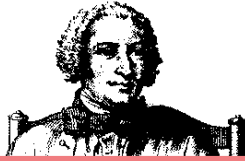


embedded



Voransichts- Version

Für Bezug des Originals
siehe FAQ auf
www.embeddedFORTH.de

Rafael Deliano
Steinbergstr.37
82110 Germering
Tel 089/8418317

V1.0 (Papier) : 11. Nov 99
V2.0 (pdf) : 24. Feb. 01
V2.1 (pdf) : 29. April 04
V2.2 (pdf) : 14. Jan 07

j_r_d@t-online.de

- 1
- 2
- 5
- 8
- 10 PRNG:
Lineare Kongruenz
- 12 Fractionals
- 13 Multiplikation
- 14 Sicherheit von
Prüfsummen

READ . ME

Die vorgesehenen Beiträge über V24 und Schrittmotoren dauern leider etwas länger.

Die neu begonnene Reihe über fehlerkorrigierende Codes macht jedoch gute Fortschritte. Sie wird bald mit zyklischen Codes weitergeführt.

Immerhin ist in diesem Heft die Reihe über Multiplikation abgeschlossen. Shift & Add wurde schonmal in einer alten VD behandelt und wohl bei Gelegenheit hier nochmal nachgedruckt.

Um noch vor Jahresende fertig zu werden mußte der Heftumfang diesmal etwas gekürzt werden. Aber die Themen weniger gründlich und damit zeitraubend zu bearbeiten wäre die schlechtere Alternative gewesen.

Die Listings sind in nanoFORTH geschrieben. Für die Konvertierung in andere FORTH-Varianten im nanoFORTH-Manual GP32 nachlesen.

Einfache fehlerkorrigierende Blockcodes

In Datenübertragungssystemen wird normalerweise anhand einer Prüfsumme das Auftreten eines Fehlers erkannt und der fehlerhafte Datenblock dann erneut vom Sender angefordert. Verwendet man statt der Prüfsumme einen fehlerkorrigierenden Code, kann man die aufgetretenen Fehler im Empfänger korrigieren.

Diese Codes sind natürlich umfangreicher als eine einfache Prüfsumme, weshalb der Datendurchsatz sinkt. Trotzdem haben sich einige Anwendungen herauskristallisiert:

- * Reparatur beschädigter Daten in Speichern. Besonders in Harddisks, aber inzwischen auch in kleinen EEPROMs.
- * Datenübertragung mit fehlendem Rückkanal. Ohne ihn kann man den Sender nicht zur erneuten Übertragung auffordern.
- * Zeitkritische Anwendungen bei denen erneute Übertragung zu lange dauern würde. Z.B. Speicher und Datenbusse in Minicomputern.
- * Wenn der Kanal sehr stark durch Einzelbitfehler gestört ist. Z.B. sehr weit entfernte Satelliten im Welt- raum. Oder Funksysteme wie Packet Radio die mit geringer Sendeenergie betrieben werden.

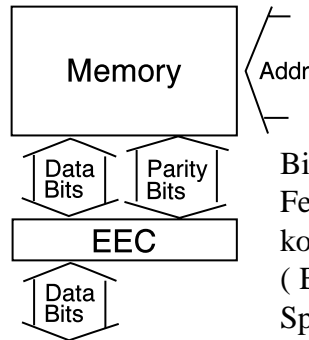
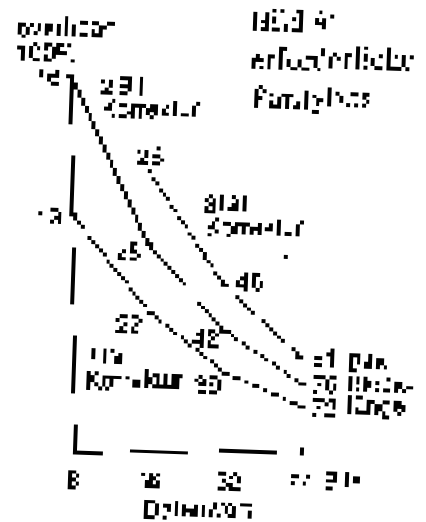


Bild 3: Fehlerkorrektur (EEC) in Speicher

LRC, „longitudinal redundancy check“ stammen noch aus der frühen Magnetbandtechnik, als man Daten mit Mehrspurköpfen (9 Spuren) in Blöcken aufzeichnete. Üblich waren Blöcke mit 8x8 Datenbit und einem Prüfbit pro Zeile/Spalte (Bild 2) Kreuzparität ist simpel in der Verarbeitung braucht aber mehr Prüfbits als nötig.

Hamming-Code

Der effizienteste Blockcode für die Korrektur von 1 Bit Fehlern ist der Hamming Code [1]. Eine wichtige Anwendung wo Korrektur von Einzelbitfehlern bereits genügt ist die Sicherung von DRAM-Speichern (Bild 3) und Bussystemen. Dabei wird die Fehlerkorrektur von digitaler Hardware durchgeführt die man heute oft als ASICs ausführt. Bild 4 zeigt an, um wieviele Prüfbits man das Datenwort erweitern muß um 1-, 2- oder 3-



Die Fehlerkorrektur in einem Speicher ist für größere Blocklängen nicht weniger praktikabel, weil man die Prüfbits nicht ausreicht, um die Fehler zu korrigieren, weil mehrere Fehler die die Daten nicht mehr korrigieren können.

Spannung (7,4)

Die Spannung (7,4) ist ein Blockcode, der die Korrektur von 1 Bit Fehlern ermöglicht. Die Spannung (7,4) ist ein Blockcode, der die Korrektur von 1 Bit Fehlern ermöglicht. Die Spannung (7,4) ist ein Blockcode, der die Korrektur von 1 Bit Fehlern ermöglicht.

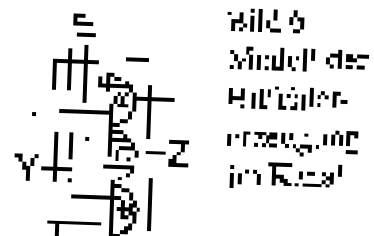


Bild 5: Modell der Fehlerkorrektur im Kanal

Kreuzparität

Dieser simpelste fehlerkorrigierende Codes wird aus Paritybits gebildet (Bild 1). Bei einem Einzelbitfehler ist ein Prüfbit in LRC und VRC fehlerhaft, wodurch im Schnittpunkt die Position des Einzelbitfehlers sichtbar wird. Als Korrektur invertiert man dann dieses Bit im Datenfeld. Die Begriffe VRC, „vertical redundancy check“ und

Bild 1:

Kreuzparität

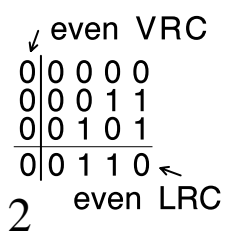


Bild 2:

Magnetband

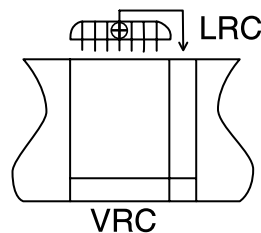
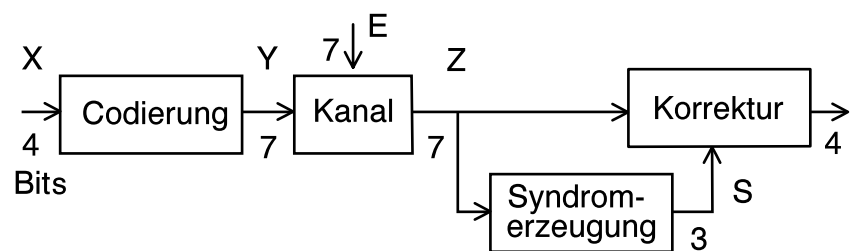


Bild 5: Hamming (7,4) in Datenübertragung



XMODEM

Einplatinencomputer werden während der Entwicklung normalerweise über V24 von PCs aus gesteuert. Im PC wird meist ein handelsübliches Terminalprogramm verwendet das entweder TTY oder VT52 emuliert. Damit können nur ASCII-Zeichen übertragen werden, Datenpakete werden nicht unterstützt und es findet keine Fehlersicherung statt. Will man Binärdaten als Paket übertragen, muß man ein Filetransferprogramm verwenden. Das älteste und verbreitetste ist XMODEM. Es wird von den meisten PC-Terminalprogrammen unterstützt. Zudem ist es einfach zu implementieren und braucht im Einplatinencomputer wenig Programm- und Datenspeicher.

Der Vorläufer von XMODEM hieß schlicht MODEM und stammt von Ward Cristensen. Keith Petersen verbesserte das Programm und taufte es XMODEM. 1977 wurde es in die public domain übergeben und verbreitete sich rasch. Es war eigentlich nur dazu gedacht, die Datenübertragung zwischen CP/M-Rechnern mit 300 Baud Modems zu ermöglichen. Daraus ergibt sich auch die Länge des XMODEM-Datenfelds von 128 Byte. Sie entspricht direkt dem Datenblock auf einer CP/M-Diskette. Im praktischen Betrieb ergaben sich mit XMODEM bald Probleme.

Sobald die Modems schneller wurden, begann die kurze Länge des Datenfelds den Durchsatz zu begrenzen. Die schwache Fehlersicherung durch eine Prüfsumme erlaubte auf Telefonleitungen keinen sicheren Betrieb, da schnellere Modems mehr Bitfehler produzieren. Bald entstanden deshalb Varianten von XMODEM. In XMODEM-1K wurde die Länge des Datenfelds auf 1024 Bytes erhöht. In XMODEM/CRC wird das Prüfsummebyte durch eine 16 Bit CRC ersetzt. Chuck Forsberg faßte die Vorteile dieser und andere XMODEM-Erweiterungen in das neue Protokoll YMODEM zusammen. Leider löste das die Probleme nicht, da von YMODEM viele inkompatible Implementierungen entstanden.

Wirklich befriedigend war aber erst Forsbergs ZMODEM. Für Datenübertragung zwischen PCs über Modem ist XMODEM heute überholt und ZMODEM besser geeignet. Für Anwendung auf Einplatinencomputern ist ZMODEM jedoch zu aufwendig. Speziell für die Übertragung von Binärfiles genügt hier XMODEM.

Protokoll

Obwohl nur in eine Richtung Daten übertragen werden, wird eine Voll duplexverbindung benötigt, damit im Rückkanal Handshakesignale empfangen werden können. Diese Quittungssignale sind ASCII-Steuerzeichen die in Tabelle 1 aufgeführt sind. Alle Zeichen werden mit 8 Bit übertragen. Der Empfänger startet die Datenübertragung indem er ein NAK-Byte sendet (Bild 1). Er zeigt damit an, daß er bereit ist, und wiederholt das NAK bis zu 10 mal. Als Pause zwischen diesen Wiederholungen ist in XMODEM ursprünglich 10sec vorgesehen. Bei hohen Baudraten ist jedoch 3sec sinnvoller.

Hat der Sender das NAK erkannt, antwortet er mit dem ersten Datenpaket. Der Empfänger quittiert jedes erfolgreich empfangene Paket mit einem ACK-Byte. Bei einem Fehler sendet er ein NAK. Danach muß der Sender dieses Paket

Tabelle 1. Verwendete Steuerzeichen mit Code in Hex

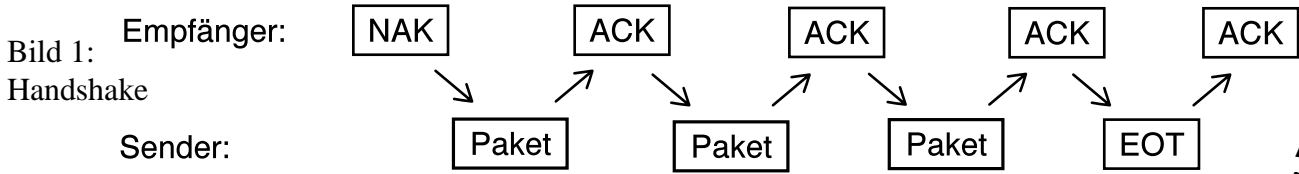
ACK „Acknowledge“	06
NAK „Negative Acknowledge“	15
SOH „Start of Header“	01
STX „Start of Text“	02
EOF „End of Transmission“	14
CAN „Cancel“	18
??	1A
C	43

weiter senden. Ein NAK-Charakter (NAK) bedeutet dieses Paket muß nicht mehr übertragen. NAK's die meist begrenzt werden werden, werden durch 10 mal wiederholt. Dann muß der Empfänger die Pakete des ACK-Wortes die Straße erfolgreich übertragen wurden. Sobald der Sender vom empfangenen Datenpaket ein EOF-Byte das zum Ende des Pakets ACK gehen werden muß. Wenn wiederholt zu Ende dieses EOF Zeichen bis zu 10 mal

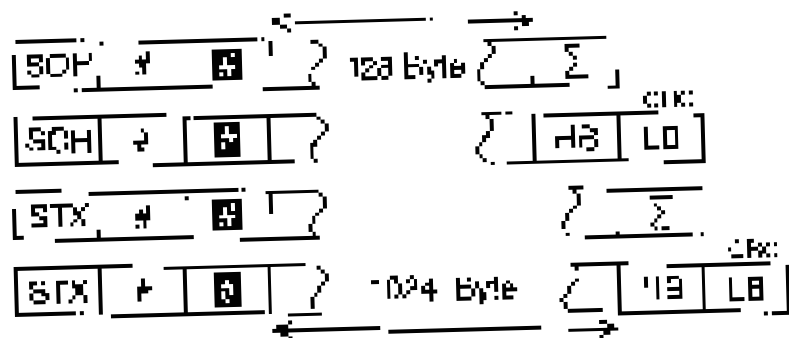
zu empfangen. XMODEM nicht vorhanden, heute aber mehr notwendig. Es ist CAN-Byte mit dem sowohl Sender als auch Empfänger einen Zeichen des Modems ganz verhindern. Es ist immer notwendig sollte es das Signal nicht mehr den Empfang eines vollständigen Datenpakets sender. Wenn es ein Fehler kann es der Empfänger nicht zu den beiden Daten unterstützen. Wenn es nicht überträgt und es nicht übertragen werden kann.

Paket

Das Protokoll hat für den geringsten XMODEM hat eine maximale Länge von 128 Byte (Bild 1). Als Standard für ein datenpaket. XMODEM hat das Datenpaket die Länge von 128 Byte. Lange Pakete werden durch das ZIP-System gekennzeichnet. Jedes Paket hat eine Prüfsumme, die im letzten Byte des Pakets immer überträgt wird. Die Länge



- RTS: XMODEM
- XMODEM + CRC
- XMODEM-1k
- XMODEM-1k mit CRC



Basieren die Normen 01. Diese Nummer wird mit jedem weiteren Paket inrechnerisch unabhängig davon ob da ist oder nicht übertragen werden. Nachdem FF erreicht ist erfolgt Überlauf auf 00. Das im nächsten Paket wird mit 00 weitergeleitet. Das zweite Byte entspricht dem ersten Byte ist aber Always inverted. Danach hat die Besondere in ihre eigene Fehlersicherung. Der Empfänger akzeptiert maximal fünfmal hintereinander Pakete Nummer, wie sie bei erfolgreicher Übertragung auftrifft. In Abhängigkeit vom Paket die Ladung in die nächste Struktur kann sein, wie die Weiterleitung einer Übertragung der Fall ist. In beiden Fällen werden die Pakete mit der Flagnummer zu lösen werden und Empfänger die Symbolisierung werden. Dieser Fehler kann nicht korrigiert werden. Der Empfänger akzeptiert die Verküpfung der Erweiterungen in einer 1000 bis 1024 Bytes.

Das zweite Byte in XMODEM immer eine feste Länge. Die Daten werden direkt in die Ladung geschickt. Sind die Pakete nicht mehr als 128 Bytes Daten vorhanden, wird die Länge des Daten Puffers aufgeführt. Wenn es sich um die Steuerwerte (z.B. die CRC) des Pakets handelt, dann wird die Länge des Pakets angegeben. Wenn die Pakete nicht mehr als 1024 Bytes Daten vorhanden sind, werden immer die bekannte CRC-Kombination in die Ladung übertragen. Jedes neue weitere Paket hat die 00-Zeichen als Daten enthält. Bei XMODEM-1k kann man diese Erweiterungen packen mit dem Empfänger abholen. Bei Paketen kann man jedes 1024 Bytes Pakete, was das gesamte Paket überträgt nach der Übertragung. Jedes ist in die Ladung des Pakets, was das 1024 Bytes.

Fehlersicherung

Zur Bildung der Prüfsumme werden in XMODEM alle vorangehenden Bytes des Pakets mit dem Empfänger. Die anderen 5 Bit der Summe werden als Modulo übertragen. Invertiert die Flagnummer wird der Empfänger 100 auf jedes Byte wird dieser Übertrag übertragen. Wenn ein NAK wird ein Fehler entdeckt, während die Übertragung wird, wird die Ladung des Pakets. Kompletter Sender ist Z.B. kann er danach werden, dass das Flag keine Summe. Wenn empfangen wurden, hat dann Sender die NAK. Das Paket wird die Übertragung wiederholen, aber das Verhalten des Empfängers.

XMODEM-CRC

Diese Option wird vom Empfänger gewählt. Es werden die die Ladung von der NAK der SOH. Byte 0. Die Pakete werden jeweils dann wieder konventionell mit NAK und NAKs quittiert. Pakete werden die CRC Daten jedoch nicht implementiert, sondern nur die Verküpfung schreiben. Die Information wird hier nur über die Flagnummern, die nicht über den Kanal geht. Die 16-Bit CRC verwendet das Generationsprogramm CRC-16/IT. Die Daten werden mit MSB zuerst verarbeitet. Wenn Empfänger wird es, das dann das zweite Byte übertragen. Das Paket wird dann mit 1024 Bytes Länge.

XMODEM-1k

Diese Option wird vom Empfänger gewählt. Hier der Empfänger die Verküpfung kann auch hier

wieder keine Übertragung zwischen. STX immer bei SOH als Steuerwert des Pakets. Die Daten die eine längere Pakete in der 1024 Bytes Länge und eines Pakets dieses Paket geschickt werden. Ein File von 1024 Bytes Länge zerlegt der Sender dementsprechend in ein langes und 1 kurze Pakete.

Man kann Programmierer akzeptieren XMODEM-1k nur CRC als Fehlersicherung. In Kauf genommen der Verküpfungsmethode wird XMODEM-1k nicht explizit aufgeführt. Einzelnen können Sie sich vor Übertragung XMODEM nach lange Pakete empfangen, die ein Zustand der Steuerung dieser Operation automatisch erkennen können.

Implementierung

Das Programm im Listing XMODEM ermöglicht Sender und Empfänger von XMODEM XMODEM-CRC mit XMODEM-1k. Es ist für die Verwendung mit einem Emulatorcomputer gedacht. Empfänger erfolgt durch die Befehle FF, 00, 00-FF. Das Paket wird auf dem Stack die Adresse und Länge der zu übertragenden Daten übertragen. FF 00 auf dem Stack ein Flag, welches die Anzahl, ob die Übertragung erfolgreich war. Sender erfolgt durch FF, 00, 00-FF. Hier die Adresse und Länge des Pakets puffer übertragen. Zugriff auf die HART erfolgt durch den Befehl 0020 der Base-Register und dem Befehl 0000-FF00. In dem Empfänger wird die Länge

Das zweite Byte im Paket, was ein 1024 Bytes Programm, welches ermöglicht 1024 Bytes in die CRC Tabelle. Verändert kann auf die CRC oder benötigt, nur im Sender

oder nur einfache. Ist sich der Sprecher bei der Aussprache nicht sicher, so kann er auf einem 2,5MHz-Kanalbereich 2000 Bytes versenden, so bis zu 9000 Bytes.

Resümee

Die Liste von XMODEM-Modellen ist lang. Folgende Punkte haben sich als die wichtigsten herausgestellt. Die meisten Modelle sind Erweiterungen von XMODEM um eine Grundfunktion, nämlich keine Verbindung zwischen dem Host und dem Modem zu stellen, sondern nur die Übertragung zu steuern. Deshalb sollte man die Bedienungsanleitung für Erweiterungen aufmerksam selbst wenn man sie nicht unbedingt benötigt. Für Kinder die sich mit dem Computer beschäftigen und keine Protokolle, die nicht mit (1) verwendet werden. Bei Erweiterungen sollte das Protokoll jedoch sehr einfachen Flusskontrolle durch XMODEM zu sein möglich. Wenn man sich auf Hardware nicht auskennt, ist es vielleicht, mit der XMODEM Software zu dem

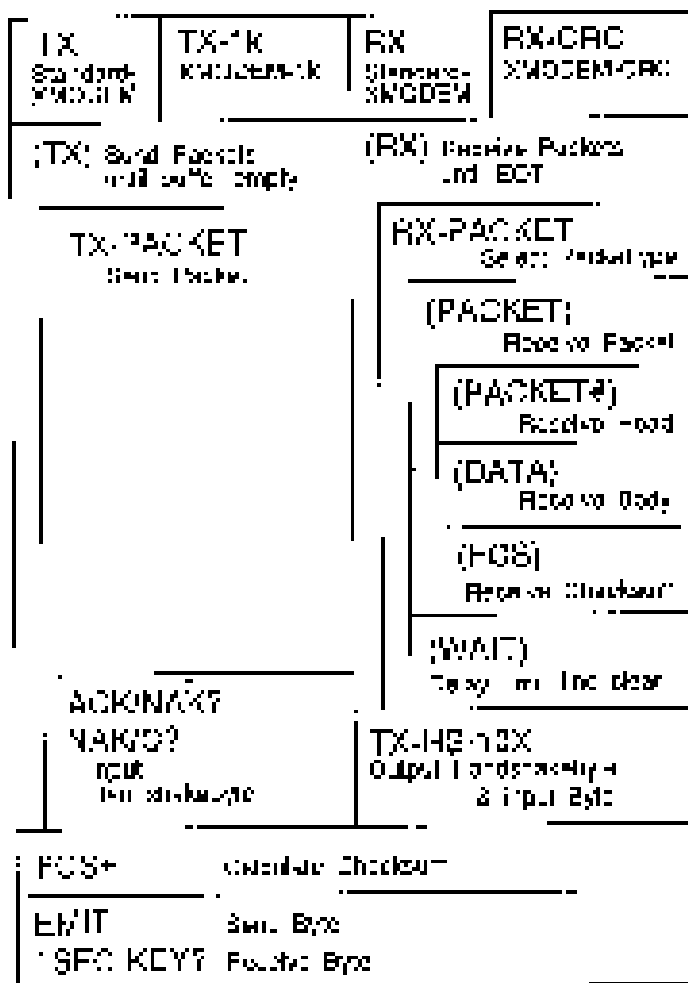
Beginn anzuprobieren. Aber es ist nicht so, dass sich Erweiterungen nicht als Lückenfüller verwenden. Die meisten Erweiterungen sind sehr einfach zu realisieren. Die meisten Erweiterungen sind sehr einfach zu realisieren. Die meisten Erweiterungen sind sehr einfach zu realisieren.

Wird Charakteristik und zu den praktischen Funktionen von XMODEM angegeben, sind folgende Punkte zu beachten. Die meisten Erweiterungen sind sehr einfach zu realisieren. Die meisten Erweiterungen sind sehr einfach zu realisieren.

- (1) Charles Leebing: XMODEM/2000 mit Protocol Reference 1988
- (2) Frank de Groot, KERMIT & File Transfer Protocol Digital Press 1987

Eine weitere XMODEM-Implementierung von AMODEM: Robert Taylor, SEND and RCV, D.J.D.U. 1984

Bild 1: Struktur des Programms



Quadratwurzel

Diese Berechnung wird häufig benötigt und ist relativ einfach. Sie hat sich damit zum Einführungsbeispiel der numerischen Mathematik entwickelt. Es gibt allerdings mehrere unterschiedliche Verfahren. Einige eignen sich mehr für Integerzahlen, andere für Float. Man sollte sie alle kennen um im jeweiligen Fall die optimale Abschätzung von Rechenzeit gegen Speicherverbrauch treffen zu können.

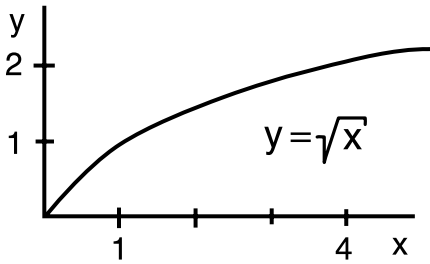


Bild 1: Quadratwurzel

Da die Funktion nur für positive Zahlen sinnvoll ist, erfolgt in vielen Implementierungen für Zahlen kleiner Null eine Fehlermeldung. In ANS-Forth wird nur vermerkt, daß x kleiner Null ein undefinierter Zustand ist.

Reihensummierung

Durch eine simple Reihenentwicklung [1] (Bild 2) kann man relativ einfach alle Quadratwerte erzeugen. Durch Vergleich mit dem Sollwert x prüft man ab, wann man größer wird als dieser (Listing SQRT1). Das Verfahren braucht keine Multiplikation, aber dauert offensichtlich um so länger je größer die Werte sind. Dafür benötigt es sehr wenig Programmspeicher. Man kann die Geschwindigkeit durch verbesserte Startwerte aus einer kurzen Tabelle verbessern, die man z.B. mit den obersten Bits des Eingangswerts adressiert. Das Resultat ist immer abwärtsgerundet. 99 ergibt hier 9. Das echte Ergebnis wäre 9,9.

Bild 2: Reihenentwicklung

$$\sum_{i=0}^n (2i+1) = (n+1)^2 = m$$

i =	0	1	2	3	4	
m =	1	4	9	16	25	
		1 ²	2 ²	3 ²	4 ²	5 ²

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

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

Bild 3: Vereinfachung von a²

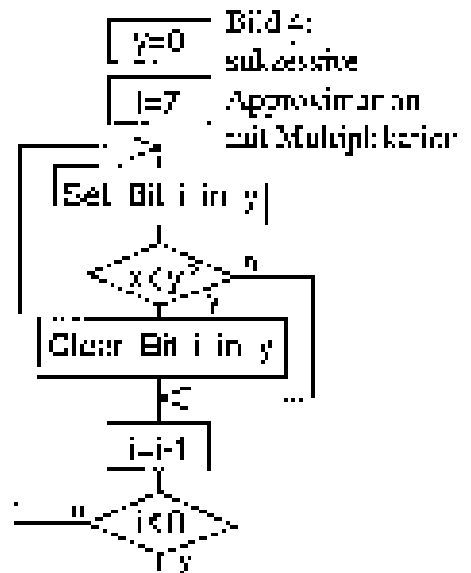
Sukzessive Approximation

Hierbei wird versuchsweise das höchstwertige Bit in einem Datenwort gesetzt und das Quadrat dieser Zahl gebildet. Ist das Ergebnis größer als der Sollwert x wird das Bit wieder gelöscht. Andernfalls wird es beibehalten und man fährt mit dem nächst niedrigeren Bit fort (Listing SQRT2, Bild 4). Das Suchverfahren erfordert in seiner Grundform schnelle Multiplikation. Wenn y nur 8 Bit Auflösung hat und der Controller über einen 8x8=16 Bit Multiplikationsbefehl verfügt, eignet sich die Grundform bereits.

Ohne Multiplikation

Das Verfahren kann durch eine Optimierung der Berechnung des Quadrats auf Controllern deutlich beschleunigt werden [2]. Dazu wird der Ausgangswert als Summe aus dem ursprünglichen Wert a sowie dem durch das gesetzte Bit neuen Zahl b angenommen (Bild 3). Die Operation 2*a ist natürlich nur ein Shift. Die Multiplikation mit b mit der Summe ist durch Shifts möglich, die Zahl b enthält ja nur Nullen und oben ein gesetztes Bit. Der Wert für a² existiert bereits vom letzten Durchlauf (Listing SQRT3, Bild 5).

Bedingt durch die nötige Speicherung von Zwischenwerten ist jedoch Codierung in FORTH umständlich. Das Rundungsverhalten



entspricht dem der Reihenentwicklung. Sukzessive Approximation macht für Integerzahlen gute Gebrauch. Für hochwertige Anwendungen wie Fließ- und hochauflösende Analogrechner ist es ungeeignet.

Newton-Iteration

Es wird hierja auch mit Newton-Iteration gearbeitet. Diese ist eine Verallgemeinerung des von Newton 1690 entwickelten Verfahrens zum Finden von Nullstellen. Die Anwendung ist nicht auf

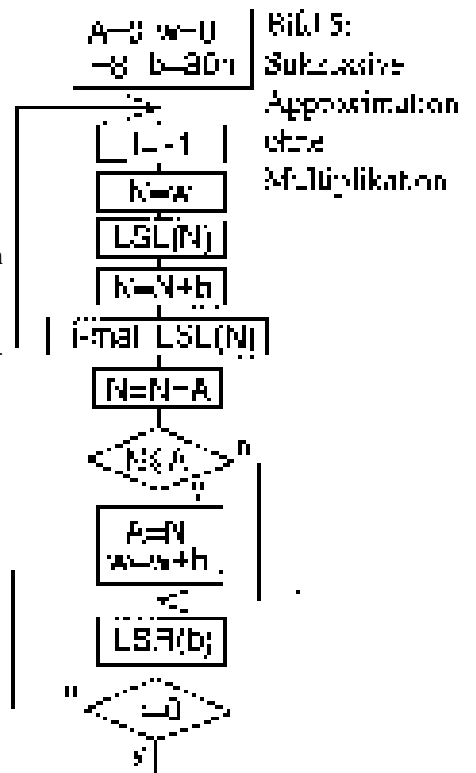


Bild 8:
Newton-Iteration

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

Bild 9:
SQRT auf
Newton
angewandt

$$f = x^2 - y$$

$$f' = 2x$$

$$x_{n+1} = x_n - \frac{x_n^2 - y}{2x_n}$$

$$x_{n+1} = \frac{1}{2} \left(x_n + \frac{y}{x_n} \right)$$

Die Bestimmung der Quadratwurzel
erschließt sich durch die Anwendung
des Newton-Verfahrens. Die Ableitung
der Funktion $f(x) = x^2 - y$ ist $f'(x) = 2x$.
Die Ableitung $f'(x)$ ist die Ableitung
von $f(x)$. Die Ableitung $f'(x)$ ist die
Ableitung von $f(x)$. Die Ableitung
von $f(x)$ ist die Ableitung von $f(x)$.
Die Ableitung von $f(x)$ ist die
Ableitung von $f(x)$. Die Ableitung
von $f(x)$ ist die Ableitung von $f(x)$.

Bild 10 zeigt die Anwendung
des Newton-Verfahrens. Das Problem wird
so umgeformt, dass es als Suche nach
einer Nullstelle erscheint. Die neue
Funktion $f(x) = x^2 - y$ ist die
Nullstelle $x = \sqrt{y}$. Die Ableitung
von $f(x)$ ist die Ableitung von $f(x)$.
Die Ableitung von $f(x)$ ist die
Ableitung von $f(x)$. Die Ableitung
von $f(x)$ ist die Ableitung von $f(x)$.

LSQRT(x)

Diese Funktion wird in der
Newton-Iteration in Bild 8 dargestellt.
Die Multiplikatoren 0,5*y wird dabei
zur um die Anfangswert durchgehend
Man beachte, dass keine Division
benötigt wird. Die Berechnung ergibt
sich aus Bild 9 und kann durch die
entsprechende Substitution

Bild 8:
Newton für LSQRT $x = \frac{1}{\sqrt{y}}$

$$x_n = x_n \left(1,5 - 0,5 * y * x_n^2 \right)$$

$$= 0,5 x_n \left(2 - y * x_n^2 \right)$$

$\frac{1}{\sqrt{y}} = \frac{1}{\sqrt{y}} * y$ Bild 9
LSQRT mit SQRT

SQRT(x) beschreiben. Es gibt deshalb
keine die diesen Zusammenhang
rückwärts lösen. Um die Berechnung
für die Newton-Iteration zu
machen sich, man bestimme die
Ableitung von $f(x)$. Die Ableitung
von $f(x)$ ist die Ableitung von $f(x)$.
Die Ableitung von $f(x)$ ist die
Ableitung von $f(x)$. Die Ableitung
von $f(x)$ ist die Ableitung von $f(x)$.

Ein effizienter Algorithmus
zur Berechnung der Quadratwurzel
ist das Newton-Verfahren. Die
Ableitung der Funktion $f(x) = x^2 - y$
ist $f'(x) = 2x$. Die Ableitung
von $f(x)$ ist die Ableitung von $f(x)$.
Die Ableitung von $f(x)$ ist die
Ableitung von $f(x)$. Die Ableitung
von $f(x)$ ist die Ableitung von $f(x)$.

Startwert

Die Wahl des Startwertes kann zu
einer Beschleunigung führen. Durch
eine gute Wahl des Startwertes
kann man sich über einen
höheren Ausgangswert für die
Iterationen hinweg setzen und die
Anzahl der Iterationen reduzieren.
Bild 10 zeigt die Anwendung
des Newton-Verfahrens. Das Problem
wird so umgeformt, dass es als
Suche nach einer Nullstelle
erscheint. Die neue Funktion
 $f(x) = x^2 - y$ ist die Nullstelle
 $x = \sqrt{y}$. Die Ableitung von
 $f(x)$ ist die Ableitung von $f(x)$.
Die Ableitung von $f(x)$ ist die
Ableitung von $f(x)$. Die Ableitung
von $f(x)$ ist die Ableitung von $f(x)$.

Sechserformel

Die in Bild 10 gezeigte Formel
kann verwendet werden, um hochauflösende
Wurzeln \sqrt{x} zu berechnen, wenn
die hier in der häufigsten Funktion
berechnet werden sind.

Es wurden keine nicht alle
Verfahren dargestellt. Einige sind
etwas anders, und sie nicht sehr
effizient. Z.B. ist Interpolation
in Tabellen möglich. Wenn man

Bild 10: Startwert für Skalarlog
 $y_0 = 0,115442 + 1,15442 * x$

Bild 11: Startwert für Skalarlog
 $y_0 = \frac{x}{2} + 0,48492$

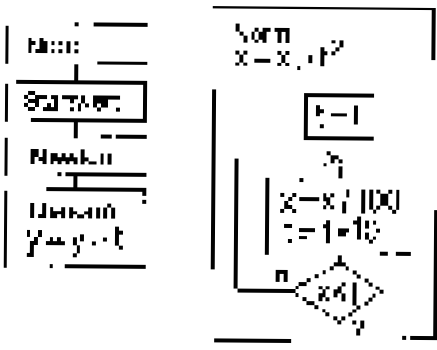


Bild 12:
Flussdiagramm
Newton

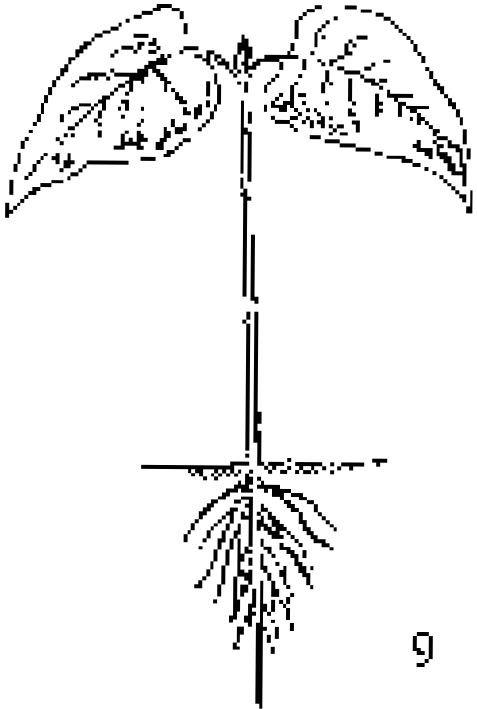
Bild 13:
Neuformel
Newton

ein geschätzter Wert herangezogen
werden. Ein Wert kann das bei
bestimmten Anwendungen wirksam
sein. Fehler sind gering. Vorteil, es
ist mit einem verbunden. Fehler werden
minimiert und Exponenten gehen
bestimmt. Die allgemeine
Einstellung hier sehr erschweren bei

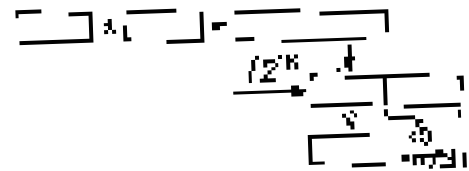
- [1] James H. Wilkinson, "The Art of
Computing", Prentice-Hall, 1971
- [2] John G. Demme, "Schnelle
Algorithmen für die Berechnung
von Wurzeln", Springer, 1991
- [3] Dennis Gillies, "Quick and Dirty
Function for the IBM PC", Dr. Dobbs
9/1978
- [4] "The Art of Computer Programming",
Vol. 2, Addison-Wesley, 1975

Bild 13:
SQRT
aus log
und 2^n

$$\sqrt{x} = 2^{(0,5 * \log_2(x))}$$



Linear kongruente Zufallszahlengeneratoren



Wegen des einfachen Aufbaus und der guten Implementierbarkeit in Software werden sie bevorzugt für Simulationen verwendet.

Bild 2 Berechnung für $m = 2^{16}$.

Bild 1: Formel

$$x_n = (a x_{n-1} + b) \bmod m$$

Das Verfahren wurde 1949 von D.H. Lehmer erfunden. Es handelt sich um die simple, rekursive Formel die in Bild 1 dargestellt ist. Dabei wird die letzte erzeugte Zufallszahl X_{n-1} mit der Konstanten a multipliziert, dann wird die Konstante b addiert und die Summe durch m dividiert. Als Ergebnis wird der ganzzahlige Rest der Division verwendet.

ist die Verteilung die man z.B. mit Würfeln erhält. Für die Simulation vieler natürlicher Prozesse sind jedoch andere Verteilungen nötig. Die bekannteste dürfte die gaussche Glockenkurve sein. Die Umwandlung auf andere Verteilung, muß durch Zusatzprogramme vorgenommen werden.

Der erste Startwert X_{n-1} , d.h. X_0 , wird im Programm als Konstante vorgegeben. Ein einfaches praktisches Beispiel für 16 Bit Wortbreite findet sich im Listing. Nachdem man den Startwert geladen hat, kann man mit dem Befehl RANDOM laufend Zufallszahlen entnehmen. Wie Bild 2 zeigt, entfällt für $m = 2^{16}$ die explizite Division, da man die oberen 16 Bit des Resultats verwerfen kann und damit der Rest der Division sofort entsteht. Aus diesem Grund sind Werte wie 2^{16} und 2^{32} für aufwandsarme Implementierung besonders populär.

Kochrezepte

Meist greift man aus Bequemlichkeit auf in der Literatur empfohlene Werte zurück von denen die Tabellen 1 bis 3 eine Auswahl bietet. Das Verfahren RANDU wird nicht empfohlen, aber man findet es trotzdem manchmal in der Literatur. 8 Bit Wortbreit kann prinzipiell auch keine guten Ergebnisse liefern. 16 Bit wird aufgrund der Gegebenheiten auf Controllern oft verwendet, auch wenn 32 Bit und mehr die besseren Ergebnisse liefern würden.

Die Popularität des Wertes 2^{31} statt 2^{32} erklärt sich wohl aus Anwendungen in den man ein Zeichenbit freihalten will. 2^{31} kann man offensichtlich durch Shifts realisieren.

In $2^{31}-1$ sind die unteren Bits zufälliger verteilt und es gibt anscheinend ein schnelles Rechenverfahren.

Man kann die Parameter auch selbst festlegen, es gibt in der Literatur reichlich Anleitungen. Das meistzitierte Rezeptbuch ist [1]. Wer selbst kochen muß allerdings auch testen und das ist mühsam. Wenn man allerdings auf die Qualität bestimmter Eigenschaften wie Zykluslänge oder Verteilung besonderen Wert legt, muß man ohnehin prüfen.

Abspecken

Die Konstante b kann zu anderen Werten. Der genaue Wert gilt aber als weniger kritisch und oft wird der alle 1 mit a verwendet. Im ersten Fall kann man sich den Restwert sparen, um auch im Fall $m = 2^k$ zu arbeiten.

Aufgaben

Wie man Qualität zu verbessern kann man in der zweiten Kapitel lesen und es dürfte zu erwarten. Werte die Zahlen für m meist zu klein sind, da dann nur die ersten p die Stelle realisiert werden können. Tabelle 3 zeigt warum das gute Werte Sie ergeben, meistens die Länge und gewisse Verteilung. Das Overlow bezieht sich auf die Produktlänge.

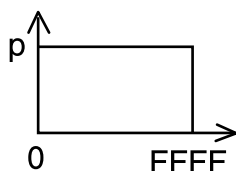
Resümee

In vielen Anwendungen braucht man sich nicht zu befürchten. Wer sich mit einem einfachen Verfahren befreit, der möge sich von Zufallszahlengeneratoren nicht verwirren lassen. Als einfaches Verfahren für geringe Platzanforderungen ist die Qualität der Zahlen relativ gut. Wer mehr Details will, sollte die Datenverteilung verwenden, weil das immer besser ist und es zu beschleunigen. Nicht Ihre absolute Qualität ist gut, aber die Verteilung von Auswertungen ist Qualität.

Eigenschaften

Der Generator erzeugt zwar laufend neue Zahlen, es ergibt sich jedoch ein Muster das zyklisch wiederholt wird. Das ist einer der wesentlichen Unterschiede zu einem echten Zufallszahlengenerator. Durch die Wortbreite m ist auch die Länge dieser Periode festgelegt. Sie kann im günstigsten Fall nicht länger als $m-1$ sein, ist aber manchmal bedeutend kürzer. Ein guter Generator erzeugt gleichverteilte Zahlen (Bild 3). Das

Bild 3: Gleichverteilung



[1] Knuth, "The Art of Computer Programming, Vol. 2: Seminumerical Algorithms", Addison-Wesley, 1997.
 [2] Steve Flannery, "Reckless Mathematics: Numerical Recipes in C: The Art of Scientific Computing", Cambridge University Press, 1994.

Ausblick

Der Mehrzweck IC 74181 und 74182 sind wesentlich höher als fast alle anderen Aus-Rechen-Elemente. Rechner mit konstante Architektur in unterschiedlichen Aufbauelementen sind weniger flexibel als Rechner mit unterschiedlichen Logik-Elementen auf dem Markt. Infolgedessen ist die dynamische Komplexität bei 16 Bit Wortlänge etwas sehr knapp. Man hat immer gegen "fliehe lauf große" sowie gegen Verschieben der Zahl nach links anknüpfen. Letztendlich sind die Anordnungen bei 24 Bit Wortlänge mit dynamischen Elementen.

(1) Wellenform, Vektor
"Practical Arithmetic"
Leserbrief 22/1983



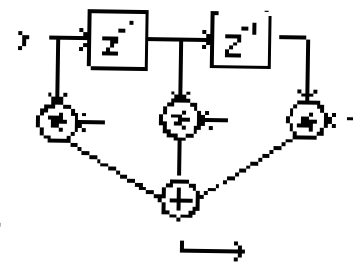
Fortsetzung von Seite 14:

Es gehen zu großen Stück auf ist für die erhoheit. CRC besser als die Fußsamme. 8 bis 128 bis gehen bis nicht besonders tiefen. Die allgemeine ist, daß die Programme unterschieden durch die Qualität der Software haben. Das ist für 8 Bit CRC's mit 16 bis 128 bis Polynom. Man könnte man mit dieser kleinen Stück die erhoheit. Die allgemeine ist, daß die Programme unterschieden durch die Qualität der Software haben. Das ist für 8 Bit CRC's mit 16 bis 128 bis Polynom. Man könnte man mit dieser kleinen Stück die erhoheit.

Multiplikation

Schnelle

Koeffizientenmultiplizierer



Wenn man nur geringe Dynamik, aber hohe Geschwindigkeit braucht und freien Programmierplatz hat, kann die Realisierung des Multiplizierers als Tabelle sinnvoll sein. Für 4096 Bit mit 8 Bit Resultat benötigt man eine 256 x 256 Tabelle. Der Aufwand reicht meist auf 200 k Operationen.

Sei ganzlogisch ergibt sich die allgemeine Lösung wegen der hohen

Man kann den Tabellenvergleich auf 128 oder 64 Bits verringern, wenn man weitere Verluste in der Dynamik in Kauf nimmt.

Eine spezielle Anwendung war BR 1983

MAX. 1.28 von Akku auf X
Tabelle 1.28 von Akku auf X

$$\begin{array}{r}
 \boxed{X_{1-8}} \boxed{X_{9-16}} * \boxed{Y_{1-8}} \boxed{Y_{9-16}} \\
 \boxed{X_{1-8}} * \boxed{Y_{1-8}} = \boxed{\quad} \boxed{\quad} \\
 \boxed{X_{9-16}} * \boxed{Y_{1-8}} = \boxed{\quad} \boxed{\quad} \\
 \boxed{X_{1-8}} * \boxed{Y_{9-16}} = \boxed{\quad} \boxed{\quad} \\
 \boxed{X_{9-16}} * \boxed{Y_{9-16}} = \boxed{\quad} \boxed{\quad} \\
 \hline
 \boxed{\quad} \boxed{\quad} \boxed{\quad} \boxed{\quad}
 \end{array}$$

Zusammenfassung

VMI

Teilmultiplikationen

Viele Computer haben zwei oder drei Multiplizierer. Die allgemeine ist, daß die Programme unterschieden durch die Qualität der Software haben. Das ist für 8 Bit CRC's mit 16 bis 128 bis Polynom. Man könnte man mit dieser kleinen Stück die erhoheit.

Die allgemeine ist, daß die Programme unterschieden durch die Qualität der Software haben. Das ist für 8 Bit CRC's mit 16 bis 128 bis Polynom. Man könnte man mit dieser kleinen Stück die erhoheit.

Bei 1,54 MHz ist die CRC steigt die Geschwindigkeit gegenüber der allgemeinen Software. Die allgemeine ist, daß die Programme unterschieden durch die Qualität der Software haben. Das ist für 8 Bit CRC's mit 16 bis 128 bis Polynom. Man könnte man mit dieser kleinen Stück die erhoheit.

Die allgemeine ist, daß die Programme unterschieden durch die Qualität der Software haben. Das ist für 8 Bit CRC's mit 16 bis 128 bis Polynom. Man könnte man mit dieser kleinen Stück die erhoheit.

Sicherheit von Prüfsummen

Ein praktisches Beispiel für Simulation unter Verwendung von Zufallszahlen. Die Fragestellung ist durchaus praxisnah: es sollen Datenpakete von 8 Byte Länge übertragen werden. Z.B. bei Transpondern (vgl MARC4-Newsletter 4). Das letzte Byte ist die Prüfsumme. Ist eine Summenbildung oder eine einfache Verknüpfung über XOR besser? Oder soll man eine CRC einsetzen? Der Rechenaufwand beim Decodieren ist dafür höher. Wenn ja welches Polynom?

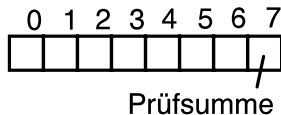


Bild 1: Datenformat

Eine Antwort auf diese Frage kann Simulation geben. Das Programm im Anhang ist für andere Problemstellungen nicht direkt verwendbar, zeigt aber die Vorgehensweise auf. Es werden hier 5 Varianten für die Prüfsumme durchprobiert:

- * Addition
- * XOR-Verknüpfung
- * CRC Polynom „TM“
- * CRC Polynom „PDV“
- * CRC Polynom „CRC8“

Als erstes bestimmt man 7 Datenbytes. Hier werden sie mittels Zufallszahlengenerator erzeugt. Da es sich bei Transpondern um Seriensnummern handelt, ist das sogar einigermaßen realistisch. Dann berechnet man die Prüfsummen und speichert sie im 8. Byte.

Nun erzeugt man einen Einzelbitfehler. Dazu betrachtet man die 8 Bytes als ein langes Datenwort mit 64 Bit. Für diesen Bereich läßt man sich vom Zufallszahlengenerator eine Bitposition errechnen. Und invertiert an dieser Bitposition das Datenbit. Da von Interesse ist, wie sich die Fehler-sicherung bei Mehrbitfehlern verhält, wird das mehrfach durchgeführt. Hier für 1 bis 8 Fehler.

RUN

Für jede der 5 Prüfsummenvarianten muß die Source umeditiert

und neu kompiliert werden. Was aber einem komplexeren Programm vorzuziehen ist. „Errors=“ gibt dann die Zahl der Bitfehler an. „Samples=“ zeigt die Größe der Stichprobe, hier 1000 Werte. Die Stichprobe sollte so groß gewählt werden, wie bei der Rechenzeit des verwendeten Systems, erträglich ist. Von der Obergrenze durch die 16 Bit Wortlänge des verwendeten FORTHS ist man hier noch weit entfernt. Der Ausdruck wird wegen der Einfachheit als Hexzahlen vorgenommen, weil die absoluten Werte ohnehin nicht von Interesse sind.

Bild 1: Ausdruck des Testprogramms

```

\ CRC TM
Errors=01 Samples=03E8 Missed=0000 Detected=03E8 Identical=0000
Errors=02 Samples=03E8 Missed=0000 Detected=03D0 Identical=0018
Errors=03 Samples=03E8 Missed=0000 Detected=03E8 Identical=0000
Errors=04 Samples=03E8 Missed=0006 Detected=03DF Identical=0003
Errors=05 Samples=03E8 Missed=0000 Detected=03E8 Identical=0000
Errors=06 Samples=03E8 Missed=0005 Detected=03E3 Identical=0000
Errors=07 Samples=03E8 Missed=0000 Detected=03E8 Identical=0000
Errors=08 Samples=03E8 Missed=0004 Detected=03E4 Identical=0000
\ CRC PDV-Bus
Errors=01 Samples=03E8 Missed=0000 Detected=03E8 Identical=0000
Errors=02 Samples=03E8 Missed=0000 Detected=03D0 Identical=0018
Errors=03 Samples=03E8 Missed=0000 Detected=03E8 Identical=0000
Errors=04 Samples=03E8 Missed=0010 Detected=03D5 Identical=0003
Errors=05 Samples=03E8 Missed=0000 Detected=03E8 Identical=0000
Errors=06 Samples=03E8 Missed=0009 Detected=03DF Identical=0000
Errors=07 Samples=03E8 Missed=0000 Detected=03E8 Identical=0000
Errors=08 Samples=03E8 Missed=0004 Detected=03E4 Identical=0000
\ CRC CRC8
Errors=01 Samples=03E8 Missed=0000 Detected=03E8 Identical=0000
Errors=02 Samples=03E8 Missed=000B Detected=03C5 Identical=0018
Errors=03 Samples=03E8 Missed=0000 Detected=03E8 Identical=0000
Errors=04 Samples=03E8 Missed=000F Detected=03D6 Identical=0003
Errors=05 Samples=03E8 Missed=0005 Detected=03E3 Identical=0000
Errors=06 Samples=03E8 Missed=000B Detected=03DD Identical=0000
Errors=07 Samples=03E8 Missed=0008 Detected=03E0 Identical=0000
Errors=08 Samples=03E8 Missed=0006 Detected=03E2 Identical=0000
\ +
Errors=01 Samples=03E8 Missed=0000 Detected=03E8 Identical=0000
Errors=02 Samples=03E8 Missed=003D Detected=0393 Identical=0018
Errors=03 Samples=03E8 Missed=0008 Detected=03E0 Identical=0000
Errors=04 Samples=03E8 Missed=0012 Detected=03D3 Identical=0003
Errors=05 Samples=03E8 Missed=0008 Detected=03E0 Identical=0000
Errors=06 Samples=03E8 Missed=0006 Detected=03E2 Identical=0000
Errors=07 Samples=03E8 Missed=0002 Detected=03E6 Identical=0000
Errors=08 Samples=03E8 Missed=000A Detected=03DE Identical=0000
\ XOR
Errors=01 Samples=03E8 Missed=0000 Detected=03E8 Identical=0000
Errors=02 Samples=03E8 Missed=0063 Detected=036D Identical=0018
Errors=03 Samples=03E8 Missed=0000 Detected=03E8 Identical=0000
Errors=04 Samples=03E8 Missed=0024 Detected=03C1 Identical=0003
Errors=05 Samples=03E8 Missed=0000 Detected=03E8 Identical=0000
Errors=06 Samples=03E8 Missed=0011 Detected=03D7 Identical=0000
Errors=07 Samples=03E8 Missed=0000 Detected=03E8 Identical=0000
Errors=08 Samples=03E8 Missed=000D Detected=03DB Identical=0000
    
```

Die Ausgabe „Missed“ zeigt, ob man sich die Prüfsumme verschaffen kann. „Detected=“ gibt die Zahl der korrekt als nachlässig erkannter Datenpakete an. Hier über „Identical“ ergibt sich heraus, daß man sich bei 3 Byteswerten in einem kleinen Zahlbereich die Prüfsumme mit Hilfe der die gleiche Prüfsumme zu einem anderen Wert durch Invertieren sich selbst, wobei man sich die Zahl der Bits unterscheidet. Man kann also wenn man Fehler zu „Missed“ noch für „Identical“ eintragen, um das Verhalten einer richtigen Zahl der Bits zu simulieren. Anders ausgedrückt: die Zahl der Samples entsprechen

Resultat

Der Ausdruck zeigt das Ergebnis der Simulation. Sie werden festgestellt für hieraus die Schlussfolgerung gezogen, denn CRC8 scheint sich am besten zu verhalten. Die Prüfsummen sind also identisch