 + z + 1$
$\alpha^8$	$(1+z)^8$	1	0	0	1	
$\alpha^9$	$(1+z)^9$	0	1	0	0	$z^4 \equiv z^3 + z^2 + z + 1$
$\alpha^{10}$	$(1+z)^{10}$	1	1	0	0	
$\alpha^{11}$	$(1+z)^{11}$	1	0	1	1	$z^4 \equiv z^3 + z^2 + z + 1$
$\alpha^{12}$	$(1+z)^{12}$	0	0	1	0	$z^4 \equiv z^3 + z^2 + z + 1$
$\alpha^{13}$	$(1+z)^{13}$	0	1	1	0	
$\alpha^{14}$	$(1+z)^{14}$	1	0	1	0	

Dette polynomium  
er ikke primitivt, men  
har  $\alpha = z+1$  som  
primitivt element:

$$\begin{aligned}
 \alpha^{11}\alpha^7 &= \alpha^{18} = \alpha^3\alpha^{15} = \alpha^3 \\
 (\alpha^3 + \alpha)(\alpha^2 + \alpha + 1) &= \alpha^5 + \alpha^4 + 1 \\
 &= \alpha P(\alpha) + \alpha^3 + \alpha^2 + \alpha + 1 \\
 &= \alpha^3 + \alpha^2 + \alpha + 1 \quad \text{mod } P(\alpha)
 \end{aligned}$$

MATLABs datatype for  
endelige legemer er  
altid genereret af  
primitive polynomier

# Endelige legemer

De mest almindeligt anvendte primitive polynomier over  $\text{F}(2)$  (fx i MATLAB)

Grad	$P(z)$
2	$z^2+z+1$
3	$z^3+z+1$
4	$z^4+z+1$
5	$z^5+z^2+1$
6	$z^6+z+1$
7	$z^7+z^3+1$
8	$z^8+z^4+z^3+z^2+1$
9	$z^9+z^4+1$
10	$z^{10}+z^3+1$
11	$z^{11}+z^2+1$
12	$z^{12}+z^6+z^4+z+1$
13	$z^{13}+z^4+z^3+z+1$
14	$z^{14}+z^{10}+z^6+z+1$
15	$z^{15}+z+1$
16	$z^{16}+z^{12}+z^3+z+1$

# Endelige legemer – diverse

- Som tidligere omtalt findes kun endelige legemer for  $q = p^m$  hvor  $p$  er et primtal og  $m \geq 1$
- Disse konstrueres ligesom  $F(2^m)$  ved hjælp af primitive polynomier,  $P(z)$ , med koefficienter i  $F(p)$  og grad  $m$ . Addition er modulo  $p$  for koefficienter og multiplikation er modulo  $P(z)$
- Her er der også et primitivt element  $\alpha$  og alle ikke-0 elementer er  $\alpha^i$
- Alle elementer  $\neq 0$  er rødder i  $x^{q-1} - 1$  da  $(\alpha^i)^{q-1} = (\alpha^{q-1})^i = 1$
- I ethvert legeme gælder at et **n'te grads polynomium højest har n rødder.**
- En udregning til senere anvendelse:

$$0 = x^{q-1} - 1 = (x - 1) \sum_{i=0}^{q-2} x^i, \text{ som giver}$$

$$\text{For } x \neq 1: \quad \sum_{i=0}^{q-2} x^i = 0 \quad \text{For } x = 1: \quad \sum_{i=0}^{q-2} 1^i = \sum_{i=0}^{p^m-2} 1 = (p^m - 1) \quad (= -1 \bmod p)$$

# Lineære koder over $F(q)$

- Lineære koder over generelle endelige legemer,  $F(q)$ , kan defineres ganske som de binære koder over  $F(2)$ , dvs.  $\mathbf{H}$  og  $\mathbf{G}$  matricer.
- Som tidligere er (Hamming-)afstanden mellem to kodeord  $\mathbf{a}$  og  $\mathbf{b}$   $d(\mathbf{a}, \mathbf{b}) =$  antallet af symboler der er forskellige (Bemærk: værdierne ikke involveret) og antallet af ikke-0 symboler i et kodeord er vægten af kodeordet,  $w(\mathbf{a})$ .
- $d(\mathbf{a}, \mathbf{b}) = w(\mathbf{a} - \mathbf{b})$  og da  $\mathbf{a} - \mathbf{b} \in$  koden (da den er lineær), kan alle afstande findes som vægte for et eller andet kodeord, så det er meget simplere at finde fx minimumsafstanden  $d$ .
- Vi ser nu på et meget vigtigt eksempel

## Reed-Solomon koder

- Som regel over  $F(2^m)$ , vi har nogle eksempler med simplere regning fx  $F(7)$
- Eksempler CD, DVD, Digital TV, QR koder, Datamatrix. StampIT
- Vi ser først på definition og kodning

# Reed-Solomon Koder

- Polynomier over  $F(q)$  med grad  $< K$ :  $u(x)$ .
- $F(q)$ -koefficienterne i  $u(x)$  kan ses som de  $K$  **informationssymboler**,  $u_{K-1}, \dots, u_0$ :

$$u(x) = u_{K-1}x^{K-1} + u_{K-2}x^{K-2} + \dots + u_1x + u_0$$

- **Kodningen** gøres i første omgang ved at udregne værdierne af  $u(x)$  i  $N \leq q$  **forskellige** elementer,  $x_i$ , i  $F(q)$ :

$$[c_{N-1}, c_{N-2}, \dots, c_1, c_0] = [u(x_{N-1}), u(x_{N-2}), \dots, u(x_1), u(x_0)]$$

- Koden er lineær da  $a u_1(x_i) + b u_2(x_i) = u(x_i)$  også er i koden
- Dimensionen er  $K$ :
  - $\leq K$  da der er  $K$  koefficienter i  $u(x)$
  - to forskellige  $u_1(x)$  og  $u_2(x)$  giver forskellige kodeord da  $u_1(x) - u_2(x)$  er et polynomium med grad  $< K$  og det kan ikke være 0 for alle  $x_i$  da det højst kan have  $K-1$  rødder.
- Minimumsafstanden er  $d = N - (K-1) = N - K + 1$  da  $u(x)$  højst har  $K-1$  rødder
- Koden kan altså rette  $T = (d - 1)/2 = (N - K)/2$  fejl

# Cyklistiske Reed-Solomon koder

- Koden bliver **cyklistisk**, hvis vi vælger  $x_i = \alpha^i$ ,  $i = 0 \dots N - 1 = q - 2$  i  $\mathbb{F}(q)$
- Kodeordet bliver altså

$$[c_{N-1}, c_{N-2}, \dots, c_1, c_0] = [u(\alpha^{N-1}), u(\alpha^{N-2}), \dots, u(\alpha^1), u(\alpha^0)]$$

- eller udtrykt som polynomium

$$C(x) = \sum_{i=0}^{N-1} c_i x^i = \sum_{i=0}^{N-1} \left( \sum_{j=0}^{K-1} u_j \alpha^{ji} \right) x^i$$

- På næste slide viser vi at  $G(x) = (x - \alpha^1)(x - \alpha^2) \dots (x - \alpha^{2T})$  er altid en faktor i  $C(x)$
- Da alle  $\alpha^j$  er rødder  $x^{q-1} - 1 = x^N - 1$  får vi at  $G(x) | (x^N - 1)$   
så  $G(x)$  er **generatorpolynomiet** for en cyklistisk Reed-Solomon kode

# Cykliste Reed-Solomon koder



- Kodeord udtrykt som polynomium

$$C(x) = \sum_{i=0}^{N-1} c_i x^i = \sum_{i=0}^{N-1} \left( \sum_{j=0}^{K-1} u_j a^{ji} \right) x^i$$

- Lad os undersøge rødderne  $\beta$  i  $C(x)$ , dvs. for hvilke  $\beta$  er  $C(\beta) = 0$ :

$$C(\beta) = \sum_{i=0}^{N-1} \sum_{j=0}^{K-1} u_j \alpha^{ji} \beta^i = \sum_{j=0}^{K-1} u_j \sum_{i=0}^{N-1} (\alpha^j \beta)^i$$

- C har altså altid  $a^{-K}, a^{-K-1}, \dots, a^{1-N}$  som rødder og da man kan addere  $q-1 = N$  i potenserne og da  $N-K = 2T$  får vi:
  - $G(x) = (x - a^1)(x - a^2)\dots(x - a^{2T})$  er altid en faktor i  $C(x)$

# Cyklistiske Reed-Solomon koder

- Kodningen af  $[u_{K-1}, u_{K-2}, \dots, u_1, u_0]$  som et polynomium

$$u(x) = u_{K-1}x^{K-1} + u_{K-2}x^{K-2} + \dots + u_1x + u_0$$

kan gøres **ikke-systematisk** som  $C(x) = u(x)G(x)$  eller som vi tidligere har set **systematisk**

$$C(x) = u(x)x^{N-K} + \underset{G(x)}{\text{Rest}}(-u(x)x^{N-K})$$

- Her ses  $[u_{K-1}, u_{K-2}, \dots, u_1, u_0]$  direkte på de første K pladser
- Man kan eventuelt bruge en implementering med skifteregister som vist for cyklistiske koder, men multiplikationerne med koeficienterne i  $G(x)$  er jo noget mere indviklede. I  $F(2^m)$  er additionerne blot en række parallelle EXOR funktioner.
- Anvendes MATLAB datatypen `gf`, kan udregningen af Rest nemt gøres med `deconv`. Det undersøges i øvelsen.

# Fejlkorrigerende og fejldetekterende kodning

Kort oversigt over afsnittet

- Definition af et legeme
- Endelige legemer,  $F(p)$ , p primtal
- Primitivt element for legeme
- Endelige legemer,  $F(2^m)$ ,  $m > 1$
- Regning i endelige legemer, specielt  $F(2^m)$
- Diverse regneregler for endelige legemer
- Lineære koder over  $F(q)$ , **G**, **H**, afstand og vægt
- Definition af Reed-Solomon (RS) koder over  $F(q)$
- Minimumsvægt
- Cykliske RS koder, generatorpolynomium
- Systematisk kodning af RS koder

## ØVELSE #5: ENDELIGE LEGEMER OG REED-SOLOMON KODER

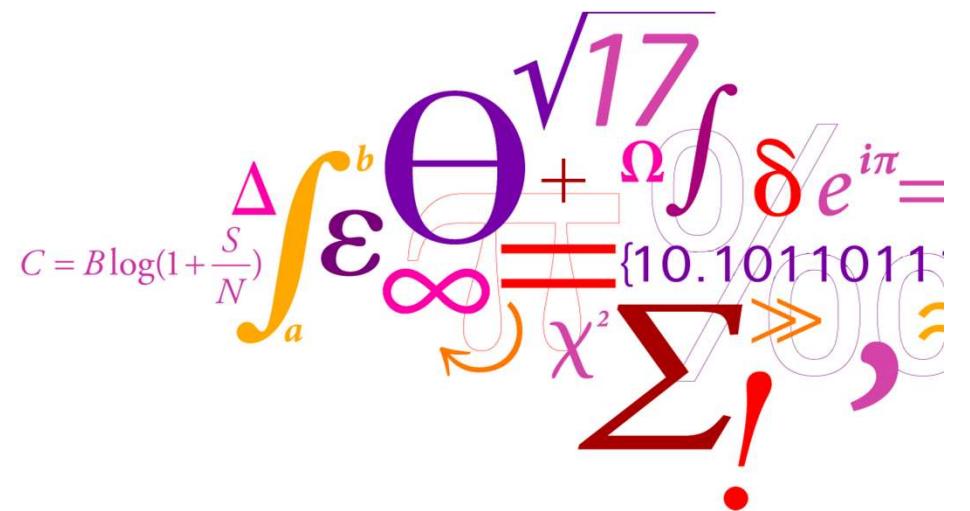
1. Lav en tabel over legemet  $F(32)$
2. MATLAB:
  1. Undersøg MATLABs funktioner for regning i endelige legemer (og sammenlign med tabellen fra punkt 1)
  2. Find generatorpolynomiet  $G(x)$  for den cykliske RS-kode (31,25) over  $F(32)$
  3. Lav et program til systematisk kodning af (31,25)
  4. Undersøg om et kodet ord har  $G(x)$  som faktor

# Dekodning af Reed-Solomon koder

**34220**

Knud J. Larsen

Slides: RSD

$$C = B \log\left(1 + \frac{S}{N}\right) \int_a^b \mathcal{E} \Theta + \Omega \int \delta e^{i\pi} =$$


# Fejlkorrigerende og fejldetekterende kodning

I denne del af kurset vil vi se på

- Fejldetection og -korrektion ved hjælp af koder
- Introducere begreber
- Hamming-kode eksempel
- Lineære koder – paritetscheckmatrix, syndromer, dekodning, minimumsvægt, generatormatrix, systematisk kodning
- Fejsandsynligheder ved anvendelse af blokkoder
- Foldningskoder og deres dekodning (Viterbi-algoritmen)
- Cykliske koder – polynomiumsbeskrivelse
- Endelige legemer og koder over disse
- Reed-Solomon koder og **deres dekodning**
- BCH-koder
- Simulering af koder på virkelige kanaler
- Sammensætning af simplere koder til stærke systemer

# Reed-Solomon Koder

## Repetition

- Polynomier over  $F(q)$  med grad  $< K$ :  $u(x)$ .
- $F(q)$ -koefficienterne i  $u(x)$  kan ses som de  $K$  **informationssymboler**,  $u_{K-1}, \dots, u_0$ :

$$u(x) = u_{K-1}x^{K-1} + u_{K-2}x^{K-2} + \dots + u_1x + u_0$$

- **Kodningen** gøres i første omgang ved at udregne værdierne af  $u(x)$  i  $N \leq q$  forskellige elementer,  $x_i$ , i  $F(q)$ :

$$[c_{N-1}, c_{N-2}, \dots, c_1, c_0] = [u(x_{N-1}), u(x_{N-2}), \dots, u(x_1), u(x_0)]$$

- Koden er lineær da  $au_1(x_i) + bu_2(x_i) = u(x_i)$  også er i koden
- Dimensionen er  $K$ :
  - $\leq K$  da der er  $K$  koefficienter i  $u(x)$
  - to forskellige  $u_1(x)$  og  $u_2(x)$  giver forskellige kodeord da  $u_1(x) - u_2(x)$  er et polynomium med grad  $< K$  og det kan ikke være 0 for alle  $x_i$  da det højest kan have  $K-1$  rødder.
- Minimumsafstanden er  $d = N - (K-1) = N - K + 1$  da  $u(x)$  højest har  $K-1$  rødder
- Koden kan altså rette  $T = (d - 1)/2 = (N - K)/2$  fejl

# Dekodning af Reed-Solomon koder

- Vi modtager  $\mathbf{r} = \mathbf{c} + \mathbf{e}$  hvor  $\mathbf{c}$  er et kodeord ( $= [u(x_i)]$ , for  $i = 0, \dots, N-1$ ) og  $\mathbf{e}$  er et fejlmønster med højst  $T = (N-K)/2$  ikke-0 positioner
- Den nyeste idé for dette er at bestemme et såkaldt interpolationspolynomium i to variable

$$Q(x, y) = Q_0(x) + yQ_1(x)$$

som skal få  $\mathbf{r}$  til at passe så godt som muligt i de punkter  $x_i$  hvor vi udregnede kodeordet  $c$ , dvs. vi vil have

$$Q(x_i, r_i) = 0 \quad \text{for } 0 \leq i \leq N-1$$

$$\ell_0 = \text{grad}(Q_0) \leq N - T - 1 \quad \ell_1 = \text{grad}(Q_1) \leq N - K - T = T$$

dvs.  $Q(x, y)$  har højst  $N+1$  ikke-0 koefficienter:

$$(N - T - 1 + 1) + (N - K - T + 1) = N + 1 \text{ da } 2T = N - K.$$

- Det kan lade sig gøre da der er  $N$  ligninger  $Q(x_i, r_i) = 0$  som skal give de  $N+1$  ubekendte koefficienter. En matrixformulering er:

# Dekodning af Reed-Solomon koder

$$\begin{bmatrix} 1 & x_0 & {x_0}^2 & \cdots & {x_0}^{\ell_0} & r_0 & r_0x_0 & \cdots & {r_0x_0}^{\ell_1} \\ 1 & x_1 & {x_1}^2 & \cdots & {x_1}^{\ell_0} & r_1 & r_1x_1 & \cdots & {r_1x_1}^{\ell_1} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & {x_{N-1}}x_{N-1}^2 & \cdots & {x_{N-1}}^{\ell_0}r_{N-1} & r_{N-1}x_{N-1} & \cdots & {r_{N-1}x_{N-1}}^{\ell_1} \end{bmatrix} \begin{bmatrix} Q_{0,0} \\ Q_{0,1} \\ Q_{0,2} \\ \vdots \\ Q_{0,\ell_0} \\ Q_{1,0} \\ Q_{1,1} \\ \vdots \\ Q_{1,\ell_1} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

Da der er N ligninger med N+1 ubekendte kan vi vælge en ubekendt frit

# Dekodning af Reed-Solomon koder

Da der er  $N$  ligninger med  $N+1$  ubekendte kan vi vælge en ubekendt frit.  
 Vi vælger  $Q_{1,\ell_1} = 1$  og får så et kvadratisk ligningssystem ved at flytte  $Q_{1,\ell_1}$   
 multipliceret med den sidste søjle over på den anden side af =:

$$\begin{bmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^{\ell_0} & r_0 & r_0x_0 & \cdots & r_0x_0^{\ell_1-1} \\ 1 & x_1 & x_1^2 & \cdots & x_1^{\ell_0} & r_1 & r_1x_1 & \cdots & r_1x_1^{\ell_1-1} \\ \vdots & \vdots & \ddots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{N-1} & x_{N-1}^2 & \cdots & x_{N-1}^{\ell_0} & r_{N-1} & r_{N-1}x_{N-1} & \cdots & r_{N-1}x_{N-1}^{\ell_1-1} \end{bmatrix} \begin{bmatrix} Q_{0,0} \\ Q_{0,1} \\ Q_{0,2} \\ \vdots \\ Q_{0,\ell_0} \\ Q_{1,0} \\ Q_{1,1} \\ \vdots \\ Q_{1,\ell_1} \end{bmatrix} = \begin{bmatrix} -r_0x_0^{\ell_1} \\ -r_1x_1^{\ell_1} \\ \vdots \\ -r_{\ell_0}x_{\ell_0}^{\ell_1} \\ -r_{\ell_0+1}x_{\ell_0+1}^{\ell_1} \\ \vdots \\ -r_{N-1}x_{N-1}^{\ell_1} \end{bmatrix}$$

Det er ikke helt sikkert at dette kan løses for mindre end  $T$  fejl og så må  
 $Q_{1,\ell_1} = 0$  og processen gentages med  $Q_{1,\ell_1-1}$ . For  $T$  fejl kan det altid løses.

MATLAB har funktioner til løsning af et sådant system: `gflineq` for et  
 primtalslegeme  $F(p)$  og `mldivide` for  $F(2^m)$

# Dekodning af Reed-Solomon koder

- Antag at vi har højst  $T$  fejl
- Så må  $r_i = u(x_i)$  i mindst  $N-T$  positioner dvs.  $Q(x_i, u(x_i)) = 0$  for  $N-T$  værdier af  $x_i$  og da graden af  $Q(x, u(x)) \leq (K-1) + (N-K-T) = N-T-1$  er 1 mindre end antallet,  $N-T$ , af rødder så må  $Q(x, u(x))$  være 0-polynomiet:

$$0 = Q(x, u(x)) = Q_0(x) + u(x)Q_1(x) \Rightarrow u(x) = -\frac{Q_0(x)}{Q_1(x)}$$

- Når vi har fundet  $Q$  (som på foregående slide), kan vi altså finde den sendte information hvis der ikke er mere end  $T$  fejl
- Indsættes denne sammenhæng  $Q_0(x) = -u(x)Q_1(x)$  i  $Q(x, r) = 0$  fås

$$0 = Q(x_i, r_i) = -u(x_i)Q_1(x_i) + r_iQ_1(x_i) = (r_i - u(x_i))Q_1(x_i)$$

- dvs.  $Q_1(x)$  har rødder hvor  $r \neq u(x)$ , altså netop der hvor fejlene er
- Derfor kaldes  $Q_1(x)$  **fejllokeringspolynomiet**

# Cyklistiske Reed-Solomon koder

Repetition

- Vi vælger  $x_i = \alpha^i$ ,  $i = 0 \dots N-1 = q-2$  i  $F(q)$
- Kodeordet bliver altså

$$[c_{N-1}, c_{N-2}, \dots, c_1, c_0] = [u(\alpha^{N-1}), u(\alpha^{N-2}), \dots, u(\alpha^1), u(\alpha^0)]$$

- eller som polynomium

$$C(x) = \sum_{i=0}^{N-1} c_i x^i = \sum_{i=0}^{N-1} \sum_{j=0}^{K-1} u_j \alpha^{ji} x^i$$

- Lad os undersøge rødderne  $\beta$  i  $C(x)$ :

$$C(\beta) = \sum_{i=0}^{N-1} \sum_{j=0}^{K-1} u_j \alpha^{ji} \beta^i = \sum_{j=0}^{K-1} u_j \sum_{i=0}^{N-1} (\alpha^j \beta)^i$$

$$\begin{cases} 0 \text{ hvis for alle } j : \alpha^j \beta \neq 1, \text{ dvs. } \beta \neq \alpha^0, \alpha^{-1}, \dots, \alpha^{1-K} \\ -u_j \text{ hvis } j \text{ findes så } \alpha^j \beta = 1, \text{ dvs. } \beta = \alpha^0, \alpha^{-1}, \dots, \alpha^{1-K} \end{cases}$$

- $C$  har altså altid  $\alpha^{-K}, \alpha^{-K-1}, \dots, \alpha^{1-N}$  som rødder og da man kan addere  $q-1 = N$  i potenserne og da  $N-K = 2T$  får vi:
- $G(x) = (x - \alpha^1)(x - \alpha^2) \dots (x - \alpha^{2T})$  er altid en faktor i  $C(x)$
- Da alle  $\alpha^j$  er rødder  $x^{q-1} - 1 = x^N - 1$  bliver koden **cyklistisk** da  $G(x)|(x^N - 1)$

# Dekodning af cykliske Reed-Solomon koder



- Når  $x_i = \alpha^i$ ,  $i = 0 \dots N - 1 = q-2$  i  $F(q)$ , bliver koden cyklistisk som vist
- I det tilfælde kan man eliminere  $Q_0$  af ligningssystemet og efter en del regninger kommer man til en simpel ligning for de  $T+1$  koefficienter i  $Q_1(x)$ :

$$\begin{bmatrix} S_1 & S_2 & \cdots & S_{T+1} \\ S_2 & S_3 & \cdots & S_{T+2} \\ \vdots & & & \vdots \\ S_T & S_{T+1} & \cdots & S_{2T} \end{bmatrix} \begin{bmatrix} Q_{1,0} \\ Q_{1,1} \\ \vdots \\ Q_{1,T} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

hvor  $S_i = r(\alpha^i) = c(\alpha^i) + e(\alpha^i)$ ,  $1 \leq i \leq 2T$ , er **syndromer** som kun afhænger af fejlene da  $c(\alpha^i) = 0$ ,  $1 \leq i \leq 2T$ . Her har vi igen én ligning mindre end antallet af ubekendte, så samme trick som ved den store matrix giver en løsning.

# Dekodning af cykliske Reed-Solomon koder



Vi får

$$\begin{bmatrix} S_1 & S_2 & \cdots & S_T \\ S_2 & S_3 & \cdots & S_{T+1} \\ \vdots & & & \vdots \\ S_T & S_{T+1} & \cdots & S_{2T-1} \end{bmatrix} \begin{bmatrix} Q_{1,0} \\ Q_{1,1} \\ \vdots \\ Q_{1,T-1} \end{bmatrix} = \begin{bmatrix} -S_{T+1} \\ -S_{T+2} \\ \vdots \\ -S_{2T} \end{bmatrix}$$

Hvis der er mindre end  $T$  fejl, har dette system måske ikke en løsning, men så tricket gentages med  $Q_{1,T} = 0$  og  $Q_{1,T-1} = 1$ .

- Fejlpositionerne,  $f$ , findes som tidligere som rødder af i  $Q_1(x)$

# Dekodning af cykliske Reed-Solomon koder

- Fejlværdierne,  $e_{f_1}, \dots, e_{f_T}$  kan findes ved at løse T lineære ligninger:

$$S_i = \sum_{j=1}^T e_{f_j} \alpha^{if_j} \text{ for } i = 1, \dots, T$$

$$\begin{bmatrix} \alpha^{f_1} & \alpha^{f_2} & \dots & \alpha^{f_T} \\ \alpha^{2f_1} & \alpha^{2f_2} & \dots & \alpha^{2f_T} \\ \vdots & & & \vdots \\ \alpha^{Tf_1} & \alpha^{Tf_2} & \dots & \alpha^{Tf_T} \end{bmatrix} \begin{bmatrix} e_{f_1} \\ e_{f_2} \\ \vdots \\ e_{f_T} \end{bmatrix} = \begin{bmatrix} S_1 \\ S_2 \\ \vdots \\ S_T \end{bmatrix}$$

- Der findes to smarte metoder til at finde  $Q_1$  ved iteration uden at skulle invertere matricen: **Berlekamp-Massey algoritmen** og anvendelse af **Euklids** algoritme. Disse giver også en simplere metode til at finde fejlværdierne

# Dekodning af cykliske Reed-Solomon koder



Eksempel – RS(N,N-2) Enkeltfejl korrigende RS

- $S_1 = r(\alpha) = c(\alpha) + e(\alpha) = e_f \alpha^f$  og
- $S_2 = r(\alpha^2) = c(\alpha^2) + e(\alpha^2) = e_f \alpha^{2f}$ , er de 2 syndromer
- Man ser da umiddelbart hvordan fejlpositionen,  $f$ , bestemmes:

$$\alpha^f = \frac{S_2}{S_1}$$

- Dette er også løsningen hvis vi opstiller syndrommatricen og finder roden  $\alpha^f$  i fejllokeringspolynomiet  $Q(x)$ :

$$\begin{bmatrix} S_1 & S_2 \end{bmatrix} \begin{bmatrix} Q_{1,0} \\ Q_{1,1} \end{bmatrix} = \begin{bmatrix} S_1 & S_2 \end{bmatrix} \begin{bmatrix} Q_{1,0} \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \Rightarrow Q(x) = x - \frac{S_2}{S_1} \Rightarrow \alpha^f = \frac{S_2}{S_1}$$

- I begge udgaver udregnes fejlværdien,  $e_f$  som:

$$e_f = S_1 \alpha^{-f} = \frac{S_1^2}{S_2}$$

- Der kan også regnes ret direkte for  $T = 2$  og  $3$ , men for  $T > 3$  bliver det sværere

# Fejsandsynlighed Reed-Solomon koder

- Lad os antage at legemet er  $F(2^m)$  dvs med  $2^m$  som hver kan repræsenteres med  $m$  bit. Sandsynlighed for fejl i en bit kaldes  $p$  og da der er  $m$  bit fås sandsynligheden for **fejl i et symbol  $P = mp$**  (tilnærmelsesvist, der kunne være flere fejl i et symbol)
- Hvis koden kan rette  $T$  symbolfejl bliver sandsynligheden for at der ikke kan rettes eller at der rettes noget forkert

$$P_e = 1 - \sum_{j=0}^T \binom{N}{j} P^j (1-P)^{N-j} \approx \binom{N}{T+1} P^{T+1} (1-P)^{N-T-1} \quad \text{hvor } \binom{N}{j} = \frac{N!}{j!(N-j)!}$$

- Ofte er man interesseret i bitfejsandsynligheden,  $P_b$ , og ovenstående tilnærmelse (som gælder når  $PN \ll T$ ) svarer til at der stort set er  $T+1$  symbolfejl og hvis  $p$  ikke er alt for stor er der ca. 1 bitfejl pr fejlagtigt symbol. Man får så

$$P_b \approx \frac{T+1}{mN} \binom{N}{T+1} P^{T+1} (1-P)^{N-T-1}$$

# Fejlkorrigerende og fejldetekterende kodning

Kort oversigt over dette afsnit

- Repetition af Reed-Solomon (RS) koder over  $F(q)$
- Dekodning ved interpolation
- Fejllokeringspolynomium
- Repetition af cykliske Reed-Solomon (RS) koder
- Bestemmelse af fejllokeringspolynomium for cykliske RS-koder ved syndromer
- Uregning af fejlværdier for cykliske RS-koder ved syndromer
- Berlekamp-Massey og Euklids algoritmer nævnes
- Fejsandsynlighed for RS-koder over  $F(2^m)$  anvendt på BSC

## ØVELSE #6: DEKODNING AF REED-SOLOMON KODER

### 1. MATLAB:

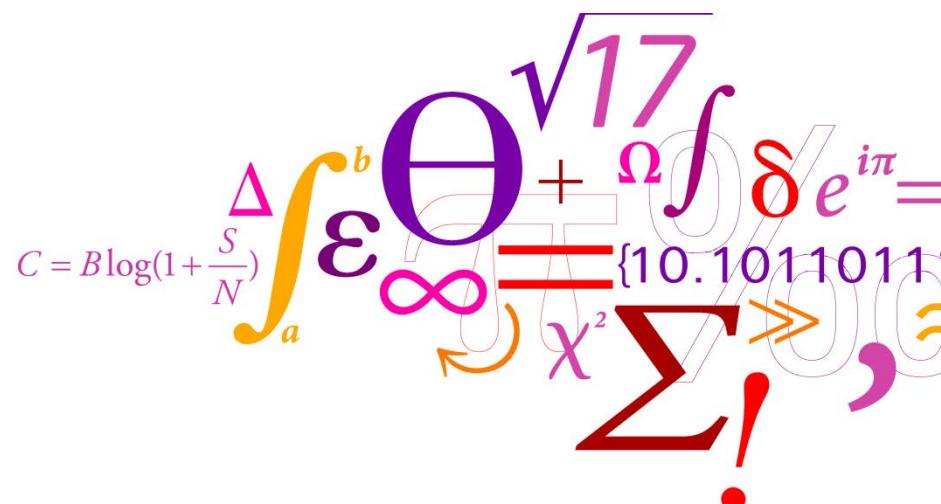
1. Vi forsætter med cykliske RS-kode (31,25) over  $F(32)$  fra Øvelse #5 og indlægger 3 fejl i et kodeord
2. Udregn syndromer for kodeordet med fejl
3. Bestem fejllokeringspolynomiet  $Q_1(x)$  og dets rødder (fejlenes positioner)
4. Find værdierne for de 3 fejl ved at løse et lineært ligningssystem

# Simulering af fejlkorrektion med foldnings- og blokkoder

34220

Knud J. Larsen

Slides: SIM

$$C = B \log\left(1 + \frac{S}{N}\right)$$


# Fejldkorrigende og fejlddetekterende kodning

I denne del af kurset vil vi se på

- Fejlddetektion og -korrektion ved hjælp af koder
- Introducere begreber
- Hamming-kode eksempel
- Lineære koder – paritetscheckmatrix, syndromer, dekodning, minimumsvægt, generatormatrix, systematisk kodning
- Fejsandsynligheder ved anvendelse af blokkoder
- Foldningskoder og deres dekodning (Viterbi-algoritmen)
- Cykliske koder – polynomiumsbeskrivelse
- Endelige legemer og koder over disse
- Reed-Solomon koder og deres dekodning
- **Simulering af koder på virkelige kanaler**
- BCH-koder
- Sammensætning af simplere koder til stærke systemer

# Fejlkorrigerende og fejldetekterende kodning

## Beregninger og simuleringer

- Vi har tidligere vist, hvordan man ud fra en sandsynlighed for fejl i en transmitteret sekvens kan **beregne** hvor stor sandsynligheden er for at en **BDD-dekoder** ikke dekoder til det rigtige eller giver op (vi skelnede ikke mellem disse og det er straks en del sværere)
- Ligeledes regnede vi også på sandsynligheden for at en **fejldetekterende kode** accepterer en sekvens givet en sandsynlighed for fejl
- I begge tilfælde antog vi at fejlene er uafhængige af hinanden (det er også det sværeste og de fleste koder er baseret på denne antagelse. Der findes specielle koder for sammenhængende fejl, såkaldte bygeføj, men ofte spredes man de modtagne sekvenser ud så de ser ud som tilfældige fejl)
- Vi kunne ikke umiddelbart regne sandsynligheden ud for en **ML-dekoder** hverken når den arbejder med hårde beslutninger eller - som vi demonstrerede for en Viterbi-dekoder - brugte bløde beslutninger for en AWGN-kanal
- Vi vil derfor gerne relatere **den fysiske virkelighed**, støj på virkelige kanaler, til kodningsystemers resulterende fejsandsynligheder, specielt for ML-dekodere

# Fejlkorrigerende og fejldetekterende kodning

## Beregninger og simuleringer

- Til det formål opstiller vi en model for støj på virkelige kanaler (AWGN)
- Indfører begrebet signal/støjforhold (SNR)
- Diskuterer den såkaldte kodningsgevinst, dvs. reduktion i nødvendigt SNR for en given resulterende fejlsandsynlighed ved anvendelse af fejlkorrigerende kodning
- Da beregninger ofte er umulige eller i hvert fald meget komplicerede giber man til **simulering af virkelige kanaler og kodningssystemer**
- Vi vil demonstrere dette i MATLAB, specielt i øvelsen
- Et værktøj er pseudotilfældige sekvenser

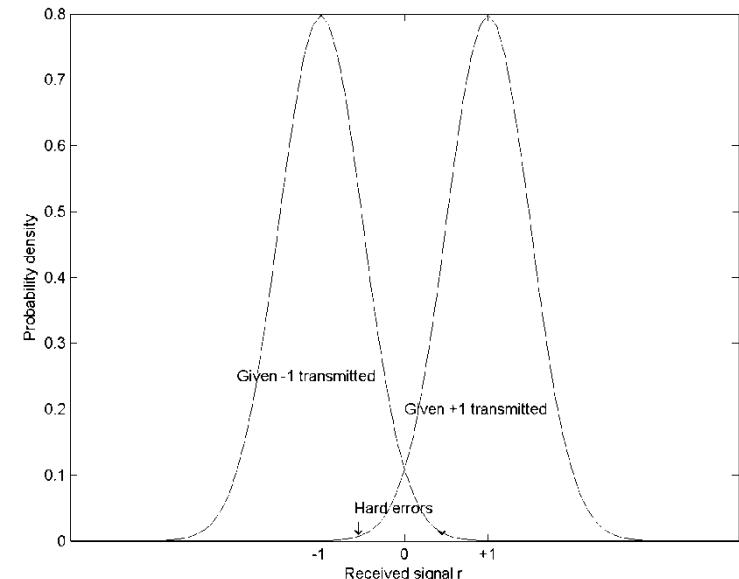
# Virkelighedens transmissionskanaler

For kanalen med **additiv hvid Gaussisk støj, AWGN**, som kendes fra kursus 34210, vil det modtagne signal efter det tilpassede filter have en fordeling som vist:

Sandsynlighedsfordelingen (helt præcist en tæthedsfunktion) er:

$$p[r | c] = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(r-c)^2}{2\sigma^2}}$$

hvor  $\sigma^2$  er støjens styrke (varians) og  $c$  det transmitterede signal  $c(j) \in \{-1, +1\}$ , dvs. det binære signal  $\{0, 1\}$  fra en koder er ændret til et bipolært signal og  $r$  modtages – som vi så allerede for Viterbi-dekoderen



Fejlene indtræffer når det modtagne signal krydser grænse( $r$ ) til andre mulige signaler – her når 0 overskrides. Vi vil give et udtryk for dette som funktion af fysiske størrelser på kanalen.

## Virkelighedens transmissionskanaler

Støjens varians,  $\sigma^2$ , vil normalt være givet da støjen fremkommer under transmissionsprocessen, så den eneste mulighed for at påvirke fejlsandsynligheden er at ændre signalets styrke. Hvis vi stedet for  $\pm 1$  transmitterer  $\pm \sqrt{E}$  får vi sandsynligheden for en fejl i en hård beslutning – en BSC-kanal som vi tidligere har set:

$$p = 2 \cdot \frac{1}{2} \int_0^\infty \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(r-(-\sqrt{E}))^2}{2\sigma^2}} dr = Q\left(\frac{\sqrt{E}}{\sigma}\right) \quad \text{hvor } Q(x) = \int_x^\infty \frac{1}{\sqrt{2\pi}} e^{-\frac{n^2}{2}} dn$$

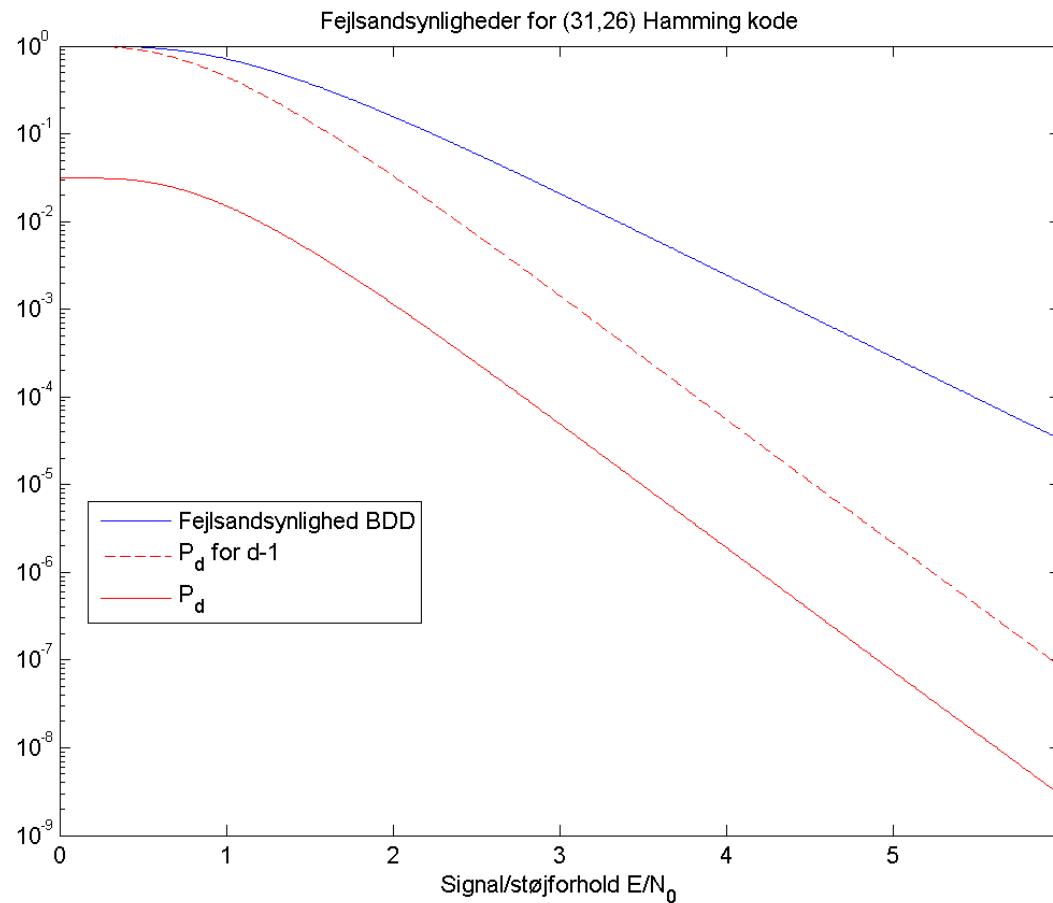
(2 pga. symmetri om 0,  $\frac{1}{2}$  er sandsynligheden for de to symboler  $\pm \sqrt{E}$ )  
 Støjens varians udtrykkes som regel ud fra styrken af støjens effektspektrum, der er helt fladt (hvidt) med styrken  $N_0/2 = \sigma^2$ . Indsættes det fås

$$p = Q\left(\sqrt{\frac{2E}{N_0}}\right)$$

hvor  $E/N_0$  kaldes **signal/støjforholdet**. I tekniske anvendelser angives signal/støjforholdet i dB og alle kurver over fejlsandsynligheder viser dem som funktion af dette:

$$\frac{E}{N_0} [\text{dB}] = 10 \log_{10} \frac{E}{N_0} \Leftrightarrow \frac{E}{N_0} = 10^{\frac{E/N_0 [\text{dB}]}{10}}$$

# Virkelighedens transmissionskanaler



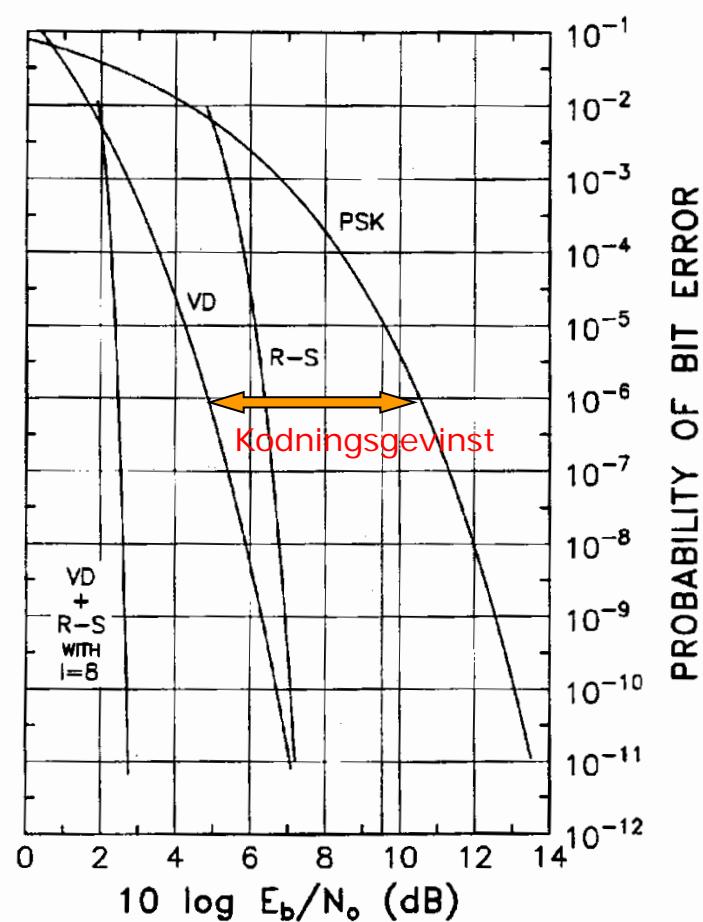
Den tidligere viste beregning for (31,26)-koden udtrykt som funktion af signal/støjforhold,  $E/N_0$ . Bemærk ”vandfaldsstrukturen” når signal/støjforhold er på abscissen.

# Virkelighedens transmissionskanaler

E er det transmitterede signals energi for hvert symbol, men da der for et system med fejlkorrektion anvendes noget af energien på redundansen er man ofte interesseret i et

- "effektivt" signal/støjforhold hvor man kun ser det som funktion af den energi,  $E_b$ , der anvendes på informationssymbolerne. Hvis koden har hastigheden R (som er  $<1$ ) får vi:
- $$\frac{E_b}{N_0} = \frac{E}{RN_0} \Leftrightarrow \frac{E_b}{N_0} [dB] = \frac{E}{N_0} [dB] - 10\log_{10} R$$

- Et eksempel på en foldningskode (VD for Viterbi dekodning, M=6, n=2) og en RS-kode ((N,K)=(255,223), m=8) er vist. (De kan også sættes sammen – mere herom senere). Her er også vist **kodningsgevinsten**, den reduktion i det nødvendige  $E_b/N_0$  man opnår ved kodning.



(R-S kurven er beregnet, VD og den sammensatte, VD+R-S, er simulerede)

# Simulering af transmissionskanaler

Vi ønsker at anvende MATLAB til simulering. Der er brug for 6 moduler

1. En kilde til frembringelse af tilfældige data. MATLAB funktionen `randi(2,1,L)-1` frembringer en vektor med  $L$  tilfældige 0 og 1. Et lignende kald til denne funktion vil kunne give tilfældige værdier i  $F(p)$  som kunne anvendes til RS simuleringer. De tilfældige 0 og 1 værdier kan også frembringes som pseudotilfældig sekvens – mere senere.
2. En koder
3. En transmissionskanal med AWGN. Det vil være smart at den konverterer fra  $\{0,1\}$  til  $c \in \{-1,+1\}$  og at man kan kontrollere signal/støjforholdet  $E/N_0 = 1/(2\sigma^2)$  (endnu smartere hvis det kan angives i dB). Her vil funktionen `sigma*randn(1,L)` være til stor hjælp. Dens værdier adderes så blot til  $c$ .
4. En modtager der laver hårde beslutninger hvis det er en sådan dekoder der simuleres eller videreleverer bløde værdier eventuelt kvantiserede i Q niveauer.
5. En dekoder
6. En optælling af fejlene i forhold til de  $L$  data og udregning af hyppighed

Bemærk at  $L$  skal være temmelig stor for at få pålidelige resultater for gode signal/støjforhold. Det vil være godt at se mindst 10 fejl, bedre med 100. Man må forvente at resultatet ved forskellige kørsler svinger mindst  $\sqrt{\text{fejl}}$ .

# Pseudotilfældige sekvenser

I eksperimenter anvender man ofte tilfældige sekvenser som i praksis ikke er helt tilfældige, men deterministiske og gentager sig efter en længde, N.

- En helt tilfældig sekvens af en længde N vil i gennemsnit have  $N/2$  0'er og  $N/2$  1'er
- men de skal selvfølgelig også komme uafhængigt af hinanden
- Dette kan man få et overblik over ved at se på hvor mange bit der bliver ens ved forskydninger af sekvensen. Man vil forvente at  $N/2$  bliver ens når sekvensen fortsættes efter N.
- En deterministisk sekvens der næsten opfylder disse krav er fx en bit i de 15 repræsentationer af  $a^0, \dots, a^{14}$  som udgør ikke-0 elementer i legemet  $F(16)$ :

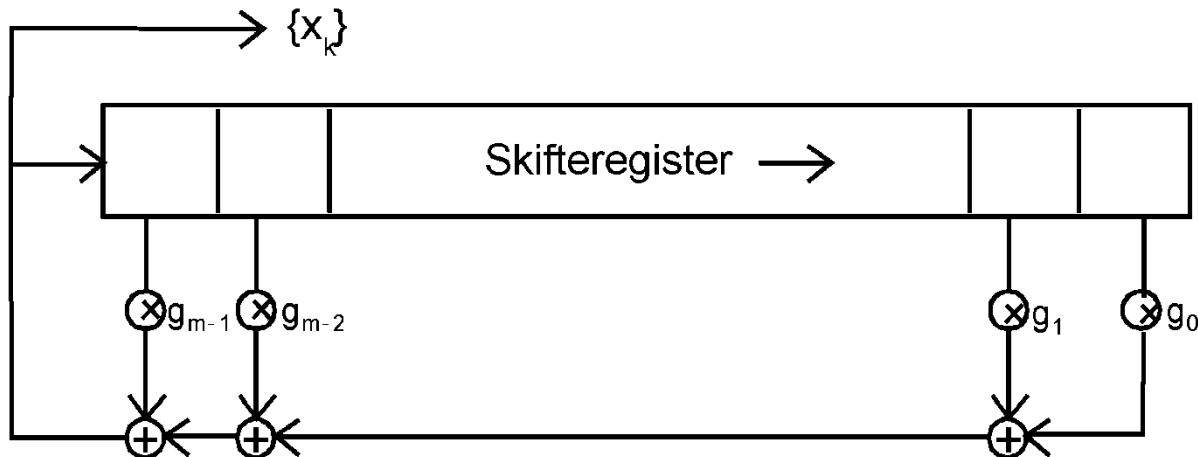
...11.000100110101111.000100110101111.000100110101111.00...

- Denne sekvens har 7 1'er og 8 0'er og alle forskydninger fra 1 til 14 har præcis 7 overensstemmelser med den originale sekvens.

# Pseudotilfældige sekvenser – m-sekvenser

Sekvenser som den nævnte kan nemt frembringes:

$$x_k = \sum_{n=1}^m g_{m-n} x_{k-n} = g_{m-1} x_{k-1} + g_{m-2} x_{k-2} + \dots + g_1 x_{k-m+1} + g_0 x_{k-m} \bmod 2$$



Hvis et register med  $m = 4$  startes 1111 frembringes den netop viste sekvens. Det er klart at den længst mulige periode vil være  $2^m - 1$  da skifteredisteret så gennemløber alle tilstande  $\neq 0\dots 0$ .

Derfor kaldes disse sekvenser **maksimallængdesekvenser** eller blot **m-sekvenser**.

De er også kendt under andre navne: PRBS eller LFSR

## Pseudotilfældige sekvenser – m-sekvenser

- Hvordan skal g-koefficienterne så vælges?
- Ikke overraskende skal de vælges som primitive polynomier der genererer legemerne  $F(2^m)$
- Sekvensen der genereres vil være en bit i  $z^k \bmod G(z)$

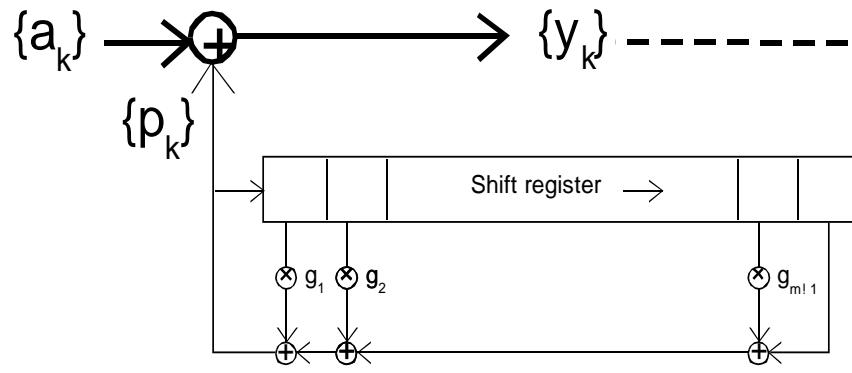
$$G(z) = \sum_{n=0}^m g_n z^n = z^m + g_{m-1} z^{m-1} + g_{m-2} z^{m-2} + \dots + g_1 z + g_0$$

m	Periode	Primitivt polynomium
3	7	$z^3 + z + 1$
4	15	$z^4 + z + 1$
5	31	$z^5 + z^2 + 1$
7	127	$z^7 + z + 1$
8	255	$z^8 + z^4 + z^3 + z^2 + 1$
12	4095	$z^{12} + z^6 + z^4 + z + 1$
16	65535	$z^{16} + z^{12} + z^3 + z + 1$

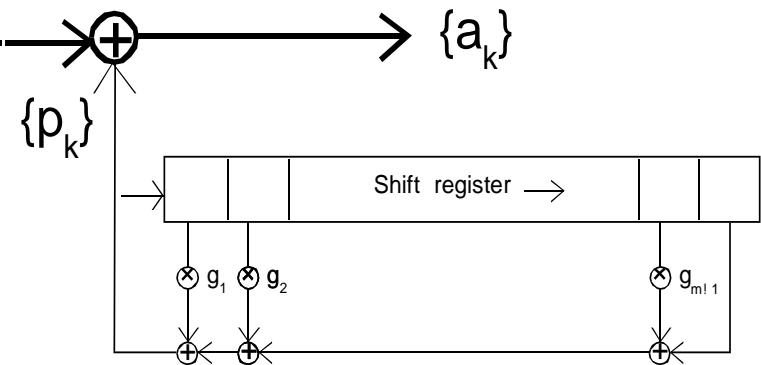
Forskellige egenskaber for maksimallængde sekvenser:

- m sammenhængende værdier,  $x_k x_{k+1} x_{k+m-1}$  antager alle N ikke-0 kombinationer præcis én gang
- Der er  $(N+1)/2$  1'er og  $(N-1)/2$  0'er
- Sekvensen er unik på nær en cyklisk forskydning (skift)
- Addition af to sekvenser forskudt cyklisk giver en tredje cyklisk forskydning af sekvensen
- I anvendelser gentages sekvensen periodisk. Alle forskydninger fra 1 til  $N-1$  har præcis  $(N-1)/2$  overensstemmelser med den originale sekvens
- Sekvenser med  $m = 7$  og/eller  $m = 23$  er ofte indbygget i apparater til bestemmelse af fejlsandsynligheder

En anden anvendelse: **Scrambling** –  
få en sekvens til at se tilfældig ud



Scrambling



Descrambling

# Fejlkorrigerende og fejldetekterende kodning

I dette afsnit så vi på

- Beregninger og simuleringer
- Model for støj på virkelige kanaler (AWGN)
- Signal/støjforhold
- Kodningsgevinst for eksempel
- Simulering af virkelige kanaler og kodningssystemer i MATLAB
- Pseudotilfældige sekvenser
- Maksimallængdesekvenser (m-sekvenser, LFSR, PRBS)
- Scrambling

## ØVELSE #7: SIMULERING AF KODNING PÅ KOMMUNIKATIONSKANALER

### 1. MATLAB:

1. M-sekvens med  $G(z) = z^7 + z + 1$
2. Kod en periode af sekvensen med en bestemt foldningskode (Øvelse #3)
3. En transmissionskanal med AWGN-støj (som kan reguleres i styrke)
4. En modtager med hårde beslutninger
5. MATLABs Viterbidekoder, vitdec (Øvelse #4)
6. En fejltæller som tæller fejl i en lang simulering ved  $E/N_0 = 0, 1$  og  $2 \text{ dB}$

# Binære BCH koder

34220

Knud J. Larsen

Slides: BCH

$$C = B \log\left(1 + \frac{S}{N}\right)$$
$$\int_a^b \mathcal{E} \Theta^{\sqrt{17}} + \Omega \int \delta e^{i\pi} =$$
$$\infty = \{10.10110110001001\}$$
$$\chi^2 \Sigma \gg,$$
$$!$$

# Fejldkorrigende og fejlddetekterende kodning

I denne del af kurset vil vi se på

- Fejlddetektion og -korrektion ved hjælp af koder
- Introducere begreber
- Hamming-kode eksempel
- Lineære koder – paritetscheckmatrix, syndromer, dekodning, minimumsvægt, generatormatrix, systematisk kodning
- Fejsandsynligheder ved anvendelse af blokkoder
- Foldningskoder og deres dekodning (Viterbi-algoritmen)
- Cykliske koder – polynomiumsbeskrivelse
- Endelige legemer og koder over disse
- Reed-Solomon koder og deres dekodning
- Simulering af koder på virkelige kanaler
- **BCH-koder**
- Sammensætning af simplere koder til stærke systemer

# Endelige legemer

Eksempel  $F(2^4) = F(16)$ ,  $P(z) = z^4 + z + 1$ , dvs  $z^4 = z + 1$

		$z^3$	$z^2$	$z^1$	$z^0$	
0	0	0	0	0	0	
$\alpha^0$	$z^0$	0	0	0	1	
$\alpha^1$	$z^1$	0	0	1	0	
$\alpha^2$	$z^2$	0	1	0	0	
$\alpha^3$	$z^3$	1	0	0	0	
$\alpha^4$	$z^4$	0	0	1	1	$z^4 \bmod z^4 + z + 1 = z + 1$
$\alpha^5$	$z^5$	0	1	1	0	
$\alpha^6$	$z^6$	1	1	0	0	
$\alpha^7$	$z^7$	1	0	1	1	$z^7 = z^6z = z^3z + z^2z = z^3 + z + 1$
$\alpha^8$	$z^8$	0	1	0	1	
$\alpha^9$	$z^9$	1	0	1	0	
$\alpha^{10}$	$z^{10}$	0	1	1	1	$z^4 = z + 1$
$\alpha^{11}$	$z^{11}$	1	1	1	0	
$\alpha^{12}$	$z^{12}$	1	1	1	1	$z^4 = z + 1$
$\alpha^{13}$	$z^{13}$	1	1	0	1	$z^4 = z + 1$
$\alpha^{14}$	$z^{14}$	1	0	0	1	$z^4 = z + 1$

Repetition

med  $\alpha = z$  som  
primitivt element:

$$\begin{aligned}
 \alpha^7\alpha^{10} &= \alpha^{17} = \alpha^2\alpha^{15} = \alpha^2 \\
 (z^3 + z + 1)(z^2 + z + 1) &= z^5 + z^4 + 1 \\
 &= (z + 1)(z^4 + z + 1) + z^2 \\
 &= z^2 \bmod P(z)
 \end{aligned}$$

# Endelige legemer – diverse

- Som tidligere omtalt findes kun endelige legemer for  $q = p^m$  hvor  $p$  er et primtal og  $m \geq 1$ . Her vil vi dog kun se på  $p = 2$ .
- Endelige legemer konstrueres ved hjælp af primitive polynomier,  $P(z)$ , med koefficienter i  $F(2)$  og grad  $m$ . Addition er modulo 2 for koefficienter og multiplikation er modulo  $P(z)$
- Her er der også et primitivt element  $a$  og alle ikke-0 elementer er  $a^i$
- Alle elementer  $\neq 0$  er rødder i  $x^{q-1} - 1$  da  $(a^i)^{q-1} = (a^{q-1})^i = 1$
- I ethvert endeligt legeme gælder at et  $n$ 'te grads polynomium højst har  $n$  rødder.
- En udregning til senere anvendelse i  $F(2^m)$ :

$$(a + b)^2 = a^2 + b^2 + 2ab = a^2 + b^2$$

Repetition

# Underkode til cykliske Reed-Solomon koder over $F(2^m)$

- Vi vælger en RS-kode med  $x_i = a^i$ ,  $i = 0 \dots N - 1 = 2^m - 2$  i  $F(2^m)$
- $G(x) = (x - a^1)(x - a^2) \dots (x - a^{2^T})$  er altid en faktor i  $C(x)$
- Da alle  $a^j$  er rødder  $x^{q-1} - 1 = x^N - 1$  bliver koden cyklistisk da  $G(x) | (x^N - 1)$
- Nogle af RS-kodeordene har alle symboler  $c_i$  i  $F(2)$ , dvs. de har  $m-1$  0'er i den binære repræsentation og sidste bit kan være 0 eller 1.
- Disse udgør en lineær kode (en underkode) da summen af to af disse kodeord selv er binær
- Længden af koden er  $N = 2^m - 1$  i bit da vi dropper de  $m-1$  0'er i hvert  $F(2^m)$ -symbol
- Den er selvfølgelig også cyklistisk så alle kodeord har en fast binær faktor,  $G_2(x)$ , som har rødderne  $a^1, a^2, \dots, a^{2^T}$  i  $F(2^m)$ .

# Binære BCH koder

- Spørgsmålet er: Hvordan kan  $G_2(x)$  findes?
- For ethvert element,  $\beta$ , i  $F(2^m)$  findes der et **binært** polynomium af mindste grad som har dette som rod. Dette kaldes **minimalpolynomiet** for  $\beta$ ,  $m_\beta(x)$ .
- Hvis  $m_\beta(x)$  har grad  $L$  vil alle dets rødder være  $\beta, \beta^2, \beta^4, \dots, \beta^{2^{L-1}}$  (husk at potenser i et endeligt legeme kan regnes modulo  $q-1 = 2^m-1$ . Dette skyldes at som vist  $(a+b)^2 = a^2+b^2$ . Så hvis  $\beta$  er rod, da er  $0 = (m_\beta(\beta))^2 = m_\beta(\beta^2)$ , så  $\beta^2$  er også rod osv.)
- De fleste minimalpolynomier har grad  $m$ , men nogle få har mindre grad (hvis  $2^m-1$  er et sammensat tal)
- Minimalpolynomiet for  $\alpha$  er det primitive polynomium der anvendtes til at opstille legemet, fx  $m_\alpha(x) = x^4 + x + 1$  for  $F(16)$ .

# Binære BCH koder

- Spørgsmålet er: Hvordan kan  $G_2(x)$  findes?
- Da  $G_2(x)$  skal have rødder  $a^1, a^2, \dots, a^{2T}$ , findes  $G_2(x)$  ud fra disses minimalpolynomier som er binære
- $G_2(x) = m_a(x)m_{a^3}(x)m_{a^5}(x)\dots$ , så mange som er nødvendige til at dække alle potenserne op til  $2T$ . For lidt større  $T$  kan der godt være færre end  $T$  af disse, da de mange rødder i  $m_\beta(x)$  måske allerede har "snuppet" en potens.
- Koden har altså længde  $N = 2^m - 1$  og antallet af informationssymboler bliver  $K = N - \text{grad}(G_2(x))$  og  $d \geq 2T + 1$  (den aktuelle  $d$  er svær at finde, men som regel er  $d = 2T + 1$ ).
- Graden af  $G_2(x)$  vokser for små  $T$  med  $m$  for hver faktor, men som omtalt ovenfor, dækker faktorerne flere  $a$ -potenser så nogle kommer med "gratis", og desuden har ikke alle minimalpolynomier grad  $m$ . Så længe alt er regelmæssigt fås koder med  $(N, K) = (2^m - 1, 2^m - mT - 1)$
- Disse koder kaldes **BCH-koder** efter opfinderne Bose, Chaudhuri og Hocquenghem

# Dekodning af BCH koder

- Dekodingsmetoderne for RS-koder virker selvfølgelig også for BCH-koder – husk at selv om det er binære koder skal man regne i  $F(2^m)$  undervejs, fx når syndromer skal udregnes. Fejlværdierne er dog ikke svære at finde, da kun 1 er mulig som fejl.
- Vi repeterer dekodningen med syndromer ( $T$  fejl antaget,  $F(2^m)$  – er nemt):

$$\begin{bmatrix} S_1 & S_2 & \cdots & S_T \\ S_2 & S_3 & \cdots & S_{T+1} \\ \vdots & & & \vdots \\ S_T & S_{T+1} & \cdots & S_{2T-1} \end{bmatrix} \begin{bmatrix} Q_{1,0} \\ Q_{1,1} \\ \vdots \\ Q_{1,T-1} \end{bmatrix} = \begin{bmatrix} S_{T+1} \\ S_{T+2} \\ \vdots \\ S_{2T} \end{bmatrix}$$

hvor  $S_i = r(\alpha^i) = c(\alpha^i) + e(\alpha^i)$ ,  $1 \leq i \leq 2T$ , er **syndromer** som kun afhænger af fejlene da  $c(\alpha^i) = 0$ ,  $1 \leq i \leq 2T$ . (Regnes i  $F(2^m)$ )

- Fejlpositionerne,  $f$ , findes som tidligere som rødder  $\alpha^f$  i  $Q_1(x)$
- Fejlværdierne,  $e_f$ , = 1
- Der findes to smarte metoder til at finde  $Q_1$  ved iteration uden at skulle invertere matricen: **Berlekamp-Massey** og anvendelse af **Euklids algoritme**. Der er dog nogle forenklinger for BCH.

# Dekodning af cyklisk Hamming kode

## Eksempel

BCH( $2^m - 1, 2^m - m - 1$ ) Enkeltfejl korrigerende BCH dvs. cyklisk Hamming

- $S_1 = r(a) = c(a) + e(a) = 1 \cdot a^f$  og det andet syndrom
- $S_2 = r(a^2) = c(a^2) + e(a^2) = 1 \cdot a^{2f}$  er slet ikke nødvendigt (og i øvrigt lig  $S_1^2$ )
- Man får da fejlpositionen,  $f$ :

$$a^f = S_1$$

- Det var netop sådan vi fandt fejl i en Hamming-kode. Fejlen var bestemt af syndromet som kunne antage de  $2^m - 1$  forskellige ikke-0 værdier  $a^f$  kan have når det repræsenteres som polynomium.

# Fejlkorrigerende og fejldetekterende kodning

Kort oversigt over dette afsnit

- Lille repetition af endelige legemer,  $F(2^m)$ ,  $m > 1$
- BCH-koder som underkoder til RS-koder
- Minimalpolynomier
- Generatorpolynomium for BCH-koder
- Dekodning af BCH-koder
- Hamming-kode som BCH-kode

# Sammensætning af koder

## Produktkoder med iterativ dekodning

### Lidt om turbo-koder

34220

Knud J. Larsen

Jakob Dahl Andersen

Slides: SAM

$$C = B \log\left(1 + \frac{S}{N}\right)$$
$$\int_a^b \mathcal{E} \Theta^{\sqrt{17}} + \Omega \int \delta e^{i\pi} =$$
$$\infty = \{10.101101100101001\}$$
$$\chi^2 \Sigma \gg,$$
$$!$$

DTU Fotonik

Department of Photonics Engineering

---

# Fejldkorrigende og fejlddetekterende kodning

I denne del af kurset vil vi se på

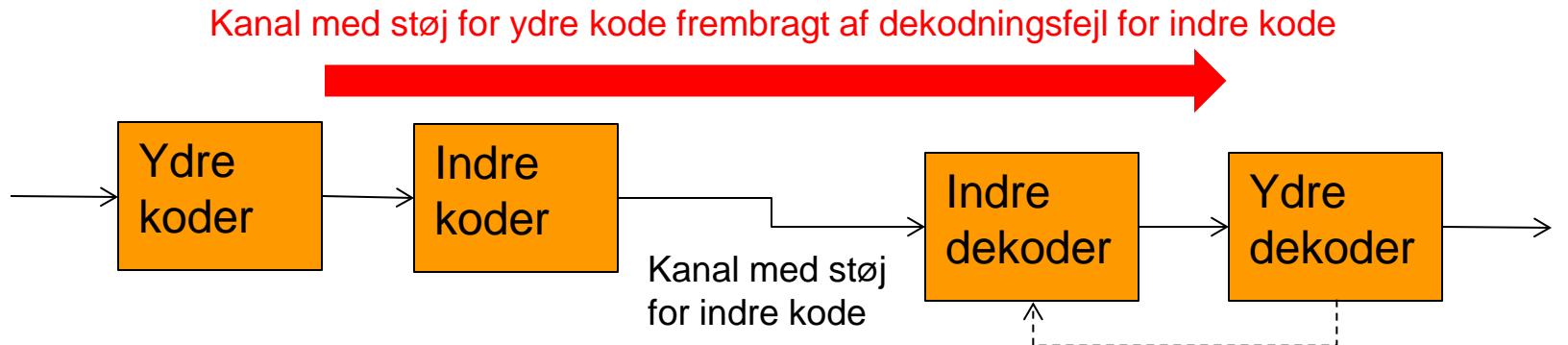
- Fejlddetektion og -korrektion ved hjælp af koder
- Introducere begreber
- Hamming-kode eksempel
- Lineære koder – paritetscheckmatrix, syndromer, dekodning, minimumsvægt, generatormatrix, systematisk kodning
- Fejsandsynligheder ved anvendelse af blokkoder
- Foldningskoder og deres dekodning (Viterbi-algoritmen)
- Cykliske koder – polynomiumsbeskrivelse
- Endelige legemer og koder over disse
- Reed-Solomon koder og deres dekodning
- Simulering af koder på virkelige kanaler
- BCH-koder
- **Sammensætning af simplere koder til stærke systemer**

# Gode koder og deres dekodning

- En fejlkorrigerende kode skal vælges ud fra karakteristika for transmissionskanalen og den ønskede ydelse, fx fejsandsynlighed
- Selvom det skulle være muligt at finde en kode er det langt fra sikkert at dekodningen kan foregå pga. kompleksitetsbegrænsninger
- Derfor er der opstået den idé at sætte flere simplere koder sammen til en såkaldt **sammensat kode**.
- Hver især har disse simple dekodingsalgoritmer og specielt hvis man kan etablere samarbejde mellem disse kan man opnå en meget stærk kombination.
- Vi vil se på 3 eksempler på sammensatte koder
  - Traditionel sammensætning af foldningskode og RS-koder
  - Produktkoder med iterativ dekodning
  - Turbo-koder med iterativ dekodning

# Traditionel sammensætning

- Traditionel sammensætning af en indre kode og en ydre kode



- Ofte anvendes en foldningskode med bløde beslutninger i Viterbi-dekoderen som indre kode
- Hver gang denne laver dekodningsfejl kommer fejlene i en byge indtil algoritmen er tilbage på rette spor
- Den ydre kode skal kunne håndtere disse og hvis ikke urimeligt lange koder skal anvendes spredes fejlene i en byge ved såkaldt *interleaving* mellem flere kodeord
- Ofte anvendes en RS-kode over  $F(2^m)$  som ydre kode og m-bit symbolerne klarer i sig selv mindre byger

Mulig  
forbedring ved  
iterationer

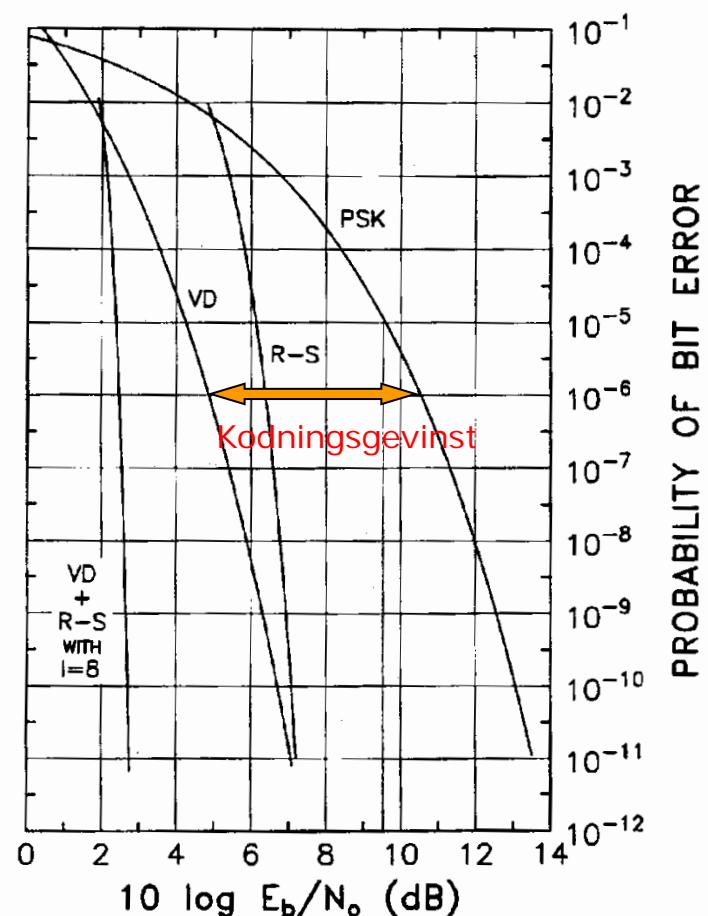
# Traditionel sammensætning

CCSDS standardkoder for satellitkommunikation

Foldningskoden (VD for Viterbi dekodning) har  $M=6$  og  $n=2$ . RS-koden har  $(N,K)=(255,223)$ ,  $m=8$  og kan således rette 16 fejlsymboler à 8 bit.

Disse sættes sammen til en stærk kombination. Her anvendes 8 RS-koder blandet så fejlbyger fra Viterbi dekoderen spredes bedst muligt. (I praksis er det 5 RS-koder men det ændrer ikke meget).

**Kodningsgevinsten**, den reduktion i det nødvendige  $E_b/N_0$  man opnår ved kodning, kommer helt op på ca. 8 dB.



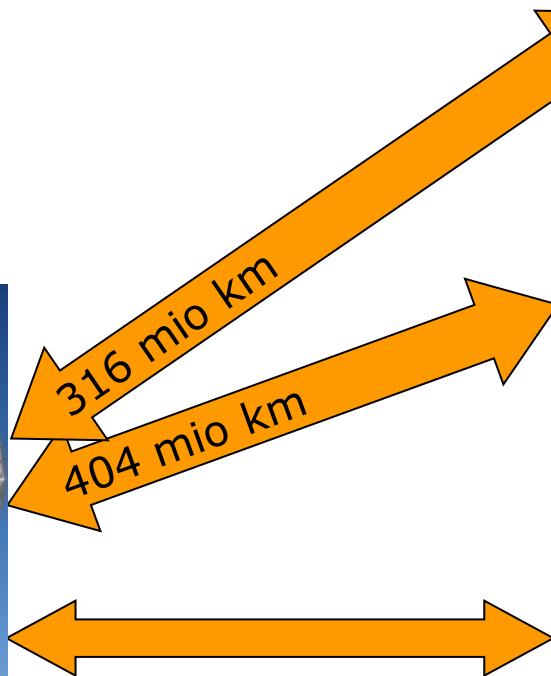
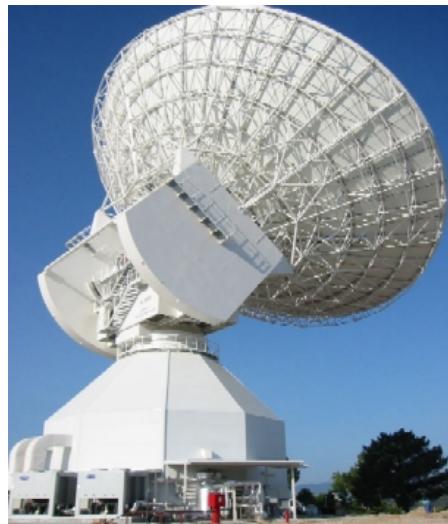
# Forbedring af den traditionelle sammensætning



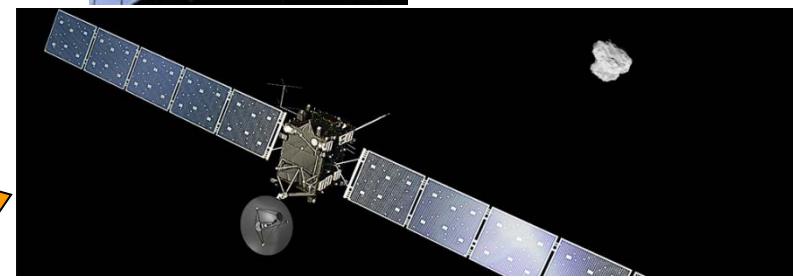
- Det sammensatte system kunne forbedres hvis der sendes information fra den ydre dekoder tilbage til den indre dekoder så denne information ved en gentagelse af den indre dekodning kan styrke beslutningerne så nogle fejlbygger kan forventes fjernet
- Dette virker specielt godt hvis nogle af de *interleavede* ydre kodeord kan korrigere mange fejl
- Som vi skal se i de næste par systemer indeholder de stærkeste kodningssystemer ofte sådanne iterative algoritmer

# Eksempel på anvendelse af fejlkorrigerende koder

Kommunikation med satellitter i rummet



Rosetta ved  
Mars marts  
2007



Ved kometen 67P/CG, aug. 2014



Modtagestation

ESA i Tyskland

# ESA Projekt

- Fejlkorrigerende kodning for rumforskning



Op til 400 mio km

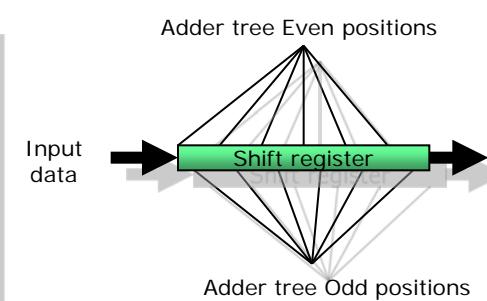
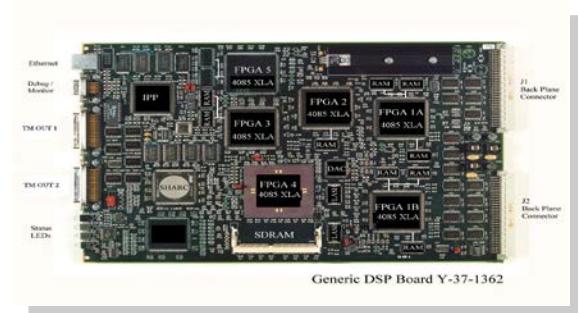
Når afstanden ikke er alt for stor anvendes det traditionelle sammensatte system



ESA satellit (her Rosetta)

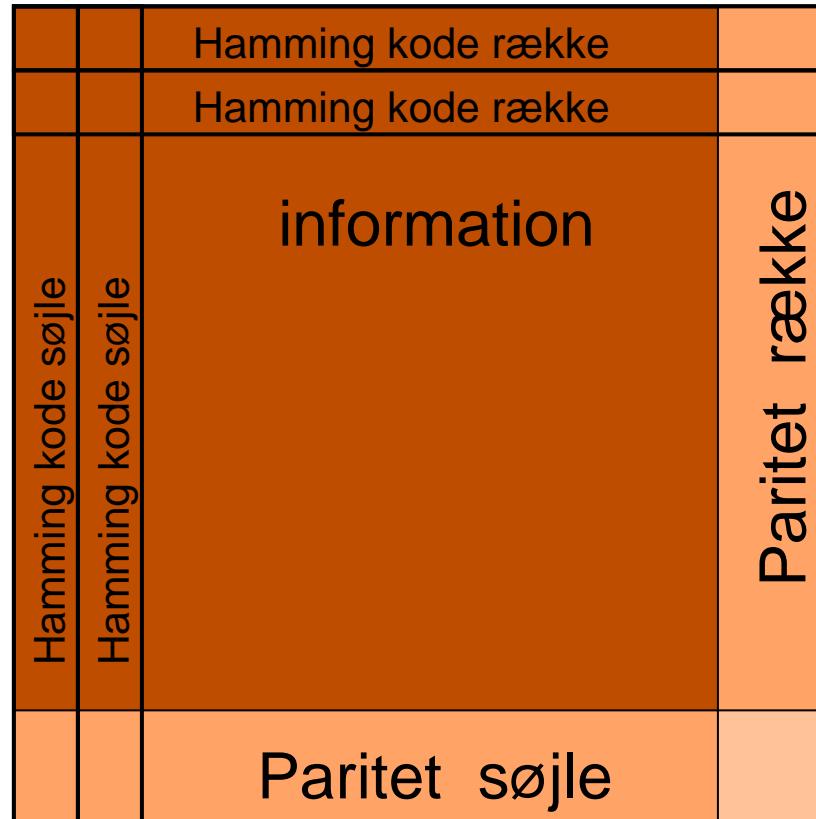
BAE Systems – DTU Fotonik  
dekoder som er standard i alle  
ESAs jordstationer

DTU Fotonik lavede  
"programmet" der er downloaded  
i en FPGA



# Produktkoder

Produktkode – sammensætning i to dimensioner  
(Her med en simpel Hamming-kode for at illustrere princip)



# Produktkoder

Produktkode – sammensætning i to dimensioner

		Hamming kode række	
		Hamming kode række	
		information	
		1	
		1      1	
		1      1	
		Paritet række	
		Paritet søjle	

Eksempel på iterativ dekodning af fejlmønster

# Produktkoder

Produktkode – sammensætning i to dimensioner

		Hamming kode række	
		Hamming kode række	
		information	
		1      1	
		1      1	
		Paritet række	
		Paritet søjle	

Eksempel på dekodning af fejlmønster, 1. iteration række med enkeltfejl rettet

# Produktkoder

Produktkode – sammensætning i to dimensioner

		Hamming kode række	
		Hamming kode række	
		information	
Hamming kode søjle	Hamming kode søjle	1            1            1	Paritet række
		1            1	
		Paritet søjle	

Eksempel på dekodning af fejlmønster, 1. iteration række med for mange fejl

# Produktkoder

Produktkode – sammensætning i to dimensioner

		Hamming kode række	
		Hamming kode række	
		information	
Hamming kode søjle	Hamming kode søjle	1            1            1	Paritet række
		1            1	1
		Paritet søjle	

Eksempel på dekodning af fejlmønster, 1. iteration række med for mange fejl

# Produktkoder

Produktkode – sammensætning i to dimensioner

		Hamming kode række	
		Hamming kode række	
		information	
		1      1	
		1      1	1 Paritet række
		Paritet søjle	

Eksempel på dekodning af fejlmønster, 1. iteration søjle, enkeltfejl rettet

# Produktkoder

Produktkode – sammensætning i to dimensioner

		Hamming kode række	
		Hamming kode række	
		information	
		1      1	
		1	1
		Paritet søjle	

Eksempel på dekodning af fejlmønster, 1. iteration søjle, enkeltfejl rettet

# Produktkoder

Produktkode – sammensætning i to dimensioner

		Hamming kode række	
		Hamming kode række	
		information	
		1	
		1      1	
		1	Paritet række
		1	
		Paritet søjle	

Eksempel på dekodning af fejlmønster, 1. iteration søjle, for mange fejl

# Produktkoder

Produktkode – sammensætning i to dimensioner

		Hamming kode række	
		Hamming kode række	
		information	
		1	
		1	
		1	
			Paritet række
		1	
			Paritet række

Eksempel på dekodning af fejlmønster, 1. iteration søjle, enkeltfejl rettet

# Produktkoder

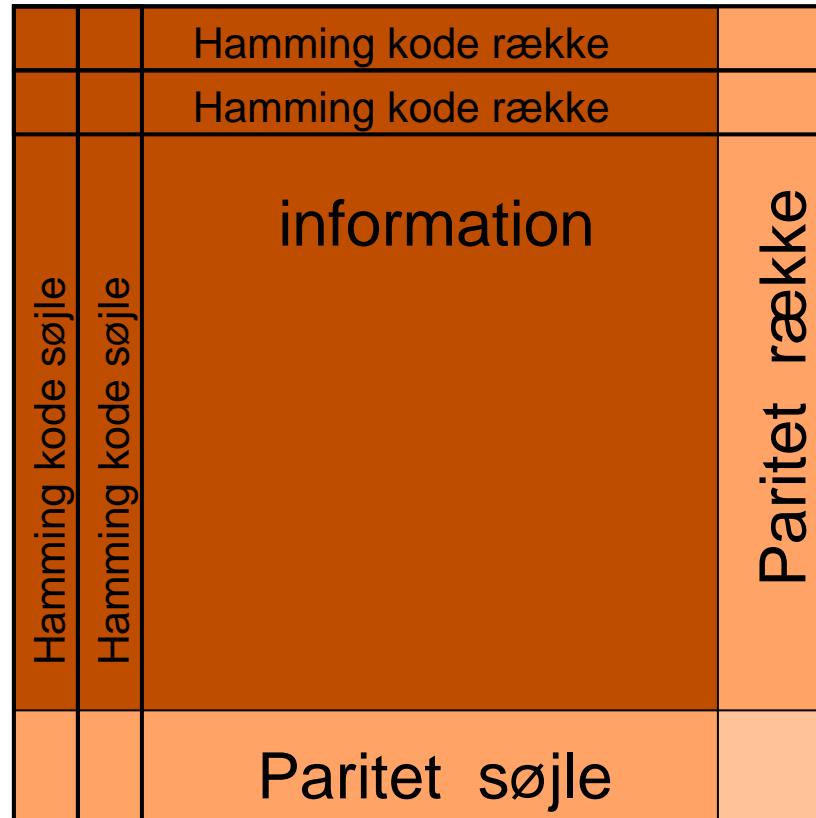
Produktkode – sammensætning i to dimensioner

		Hamming kode række	
		Hamming kode række	
		information	
		1	
		1	
		1	
			Paritet række
		Paritet søjle	

Eksempel på dekodning af fejlmønster, 1. iteration søjle, enkeltfejl rettet

# Produktkoder

Produktkode – sammensætning i to dimensioner



Eksempel på dekodning af fejlmønster, 2. iteration række, 3 enkeltfejl rettet

# Produktkoder

Produktkode – sammensætning i to dimensioner

		Hamming kode række	
		Hamming kode række	
		information	
		1	1
		1	1
			Paritet række
		Paritet søjle	

Produktkoden har  $d = 3 \cdot 3 = 9$  og skulle kunne rette 4 fejl, men den anvendte metode kan ikke klare fejlmønstre som det viste.

Til gengæld gik det nemt med at rette 5 fejl i eksemplet!

Eksempel på dekodning af fejlmønster, umuligt fejlmønster

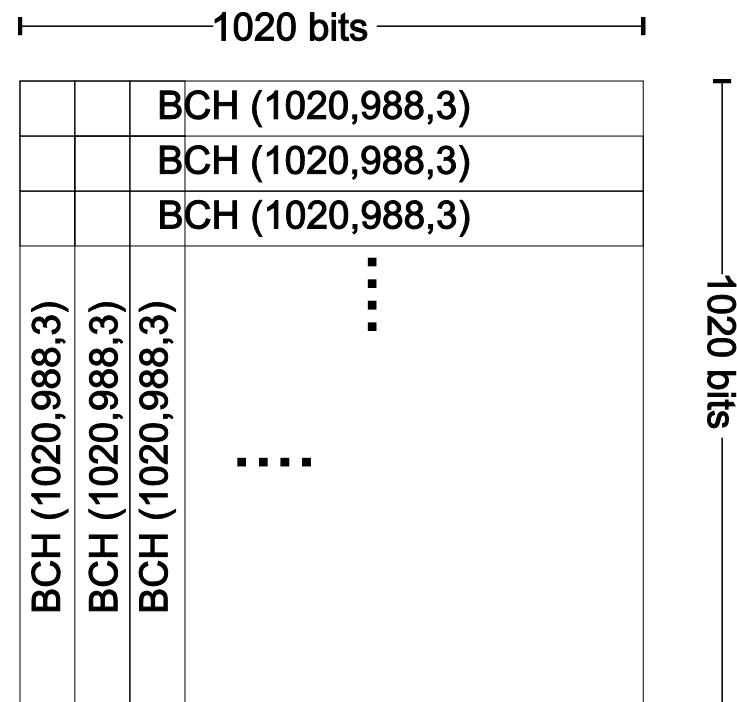
# Produktkoder

## Produktkode - anvendelse i optisk kommunikation

Til 40 Gbit/s optisk kommunikation har vi brugt en BCH kode der retter 3 fejl og detekterer 4,  $d = 8$

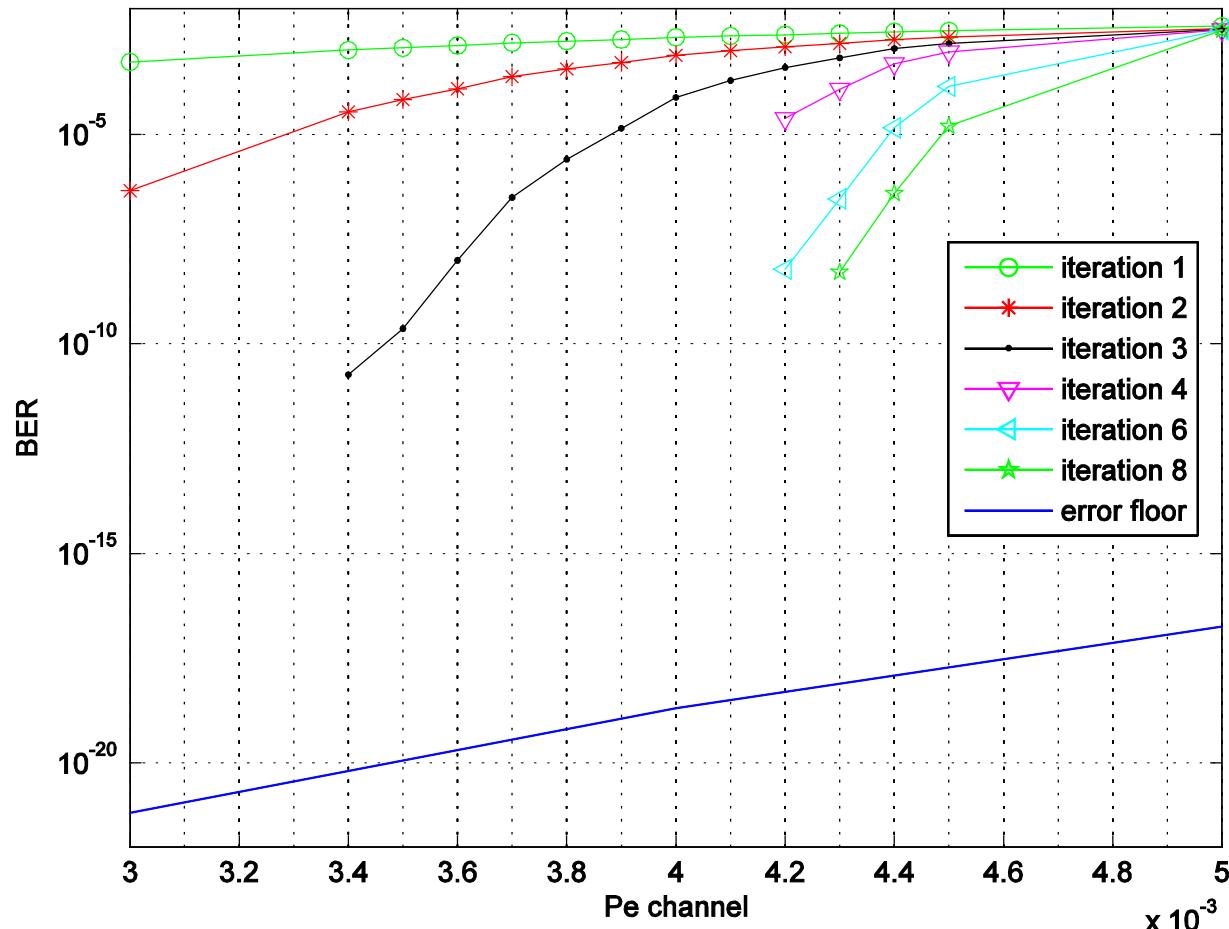
Produktkoden har  
976144 informationsbit  
64526 paritetsbit  
Dvs. et “overhead” på 6.6 %

Vi har implementeret en dekoder i én FPGA som virker op til 1600 Gbit/s



MATLAB demo af tærskel, DemoProd

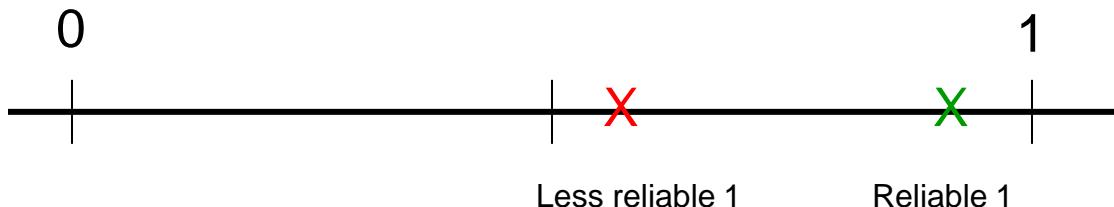
# Produktkoder



Input har fejl for hver 300 bit - output for mindre end hver 1000000000000000 bit  
Det er ca. en gang om dagen med 10 Gbit/s  
"Fejlgulvet" skyldes de umulige mønstre på 4 gange 4 fejl

# Turbo-koder

For produktkoderne vi lige har set foregik det hele i "hard-decision" (0 var 0 og 1 var 1). Man kunne forestille sig at det ville være meget bedre hvis vi havde "soft-decision", både som input fra kanalen og mellem de forskellige iterationer. Soft decision kan illustreres med en linje



Positionen X svarer til

$$LLR(u_l) = \log \frac{P\{u_l = 1\}}{P\{u_l = 0\}}$$

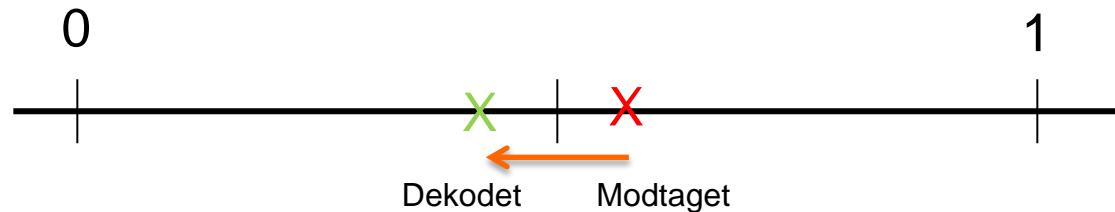
hvor vi tager logaritmen så pålidelige 1'er bliver et stort positivt tal og pålidelige 0'er et stort negativt tal. Desuden har det den fordel at vi umiddelbart kan addere forskellige bidrag i stedet for at skulle multiplicere sandsynligheder

$$LLR(u_l) = \log \frac{P\{u_l = 1, \mathbf{r}\}}{P\{u_l = 0, \mathbf{r}\}} = \log \frac{P_{apriori}\{u_l = 1\}}{P_{apriori}\{u_l = 0\}} + \log \frac{P\{\mathbf{r} | u_l = 1\}}{P\{\mathbf{r} | u_l = 0\}}$$

# Turbo-koder

Når vi dekoder (dele af) den fejlkorrigerende kode ændres vores estimat af  $P\{u_i=0\}$  og  $P\{u_i=1\}$ . Det svarer til at flytte krydset på linjen.

Man kan sige at vi indsamler information om vores bit og hver gang vi får ny information ændres vores mening om  $P\{u_i=0\}$  og  $P\{u_i=1\}$



Dekodning af blokkoder med hard-decision er virkelig elegant – men med soft-decision er det noget rod

For foldningskoder derimod har vi allerede set en smart måde at udnytte soft-decision på – nemlig Viterbi-dekodning

# Turbo-koder

Man kan udvide Viterbi dekoderen så der også er soft-decision for output. En sådan dekoder kaldes en MAP dekoder (maximum a posteriori probability) eller en BCJR dekoder efter opfinderne.

I principippet kigger man på sandsynligheden for alle de mulige sekvenser, og adderer dem hvor  $u_i=0$  og dem hvor  $u_i=1$  hvorefter vi har  $P\{u_i=0\}$  og  $P\{u_i=1\}$

Eksempel : Antag at vi har en sekvens med 5 bit ( $u_0, u_1, u_2, t_0, t_1$ ) (svarende til trelliset på næste slide) og har beregnet følgende sandsynligheder

$$P(00000) = 0.03$$

$$P(00101) = 0.01$$

$$P(01010) = 0.04$$

$$P(01111) = 0.02$$

$$P(10001) = 0.01$$

$$P(10100) = 0.62$$

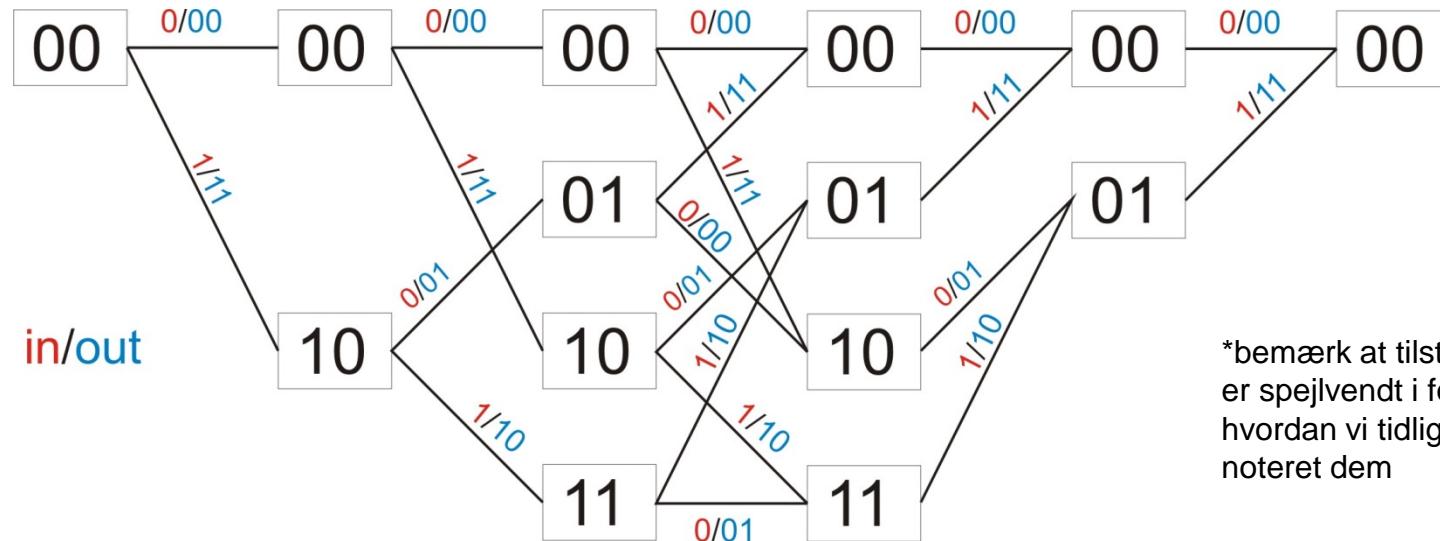
$$P(11011) = 0.06$$

$$P(11110) = 0.21$$

$$\text{Så er } P\{u_2=0\} = 0.03 + 0.04 + 0.01 + 0.06 = 0.14 \text{ og } P\{u_2=1\} = 0.01 + 0.02 + 0.62 + 0.21 = 0.86$$

# Turbo-koder

I praksis må man beregne det ud fra trellisgrafen efter samme principper som Viterbi dekodning – men en del mere kompliceret



\*bemærk at tilstandene her  
er spejlvendt i forhold til  
hvordan vi tidligere har  
noteret dem

For hver tilstand kan man beregne sandsynligheden for at være i den tilstand givet den modtagne sekvens indtil da (fortiden). Tilsvarende kan man beregne sandsynligheden for at have modtaget den følgende sekvens givet man er i en bestemt tilstand (fremtiden) og endelig kan man beregne sandsynligheden for en givet tilstandsovergang givet den aktuelle modtagne værdi (nutiden).

# Turbo-koder

Da vi så på produktkoder med blokkoder tog vi det for givet at de var systematiske.

De foldningskoder vi har set indtil nu har derimod ikke være systematiske, men det kan de blive !

Først skal vi lige se at vi kan skrive generatorvektorerne som polynomier

Generatorerne  $\mathbf{g}^{(0)} = (1 \ 0 \ 1)$  og  $\mathbf{g}^{(1)} = (1 \ 1 \ 1)$  skrives så som

$$(1+D^2, 1+D+D^2)$$

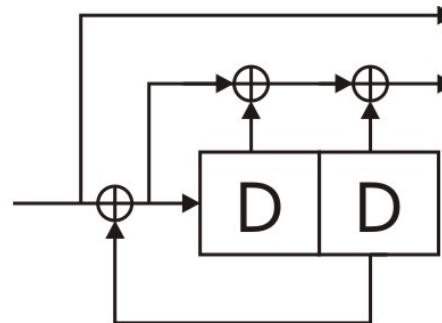
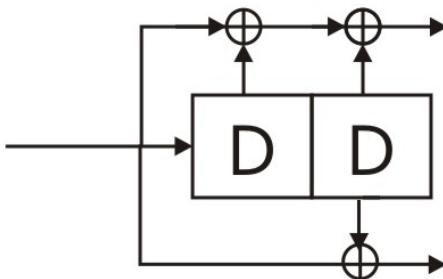
hvor vi bruger D for at indikere at der er tale om et delay (forsinkelse).  $D^2$  er så et delay på 2.

# Turbo-koder

Foldningskoden bliver systematisk ved at dividere med et af generatorpolynomierne

$$(1+D^2, 1+D+D^2) \text{ bliver så } (1, (1+D+D^2)/(1+D^2))$$

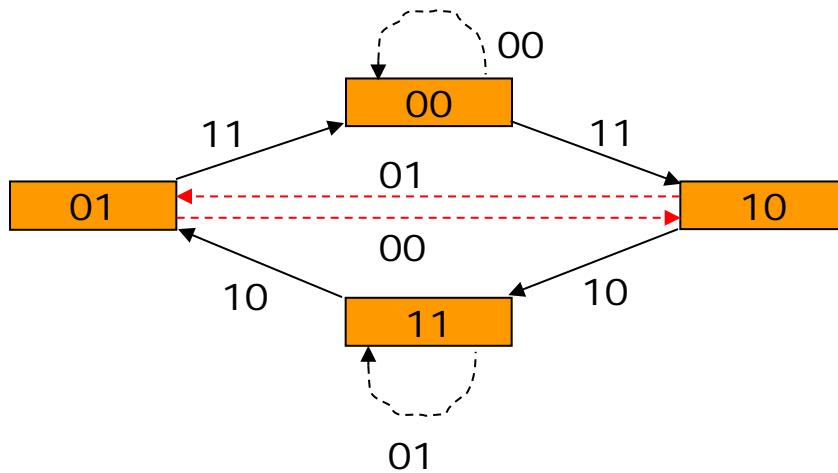
og som "hardware" får vi



Det giver den samme kode men en anden "mapping" af input sekvenser til output sekvenser. Vi ser lidt på det i næste slide.

# Turbo-koder

Man kunne faktisk også bruge den koder er udviklet i Øvelse #3 (Se slide FOL-8), ved at ombytte indgangsbitten for de to udgangsgrene i de tilstænde, hvor den første bit i udgangsblokken ikke svarer til indgangsbitten; det er for tilstandene 11 og 01 ( i den notation vi bruger her):



Systematisk version af koden i FOL-8

Stiplet er overgangen for  $u = 0$   
Fuldt optrukket er for  $u = 1$

Det giver den samme kode men en anden "mapping" af input sekvenser til output sekvenser. Blandt andet bliver ...0001000... til en outputsekvens med uendelig mange 1'er – langs den røde cyklus - og det viser sig at være en enorm fordel !

# Turbo-koder

Man kunne umiddelbart forstille sig at minimumsvægt ordet kom fra et input med et enkelt 1,

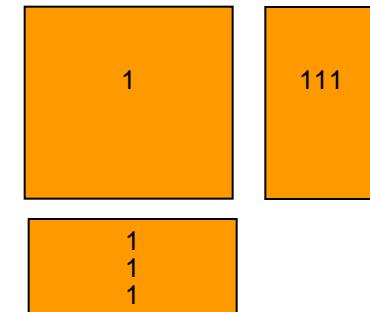
altså ...0001000... sådan er det også for de ikke systematiske foldningskoder

Men for de systematiske foldningskoder vil ...0001000... give et semi-uendeligt svar.

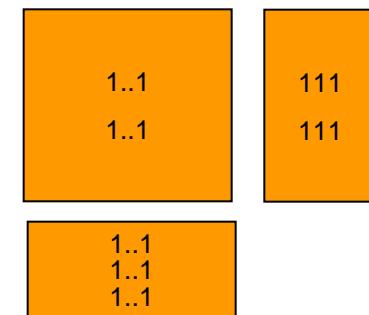
Der er brug for endnu et 1 for at "stoppe" divisionen, fx ...0010100...

Det har naturligvis en gavnlig effekt på produktkodens samlede minimum vægt som illustreret

Men det bliver bedre endnu. Hvis vi i stedet for den viste blok-interleaver bruger en pseudo-tilfældig interleaving er det meget mindre sandsynlig at sekvenser der giver lav vægt i den ene dimension også gør det i den anden. Dvs at firkantmønsteret fra figuren bliver meget mindre sandsynligt



Ikke systematisk foldningskode  
(enhver interleaver)



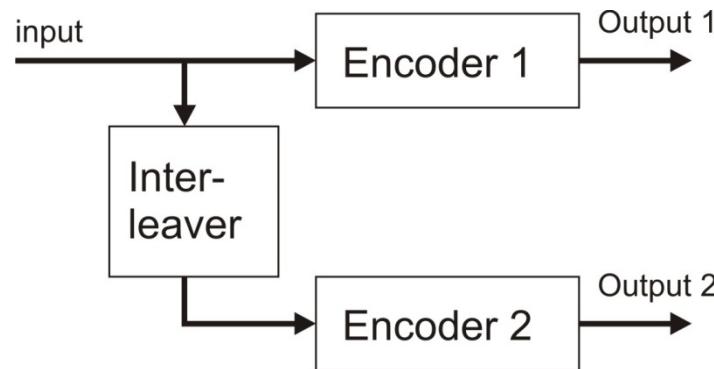
systematisk foldningskode  
(blok-interleaver)

# Turbo-koder

Vi kan nu introducere Turbo indkoderen

Informationssekvensen bliver delt op i blokke som hver bliver indkodet med to systematiske foldningskoder. Inden den anden indkodning bliver blokken omordnet med en pseudo-random interleaver eller lignende.

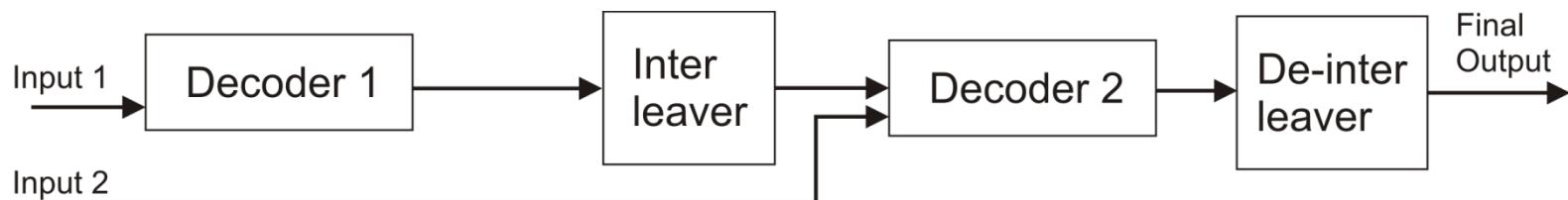
Output fra de to foldningskoder bliver samlet (multiplekset) og sendt



# Turbo-koder

I dekoderen bliver de to sekvenser splittet op igen og input 1 bliver dekodet med en MAP dekoder der giver soft-decision output

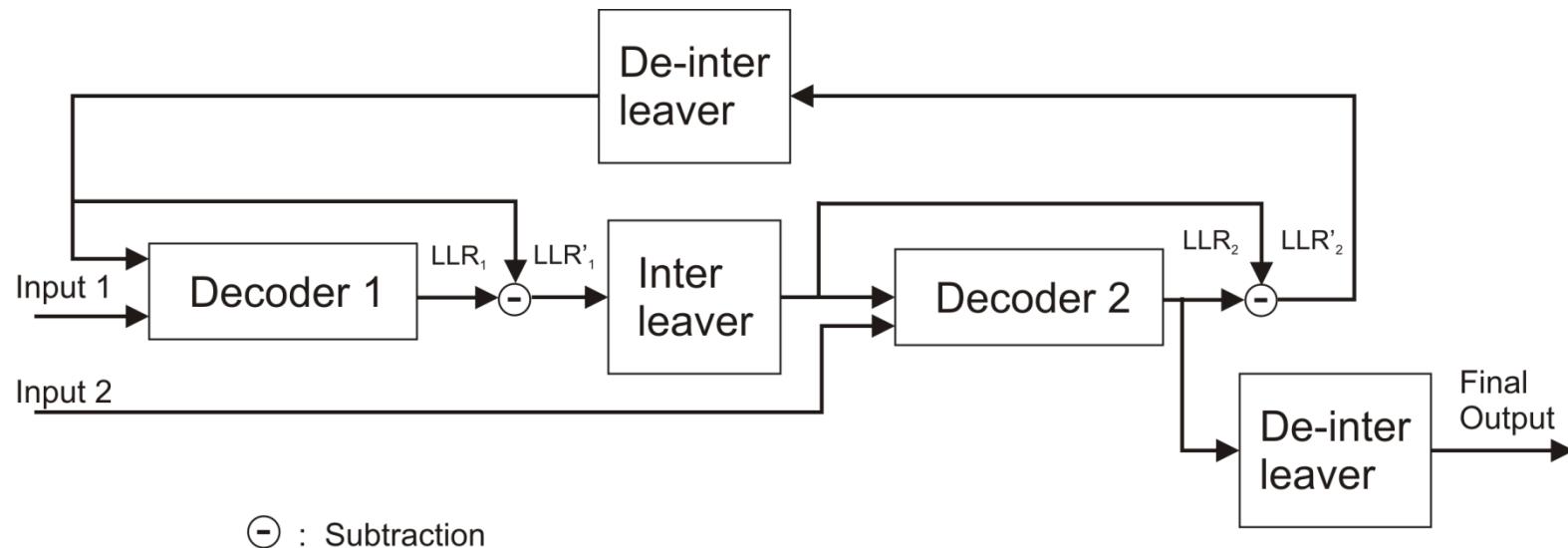
Input 2 bliver tilsvarende dekodet med en MAP dekoder men her bruger man resultatet fra den første dekoder som "a priori" information – eller man kan se det som modtagne værdier for informationsbittene.



# Turbo-koder

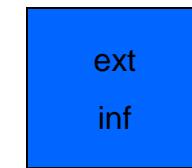
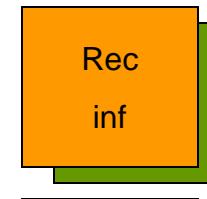
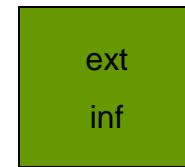
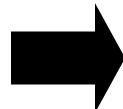
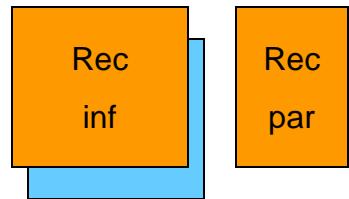
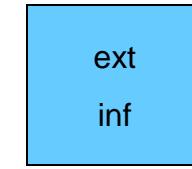
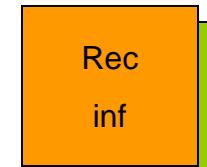
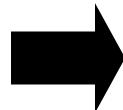
Man kan nu fortsætte med iterativ dekodning ved at koble resultatet fra dekoder 2 tilbage til dekoder 1 og dekode input 1 igen.

Man skal dog fjerne den del af informationen der oprindeligt kom fra dekoder 1 – ellers får man en ubehagelig positiv tilbagekobling.  
Tilsvarende når man laver næste iteration i dekoder 2 ....osv



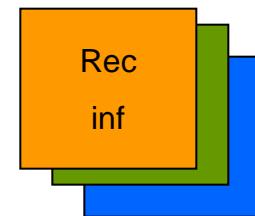
⊖ : Subtraction

# Turbo-koder



Final result :

Overførslen af information mellem dekoderne



# Turbo-koder

## Turbo indkoder for European Space Agency

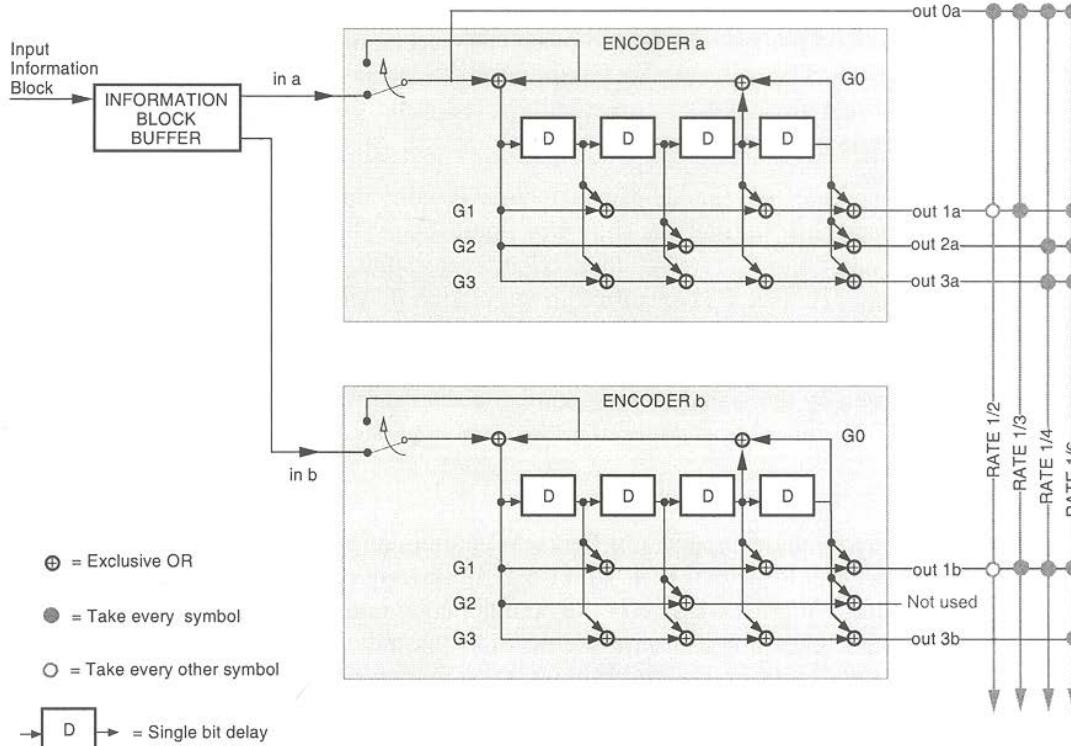


Figure 5-2: Turbo Encoder Block Diagram

Rates

1/2, 1/3, 1/4 and 1/6

Block sizes

1784

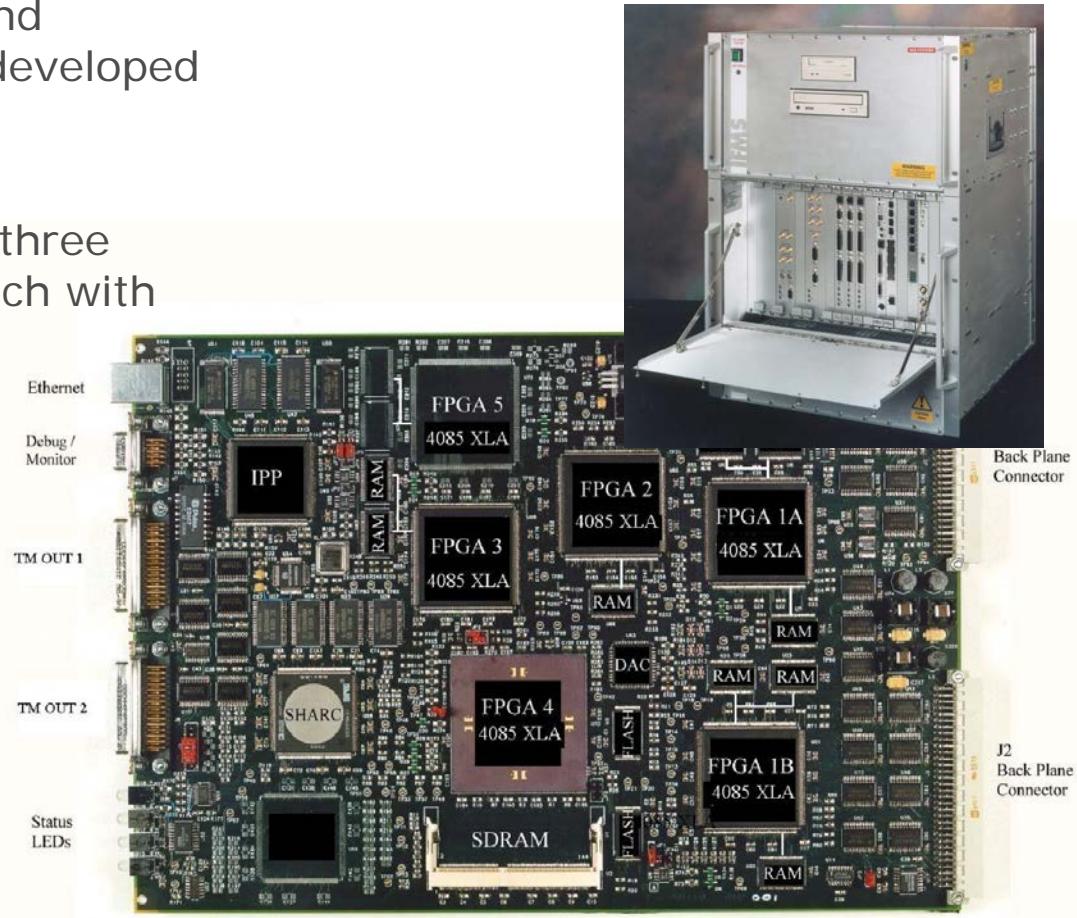
3568

7136

8920

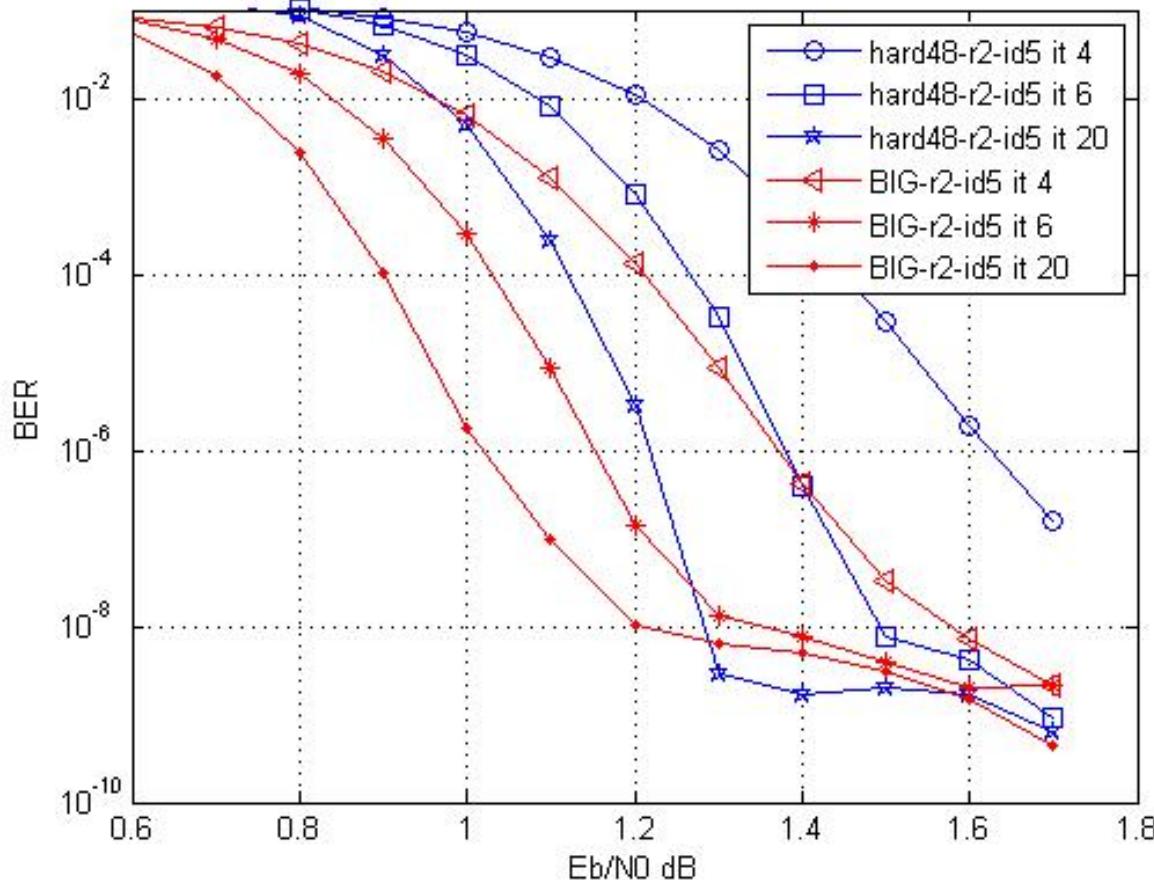
# Turbo-koder

- The Turbo decoder System is implemented as part of the Intermediate Frequency and Modulation System IFMS developed by BAE systems
- The available hardware is three FPGAs (Xilinx XC 4085) each with 64kx16 associated RAM
- App. 1 Mbit/s with 6 iterations
- App. 0.2 dB implementation loss



# Turbo-koder

Simulerings resultater : Rate  $\frac{1}{2}$  blokstørrelse 8920



De blå kurver er vores hardware løsning. De røde er en den optimale dekodning (som der ikke var plads til i det givne system). Som det fremgår har Turbo koder også problemer med "error-floor" som vi så tidligere

# Fejlkorrigerende og fejldetekterende kodning

I denne sidste forelæsning om fejlkorrigerende kodning så vi på

- Sammensætning af simplere koder for at reducere kompleksiteten
- Traditionel sammensætning af indre og ydre koder
- Eksempel fra satellitkommunikation
- Produktkoder
- Eksempel fra optisk kommunikation
- Turbokoder
- Eksempel fra satellitkommunikation

## ØVELSE #8: ITERATIV DEKODNING AF PRODUKTKODER

1. Produktkode baseret på BCH-koder med  $(N,K) = (31,16)$  som kan rette 3 fejl
2. MATLAB:
  1. Lav et program der bruger MATLABs `bchenc` til at kode en  $31 \times 31$  matrix
  2. Dekodning med MATLABs `bchdec` skal nu ske iterativt: Først alle rækkerne og så søjlerne; derefter rækker og søjler igen osv.  
Lav et program til dette og se hvor mange af disse iterationer (rækker+søjler) der skal til at dekode forskellige fejlmønstre
  3. Prøv dekodning med tilfældige fejlmønstre med fejlsandsynlighed  $p$
  4. Kan I finde en tærskelværdi for  $p$ , hvor dekodning begynder at gå godt?
  5. Fejlmønstre der ikke kan rettes