Bab

10

Deteksi dan Koreksi Error

Bab ini membahas mengenai cara-cara untuk melakukan deteksi dan koreksi error.

Data dapat rusak selama transmisi. Jadi untuk komunikasi yang reliabel, error harus dideteksi dan diperbaiki/ koreksi.

Pendahuluan Deteksi dan Koreksi Error

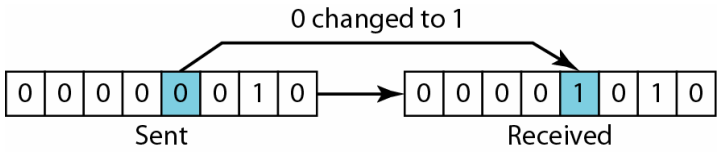
Berikut dibahas mengenai isu-isu yang berkaitan dengan deteksi dan koreksi error.

Tipe Error

Ketika bit mengalir dari satu titik ke titik lainnya, mereka adalah subjek terhadap perubahan tak terduga karena interferensi. Interferensi dapat merubah bentuk sinyal.

Single bit error

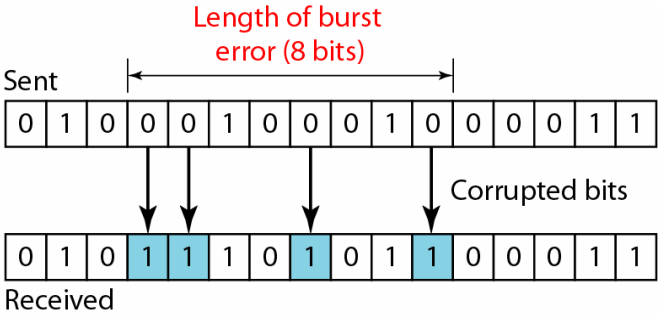
Istilah single bit error berarti hanya satu bit dari unit data yang dikirim berubah.



Gambar 10.1 – single bit error

Burst error

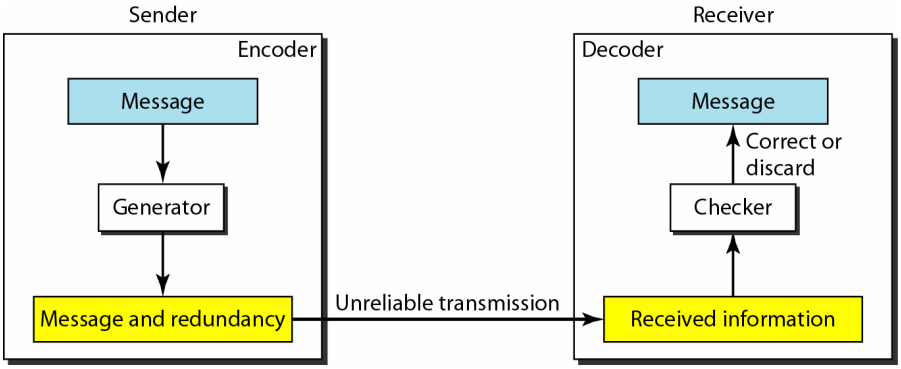
Istilah burst error berarti dua atau lebih bit dalam unit data berubah dari 1 jadi 0 atau dari 0 jadi 1.



Gambar 10.2 – burst error

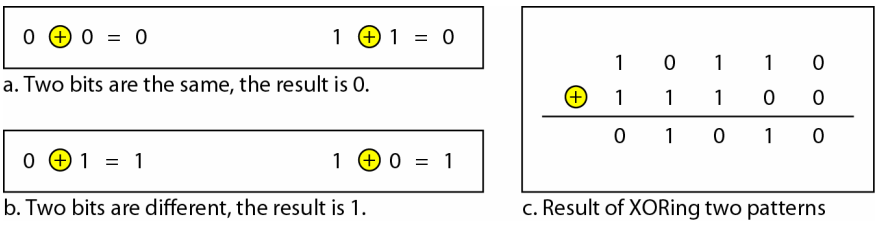
Redundansi

Untuk mendeteksi error atau memperbaiki data, kita perlu mengirimkan bit ekstra (redundan) beserta data.



Gambar 10.3 – Struktur Encoder dan Decoder

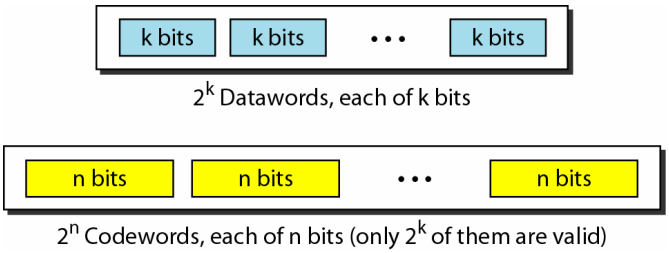
Dalam aritmetik modulo-N, kita hanya menggunakan bilangan bulat antara 0 s.d. N-1, inklusif.



Gambar 10.4 – XORing dari dua bit tunggal atau dua word

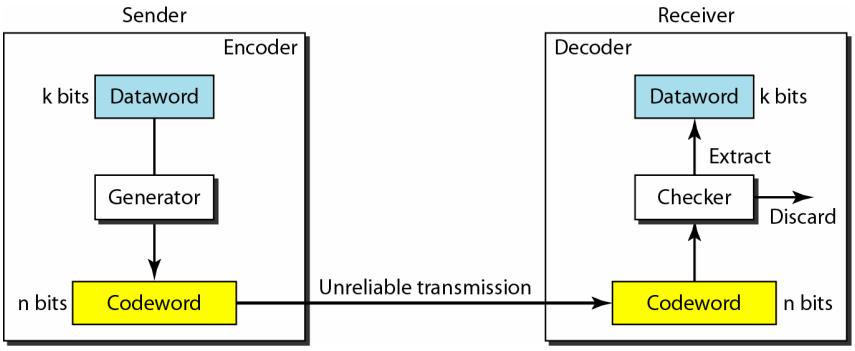
Block Coding

Dalam block coding, kita membagi pesan menjadi blok-blok, setiap k bit, dinamakan dataword. Kita menambahkan r bit yang redundan ke setiap blok sehingga panjang menjadi n = k + r. Blok n bit hasilnya dinamakan codeword.



Gambar 10.5 – dataword dan codeword

Block coding 4B/5B adalah salah satu contoh coding. Dengan skema coding ini, k=4 dan n=5. Seperti kita lihat, kita memiliki $2^k = 16$ dataword dan $2^n = 32$ codeword. Kita lihat bahwa 16 dari 32 codeword digunakan untuk pengiriman pesan, dan sisanya digunakan untuk hal lain atau tidak digunakan.



Gambar 10.6 – Proses deteksi error dalam block coding

Misalkan k = 2 dan n = 3. Tabel di bawah ini menyatakan dataword dan codewordnya. Kita lihat bagaimana menurunkan codeword dari dataword.

Asumsikan pengirim meng-encode dataword 01 sebagai 011 dan mengirimkannya ke penerima. Pertimbangkan kasus-kasus berikut :

- Penerima menerima 011. Ia adalah codeword yang valid. Penerima mengekstraksi dataword 01 dari codewordnya.
- Codeword berubah/ korup selama transmisi, dan yang diterima 111. Ia adalah codeword yang tidak valid dan diabaikan.
- Codeword berubah/ korup selama transmisi, dan yang diterima 000. Ini adalah codeword yang valid. Penerima mengekstraksi dataword yang salah yaitu 00. Dua bit yang korup menjadikan error tidak terdeteksi.

Kode untuk deteksi error.

Dataword	*Codeword*
00	000
01	011
10	101
11	110

Misalkan kita tambahkan bit-bit redundan ke contoh di atas untuk melihat jika penerima dapat memperbaiki error tanpa mengetahui apa yang dikirim. Kita tambahkan 3 bit redundan ke 2 bit dataword menjadi 5 bit codeword. Tabel berikut menyatakan dataword dan codewordnya.

Dataword	*Codeword*
00	00000
01	01011
10	10101
11	11110

Misalkan datawordnya 01. Pengirim membuat codeword 01011. Codeword berubah dalam transmisi, dan 01001 yang diterima. Pertama-tama penerima menemukan bahwa codeword yang diterima tidak ada dalam tabel. Artinya terjadi error. Penerima mengasumsikan bahwa hanya 1 bit yang berubah, gunakan strategi berikut untuk menebak dataword yang benar.

- Bandingkan codeword yang diterima dengan codeword pertama dalam tabel (01001 vs 00000), penerima memutuskan bahwa codeword pertama bukan yang dikirim karena ada dua bit yang berbeda.
- Dengan alasan yang sama, codeword bukan berupa data ketiga atau data keempat dalam tabel.
- Codeword asli adalah data kedua datam tabel, karena hanya ini yang berbeda 1 bit. Penerima menimpa 01001 dengan 01011 dan mengacu ke tabel untuk menemukan dataword 01.

Hamming distance/ Jarak Hamming

Jarak Hamming antara dua word adalah jumlah perbedaan antara bit-bit yang berkorespondensi.

Misalkan kita hitung Jarak Hamming antara dua pasang word :

- Jarak Hamming d(000, 011) adalah 2 karena

$000 \oplus 011$ is 011 (two 1s)

- Jarak Hamming d(10101, 11110) adalah 3 karena

$10101 \oplus 11110$ is 01011 (three 1s)

Jarak Hamming minimum adalah Jarak Hamming terkecil di antara semua pasangan yang mungkin dalam sekumpulan word.

Untuk menjamin deteksi sampai dengan s buah error dalam semua kasus, Jarak Hamming minimum dalam block code harus $d_{min} = s + 1$.

Jarak Hamming minimum untuk skema pada tabel pertama di atas adalah 2. Kode ini menjamin deteksi hanya pada error tunggal. Contohnya, jika codeword (101) dikirim dan error terjadi, codeword yang diterima tidak cocok dengan codeword valid yang ada. Jika dua error terjadi, codeword yang diterima mungkin valid dan error tidak terdeteksi..

Skema pada tabel ke dua, nilai d_{min} = 3. Kode ini dapat mendeteksi dua error. Tetapi jika tiga error terjadi, mungkin saja codeword yang diterima tetap valid dan error tidak terdeteksi.

Untuk menjamin koreksi terhadap t buah error dalam semua kasus, Jarak Hamming minimum dalam block code harus $d_{min} = 2t + 1$.

Linear Block Code

Hampir semua block code yang digunakan saat ini termasuk pada linear block code. Linear block code adalah kode yang mana exclusive OR (penambahan modulo 2) dari dua codeword valid menghasilkan codeword valid lainnya.

Misalkan jika kedua kode pata tabel di atas adalah kelas dari linear block code, maka:

- Skema pada tabel pertama adalah linear block code karena hasil XOR codeword apapun dengan codeword lainnya adalah codeword valid.
- Skema pada tabel kedua juga linear block code. Kita dapat membuat semua codeword dengan XOR dari codeword lainnya.

Parity Check

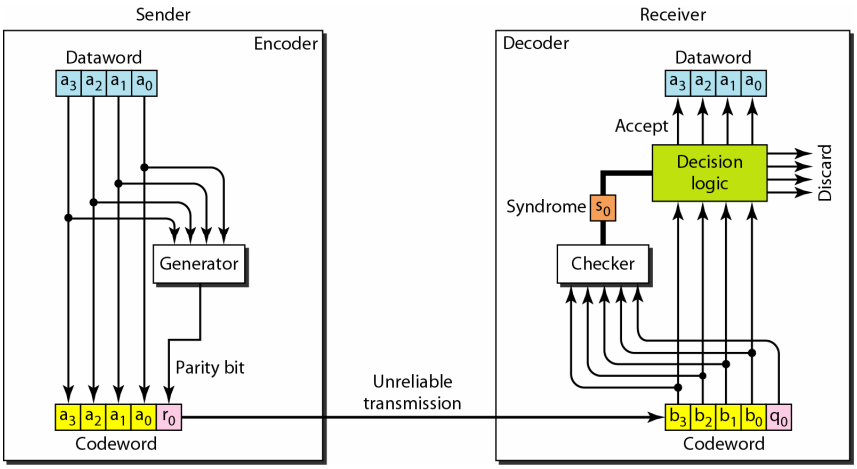
Simple Parity Check

Kode simple parity check adalah kode bit tunggal untuk deteksi error yang n = k +1 dan d_{min} = 2.

Tabel di bawah adalah kode single parity check C(5,4)

Dataword	*Codeword*	*Dataword*	*Codeword*
0000	00000	1000	10001
0001	00011	1001	10010
0010	00101	1010	10100
0011	00110	1011	10111
0100	01001	1100	11000
0101	01010	1101	11011
0110	01100	1110	11101
0111	01111	1111	11110

Dengan teknik ini, bit redundan yang dinamakan parity bit ditambahkan pada setiap dataword/ unit data, sehingga jumlah bit 1 pada unit tersebut/ codeword menjadi genap (atau ganjil).



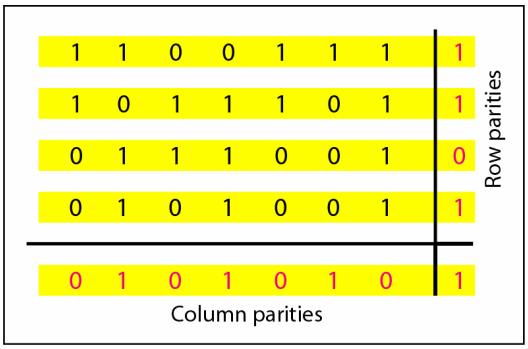
Gambar 10.7 – Encoder dan decoder untu simple parity check

Misalkan kita mengirim data 1011. Codeword yang dihasilkan adalah 10111, yang kemudian dikirim ke penerima. Kita lihat lima kasus berikut :

- Tidak terjadi error, codeword yang diterima adalah 10111. Syndrome adalah 0. Dataword 1011 yang dibuat.
- Terjadi error bit tunggal pada a1, codeword yang diterima adalah 10011. Syndrome adalah 1. Tidak ada dataword yang dibuat.
- Terjadi error bit tunggal pada r0. Codeword yang diterima adalah 10110. Syndrome adalah 1. Tidak ada dataword yang dibuat.
- Terjadi error yang mengubah r0 dan a3. Codeword yang diterima adalah 00110. Syndome adalah 0. Dataword 0011 yang dibuat. Perhatikan di sini dataword dibuat karena nilai syndrome adalah 0, walaupun nilai dataword salah.
- Tiga bit a3, a2 dan a1 berubah. Codeword yang diterima adalah 01011. Syndrome 1. Dataword tidak dibuat. Ini menunjukkan simple parity check, menjamin pendeteksian error tunggal, juga dapat mendeteksi jumlah error yang ganjil.

Two dimensional parity check

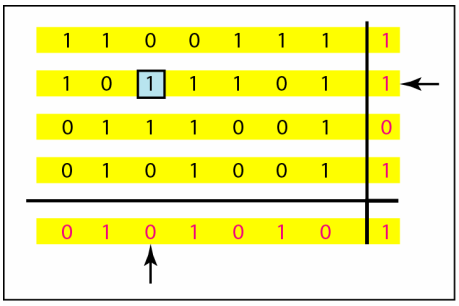
Metode ini lebih baik dari single parity check. Dalam metode ini blok bit diorganisasikan ke dalam tabel (terdiri dari baris dan kolom). Pertama-tama kita hitung parity bit untuk setiap unit data, kemudian diorganisasikan ke dalam tabel. Kemudian kita hitung parity bit untuk setiap kolom.



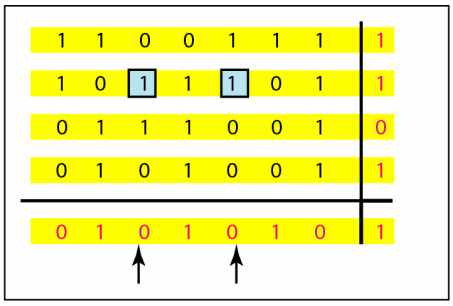
a. Design of row and column parities

Gambar 10.8 – Parity check dua dimensi

Kemungkinan terjadi error dapat dilihat pada gambar-gambar berikut :

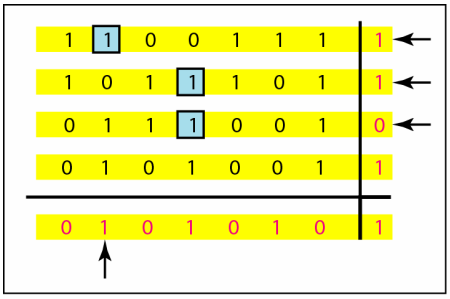


b. One error affects two parities

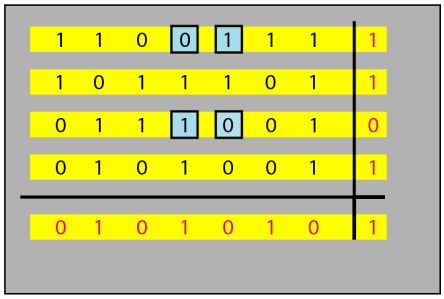


c. Two errors affect two parities

Gambar 10.9 – parity check dua dimensi jika terjadi error



d. Three errors affect four parities



e. Four errors cannot be detected

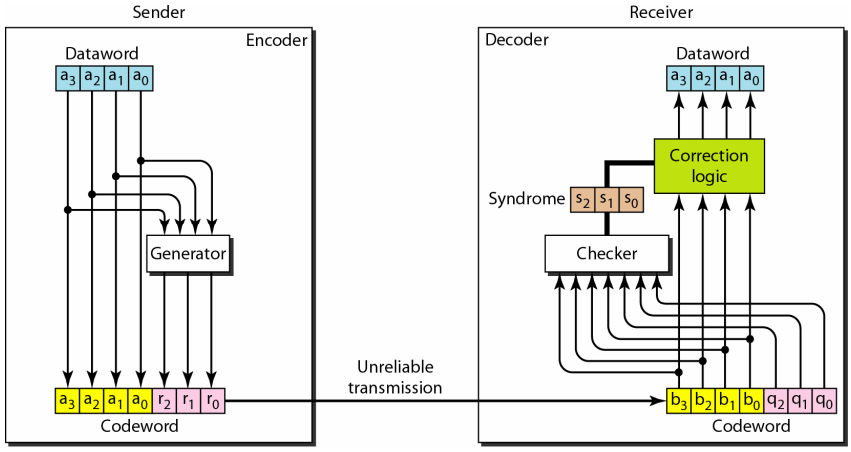
Gambar 10.10 – parity check dua dimensi jika terjadi error

Hamming Code

Hamming menyediakan solusi praktis dengan Hamming code. Hamming code dapat diaplikasikan pada panjang data berapapun dan menggunakan hubungan antara data dan bit redundan. Contohnya pada data 7 bit ditambahkan 4 bit redundan.

Berikut tabel Hamming code C(7,4)

Dataword	Codeword	Dataword	Codeword
0000	0000000	1000	1000110
0001	0001101	1001	1001011
0010	0010111	1010	1010001
0011	0011010	1011	1011100
0100	0100011	1100	1100101
0101	0101110	1101	1101000
0110	0110100	1110	1110010
0111	0111001	1111	1111111



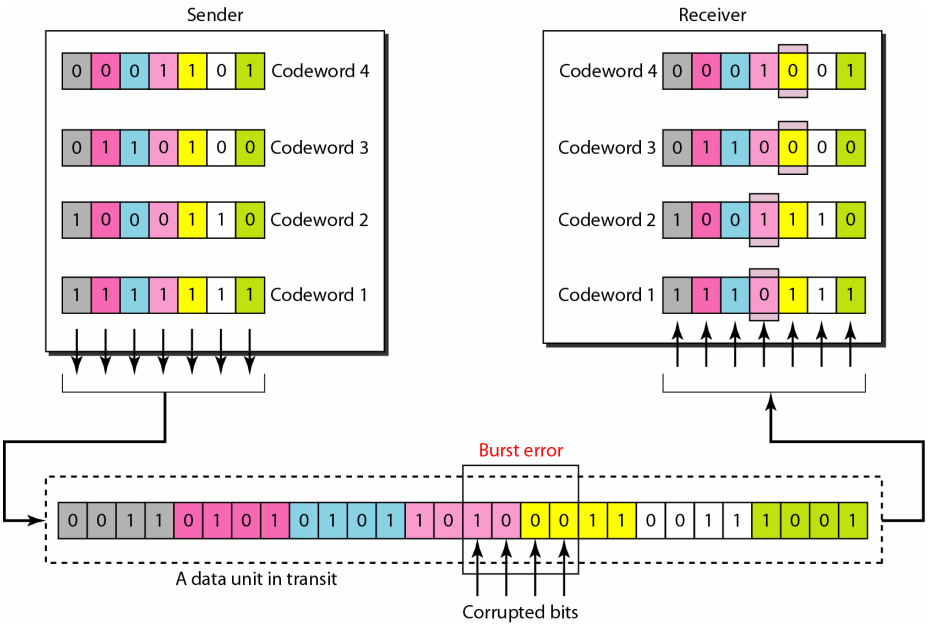
Gambar 10.11 – Encoder dan decoder menggunakan Hamming code

Syndrome	000	001	010	011	100	101	110	111
Error	none	q_0	q_1	b_2	q_2	b_0	b_3	b_1

Keputusan logis oleh correction logic analyzer

Mari kita menelusuri jalur dari tiga dataword dari pengirim ke tujuan.

- Dataword 0100 menjadi 0100011. yang diterima codeword 0100011. Syndrome adalah 000, dataword akhir adalah 0100.
- Dataword 0111 menjadi codeword 0111001. Syndrome adalah 011. Setelah mengubah b2 (dari 1 menjadi 0), dataword akhir menjadi 0111.
- Dataword 1101 menjadi codeword 1101000. Syndrome 101. setelah mengubah b0, kita memperoleh 0000, dataword yang salah. Kita lihat Hamming code tidak bisa memperbaiki dua error.



Gambar 10.12 – Burst error menggunakan Hamming code

Cyclic Codes

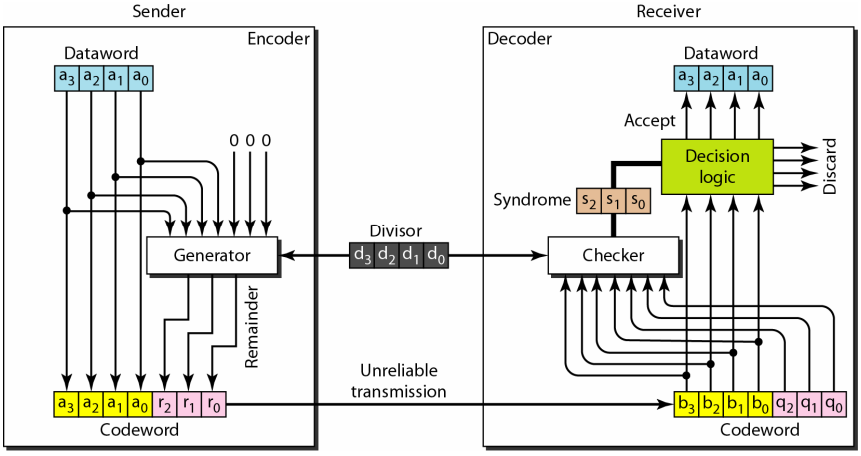
Cyclic code adalah linear block code khusus dengan satu properti ekstra. Dalam cyclic code , jika codeword berotasi (bergeser secara berputar), hasilnya adalah codeword lainnya.

Cyclic redundancy check (CRC)

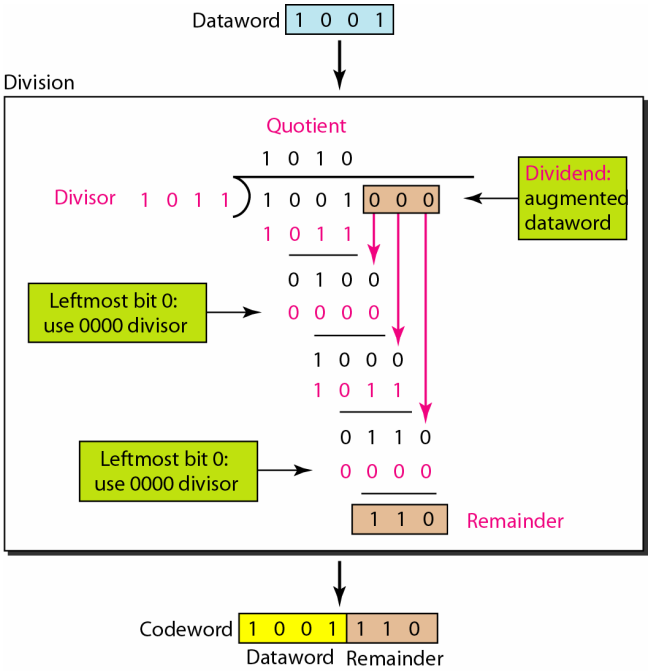
CRC berbasis pada pembagian biner. Sekumpulan bit redundan memanggil CRC atau sisa CRC, ditambahkan pada akhir data sehingga data hasil dapat dibagi tepat dengan detik, bilangan biner yang ditentukan dulu. Di tujuan, unit data yang dibagi dengan bilangan yang sama. Jika pada tahap ini tidak ada sisa, unit data diterima. Sisa mengindikasikan unit data rusak dalam perjalanan dan harus ditolak.

Bit redundan yang digunakan dalam CRC diturunkan dengan membagi unit data dengan pembagi; sisanya/ remainder adalah CRC. Supaya valid, CRC memiliki dua sifat, yaitu ia harus persis kurang satu dari pembagi, dan dengan menambahkan pada akhir string data harus membuat bit-bit hasil dapat dibagi dengan pembagi.

Tabel berikut adalah CRC code C(7,4)



Gambar 10.13 –CRC encoder dan decoder



Gambar 10.14 – Pembagian pada CRC encoder

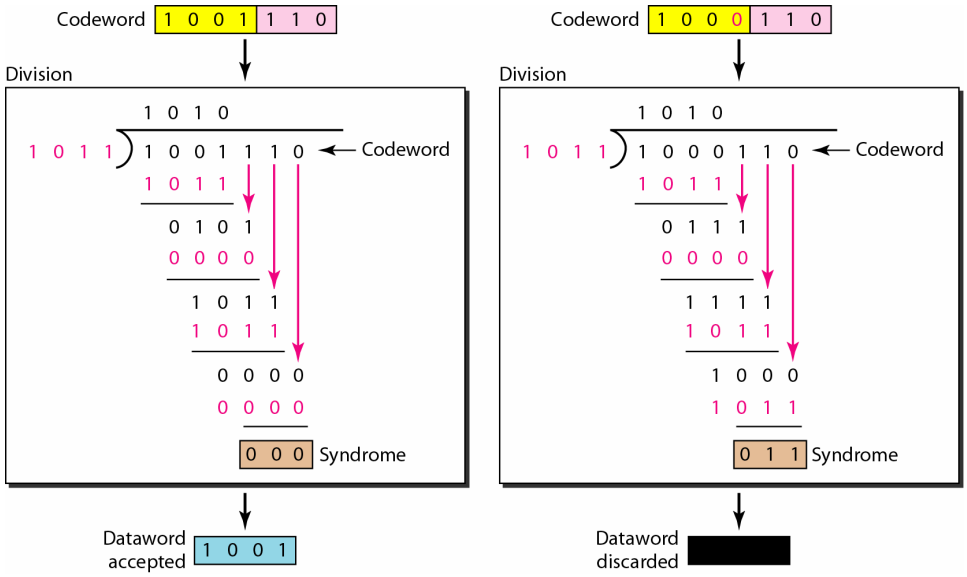
Pertama-tama string 0 sepanjang n ditambahkan ke unit data. Besar n adalah kurang 1 dari jumlah bit dalam pembagi (pembagi = n + 1).

Kedua, unit data yang telah ditambah bit 0 dibagi dengan pembagi, menggunakan proses pembagian biner. Sisa

Ketiga, CRC dari n bit diturunkan pada tahap dua mengganti bit 0 yang ditambahkan pada akhir data.

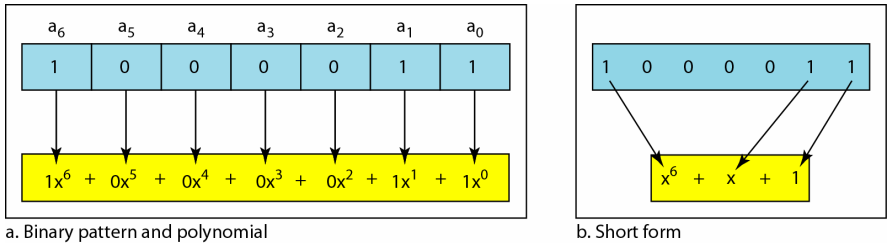
Unit data datang pada penerima terlebih dulu, baru CRC. Penerima memperlakukan seluruh string sebagau satu unit dan membaginya dengan pembagi yang sama yang digunakan oleh pengirim.

Jika string yang diterima tanpa error, CRC checker memperoleh sisa 0 dan unit data diterima. Jika string berubah dalam transmisi, sisanya bukan 0 dan unit data tidak diterima.



Gambar 10.15 – Kasus pembagian dalam CRC decoder

Pembagi dalam CRC generator seringkali dinyatakan tidak dalam string 1 dan 0, tetapi dalam polinomial aljabar. Format polinomial berguna karena dua hal, pertama karena singkat dan kedua karena ia dapat digunakan untuk membuktikan konsep secara matematis.



Gambar 10.16 – polinom untuk menyatakan data biner

Polinom harus dipilih untuk memiliki paling sedikit dua karakteristik berikut :

- Ia harus tidak dapat dibagi oleh x
- Ia harus dapat dibagi oleh x+1

CRC adalah metode deteksi error yang sangat efektif. Jika pembagi dipilih berdasarkan aturan, maka :

- CRC dapat mendeteksi semua burst error terhadap bit nomor ganjil.
- CRC dapat mendeteksi semua burst error dengan panjang kurang dari atau sama dengan derajat polinom.

- CRC dapat mendeteksi dengan probabilitas tinggi, burst error dengan panjang lebih dari derajat polinom.

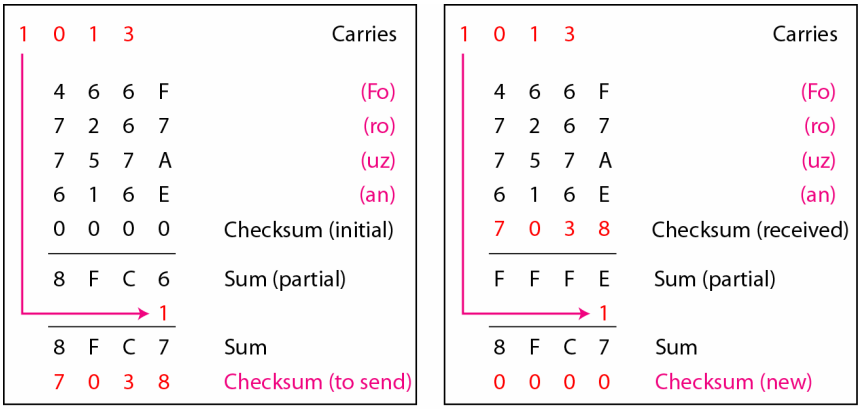
Nama	*Polinom*	*Aplikasi*
CRC-8	$x^8 + x^2 + x + 1$	ATM header
CRC-10	$x^{10} + x^9 + x^5 + x^4 + x^2 + 1$	ATM AAL
CRC-16	$x^{16} + x^{12} + x^5 + 1$	HDLC
CRC-32	$x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$	LAN

Checksum

Checksum digunakan dalam Internet oleh beberapa protokol meskipun bukan bagian dari data link layer. Checksum menggunakan konsep redundansi untuk mendeteksi error.

Pada sisi pengirim, checksum generator membagi-bagi unit data menjadi segmen-segmen yang sama jumlah bitnya (biasanya 16 bit). Segmen-segmen ini dijumlahkan, sehingga panjang total bitnya tetap sama. Totalnya ini dikomplemenkan dan ditambhakna ke unit data aslinya sebagai bit redundan yang merupakan field checksum. Unit data yang sudah bertambah ini yang dikirimkan.

Penerima membagi-bagi lagi unit data dan menjumlahkan semua segmen dan mencari komplemen dari hasilnya. Komplemen ini dibandingkan dengan data yang ada pada field checksum.



a. Checksum at the sender site

a. Checksum at the receiver site

Gambar 10.17 – contoh penggunaan checksum