Transmissiefouten en redundantie.

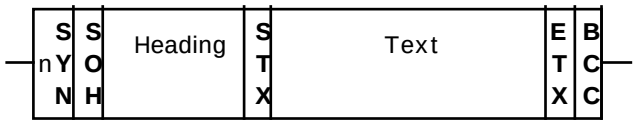
Transmissiefouten zijn volkomen toevallige verschijnselen. De oorzaken van deze fouten zijn van verschillende oorsprong, de datatransmissieketen is immers lang, en de zwakste schakel zal de meeste fouten veroorzaken. Mogelijke foutbronnen zijn: De mens, de data drager (transmissielijn), de terminal, de datatransmissie apparatuur, het transmissiemedium, de datakommunikatiekontroller (DDC) en de komputer. Het grootste aantal fouten wordt gemaakt door de mens, die een foutfrekwentie heeft die hoger ligt dan de 10^{-6} tot 10^{-10} wat normale waarden zijn voor de elektronische apparatuur.

De storingsgevoeligheid van deze verschillende elementen kan verminderd worden door aan het bericht redundante bits toe te voegen. Deze extra bits geven de mogelijkheid om tot op een zekere hoogte aan foutdetektie en eventueel foutkorrektie te doen. Meestal zal men een bericht in blokken verdelen, 80 tot 512 bytes of karakters per blok, en beperkt men zicht tot een foutdetektie per blok en indien er een fout is vraagt men een hertransmissie van dit blok aan.
Het aantal redundante bits die we aan een bericht toevoegen kan hoog oplopen indien we naast foutdetektie ook aan foutkorrektie wensen te doen, dit is namelijk de enige mogelijkheid wanneer het onmogelijk is om een hertransmissie te vragen. B.v. op compact disk schijven zet men 4,3 MB aan informatie, terwijl er maar 1,5 MB muziekinformatie bevatten, de andere bits zijn voor formatering, synchronisatie en foutkorrektie.
Om foutdetektie te bewerkstelligen gebruikt men foutdetekterende kodes, terwijl foutkorrigerende kodes de fouten kunnen verbeteren.
Door het gebruik van deze kodes wordt de maximale informatiesnelheid echter gereduceerd en zal de transmissie van een bepaald bericht langer duren (hogere kostprijs), Zullen er minder nuttige bits op een schijf of band kunnen, zal men voor een 8 bit breed geheugen er een van 9 (foutdetektie) tot 12 (foutkorrektie) bits breed moeten voorzien. En de nodige hard- en software moeten voorzien.

Pariteitskontrole (Parity check).

De eenvoudigste en oudste vorm van foutdetekterende kodering is de pariteitskontrole. Hier wordt bij een bepaald aantal bits één extra bit toe-gevoegd, de pariteitsbit genoemd, waardoor het totaal aantal bits die een zijn even moet zijn bij even pariteit en oneven bij oneven pariteit.
Naargelang de wijze waarop men deze pariteitsbits toevoegt onderscheidt men drie types:

- Pariteitskontrole per karakter (VRC)
- Pariteitskontrole per blok (LRC)
- Pariteitskontrole per karakter en per blok (VRC/LRC)



Figuur 1.1 Tekstblokframe.

Data wordt dikwijls doorgezonden met behulp van zogenaamde tekstblokken. Deze bevatten een aantal karakters of tekens. Voor het begin van zo een blok wordt meestal het STX karakter (*start of text*), en na het blok volgt het ETX karakter (*end of text*),eventueel gevolgd met een BCC (*block check character*) dat de pariteitskontrole bevat.
Een tekstblok wordt dan doorgezonden zoals in figuur 1.1

Pariteitskontrole per karakter (VRC)

Dit type wordt ook *vertical redundancy checking* of VRC genoemd. Aan elk door te sturen karakter wordt een pariteitsbit toegevoegd, zodat men voor elk teken een even of een oneven pariteit bekomt. Men spreekt van een vertikale beveiliging omdat volgens de voorstelling van informatie zoals die gegeven wordt in figuur 1.1 en 2.1, de bits die behoren bij een bepaald karakter of teken, vertikaal onder elkander voorgesteld worden.

	K1	K2	K3	K4
b0	0	1	1	0
b1	0	1	0	1
b2	1	0	0	0
b3	0	1	0	1
b4	1	1	0	0
b5	0	1	1	1
b6	0	1	1	1
P	0	0	1	0

a) even pariteit

	K1	K2	K3	K4
b0	0	1	1	0
b1	0	1	0	1
b2	1	0	0	0
b3	0	1	0	1
b4	1	1	0	0
b5	0	1	1	1
b6	0	1	1	1
P	1	1	0	1

b) oneven pariteit

Figuur 2.1 Pariteitskontrole per karakter

In bovenstaand voorbeeld is gebruik gemaakt van een 7-bits kode met een parteitsbit per karakter. Op deze manier kan men fouten opsporen waarbij er een oneven aantal bits vervalst zijn. Indien er een even aantal bits fout zijn werkt deze methode niet. Indien de pariteit klopt wil dit dus niet zeggen dat de transmissie foutloos verlopen is. De waarschijnlijkheid is echter klein dat er in een transmissie twee bits foutief zijn, zodat een pariteitskontrole er de meeste fouten uithaalt.
Deze pariteitskontrole wordt zeer veel gebruikt, onder andere voor stokkering van data op papier en magneetbanden, en in dynamische RAM- geheugens. Bij datakommunikatie gebruikt men vaak de 7-bits ASCII code, waaraan een achtste pariteitsbit toegevoegd is.

Pariteitskontrole per blok (LRC)

Dit type wordt ook *logitudinal redundancy checking* of LRC genoemd. Hierbij wordt de pariteit gekontroleerd van de bits van gelijke rangorde en van alle doorgezonden karakters. Aldus bekomt men een bijkomend karakter, bestaande uit de verschillende pariteitsbits. Er zijn evenveel pariteitsbits als er bits in een karakter zijn. Dit bijkomend karakter wordt *Block check character* of BCC genoemd. Voor de bepaling van het BCC worden alle karakters van de volledige tekst in rekening gebracht, zie het BISYNC protokol. (BInair SYNchroon Communicatie protocol)

	K1	K2	K3	K4	K5	K6	K7	**BCC**
b0	0	1	1	0	0	0	0	**0**
b1	0	1	0	1	0	1	1	**0**
b2	1	0	0	0	0	1	0	**0**
b3	0	1	0	1	0	1	0	**1**
b4	1	1	0	0	0	0	0	**0**
b5	0	1	1	1	1	0	1	**1**
b6	0	1	1	0	1	1	1	**1**

Figuur 3.1 Pariteitskontrole per blok

Door de zender wordt het BCC gegenereerd en meegezonden met het bericht. Aan de ontvangstzijde wordt dit eveneens gedaan en beide worden met elkander vergeleken. Zo beide verschillend zijn is een fout opgetreden bij de transmissie. Zijn beide gelijk dan is er een grote kans (echter ≠ 100%) dat het bericht goed ontvangen is. Indien er meerdere bits foutief zijn, kan deze methode falen. Uiteraard moeten zender en ontvanger met dezelfde even of oneven pariteit werken.

Pariteitskontrole per karakter en per blok (VRC/LRC)

Deze zogenaamde kruisbeveiliging is een kombinatie van beide beveiligingsprincipes (VRC en LRC). Met de VRC kan men een oneven aantal fouten detekteren doch een even aantal fouten ontsnappen aan deze kontrole. Terwijl men met de LRC een oneven aantal bitfouten in een rij kan detecteren doch een even aantal bitfouten ontsnappen ook deze kontrole. Evenmin is er enige mogelijkheid tot korrektie, tenzij door retransmissie, van een fout.

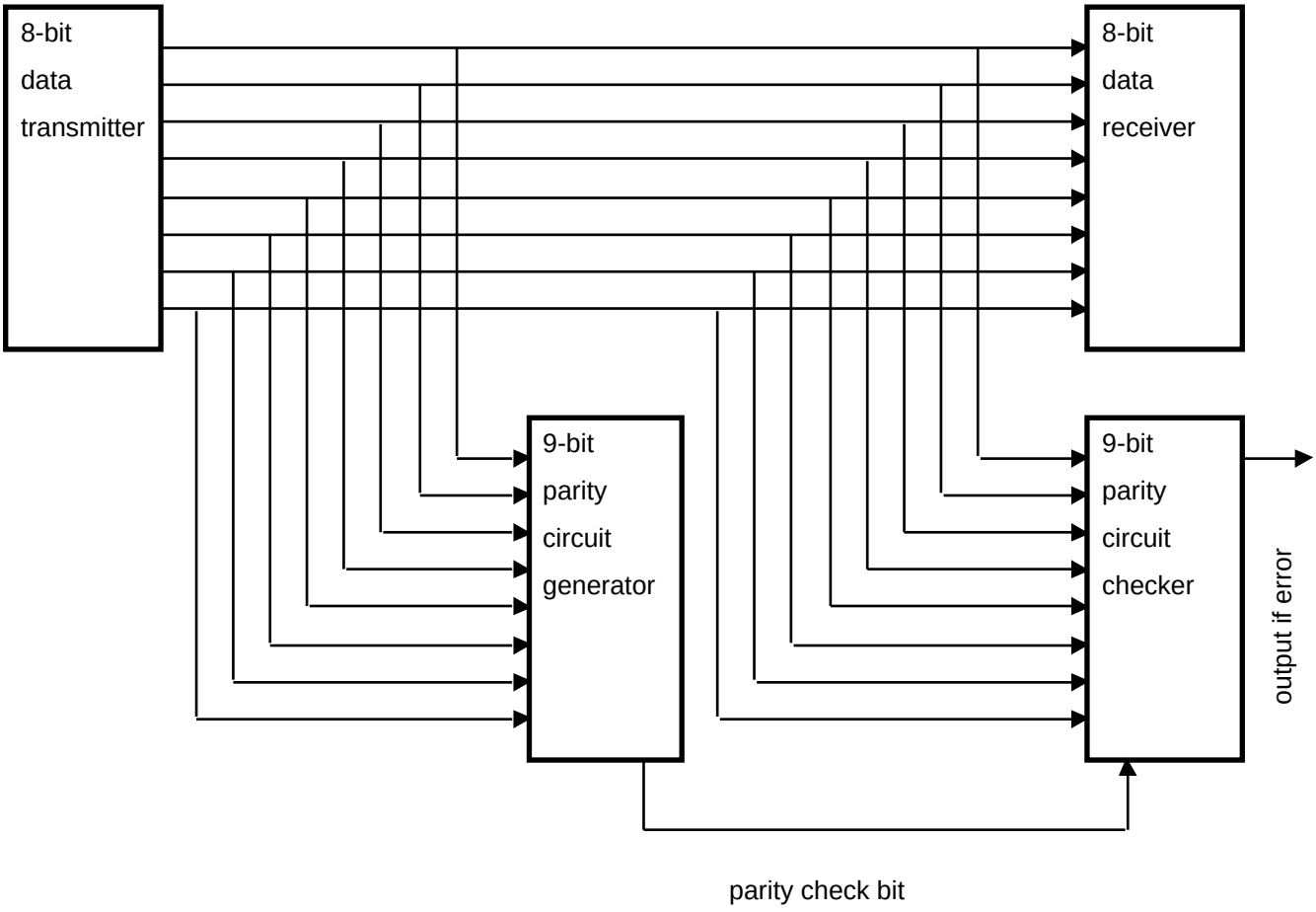
	K1	K2	K3	K4	K5	K6	K7	**BCC**
b0	0	1	1	0	0	0	0	**0**
b1	0	1	0	1	0	1	1	**0**
b2	1	0	0	0	0	1	0	**0**
b3	0	1	0	1	0	1	0	**1**
b4	1	1	0	0	0	0	0	**0**
b5	0	1	1	1	1	0	1	**1**
b6	0	1	1	0	1	1	1	**1**
P	0	0	1	1	0	0	1	**1**

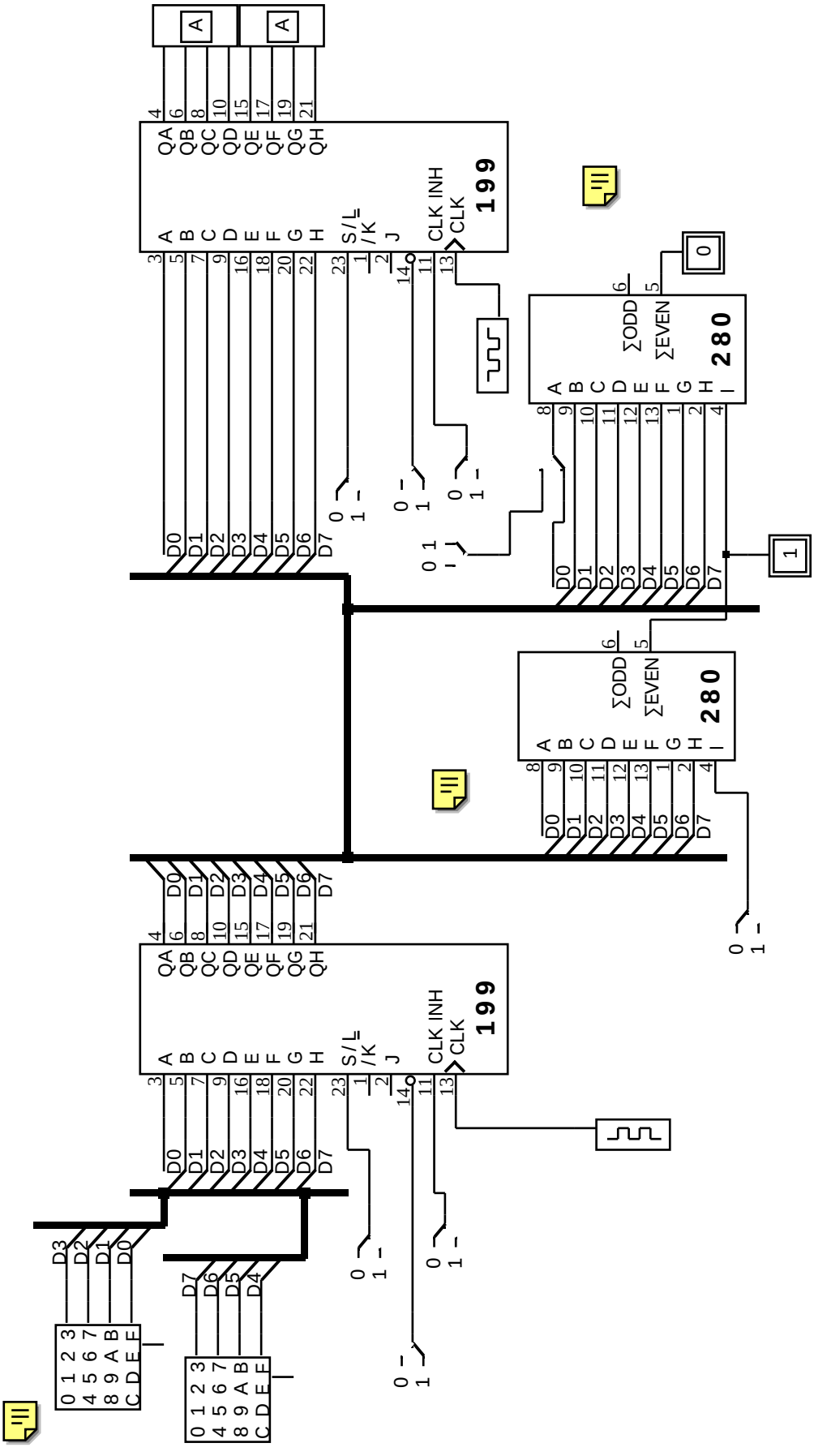
Figuur 3.2 Pariteitskontrole per karakter en per blok.

Door de kombinatie van VRC en LRC kontrole wordt de beveiliging sterk verbeterd. Indien er maar één bit foutief is kan deze niet alleen gedetecteerd worden doch ook verbeterd.

Voor de laatste bit rechtsonder kan men kiezen tussen ofwel deze gelijk nemen aan de pariteit van de pariteitsbits (LRC), ofwel vormt deze de pariteit van het BCC (VRC).

Een transmissiekanaal met parity check.

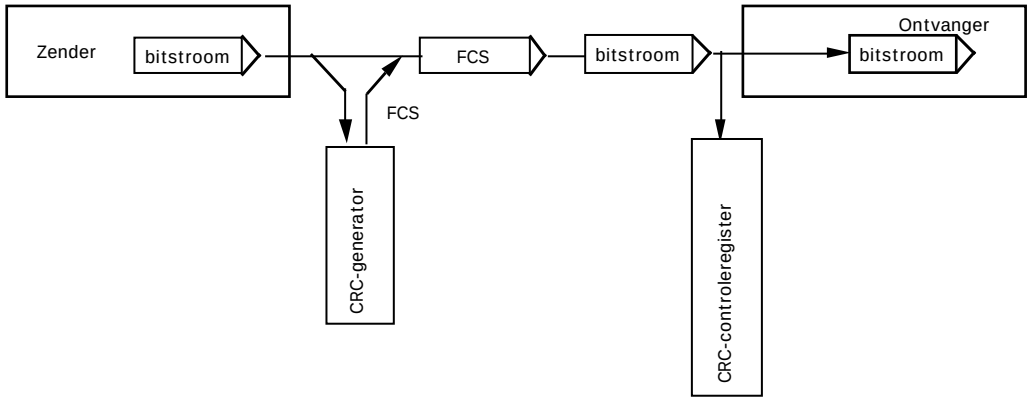




Een transmissiekanaal met parity check.

Cyclic Redundancy Check (CRC)

Bij deze foutcontrole is de kans op het detecteren van transmissiefouten groter dan bij de VRC en LRC foutcontrole.
Deze **CRC foutcontrole** heeft het voordeel dat niet voor elk karakter een apart pariteitsbit behoeft te worden toegevoegd, waardoor de 'overhead' aan bits ten behoeve van de foutcontrole beduidend kleiner is geworden.
Bij de CRC-controle wordt het te controleren bericht beschouwd als één stroom bits die in volgorde, **cyclisch**, ter controle wordt aangeboden. Alle bits van het bericht die worden verstuurd, worden ook binnengeschoven in een **CRC-generator**.
In dit register wordt volgens een bepaalde rekenkundige formule een controlegetal van het bericht gegenereerd. Dit getal wordt aan het einde van het bericht ook meegezonden naar de ontvanger.
Aan de ontvangende zijde zal hetzelfde proces weer worden herhaald, maar nu in het **CRC-controleregister**.
Na ontvangst van het complete bericht zal het controleregister een bepaalde, vooraf bekende en constante waarde hebben gegenereerd. Als de generatie van het CRC-karakter in een gesimplificeerde rekenkundige taal wordt verklaard, kan worden gesteld dat in de CRC-generator het bericht wordt gedeeld door een constante waarde. Het resultaat van deze deling wordt dan verder genegeerd, maar de restwaarde van de deling wordt als controlegetal na het bericht vertuurd.



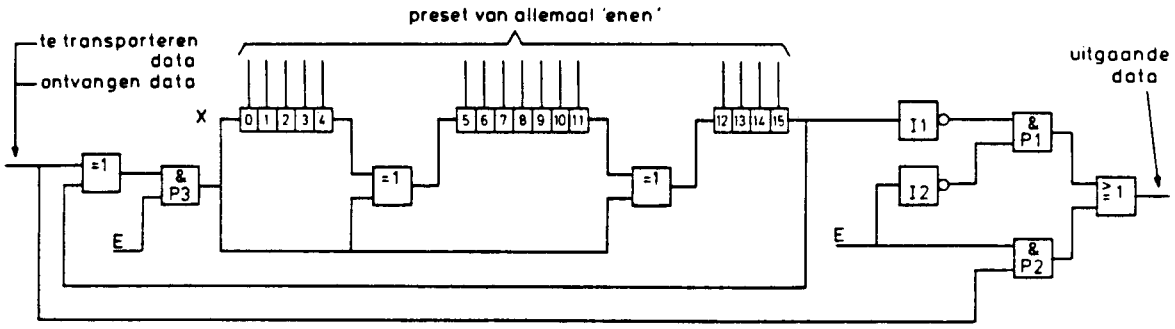
De realisatie van de CRC-controle kan zowel door middel van de hardware van de DC-controller als door middel van een software-routine geschieden. (DC-controller = datacommunicatie-controller)
In de hardware wordt daartoe gebruik gemaakt van een speciaal schuifregister bestaande uit een bepaald aantal bits, in ons geval zestien bits. Er kunnen echter ook controleregisters van twaalf en acht bits worden toegepast. Op verschillende plaatsen in het schuifregister bevinden zich EX-OR poorten via welke de informatiebits in het register kunnen worden geschoven. Elke keer dat een bit wordt verstuurd, wordt ook het schuifregister geklokt, hetgeen tot gevolg heeft dat de informatie in het register over één positie wordt rondgeschoven.
De plaats van de EX-OR poorten in het register bepaald de wiskundige formule volgens welke het bericht wordt gecontroleerd, de zogenaamde **Polynomial**.
Nadat alle bits van het bericht zijn verstuurd, vormt de inhoud van het register de restwaarde van de berekening die dan serieel wordt weggestuurd als **Frame Check Sequence,** onmiddelijk aansluitend op de informatiebits.
De ontvanger beschikt over een soorgelijk register waar het complete bericht, dus inclusief de Frame Check Sequence, wordt binnengeschoven.

In dit register wordt met dezelfde Polynomial gewerkt. Bij een foutloos transport zal de eindwaarde in het register voorspelbaar zijn, terwijl bij transmissiefouten de deling niet uitkomt en dus een totaal andere restwaarde te zien zal geven.
Het zal nu wel duidelijk zijn dat door de ingewikkelde formule, bepaald door de verschillende exclusieve-OF-poorten, het vrijwel onmogelijk is dat meerdere fouten in het transport toch 'per ongeluk' een goede uitkomst zouden geven. Deze verhoogde pakkans maakt de CRC-controle bijzonder aantrekkelijk. Deze controle biedt ook de mogelijkheid om te worden toegepast in niet-karakter-georiënteerde systemen, aangezien in pricipe elke lengte van het bericht in bits mogelijk is. Dit is dan ook de reden dat deze foutcontrole, behalve voor synchrone karakter-georiënteerde terminals, ook voor synchrone bitgeoriënteerde terminals wordt toegepast.
Hierna volgen de wiskundige notaties van de meest toegepaste polynomen, waarbij we in het bijzonder willen wijzen op de CCITT-polynoom die vooral wordt toegepast in de standaard synchrone bit-georiëenteerde terminals.

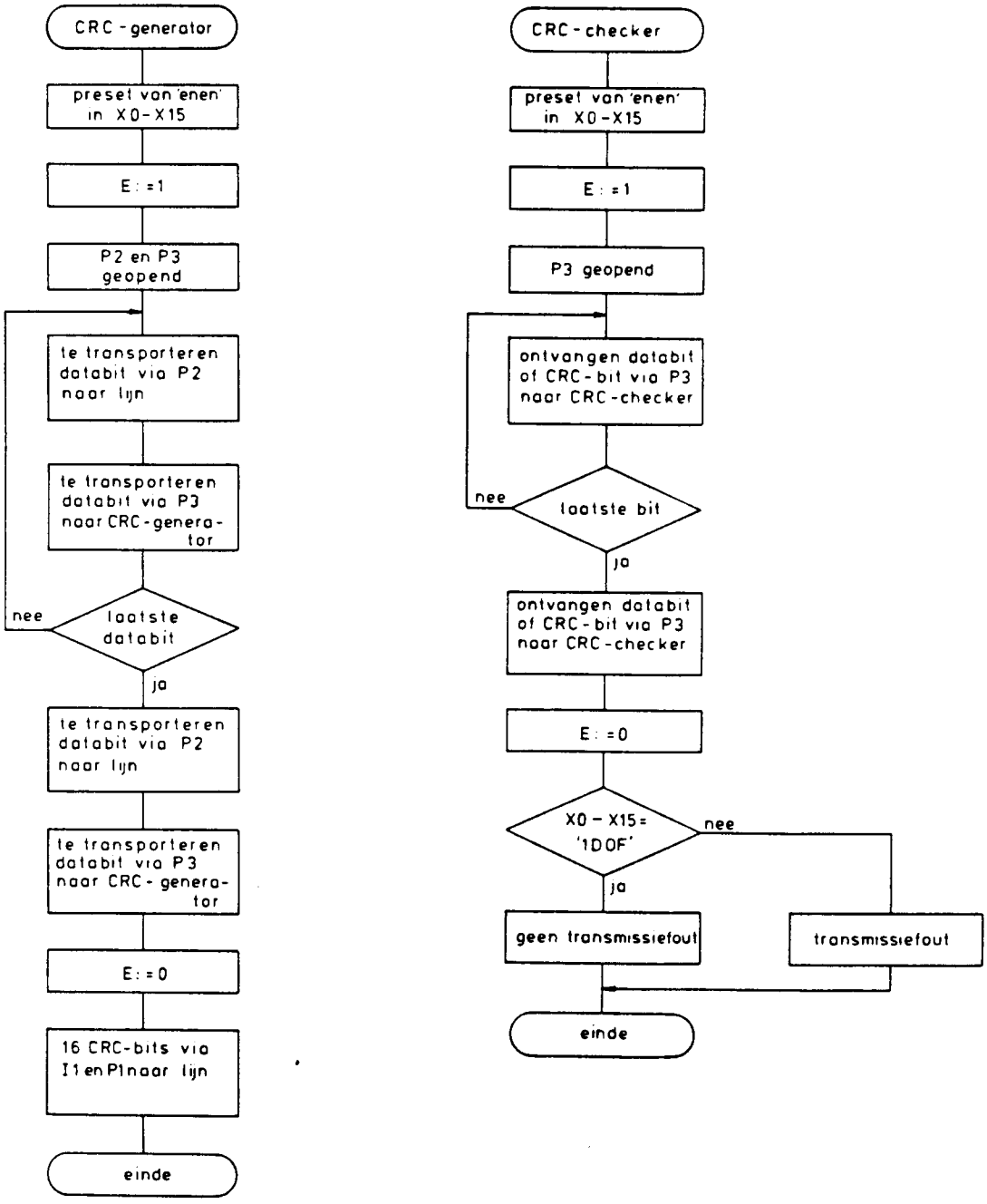
Polynomen:

CCITT	$G(x) = x^{16} + x^2 + x^5 + 1$
CRC-16 (IBM)	$G(x) = x^{16} + x^{15} + x^2 + 1$
CRC-12	$G(x) = x^{12} + x^{11} + x^3 + x^2 + x + 1$
CRC-8(LRC)	$G(x) = x^8 + 1$

Het zou buiten het kader van deze cursus vallen om u verder nog lastig te vallen met de achtergronden die aan deze berekening ten grondslag liggen. Voor een goed begrip willen we slechts wijzen op de relatie tussen de wiskundige notatie en de plaats van de modulo-2 poorten in het schuifregister van onderstaande afbeelding.
De laatstgenoemde polynoom (CRC-8) is in feite een andere weergave van de eerder beschreven LRC-controle.



$G(x) = x^{16} + x^{12} + x^5 + 1$



Cyclic Redundancy Check (CRC)

a. IA5 - bericht (International Alphabet Nr 5) bestaat uit de volgende ASCII-code.

STX	A	ETX

b. Berichtgedeelte voor cyclische berekening. Op STX wordt geen CRC berekening uitgevoerd.

A	ETX

c. Binair berichtgetal

0100000100000011

d. Berichtveelterm

4 1 0 3

$2^{14} + 2^{8} + 2^{1} + 2^{0}$

e. Decimaal berichtgetal M

M = 16643 =4103H

Voorbeelden van generatorveeltermen G(x).

$G = X^{16} + X^{15} + X^2 + 1$ (CRC - 16)
$G = X^{16} + X^{12} + X^5 + 1$ (CRC - CCITT)

Voorbeeld met als generatorterm

$G = X^8 + X^0$

Het decimale generatorgetal is dan G = 257

De berichtveelterm is M = 16643

De cyclische CRC - code is dan de rest van het quotiët van M met G.

16643/257 = 64.758754863813
0,758754863813*257 = 195

CRC = 195 dit is de rest van M/G =C3H

Het decimale codebericht is dan $C = 2^8 * M + CRC$

$C = 256 * 16643 + 195 = 4260803_{10}$

Het binaire codebericht is dan

0100000100000011	11000011

Het codebericht in de IA5-code is dan

4 1 0 3 C 3_{16}

A	ETX	CRC

Het uitgezonden IA5-bericht is dan
met CRC = 11000011

STX	A	ETX	CRC

Het ontvangen IA5-bericht is dan

met CRC = 11000011

STX	A	ETX	CRC

Berichtgedeelte voor de cyclische berekening

A	ETX	CRC

Binair berichtgetal M

0100000100000011	11000011

Berichtveelterm M $2^{22} + 2^{16} + 2^9 + 2^8 + 2^7 + 2^6 + 2^1 + 2^0$

Het decimale berichtgetal M is dan 4260803_{10}

De ontvanger berekent op het berichtgetal M de CRC - code door het berichtgetal te delen door het generator-getal G.
Bij foutloze onvangst van het bericht zal de CRC - code gelijk zijn aan nul.

want 4260803 / 257 = 16579 met rest gelijk aan nul

Oefening 8.1.
Bereken de CRC - code op het bericht "EMT".

Pakkans is 99,995% CRC16
Procescontrol 99,99999995% CRC32
100% CRC oneindig

DISTANCE-1, DISTANCE-2 CODES

Onderstaande tabel geeft de 8 verschilende biniare codes weer die kunnen weggezonden worden in een ***distance-1*** of ***distance-2-code*** .

Bij een disstance-1 code verschillen de twee opéénvolgende symbolen minimum één bit van mekaar.

Bij een distance-2 code verschillen de twee opéénvolgende symbolen minimum twee bits van mekaar. De tabel geeft een voorbeeld van een oneven of even pariteit.

symbol	distance-1 X2 X1 X0	distance-2 X2 X1 X0	distance-2 X2 X1 X0
0	0 0 0	0 0 0 | 1	0 0 0 | 0
1	0 0 1	0 0 1 | 0	0 0 1 | 1
2	0 1 0	0 1 0 | 0	0 1 0 | 1
3	0 1 1	0 1 1 | 1	0 1 1 | 0
4	1 0 0	1 0 0 | 0	1 0 0 | 1
5	1 0 1	1 0 1 | 1	1 0 1 | 0
6	1 1 0	1 1 0 | 1	1 1 0 | 0
7	1 1 1	1 1 1 | 0	1 1 1 | 1
		odd p.	**even p.**

HAMMINGCODE

Bij de distance-2 code kan er slecht een fout worden ontdekt, deze fout kan niet verbeterd worden. Om de fout te kunnen verbeteren moet de distance tussen twee opéénvolgende symbolen of codewoorden d >= 3.

d = 3 ⟶ er kan 1 fout worden gecorrigeerd in het codewoord.

d = 5 ⟶ er kunnen 2 fouten worden gecorrigeerd in het codewoord.

De ***Hammingcode*** is een foutcorrigerende code. Het is een distance-3 code. Dit wil zeggen dat een foutief ontvangen bericht kan verbeterd worden.

Er wordt aan het binaire codewoord bestaande uit I - bits H redundante controlebits toegevoegd. Zodat het doorgezonden codewoord bestaat uit N bits, waarbij N = I + H.
Voor het bepalen van het aantal redundante controlebits wordt er vanuit gegaan dat er maar één bit foutief kan ontvangen worden in het bericht.
De redundante bits worden geplaatst op de plaatsen

$$2^0\ 2^1\ 2^2\ 2^3\ 2^4 \ldots\ldots 2^{H-1}$$

Als alle redundante bits op nul staan dan is de transmissie foutloos gebeurt. Om een fout aan te duiden blijven er dan 2^H - 1 kombinaties over. Hieruit volgt dat

$$2^H - 1 >= N \qquad \text{met } N = I + H$$

$$2^H >= I + H + 1$$

Uit voorgaande vergelijking kunnen we dan onderstaande tabel aflijden.

Aantal controlebit H	Aantal informatiebits I	Totale woordlengte N = I + H
2	1	3
3	4	7
4	11	15
5	26	31
6	57	63
7	120	127

Veronderstel dat we volgend bericht bestaande uit vier bits wensen door te sturen.

I3 I2 I1 I0

D B C A

1 0 1 1

			2^2		2^1	2^0	
rang:	7	6	5	4	3	2	1
bit:	I3	I2	I1	H2	I0	H1	H0
	1	0	1	**0**	1	**0**	**1**

H0 staat op de 2^0 de plaats van het bericht.

H1 staat op de 2^1 de plaats van het bericht.

H2 staat op de 2^2 de plaats van het bericht.

foutieve bitnummer	corresponderende controlebit H2 H1 H0
GEEN FOUT	0 0 0
1 H0	0 0 1
2 H1	0 1 0
3 I0	0 1 1
4 H2	1 0 0
5 I1	1 0 1
6 I2	1 1 0
7 I3	1 1 1

Om de hammingcode H2 H1 H0 te berekenen van het bericht 1011 wordt gebruik gemaakt van de even party check.

H0 = I0 + I1 + I3 = **1**

H1 = I0 + I2 + I3 = **0**

H2 = I1 + I2 + I3 = **0**

zie bovenstaande tabel welke informatiebits **In** er moeten genomen worden voor de berekening van de **Hn-bits**

Veronderstel dat er een foutief bericht binnenkomt:(bit-7 is 0 i.p.v 1)

rang:	7	6	5	4	3	2	1
bericht:	0	0	1	0	1	0	1

Aan de ontvangstzijde ondergaat het bericht dan een even parity check, onder de volgende vorm.

C0 = **H0** + I0 + I1 + I3 = **1+1+1+0=1**

C1 = **H1** + I0 + I2 + I3 = **0+1+0+0=1**

C2 = **H2** + I1 + I2 + I3 = **0+1+0+0=1**

Dan verkrijgen we het binair getal C2C1C0 = 111.

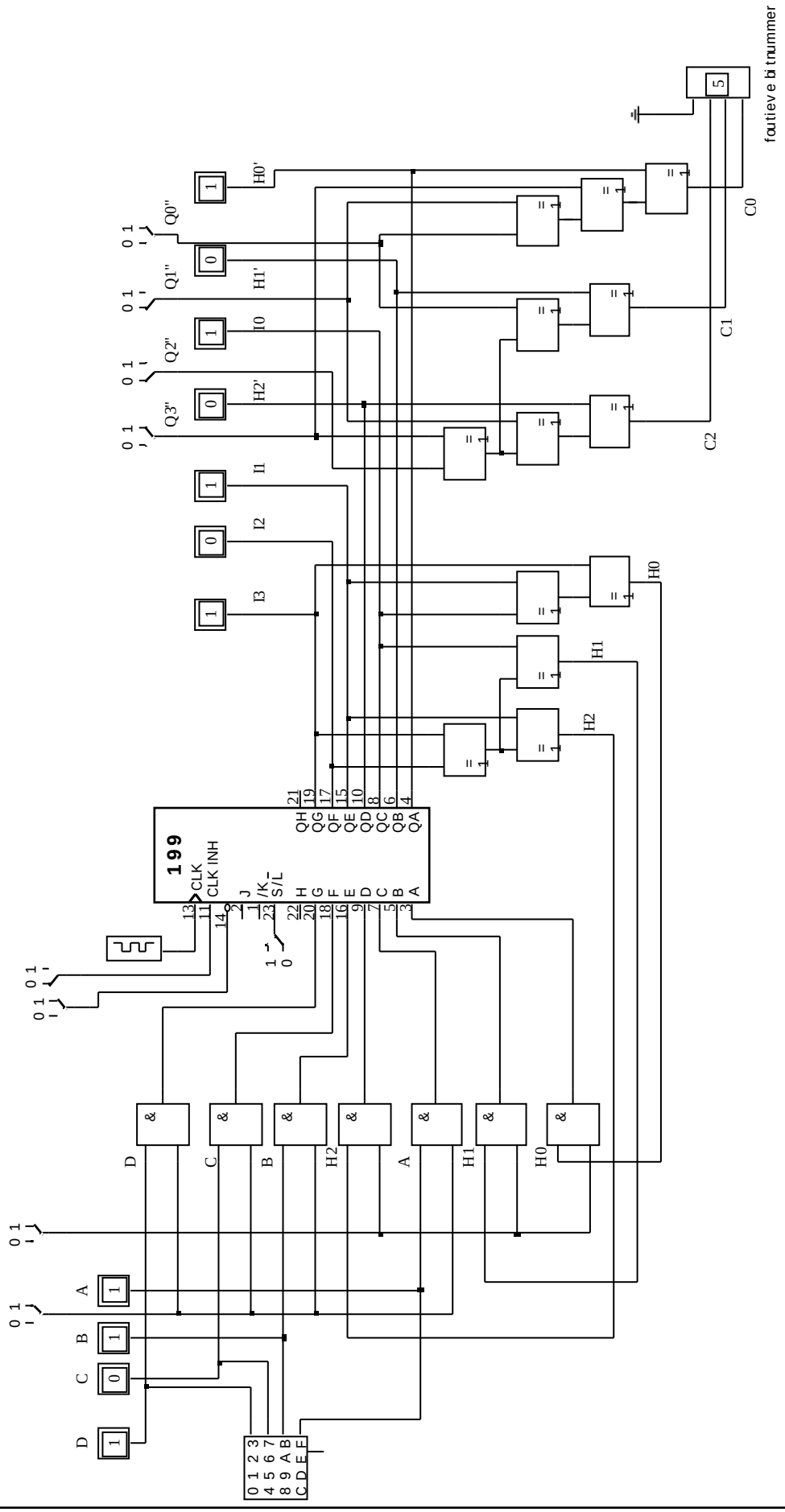
Dit wil zeggen dat bit nummer 7 fout ontvangen werd. Indien er geen transmissifout was geweest dan was C2C1C0 = 000.

Indien er twee bits foutief werden ontvangen dan was C2C1C0 ook gelijk aan 000, doch de berekende bitpositie had geen zin omdat de Hammingcode een distance-3-code is en er maar één fout kan gecorrigeerd worden.

Oefening: Bereken de opéénvolgende hammingcodes voor de getallen

0_{16} tot en met F_{16}

Hammingcode.



Inhoud