CRC Cyclic Redundancy Check
Analysing and Correcting Errors

Prof. Dr. W. Kowalk
Universität Oldenburg
Fachbereich Informatik
August 2006

Oldenburg, 2003
Author's E-Mail-Address
kowalk@informatik.uni-oldenburg.de
The Text has been written with StarOffice 8 and exported to PDF

Contents

1 Introduction ... 3
1.1 About this report ... 3
1.2 Overview ... 3
2 Applications ... 3
3 Mathematical foundations ... 4
3.1 The operator & ... 4
3.1.1 Properties of the operator & ... 5
3.1.2 One polynomial and zero polynomial ... 5
3.1.3 Complement of a polynomial ... 5
4 Bitfilter ... 6
4.1 Even and uneven bitfilters ... 6
4.2 Properties of even bitfilters ... 7
4.3 Example ... 7
4.4 Construction of bitfilters ... 8
5 The CRC method ... 9
5.1 Types of errors ... 10
5.1.1 Error bursts ... 10
5.1.2 Two bit errors ... 10
5.1.3 Uneven number of bit errors ... 11
5.1.4 Error correction ... 11
5.1.5 Maximum period of bitfilters ... 12
6 References ... 12
7 Appendix ... 12
7.1 Some optimal Generators ... 12
7.2 Programme ... 13

1 Introduction

1.1 About this report

This report analyses a method called cyclic redundancy check (CRC), which is in widespread use currently. This method can detect and correct errors in sequences of bits and can therefor be used in data transmission as well as in data storage to protect files from errors.

CRC is in use in many applications and standards, and is easily integrated in hardware. The method can be described by polynomial division, but since its property seem to be obscure to many applicants we will give a detailed explanations of its properties and how we can prove them.

Main goal of this paper is to introduce a new and simple theory to characterize the types of errors that can be detected or corrected by this method.

1.2 Overview

The method presented uses bitfilters to describe important properties of CRC method. A bitfilter considers only some to the terms of an error polynomial and derives from its structure the properties of the error. Bitfilters can be analysed independently of an error so that many new and important properties of CRC method can be derived.

We explain general foundations of the polynomial representation of sequences of bits, introduces operations on those bit sequences, and introduces then bitfilters in general as well as special bitfilters, which can be specified for each generator polynomial. Analyses of bitfilters with even parity to a a generator produces the results of this paper. This encloses exact statements of the kind of recognized errors, as well as the possibility to correct single bit errors.

This report is a shortened version of longer report, which includes more examples and more precise proofs.

2 Applications

Data are stored in bytes, which consist of 8 bits each. Thus any set of data can be considered as sequence of bytes or bits. In computer communication usually the bits are considered to be independent and a bit sequence is limited by the size of data blocks (or frames, packets, datagrams etc.). Thus we consider the problem to check whether a finite sequence of bits is changed unintentionally, e.g. by transmission errors or faulty memories.

To solve this problem (namely to protect data from unintentional errors) we add some redundant information to the bit sequence (which represents the data). To get those redundancy information we perform some operations on the bit sequence, the result of which is stored as redundancy information. The receiver of a message, or the reader of memory input can make the same operations and compare against the stored redundancy information. If both are the same there

is a good chance no error occurred. If both differ there is definitely an error; however, this may be in the redundancy information as well as in the data, so in this case there is as well only a good chance for an error.

Cyclic redundancy check performs a special operation that can be interpreted as polynomial division. The residual of this division is used as redundancy information and we will explore in this paper which types of errors can be detected by this method. The dividend is usually called generator and we will write $G(X)$ or G for this in the rest of this paper.

3 Mathematical foundations

A sequence of bits *abcdef*..., where *a* is 0 or 1, *b* is 0 or 1 etc. is called a bit sequence. We use the field 0-1 to perform some computation on the bits, i.e. add and multiply those bits; its well known that the only difference to integer arithmetic is 1+1=0.

Instead of a sequence of bits one can use polynomials, where

$$A(X)=\sum_{i=0}^{\infty} a_i \cdot X^i \text{ where } a_i, X \in \{0,1\}.$$

We use here the following convention:

$$abcdef\ldots \text{ is mapped to: } A(X)=a+b\cdot X+c\cdot X^2+d\cdot X^3+e\cdot X^4+f\cdot X^5\ldots.$$

Thus we use a 'high ending' sequence to map between polynomials and bit sequences.

Some polynomial operations like addition ('+') and multiplication ('·') as well as division ('/') are defined, which we will use as well. Addition and subtraction are the same, because in the field 0-1 we have always

a+b = a–b, e.g. 1+1=1–1=0.

We will use the notions of bit sequences and polynomials interchangeable, i.e. we might say bit *k* of a sequence A is one, which is the same as $A(X)$ holds X^k; or we talk about a bit sequence $X^3+X^5+X^6$, which means 0001011..., etc.

The number of terms in $A(X)$ (or the number of 1s in A) is called the parity of $A(X)$ or A. We are usually interested in even or odd parity which means even or odd terms in $A(X)$ or 1s in A.

3.1 The operator &

We introduce a new operator for polynomials, namely '&' (or '∧') with the meaning of multiplication of each corresponding terms of two polynomials. Formally we can state

$$A(X)\wedge B(X)=\sum_{i=0}^{\infty} a_i\cdot X^i \ \ \& \ \ \sum_{i=0}^{\infty} b_i\cdot X^i=\sum_{i=0}^{\infty}(a_i\cdot b_i)\cdot X^i.$$

The meaning of this definition is of course that the resulting term exist if and only if each corresponding term of the parameter polynomials exist. Logically we can explain this as term by

term conjunction.

Using this operator '&' a polynomial $F(X)$ can be used to select (or filter) some bits of another polynomial $A(X)$ by setting the corresponding terms of $F(X)$. The resulting polynomial $S(X)=F(X)\&A(X)$ has a term X^k if and only if $F(X)$ has the term X^k, and the corresponding term X^k exists in $A(X)$, as well. Thus we call such a polynomial $F(X)$ *bitfilter*, although its a normal polynomial with nothing else special about it, besides its special application.

3.1.1 Properties of the operator &

When the operator '&' is used in expressions together with other operators it has the following properties:

a) Operator '&' is commutative and associative.

b) Operator '&' is distributive over '+'.

c) Operator '&' is not distributive over '·'.

d) Neither '+' nor '·' are distributive over '&'.

a) and b) are easily proved, since multiplication is commutative, associative and distributive over addition. c) and d) can be proved by counter examples.

3.1.2 One polynomial and zero polynomial

Let be $1(X) = 1+X+X^2+X^3+X^4+...+X^{k-1}+X^k+X^{k+1}+...$ (or as a bit sequence 11111111...111...). This is called the 1-polynomial. We call 0(X) = 0 the 0-polynomial (or as a bit sequence 0000000000..000..). While the one polynomial is very important in our theory, the zero polynomial will only be used for systematic reasons in some special cases.

3.1.3 Complement of a polynomial

We call $F^c(X) = F(X) + 1(X)$ the complement of $F(X)$. $F^c(X)$ holds exactly those terms that $F(X)$ does not hold. Obviously we have

$1^c(X) = 0(X)$,

$F^c(X) + F(X) = 1(X)$,

$F^c(X) \& F(X) = 0(X)$.

As a bit sequence we can write

- $(111111...)^c = 000000...$
- $(abcdefg...)^c = a^c b^c c^c d^c e^c f^c g^c ...$
- $(abcdefg...)^c + abcdefg\ ... = 111111...$
- $(abcdefg...)^c \& abcdefg ... = 000000...$

4 Bitfilter

A **bitfilter** describes a subset of a sequence of bits. This subset can be described by a sequence of bits, where bit k is set if and only if the bit k of another sequence of bits is to be selected, or 'filtered'. Thus the bitfilter is a sequence of bits, or a polynomial that is used to describe such sequence of bits, as has been shown above. We write $F(X)$ for a polynomial that is used as a bitfilter.

The length of a bitfilter is usually unlimited, although we will soon see that there is (in our special applications) always a cycle in that sequence, so that the sequence of bits repeats again and again. This cycle's period is usually very long and will be called p. If p is the period of bitfilter $F(X)$, then X^k is a term of $F(X)$ if and only if X^{k+p} , X^{k+2p} , .. are terms of $F(X)$, or $X^k=X^{k+jp}$ for all positive integer $k\geq 0$ and $j>0$. Thus it is sufficient to consider only the terms $1, X, X^2 \ldots X^{p-1}$ of $F(X)$.

4.1 Even and uneven bitfilters

The following property sets a relationship between a generator polynomial $G(X)$ and a bitfilter $F(X)$. A bitfilter $F(X)$ is defined to have even parity (in relation) to $G(X)$, if at all positions of the generator $G(X)$ in $F(X)$ the number of terms in both polynomials at the same position is even. To state this formally we only have to write

$F(X)$ & ($G(X) \cdot X^k$) $|_{X=1} = 0$, for all $k\geq 0$.

The formula from above means that you replace in $F(X)$ X by 1. If $F(X)$ has an even number of terms, then $F(1) = 0$, otherwise $F(1) = 1$. For example $F(X) = X^9+X^6+X^4+X^3+1 \,|_{X=1} = 1$, while $F(X) = X^9+X^6+X^3+1 \,|_{X=1} = 0$. Remember, that + means addition in the field 0-1, thus 1+1=0!

To give an example with a bit sequence, let be

G = `1100101`

then we find for the filter sequence of a filter F with even parity to G:

F = **1**`00000`**1**`0`**1**`0`**1**`00`**1**`1`**0**`1`**1**`11`**0**`00...`

G **1**`100`**1**`0`**1** `1`**1**`00`**1**`0`**1** `1`**1**`00`**1**`01`

Wherever you put the sequence G, you find an even number of 1s in G that correspond to a 1 in F. We usually say shortly: $F(X)$ is an even bitfilter to $G(X)$.

If $F'(X)$ & ($G(X) \cdot X^k$) has uneven parity for all k>0, then $F'(X)$ is said to have uneven parity (in relation) to $G(X)$. It should be clear that this properties restrict the number of bitfilters very much. There are of course many polynomials that have neither even nor uneven parity to a generator, but they are of no interest to us in the rest of this paper. Mainly we will consider only bitfilters with even parity in relation to a generator.

The reason for this definition will become clear quite soon. Since division of $A(X)$ by $G(X)$ means nothing else but adding a generator $G(X)$ in different positions to a $A(X)$, we only add terms

that preserve the parity of $A(X)\&F(X)$. Thus an error with uneven parity in $A(X)\&F(X)$ will always be preserved, since there must be at least one term left. Details of this will be given soon.

4.2 Properties of even bitfilters

An even bitfilter $F(X)$ to a generator $G(X)$ has some interesting properties. Let $A(X)$ be an arbitrary polynomial, then

$$F(X)\ \&\ (V(X)+G(X)*X^k)\big|_{X=1}=F(X)\ \&\ V(X)\big|_{X=1}\ \text{for all}\ k\geq 0.$$

This means that addition of $G(X)$ to a polynomial $V(X)$ at any position does not change the parity of the filtered bits of $F(X)\&V(X)$, provided $F(X)$ is an even bitfilter to $G(X)$. The prove is simple, since

$$F(X)\ \&\ \left(V(X)+G(X)*X^k\right)\big|_{X=1}=\left(F(X)\ \&\ V(X)+F(X)\ \&\ G(X)*X^k\right)\big|_{X=1}=$$
$$=F(X)\ \&\ V(X)\big|_{X=1}\ \text{for all}\ k\geq 0,$$

and the second term in the second expression is zero from definition of even bitfilter.

Since division is performed by addition of the generator at some positions this result says that parity isn't changed by division for the set of filtered bits. From this we will soon see we can derive many interesting results.

A very simple even bitfilter to a generator $G(X)$ with even parity is $1(X)$. From this follows already that all generators with even terms allow to detect all uneven number of bit errors.

The complement of a bitfilter has similar properties as the bitfilter. From

$$F^c(X)\&G(X)\cdot X^k=\left(1(X)+F(X)\right)\&G(X)\cdot X^k=G(X)\cdot X^k+F(X)\&G(X)\cdot X^k$$

follows that with even parity of $G(X)$ also the complement of an even bitfilter has even parity, while with uneven parity of $G(X)$ the complement of an even bitfilter has uneven parity (in relation to $G(X)$, respectively).

4.3 Example

An example might help to understand these properties. Since we have to divide we write the smallest Bit (i.e. X^0) to the right! The generator be **1**0**1**0**11** or $G(X)=X^0+X^1+X^3+X^5$. The error be 00000**1**00**111**00000 or $E(X)=X^5+X^6+X^7+X^{10}$. A bitfilter with even parity to $G(X)$ is ...0**11**0**11**10000**1**0**1**0 or $F(X)=X^1+X^3+X^8+X^9+X^{10}+X^{12}+X^{13}+...$ (Construction of bitfilters follows). Then follows $F(X)\&E(X)=X^{10}$, i.e. there is an odd number of terms in $F(X)\&E(X)$, thus there must be an residual when $E(X)$ is divided by $G(X)$. E.g. $F(X)\&(E(X)+G(X)\cdot X^{10})_{|X=1}\neq 0$, etc.

F = 00**11**0**111**0000**1**0**1**0

E = 00000**1**00**111**00000 $F\&E$ = 00000**1**0000000000

G_1 = **1**0**1**0**11** $G_1\&E$= 0000000**1**0**11**00000

The reader should check that $E(X)$ divided by $G(X)$ leaves an residual. As an counter example take

the error **1**000000000000000**1** or $E(X)=X^0+X^{15}$. This does not leave a residual when divided by $G(X)$, as can be checked. The reason is that there is no bitfilter with which you can filter exactly one of the two bits, since all bitfilters with even parity to $G(X)$ have a cycle with period 15, as will be shown later.

4.4 Construction of bitfilters

To construct a bitfilter $F(X)$ of even parity to a generator $G(X)$ of degree n one can use an arbitrary sequence of n bits. Since $G(X)$ is one bit longer not all bits match $G(X)$, so we can uniquely determine which is the next bit at the left and right side. For example in the sequence below, one starts with a sequence (here 100000), sets at first G at this position and counts the number of matching 1s. If they are uneven a 1 is to be appended to F, otherwise a 0 is to be appended. Then shift G one position from left to right and proceed.

F = **1**00000**1**010**1**1001**1**10**1**111**1**000...

G **1**100101**1**...

This means that n bits or 2^n distinct combinations of bits determine any even bitfilter to G. Since after some time there must be a repetition of an already produced combination of n bits, there is a cycle in any even bitfilter to a generator G. Since left and right side are uniquely determined, there is no bifurcation of such sequences.

The period of a bitfilter is a very important property that will be considered soon as the key to the main properties of CRC. Although there are 2^n distinct bitfilters there should be some with similar properties. Since there are cycles in these bitfilters we call bitfilters equivalent when the sequence of bits is the same, although they do not have the same phase.

Let $F(X)//X^k$ be defined as

$$F(X)//X^k=\sum_{i=0}^{\infty} f_{i+k}\cdot X^i.$$

then two distinct bitfilters $F(X)$ and $F'(X)$ are equivalent, if and only if there is a k so that $F(X)//X^k=F'(X)$. If the period of $F(X)$ (and $F'(X)$) is p, then also $F'(X)//X^{p-k}=F(X)$ for the same $k<p$. Also we have always $F(X)//X^p=F(X)$.

The combination of equivalent and complement bitfilters now shows that there can be only a very small number of essentially distinct bitfilters. The complement to $1(X)$ is of course $0(X)$, both of which are bitfilters of even parity to a generator $G(X)$ with even parity. Thus there are 2^n-2 other bitfilters. If a bitfilter has an uneven period than either the number of 1s or the number 0s in a period is uneven. For the complement bitfilter the opposite holds, i.e. the number of 0s or the number 1s is uneven. In both cases however, there can never be a complement of $F(X)$ that is equivalent to $F(X)$, since in that case the number of 0s and 1s must be the same.

Let there be a bitfilter $F(X)$ with even parity to $G(X)$ with (uneven) length $p=2^{n-1}-1$, then $F^c(X)$ cannot be equivalent to $F(X)$, thus we have two essentially distinct bitfilters with period p, or together with the trivial bitfilters $1(X)$ and $0(X)$ we have $2\cdot p+2= 2^n$ bitfilters; these are therefore all

possible bitfilters.

It is possible to find generators $G(X)$ with more than 4 essentially distinct bitfilters; an example is $G(X)=1+X+X^2+X^7$ with the bitfilters:

0, 1, 111000010100110, 100100011110101, 1001, 01, 010000011000100101111100111011, and 100000010000111001010111010010011111101111000110101000101101.

It will soon become clear that generators with small bitfilter periods are not really useful. But the reader may observe that the period of all the bitfilters in the last example divides always 60. It is a general property of all bitfilters (with even parity in relationship to a generator with even number of terms) that their period divides the period of the greatest length of any bitfilter. This restricts the number of bitfilters very much. The proof will be sketched here.

The proof uses the property of a bitfilter sum, i.e. if $F(X)$ and $F'(X)$ are bitfilters with even parity to a generator $G(X)$ with even parity, then $F(X)+F'(X)$ is also a bitfilter with this property. Now, using the bitfilter that is generated by 100000.., one can construct further *basis* bitfilters with the starting bit sequence x10000..., xx10000..., xxx10000..., etc. and show that all other bitfilters can be constructed by the sum of some of these n basis bitfilters. Since these basis bitfilters have a minimum period p, all other have the period p, and if there minimum period is smaller it must be a divider of p. Also follows that the bitfilter with starting bit sequence 100000... is a bitfilter with the longest period p.

5 The CRC method

CRC uses the rest of polynomial division as redundancy information for detecting errors. The following steps are to be performed:

a) Let $N(X)$ be the data polynomial.
b) Let $G(X)$ be a generator of degree n.
c) Let $R(X)$ be the residual after division of $N(X)\cdot X^n$ by $G(X)$. Thus we have $G(X)\cdot Q(X)+R(X)=N(X)\cdot X^n$ or $G(X)\cdot Q(X)=N(X)\cdot X^n+R(X)$. Thus $N(X)\cdot X^n+R(X)$ can be divided by $G(X)$.
d) The sender sends $S(X)=N(X)\cdot X^n+R(X)$.
e) The receiver gets $H(X)=S(X)+E(X)$. $E(X)$ is called error polynomial. It has a term X^k if and only if there is an error (i.e. 0 became 1 or vice versa) at the bit k (k=0,1,2,...).
f) The receiver divides $H(X)/G(X)=(S(X)+E(X))/G(X)$.

$$\frac{H(X)}{G(X)}=\frac{S(X)+E(X)}{G(X)}=\frac{S(X)}{G(X)}+\frac{E(X)}{G(X)}=\frac{N(X)\cdot X^k+R(X)}{G(X)}+\frac{E(X)}{G(X)}=Q(X)+\frac{E(X)}{G(X)}$$

The term $Q(X)$ has no residual, thus any non-zero residual results from a non-zero error $E(X)$.

If any non-zero residual is found it is originated by the error during transmission or storage of the data $N(X)$. Thus in the following we will only consider the error polynomial $E(X)$.

5.1 Types of errors

We start with a simple definition of length and distance. The *distance* of two bits or terms X^k and X^j is defined to be $|k–j|$. Thus X^k and X^k have the distance 0, or X^k and X^{k+1} have the distance 1. The *length* of a sequence of bits with terms $X^k+...+X^j$ is defined to be $|j-k|+1$. Thus the sequence X^k has the length 1 and the sequence $X^k+X^{k+1}+X^{k+2}$ has the length $|k+2–k|+1=3$.

5.1.1 Error bursts

Let us consider an arbitrary generator of length $n+1$: $G(X)=1+...+X^n$. A *burst* of length k is defined to be a sequence of bits with length k. Error bursts occur often in real world systems. If a burst $E(X)=X^r+...+X^{r+n+1}$ of length $n+1$ is given and the pattern of that burst is the same as the pattern of the generator, i.e. $E(X)=G(X)\cdot X^r$, then the residual of $E(X)/G(X)$ is 0, thus the error is not detected. However, in all other cases a burst of length $n+1$ leaves a non-zero residual, as will be proved now.

To divide $E(X)$ by $G(X)$ we add repeatedly $G(X)\cdot X^q$ with some appropriate q's. If $E(X)=X^r+...+X^k+...X^j+...+X^{r+n}$ has length $n+1$, then $E(X)+G(X)\cdot X^r$ has length less than $n+1$, since $X^r+...+a_k\cdot X^k+...+a_j\cdot X^j+...+X^{r+n} + (X^r+...+X^{r+n}) = X^k+...X^j$, where k and j are the lowest and highest indices of terms in $E(X)$, that differ from the corresponding terms in $G(X)$. Obviously, k and j are neither r nor $r+n$, so $|k–j|+1<n+1$. Thus the resulting sequence is a burst of length less than that of $G(X)$ and it is not zero, since we assumed at least one different bit in $E(X)$ and $G(X)\cdot X^r$. Now we add $G(X)\cdot X^r$ to a burst of length less than $n+1$, so that $a_{r+1}\cdot X^{r+1}+...+X^k+...+X^j+...+X^{r+n} + (X^r+...+X^{r+n}) = X^r+...+X^j$, where again $|r–j|+1<n+1$, and again this term cannot vanish, since the lowest term X^r will always be non-zero. We can now proceed in this way and finally come to a burst with highest bit less then X^n which means it cannot be divided by $G(X)$ anymore.

The result of these simple considerations is of course, that all but <u>one</u> bursts not longer than $G(X)$ will be discovered by CRC. This does not depend on the structure of $G(X)$, i.e. whether its parity is odd or even. This encloses single bit errors (as 'burst' of length 1) as well.

We could prove this by use of bitfilters, where however we have to assume even parity to the generator $G(X)$. Since this is less general than the proof above, we leave it as an exercise to the reader.

5.1.2 Two bit errors

The main advantage of bitfilters is given by classifying the condition when two bit errors can be detected. This relies heavily on the period of an even bitfilter.

We assume a generator $G(X)=1+...+X^n$ with degree n and even parity and a period p of the longest bitfilter with even parity to $G(X)$. Let there be two bit errors $E(X)=X^k+X^j$, where $|k–j|$ is not a multiple of p. Then there <u>must</u> be a bitfilter $F(X)$ with even parity to $G(X)$ that filters exactly one of those two erroneous bits (as an example see section 4.3). If in each (or all) phases $F(X)$ filters both bits or none of $E(X)=X^k+X^j$, then $F(X)$ has in a distance $|k–j|$ exactly the same bits, and thus its period would be $|k–j|$, which we had excluded. Thus there exists such a bitfilter with uneven parity

to $E(X)$, which means division leaves an uneven residual, i.e. the residual will be non-zero.

This finishes the proof that any two bit errors can be detected by CRC, besides those in a multiple of the period of the (longest) bitfilter with even parity to the generator. This shows how important it is to use generators with longest bitfilters, which is in many cases simple to achieve. E.g. common generators have degree 16, and the maximum bitfilter length of this generators is $p=2^{15}-1$, which means more than 4000 bytes. In most protocols packets are much shorter (besides FDDI with maximum packet lengths of 4500 bytes; in IP with packet length of 2^{64} bytes there is no error check at all). The well known generators $G(X)=1+X^2+X^{15}+X^{16}$ or $G(X)=1+X+X^{14}+X^{16}$ do this very well. See appendix 7.1 for more examples.

5.1.3 Uneven number of bit errors

We have already mentioned a very simple proof that any odd number of errors will be discovered by CRC, provided that the generator's parity is even. Since $1(X)$ is a bitfilter with even parity to $G(X)$, all uneven errors have uneven parity to $1(X)$, which proves the theorem.

5.1.4 Error correction

It is useful to correct errors, provided one knows the exact position of the erroneous bits. To correct 1-bit errors, one can use CRC as well.

Let us assume there is a 1-bit error. To prove that the receiver can correct this error we only have to prove that the residuals can be distinguished from any other distinct single bit errors. Thus, let us assume that X^k and X^j leave the same residual $R(X)$, i.e.

$$X^k = G(X)\cdot Q_k(X) + R(X),$$

$$X^j = G(X)\cdot Q_j(X) + R(X),$$

Summing these equations we get

$$\begin{aligned} X^k+X^j &= G(X)\cdot Q_k(X) + R_k(X)+G(X)\cdot Q_j(X) + R(X) = \\ &= G(X)\cdot(Q_k(X) + Q_j(X)) + R(X) + R(X) = \\ &= G(X)\cdot(Q_k(X) + Q_j(X)). \end{aligned}$$

This means of course that dividing two bits leaves no residual, which is the case only when their distance is a multiple of the period of all bitfilters. This shows that single bit errors can be corrected uniquely, if the length of the data block is no longer than the maximum period of all bitfilters. This shows again how important it is to use generators with longest possible bitfilters, since this means that longer blocks can be corrected as well.

It should be stated explicitly that three bit errors can generate the same residual as one bit error. Two or any even number of bit errors leaves always residuals with even number of terms, while odd number of bit errors leave always an uneven number of terms as residual (if the parity of $G(X)$ is even). Thus error correction of single bit errors should be used only when the chance for three bit errors is very low. Some applications that use error correction by CRC are bluetooth (FEC

1/3) and ATM, where error correction in the header (HEC: header error correction) is performed in each cell header, while an error in the next header requires new resynchronisation of the cell stream.

5.1.5 Maximum period of bitfilters

Bitfilters have been proved to be useful for checking for errors and correcting errors. However, they must have even parity to a generator and they should be of maximum length. The maximum length for a bitfilter with these properties follows from simple considerations.

The number of residuals with a generator of degree n is 2^n, since there are 2^n combinations of the terms 1, X, ...,X^{n-1}. Half of these, i.e. 2^{n-1}, have odd parity, and since the polynomials 1, X, ...,X^{p-1} all have distinct odd parity residuals, maximum period $p \leq 2^{n-1}$. We have seen that $p = 2^{n-1}–1$ is maximum period of bitfilters with even parity to even generators. There is always one residual with odd number of terms that is not a residual of a single bit error. It can be found by solving the equation $R(X)\cdot X+G(X)=R(X)$, which can be solved uniquely if $G(X)$ has even parity. Then follows

$R(X)\cdot X^k+G(X)\cdot X^{k-1}=R(X)\cdot X^{k-1}$

$R(X)\cdot X^{k-1}+G(X)\cdot X^{k-2}=R(X)\cdot X^{k-2}$

..

$R(X)\cdot X+G(X)=R(X)$

This is the missing residual among those 2^{n-1} odd parity residuals from above.

6 References

W. Kowalk: "CRC-Analyseverfahren mit Bitfiltern".
“http://einstein.informatik.uni-oldenburg.de/papers/Bitfilter.pdf”

7 Appendix

7.1 Some optimal Generators

Some optimal generators of different degree with 4 terms:

$1+X+X^2+X^4$, $1+X+X^3+X^5$, $1+X+X^2+X^6$, $1+X+X^5+X^7$, $1+X+X^6+X^8$, $1+X^2+X^6+X^9$, $1+X^2+X^5+X^{10}$, $1+X+X^3+X^{11}$, $1+X^2+X^7+X^{12}$, $1+X+X^2+X^7+X^{12}+X^{13}$, $1+X+X^2+X^{14}$, $1+X^2+X^5+X^{15}$, $1+X+X^{14}+X^{16}$, $1+X+X^{11}+X^{17}$, $1+X+X^{14}+X^{18}$, $1+X+X^{13}+X^{19}$, $1+X+X^{14}+X^{20}$, $1+X+X^{11}+X^{21}$, $1+X+X^{16}+X^{22}$, $1+X+X^{21}+X^{23}$, $1+X+X^{18}+X^{24}$, $1+X^2+X^{14}+X^{25}$, $1+X+X^{22}+X^{26}$, $1+X+X^{15}+X^{27}$, $1+X^2+X^{15}+X^{28}$, $1+X+X^{17}+X^{29}$, $1+X^2+X^{13}+X^{30}$, $1+X^2+X^{12}+X^{31}$, $1+X+X^{28}+X^{32}$.

For degree 13 no generator with four terms exists; we added one with six terms.

7.2 Programme

A program to compute maximum period was been implemented in Java™.

```
/**
* Computes period of the even bitfilter to a generator.
* The generator is a long number, where its degree is the second parameter.
* The generator highest bit is 1L of the generator field, the lowest bit is at position
* 2^(degreeGenerator) in generator field, or as bit pattern: 1L<<(degreeGenerator).
* @param generator Generator of the bitfilter, as specified above.
* @param degreeGenerator Degree of the generator
* @return Returns the period of the longest bitfilter
*/
static long bitfilterPeriod(long generator, long degreeGenerator) {
  long bitMask = 1L<<degreeGenerator;      // mask the bit to compare for even parity
  long mask = ~((-1L)<<(degreeGenerator)); // mask the result, to detect the period
  long period = 1;                         // counts the length of the bitfilter
  long bitfilter = (1L)<<(degreeGenerator-1); // bitfilter's bit pattern
  long startBitfilter = bitfilter;         // start bitfilter, to compare with
  long limit = (1L)<<(degreeGenerator);    // limit loop, if something is wrong

  for(;period<=limit;) { // cycle for all bits in bitfilter
    long m = bitMask; // copy bit mask to count number of bits in generator and bitfilter
    int cnt = 0;      // counter for bit count
    bitfilter<<=1;    // shift bitfilter one position
    long v = bitfilter&generator; // count only bits at same position in
                                  // generator and bitfilter
    while(m!=0) {         // for all bits, selected by bit mask
      if((m&v)!=0) cnt++; // if bit in generator and bitfilter is set, count it!
      m>>=1;              // next bit mask
    }
    if((cnt&1)==1)bitfilter+=1L; // if count is odd set next bit of bitfilter
    if((mask&bitfilter)==startBitfilter) return period; // if bitfilter is the same
                                                        // as start, cycle found!
    period++;             // analyse next bit of bitfilter!
  }
  System.out.println("undefined end; period = "+period);
  return 0;
}
```