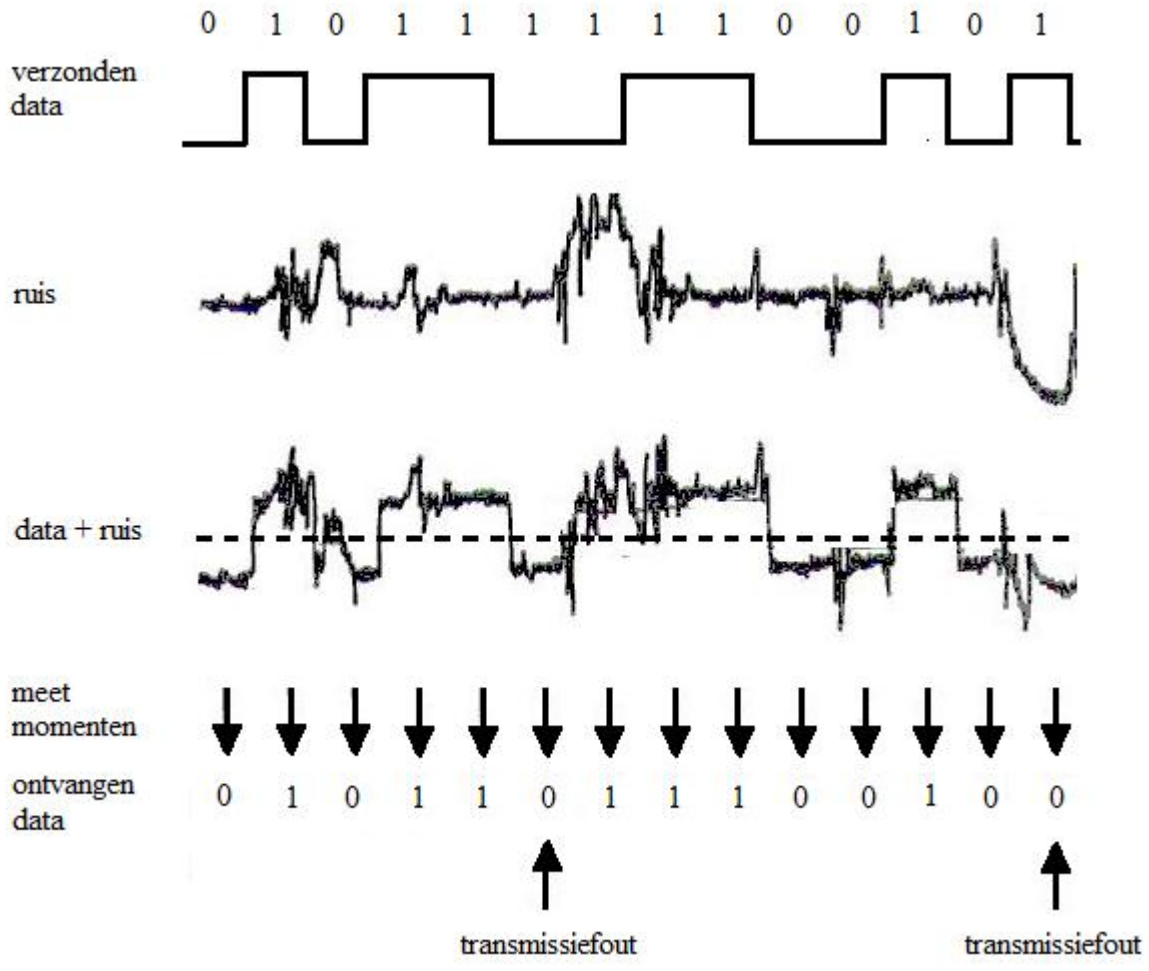
Les D-04 Foutdetectie en correctie

In deze les staan we stil bij het ontdekken (detectie) van fouten bij datacommunicatie en bij het herstellen (correctie) van fouten bij datacommunicatie. We bespreken hoe dat kan met behulp van de begrippen pariteit, Hammingcode en de Cyclische Redundantie Controle.

4.1 Het optreden van fouten bij datacommunicatie

Bij het opslaan en verzenden van digitale informatie kan er door storingen, bijvoorbeeld door de aanwezigheid van een magnetisch veld, ruis optreden. Daardoor kan het zijn dat een bitreeks die verzonden wordt niet ontvangen wordt zoals deze verzonden is.

Hieronder zie je in een eenvoudig voorbeeld hoe dat kan gebeuren. Een digitaal signaal wordt verzonden met behulp van gelijkstroom. Aan dit signaal wordt een verstorend signaal toegevoegd. De ontvanger krijgt een signaal binnen dat de optelling is van het oorspronkelijke signaal plus het verstorende signaal en meet op enkele aftastmomenten een andere waarde dan oorspronkelijk was verzonden. Er treden **transmissiefouten** op.



Optreden van transmissiefouten door een verstorend signaal (ruis)

4.2 Foutontdekkende codes: even en oneven pariteit

Onder de pariteit van een getal verstaan we het even of oneven zijn van dat getal. Onder de **pariteit** van een bitreeks verstaan we het even of oneven zijn van het aantal enen in die bitreeks.

We kunnen aan een bitreeks een **pariteitsbit** toevoegen. Werken we met **even pariteit**, dan maakt het pariteitsbit de hoeveelheid enen in de bitreeks even. In het onderstaande voorbeeld is aan een zestal bytes een pariteisbit toegevoegd.

0	1	1	0	0	1	1	0	**0**
1	0	1	1	0	0	1	0	**0**
1	1	0	0	1	1	0	1	**1**
1	0	0	0	1	0	0	1	**1**
0	1	0	0	1	0	1	0	**1**
0	0	1	1	1	0	0	1	**0**

toevoegen van pariteitsbits bij even pariteit

Het toevoegen van de pariteitsbit maakt het ontdekken van enkelvoudige fouten (**foutdetectie**) mogelijk. De ontvanger kan namelijk een **pariteitscheck** uitvoeren. Als de ontvanger een reeks met een oneven aantal enen ontvangt dan moet er iets misgegaan zijn. We weten dan alleen nog niet welke bit fout ontvangen wordt. Het is natuurlijk wel van belang dat zender en ontvanger afstemmen of er met even of oneven pariteit gewerkt wordt. Hieronder wordt geïllustreerd hoe het optreden van een fout in een bitreeks wordt opgemerkt.

0	1	1	0	0	1	1	0	**0**
1	0	1	1	0	0	1	0	**0**
1	1	0	0	1	1	0	1	**1**
1	0	0	0	0	0	0	1	**1**
0	1	0	0	1	0	1	0	**1**
0	0	1	1	1	0	0	1	**0**

controle van pariteitsbits geeft een fout aan

De pariteitsbit kan naast horizontaal ook verticaal toegepast worden. Niet alleen horizontaal moet in een bitreeks dan een even aantal enen voorkomen, ook verticaal moet dat het geval zijn:

0	1	1	0	0	1	1	0	**0**
1	0	1	1	0	0	1	0	**0**
1	1	0	0	1	1	0	1	**1**
1	0	0	0	1	0	0	1	**1**
0	1	0	0	1	0	1	0	**1**
0	0	1	1	1	0	0	1	**0**
1	**1**	**1**	**0**	**0**	**0**	**1**	**1**	**1**

De ontvanger kan nu een **tweedimensionale pariteitscheck** (zowel horizontaal als verticaal) uitvoeren. Daarbij kan een enkele fout niet alleen worden gevonden (gedetecteerd) maar ook worden verbeterd (gecorrigeerd).

Hieronder wordt geïllustreerd hoe het optreden van een fout met behulp van een tweedimensionale pariteitscheck kan worden gedetecteerd.

0	1	1	0	0	1	1	0	**0**
1	0	1	1	0	0	1	0	**0**
1	1	0	0	1	1	0	1	**1**
1	0	0	**1**	1	0	0	1	**1**
0	1	0	0	1	0	1	0	**1**
0	0	1	1	1	0	0	1	**0**
1	**1**	**1**	**0**	**0**	**0**	**1**	**1**	**1**

detectie van één enkele fout
bij een tweedimensionale pariteitscheck

Ook als meerdere bits in een rij fout zijn wordt dat bij een tweedimensionale pariteitscheck ontdekt. Bij verstoringen komen fouten komen meestal met meerdere achter elkaar voor (in zogenaamde **bursts**). De tweedimensionale pariteitscheck biedt een goede methode om dit soort fouten te detecteren en te corrigeren.

4.3 Hammingcodes

Een verzonden letter ‘i’, met bijbehorende ASCII-codering 1101001, kan bij het fout transporteren van één enkele bit al worden ontvangen als de letter ‘m’, met bijbehorende ASCII-codering 1101101.
De codes verschillen slechts één bit. We merken op dat door het toevoegen van de pariteitsbit het verschil tussen twee codes,

‘i’ = 1101001 = 11010010

en

‘m’ = 1101101 = 11011011

wordt vergroot van één verschillende bit in twee verschillende bits. Het aantal bits dat twee codes verschillen wordt ook wel de **Hamming-afstand** genoemd. Door het toevoegen van de pariteitsbit wordt de Hamming-afstand tussen de coderingen van de letters ‘i’ en ‘m’ vergroot van 1 naar 2, hetgeen de detectie van 1 fout mogelijk heeft gemaakt. Zonder pariteitsbit wordt een boodschap met 1 fout gewoon verkeerd ontvangen zonder dat de fout ontdekt wordt.

Opdrachten

Opdracht 4.1
Wat gebeurt er als juist de pariteitsbit fout wordt verzonden?

Opdracht 4.2
Wat gebeurt er als er meer dan één fout in het transport optreedt?

Om meer dan één fout te kunnen ontdekken moet er aan de bitreeks die je wilt verzenden meer dan één bit toegevoegd worden. We zullen nu beschrijven hoe we door informatie toe te voegen aan een bitreeks niet alleen foutdetectie mogelijk kunnen maken maar ook foutcorrectie mogelijk kunnen maken.

Een simpele oplossing voor het probleem dat fouten in codes ook moeten kunnen worden verbeterd is dat elke bit uit de bitreeks een aantal malen herhaald wordt.

Voorbeeld

1. Codering
Coderen we de letter ‘i’ = 1101001 op deze manier (elke bit wordt vijf maal herhaald) dan ontstaat de codering:

11111 11111 00000 11111 00000 00000 11111

2. Decodering
Het aantal bits is hiermee vervijfvoudigd en het optreden van **twee fouten** wordt bij deze manier van coderen opgevangen. De code:

11111 11111 00000 11111 **01100** 00000 11111

zal nog steeds worden ontvangen als 1101001, de letter ‘i’. Er worden twee fouten gedetecteerd en gecorrigeerd. Bij het optreden van **drie fouten** kan de code worden ontvangen als de letter ‘m’, bijvoorbeeld bij de ontvangst van:

11111 11111 00000 11111 **01110** 00000 11111

De code wordt ontvangen als letter ‘m’ waarbij de ontvanger denkt twee fouten te detecteren en corrigeren. Het verschil van deze ontvangen code met de code 1101101=’m’ (de Hamming-afstand) is immers kleiner dan het verschil met de code 1101001=’i’.

Opdracht 4.3

Hoeveel fouten kunnen er in het bovenstaande voorbeeld in het transport maximaal worden gemaakt zodat de boodschap toch nog goed aankomt?

Bovenstaand voorbeeld geeft aan dat er ontzettend veel bits aan een bitreeks moeten worden toegevoegd om het optreden van één of meerdere fouten te kunnen detecteren en corrigeren. De bitreeks wordt vijf keer zo lang! De mate waarin overtollige bits aan een boodschap worden toegevoegd wordt ook wel **redundantie** genoemd. Een van de belangrijkste kwaliteiten waarover een codering van een bitreeks moet beschikken is dat er zo weinig mogelijk overtollige bits hoeven te worden toegevoegd.

We merken op dat met de coderingsmethode uit het bovenstaande voorbeeld de Hamming-afstand tussen de code ‘i’= 11010010 en ‘m’= 11011011 verder toegenomen is van 2 naar 5, hetgeen de correctie van 2 fouten mogelijk heeft gemaakt. Je begrijpt misschien wel dat wanneer de afstand tussen de verschillende codes die je gebruikt toeneemt, de mogelijkheid om fouten te ontdekken en te verbeteren in het transport van de codes toeneemt.

In het algemeen is het zo dat voor het *ontdekken* van n fouten in een codering van boodschappen met een onderlinge Hamming-afstand 1 een afstand n+1 nodig is en voor het *verbeteren* van n fouten in een codering van boodschappen met een onderlinge Hamming-afstand 1 een afstand 2n+1 nodig is.

Voorbeeld

We zagen in de voorgaande voorbeelden:

'i'= 1101001 en 'm'= 1101101 (Hamming-afstand 1, dus n=1)

Het toevoegen van de pariteitsbit zorgde voor een codering:

'i'= 11010010 en 'm'= 11011011 (Hamming-afstand 2)

en maakte het detecteren van één fout mogelijk.
Voor het ontdekken van één fout is een afstand 2 nodig (= n+1)

Het vijf maal herhalen van bits zorgde voor een codering:

'i' = 11111 11111 00000 11111 00000 00000 11111 en
'm' = 11111 11111 00000 11111 11111 00000 11111 (Hamming-afstand 5)

en maakte het corrigeren van twee fouten mogelijk.
Voor het verbeteren van twee fouten is een afstand 5 nodig (= 2n+1)

Om één fout te kunnen corrigeren moeten in de besproken methode de bits van de bitreeks drie maal worden herhaald. De letter 'i' met ASCII codering 1101001 wordt dan gecodeerd als de 21-bits codering: 'i'= 111 111 000 111 000 000 111.

Met deze codering hebben de letters 'i' en 'm' een Hamming-afstand 3 zodat bij het maken van één transmissiefout de codering nog steeds het dichtst bij die van de letter 'i' ligt en de fout gecorrigeerd kan worden.

Een codering van een reeks van 7-bits met een 21-bits codering met het doel foutcorrectie mogelijk te maken is een niet zo efficiënte manier van coderen.

Er is een meer efficiënte / minder redundante (=overtollige) codering mogelijk die door Richard Hamming in 1950 verzonnen is. Met deze coderingsmethode is het mogelijk om een reeks van 7-bits boodschap met een 11-bits codering te coderen waarbij foutcorrectie mogelijk is.

We leggen de werking van de Hamming code uit aan de hand van een voorbeeld.

1.Codering.
We gaan weer uit van de letter ‘i’ = 1101001.
Deze wordt verwerkt in een codering, waarbij de **informatiebits** worden verwerkt in de plaatsen 3, 5, 6, 7, 9, 10, 11, 12 enz.

		1		**1**	**0**	**1**		**0**	**0**	**1**
1	2	3	4	5	6	7	8	9	10	11

De plaatsen 1, 2, 4, 8, enz. zijn gereserveerd voor zogenaamde **controlebits.**
Hoe wordt de waarde van deze bits bepaald? Hamming bedacht dat elk controlebit de plaatsen moet controleren waar hij in het binaire getallensysteem voorkomt. Controlebit 1 controleert informatiebits 3 (=**1** + 2), 5 (=**1** + 4), 7 (= **1** + 2 + 4), enz., maar niet informatiebit 6 (= 2 + 4).

In het schema hieronder wordt duidelijk gemaakt dat controlebit 1 de informatiebits 3, 5, 7, 9 en 11 controleert, controlebit 2 de informatiebits 3, 6, 7, 10 en 11, controlebit 4 de informatiebits 5, 6 en 7 en controlebit 8 de informatiebits 9, 10 en 11.

		1		**1**	**0**	**1**		**0**	**0**	**1**
1	2	**3**	4	**5**	**6**	**7**	8	**9**	**10**	**11**
	1									
	2									
	4									
	8									

Vervolgens wordt per controlebit de pariteit bekeken van de informatiebits die hij controleert. Voor controlebit 1 geldt: de pariteit van de informatiebits 3 (**1**), 5 (**1**), 7 (**1**), 9 (**0**) en 11 (**1**) is even zodat controlebit 1 de waarde ‘0’ krijgt.

Opdracht 4.4

Ga na dat de controlebits 2, 4 en 8 respectievelijk de waarden ‘1’, ‘0’en ‘1’krijgen.

Vervolg voorbeeld

De codering die op deze manier ontstaat is de volgende:

0	1	**1**	0	**1**	**0**	**1**	1	**0**	**0**	**1**
1	2	3	4	5	6	7	8	9	10	11

Deze codering, ‘i’ = 01101011001, wordt ook wel de **Hammingcode** genoemd

Opdracht 4.5

Bepaal de Hamming-code van de letter ‘m’ = 1101101

		1		1	0	1		1	0	1
1	2	3	4	5	6	7	8	9	10	11
	1									
	2									
	4									
	8									

Hoe wordt nu een Hamming gecodeerde bitreeks gedecodeerd? We vervolgen het voorbeeld van de letter ‘i’:

Vervolg voorbeeld

2. Decodering
De Hammingcodering ‘i’ = 01101011001 kan goed worden ontvangen of worden ontvangen met een foute informatiebit, bijvoorbeeld als:

0	1	1	0	1	0	1	1	1	0	1
1	2	3	4	5	6	7	8	9	10	11

Hoe gaat de detectie en correctie van deze fout in zijn werk? We redeneren omgekeerd met het codeerschema:

0	1	1	0	1	0	1	1	1	0	1
1	2	3	4	5	6	7	8	9	10	11
	1									
	2									
	4									
	8									

Controlebit 1 is ‘1’ en dat klopt niet met informatiebits 3 (1), 5 (1), 7 (1), 9 (1) en 11 (1).
Controlebit 2 is ‘1’ en dat klopt met informatiebits 3 (1), 6 (0), 7 (1), 10 (0) en 11(1)
Controlebit 4 is ‘0’ en dat klopt met informatiebits 5 (1), 6 (0) en 7 (1)
Controlebit 8 is ‘1’ en dat klopt niet met informatiebits 9 (1), 10 (0) en 11 (1)

De foute informatiebit is hiermee gevonden: 1 + 8 = 9!

Opdracht 4.6

Waarom is de foute informatiebit de bit met de plaats van de som van de niet kloppende controlebits?

Opdracht 4.7

Hoe groot is de Hamming-afstand tussen de Hamming-codes van de letters ‘i’ en ‘m’?

‘i’ = 01101011001
‘m’ = 11101010101

Opdracht 4.8

Gegeven is de ASCII-coderingstabel:

‘a’	= 97	= 1100001
‘b’	= 98	= 1100010
‘c’	= 99	= 1100011
‘d’	= 100	= 1100100
‘e’	= 101	= 1100101
‘f’	= 102	= 1100110
‘g’	= 103	= 1100111
‘h’	= 104	= 1101000
‘i’	= 105	= 1101001
‘j’	= 106	= 1101010
‘k’	= 107	= 1101011
‘l’	= 108	= 1101100
‘m’	= 109	= 1101101

‘n’	= 110	= 1101110
‘o’	= 111	= 1101111
‘p’	= 112	= 1110000
‘q’	= 113	= 1110001
‘r’	= 114	= 1110010
‘s’	= 115	= 1110011
‘t’	= 116	= 1110100
‘u’	= 117	= 1110101
‘v’	= 118	= 1110110
‘w’	= 119	= 1110111
‘x’	= 120	= 1111000
‘y’	= 121	= 1111001
‘z’	= 122	= 1111010

Welke letter hoort er bij een ontvangen Hamming-code:

a) 11111011100
b) 01101100101

1	2	**3**	4	**5**	**6**	**7**	8	**9**	**10**	**11**
	1									
	2									
	4									
	8									

Wat gebeurt er nu als juist één van de controlebits fout overkomt?

Voorbeeld

We controleren aan de hand van een voorbeeld of de hierboven beschreven decoderingsmethode met behulp van het omgekeerde coderingsschema ook werkt als niet een informatiebit maar een controlebit fout is aangekomen.

De Hammingcodering 'i' = 01101011001 kan worden ontvangen met een foute controlebit, bijvoorbeeld als controlebit 4 fout aankomt:

0	1	1	**1**	1	0	1	1	0	0	1
1	2	3	4	5	6	7	8	9	10	11

Hoe gaat de detectie en correctie van deze fout in zijn werk? We redeneren omgekeerd met het codeerschema:

0	1	**1**	1	**1**	**0**	**1**	1	**0**	**0**	**1**
1	2	**3**	4	**5**	**6**	**7**	8	**9**	**10**	**11**
	1									
	2									
	4									
	8									

Controlebit 1 is '0' en dat klopt met informatiebits 3 (**1**), 5 (**1**), 7 (**1**), 9 (**0**) en 11 (**1**).
Controlebit 2 is '1' en dat klopt met informatiebits 3 (**1**), 6 (**0**), 7 (**1**), 10 (**0**) en 11(**1**)
Controlebit 4 is '1' en dat klopt niet met informatiebits 5 (**1**), 6 (**0**) en 7 (**1**)
Controlebit 8 is '1' en dat klopt met informatiebits 9 (**0**), 10 (**0**) en 11 (**1**)

De foute informatiebit is hiermee gevonden: 4 + 0 = 4!

Een transmissiefout in een controlebit kan met behulp van de methode van het omgekeerde codeerschema dus op een zelfde manier worden gedetecteerd en gecorrigeerd!

In het laatste deel van deze lesbrief zullen we ons buigen over het probleem dat er meerdere fouten op kunnen treden. Hoe kunnen boodschappen worden gecodeerd zodat er meerdere fouten kunnen worden gedetecteerd en gecorrigeerd?

Transmissiefouten treden vaak niet alleen op maar in blokken of "bursts". Storingen, roestige stukjes koperdraad of een te grote knik in een glasvezelkabel zijn mogelijke oorzaken van zulke zogenaamde burst-fouten.

Een oplossing voor de mogelijke detectie en correctie van zulke burst-fouten is de volgende: een boodschap wordt vertaald in Hamming-codes en deze worden niet achter elkaar geplaatst maar in een matrix (getallenblok).

Zo'n matrix ziet er als volgt uit:

	H	a	m	m	i	n	g	c	o	d	e
1	0	1	1	1	0	0	1	1	0	1	0
2	0	0	1	1	1	1	1	1	0	1	0
3	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
4	1	1	0	0	0	0	1	1	0	1	1
5	**0**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
6	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
7	**1**	**0**	**1**	**1**	**1**	**1**	**0**	**0**	**1**	**0**	**0**
8	0	1	0	0	1	0	1	0	1	1	0
9	**0**	**0**	**1**	**1**	**0**	**1**	**1**	**0**	**1**	**1**	**1**
10	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**1**	**1**	**0**	**0**
11	**0**	**1**	**1**	**1**	**1**	**0**	**1**	**1**	**1**	**0**	**1**

In de matrix staan de Hamming-codes niet van links naar rechts (in de rijen) maar van boven naar beneden (in de kolommen) gerangschikt. De codes worden vervolgens verzonden. Bij aankomst wordt de matrix wel van links naar rechts gelezen. Deze manier van coderen maakt het dus mogelijk dat er een burst of rij van 11 fouten op kan treden tijdens het datatransport die nog gedetecteerd en gecorrigeerd kan worden omdat elke afzonderlijke Hamming-code in de kolommen dan maar één fout bevat.

4.4 Cyclische Redundantie Controle (CRC)

Een andere, nog iets complexere methode om foutdetectie mogelijk te maken is door Cyclische Redundantie Controle (CRC = Cyclic Redundancy Check). Deze methode wordt toegepast bij datatransport en bij het opslaan van ZIP-bestanden.

Aan de hand van het verzenden van datapakketjes volgens het 802.3 (Ethernet) zullen we toelichten hoe CRC werkt. Een datapakketje ziet er bij dit protocol als volgt uit:

Onderdeel	**Betekenis**
7 bytes "10101010"	synchronisatiebytes
1 byte "10101011"	Framesttartsein
6 bytes	MAC –adres ontvanger
6 bytes	MAC –adres zender
2 bytes	lengte dataveld
0-1540 bytes	IP-pakketje
4 bytes	frame check

datapakketje volgens het 802.3 protocol

De laatste 4 bytes van het frame vormen de frame check, die plaats vindt volgens CRC.
Het idee achter CRC is dat het datapakket gezien wordt als één groot binair getal. Dit getal wordt gedeeld door een bepaald getal. De rest na deling wordt opgeslagen in de 4 bytes frame check. De ontvanger deelt de binaire waarde van het ontvangen datapakket door eenzelfde getal en kijkt dan of hij op dezelfde rest uitkomt. Is dat niet het geval, dan wordt een foutmelding afgegeven.

Een dergelijke melding krijg je ook als een ZIP-bestand, een CD of DVD beschadigd is. In al deze gevallen wordt met CRC gewerkt ("file corrupted").

Merk op dat de CRC erg efficiënt is, want de codering neemt maar 4 van de 1566 bytes in beslag.

Hoe werkt dan de Cyclische Redundantie Controle? We illustreren dit aan de hand van een voorbeeld.

1.Codering.
We gaan uit van de letter 'i' = 105 = 1101001.
We delen dit getal door het getal 19 = 10011.
Dit getal, dat bij zender en ontvanger bekend moet zijn, noemen we de **CRC-generator**.

Zo'n deling gaat als volgt in zijn werk.
Als je nooit hebt leren staartdelen met rest hier eerst even een voorbeeld:

43568 : 123 = 354 met rest 96,
want je kunt achtereenvolgens 300 keer, 50 keer en 4 keer 123 van 43568 afhalen
er blijft dan 96 over

4	3	5	6	8	\	300	+	50	+	4	=	354
3	6	9	0	0								
	6	6	6	8								
	6	1	5	0								
		5	8	8								
		4	9	2								
			9	6								

Binair werkt dat net zo:

1101001 : 10011 = 100 + 1 = 101 (105 :19 = 5)
er blijft dan 1010 over (10)

1	1	0	1	0	0	1	\	100	+	1	=	101
1	0	0	1	1	0	0						
	1	0	0	0	0	1						
		1	0	0	1	1						
			1	0	1	0						

De rest na deling door 19 is dus 1010.
Bij deling door 19 is de maximale rest die je kunt hebben 18 = 10010.
Je hebt dus 5 bits aan CRC code nodig. Deze plak je achter de informatiebits.

De **CRC codering** van de letter "i" wordt nu: **1101001**01010

Wat gebeurt er nu aan de ontvangstkant? Hoe kunnen enkelvoudige fouten worden ontdekt?

2. Decodering.
We gaan uit van de letter 'i' = 105 = 1101001.
De CRC codering daarvan is **110100101010**.

Stel dat de ontvanger **110110101010** ontvangt (enkelvoudige fout).

De ontvanger voert de CRC check uit door de informatiebits te delen door 19.

1101101 : 10011 = 100 + 1 = 101 (109 :19 = 5)
er blijft dan 1010 over (14)

1	1	0	1	**1**	0	1	\	100	+	1	=	101
1	0	0	1	1	0	0						
	1	0	0	0	0	1						
		1	0	0	1	1						
			1	1	1	0						

De rest na deling door 19 is dus 1110.
Er is echter ontvangen **110110101010** en niet **110110101110**
Er is dus een fout gedetecteerd.

Ook twee fouten worden door de CRC check nog opgemerkt.

Op het moment dat er bij datatransport fouten worden zal de ontvanger een verzoek terugsturen om een pakketje nogmaals te sturen. Bij CRC-32 (de 4 bytes bij het 802.3 protocol) wordt meestal het getal 100000100110000010001110110110111 als deler genomen.

4.5 Samenvatting

Tijdens datatransport kunnen verstoringen in het signaal voorkomen waardoor **transmissiefouten** kunnen optreden. Door **pariteitsbits** toe te voegen aan de bitreeks zie je wilt verzenden, kan de ontvangende partij kan een **pariteitscheck** uitvoeren en nagaan of er een fout in de reeks is opgetreden. Dit noemen we **foutdetectie**.

Een bitreeks kan ook zodanig worden gecodeerd dat fouten niet alleen kunnen worden gedetecteerd maar ook kunnen worden gecorrigeerd. De **Hammingcode** is een coderingsmethode die **foutcorrectie** in bitreeksen mogelijk maakt.

De Hammingcode voegt aan de **informatiebits** uit een bitreeks **controlebits** toe. Aan de hand van deze controlebits kunnen fouten worden gedetecteerd. De methode werkt niet alleen voor detectie van fouten in de informatiebits, maar ook voor detectie van fouten in de controlebits.

De **CRC** (**Cyclic Redundancy Control**) code is een andere methode om bitreeksen zodanig te coderen dat foutdetectie mogelijk is. Bij deze methode wordt een bitreeks uitgebreid met de rest na deling door een bepaald, vastgelegd getal. Dit getal noemen we de **CRC generator**. Als de bitreeks goed is overgekomen moet de ontvanger dezelfde rest na deling hebben. Deze methode wordt bijvoorbeeld toegepast bij de datatransport volgens het 802.3 protocol en bij opslag van ZIP bestanden.

ANTWOORDEN

Opdracht 4.1
Wat gebeurt er als juist de pariteitsbit fout wordt verzonden?

Dan wordt ook de ontvangen code als fout aangemerkt!
Een ontvangen code 11011010 kan dus een ‘m’ zijn waarvan de informatiebits goed ontvangen zijn maar waarvan de controlebit fout ontvangen is!
Het is natuurlijk duidelijk dat dit probleem maar in een zeer beperkt deel (1/8) van de fout ontvangen boodschappen optreedt.

Opdracht 4.2
Wat gebeurt er als er meer dan één fout optreedt?

Natuurlijk kan het ook zo zijn dat er meerdere fouten in het transport van een code op kunnen treden! Het toevoegen van één enkele pariteitsbit biedt geen mogelijkheid om dit soort gevallen op te sporen. Zoals reeds eerder uitgelegd wordt een ‘i’ gecodeerd met 11010010 en wordt een ‘m’ gecodeerd met 11011011.
Een ontvangen code 11011101 zou zowel een ‘i’ met vier fouten als een ‘m’ met drie fouten kunnen zijn. Aangezien de pariteit van de code even is wordt er echter geen fout gedetecteerd en de code gedecodeerd als 1101110 = ‘n’.

Opdracht 4.3
Hoeveel fouten kunnen er in het bovenstaande voorbeeld in het transport maximaal worden gemaakt zodat de boodschap toch nog goed aankomt?

In elk vijftal bits kunnen twee fouten worden gemaakt:
In totaal zijn dat maximaal 7 x 2 = 14 fouten!

Opdracht 4.4
Ga na dat de controlebits 2, 4 en 8 respectievelijk de waarden ‘1’, ‘0’en ‘1’krijgen.

		1		1	0	1		0	0	1
1	2	3	4	5	6	7	8	9	10	11
	1	■		■		■		■		■
	2	■			■	■			■	■
	4			■	■	■				
	8							■	■	■

Controlebit 2 controleert de informatiebits 3 (**1**), 6 (**0**), 7 (**1**), 10 (**0**) en 11(**1**) en krijgt dus de waarde ‘1’ zodat de pariteit even wordt.
Controlebit 4 controleert de bits 5 (**1**), 6 (**0**) en 7 (**1**) en wordt dus ‘0’.
Controlebit 8 controleert de bits 9 (**0**), 10 (**0**) en 11 (**1**) en wordt dus ‘1’.

Opdracht 4.5
Bepaal de Hamming-code van de letter ‘m’ = 1101101

We gaan weer uit van de ASCII-codering ‘m’=1101101 en bekijken het bijbehorende schema waarmee de Hammingcode kan worden bepaald.

		1		**1**	**0**	**1**		**1**	**0**	**1**
1	2	**3**	4	**5**	**6**	**7**	8	**9**	**10**	**11**
	1									
	2									
	4									
	8									

Controlebit 1 controleert bijbehorende informatiebits 3, 5, 7, 9 en 11 en wordt ‘1’ om de pariteit van de even te maken, controlebit 2 controleert informatiebits 3, 6, 7, 10 en 11 en wordt ‘1’, controlebit 4 controleert informatiebits 5, 6 en 7 en wordt ‘0’ en controlebit 8 controleert informatiebits 9, 10 en 11 wordt ‘0’.

De codering die ontstaat is de volgende:

1	1	**1**	0	**1**	**0**	**1**	0	**0**	**0**	**1**
1	2	3	4	5	6	7	8	9	10	11

Dit geeft de Hamming-code ‘m’ = 11101010001

Opdracht 4.6
Waarom is de foute informatiebit de bit met de plaats van de som van de niet kloppende controlebits?

Ten eerste stel je vast **dat** er een transmissiefout is opgetreden **omdat** de controlebits niet kloppen. **Eén** van de informatiebits moet hiervoor verantwoordelijk zijn. Door die informatiebit te wijzigen die bij **alle** niet kloppende controlebits hoort kan je de code weer kloppend krijgen. Die informatiebit is precies die informatiebit die als plaats de som van de niet kloppende informatiebits heeft!

Opdracht 4.7
Hoe groot is de Hamming-afstand tussen de Hamming-codes van de letters ‘i’ en ‘m’?
‘i’ = **0**110101**10**01
‘m’ = **1**110101**01**01

De codering van de letters heeft dus de noodzakelijke Hamming-afstand 3 om detectie en correctie van één transmissiefout mogelijk te maken.

Opdracht 4.8
Welke letter hoort er bij een ontvangen Hamming-code:
a) 11**1**11**0**11**00**
b) 01**1**0**1**0**0**1**01**

a) een fout ontvangen ‘d’, fout in bit 7 (controlebits 1, 2 en 4 kloppen niet)
b) een correct ontvangen ‘u’