

embedded

Rafael Deliano
Steinbergstr.37
82110 Germering
Tel 089/8418317

`j_r_d@t-online.de`

V1.0 (pdf) : 20. Mai 04
V1.1 (pdf) : Nov 06
V1.2 (pdf) : 14. Jan 07



Voransichts- Version

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

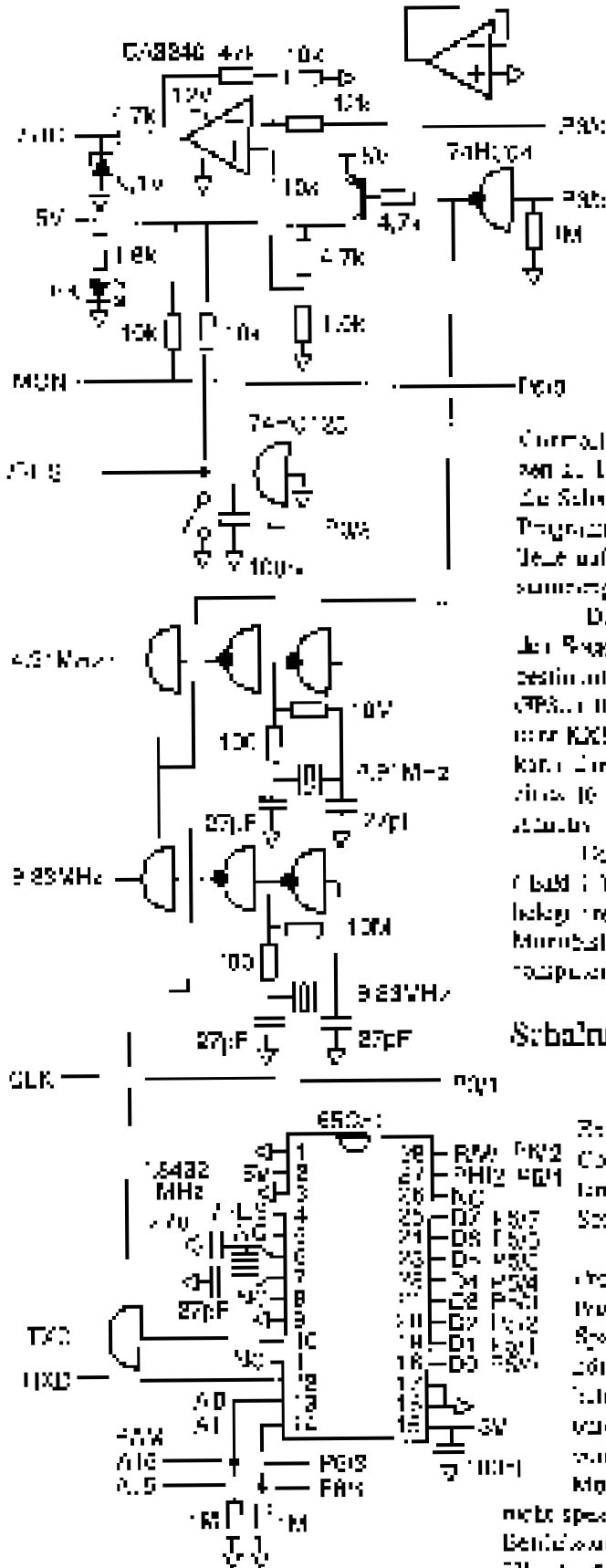
- 1 Inhalt, Impressum
- 2 Programmieren auf dem
68HC908
- 8 OCR-Eingabegerät
- 10 Einfacher Logarithmus
- 12 Compact Flash:
Hardware
- 17 Fletcher Prüfsumme
- 18 Hashing für RFIDs
- 20 Lineare Interpolation
in Tabellen

E

ng auf den
jetzt verfügbar,
eit für die Zeit-
schrift. Es ist nicht die beabsichtigte
Zusammenstellung von Artikeln
geworden. Aber es hat sich inzwi-
schen soviel Material angesammelt
daß es für eine weitere Ausgabe
reicht. Vorgesehen für das nächste
Heft ist u.a. das Filesystem für die
CompactFlash-Karte.

Die Listings sind in
nanoFORTH geschrieben. Für die
Konvertierung in andere FORTH-
Varianten sollte man im nanoFORTH-
Manual nachlesen das jetzt in der
F08-Version verfügbar ist.

Programmiergerät 68HC908



Motorolas FLASH-Controller haben eine weitgehend einheitliche serielle Schnittstelle die Programmierung und begrenzt Debugging von Software ermöglicht. Hier ist die Anschaltung mittels Einplatinencomputer dargestellt.

Die Versorgungsspannung 5V sollte speziell beim GP32 mit Maskenfehler (Code 3J20X) einen niederohmigen Entladewiderstand haben, hier 4,7k + 1,0k. Damit wird der 100nF Kerko im Adapter schnell auf unter 100mV entladen was für die Funktion des PowerOn Resets in diesen Controllern wünschenswert ist.

Die Monitorschnittstelle besteht hier nur aus einem Portpin mit Pullup-Widerstand. Das Datenformat entspricht einer UART. Aber da das Protokoll halbduplex ist, kann diese leicht in Software nachgebildet werden.

Der Resetpin kann wahlweise manuell über Taster geschaltet werden, wofür der Kerko als Entprellung nötig ist. Der kleine Kerko auf dem GP32-Adapter verhindert Übersprechen vom 9,8MHz Takt und ist nur bei ungünstiger Leiterbahnführung erforderlich. Alternativ kann man Resetpuls per Software auslösen, wobei nach Hochschalten eine Verzögerung im Programm nötig ist, um die Zeitkonstante des Kerkos auszugleichen.

Schaltung

Während des Programmierbetriebs wird die CPU durch einen externen Takt versorgt. Die beiden üblichsten Frequenzen werden deshalb im Grundgerät erzeugt. Typisch wird 9,8MHz benötigt.

Einige Controller haben intern als RC-Generatoren statt externe Quarze. Diese sind per FLASH abgleichbar. Den Abgleich kann das Programmiergerät machen. Zur Ausgabe des Takts für Messung ist es nützlich vom Adapter an einen Pin des Mitsubishi eine Leitung zu führen, die hier als CLK bezeichnet ist.

Abbildung 1:
Grundschialtung

OCR-Eingabegerät

Eigenbau ist wegen der benötigten Mechanik und Optik mühsam. Alternative ist die „Pistole“ (Bild 1) eines Optoline 3130 bzw 3140 der CGK Konstanz. Die Geräte wurden ab ca. 1983 hergestellt und sind heute oft ausgemustert preiswert erhältlich.

Im Kopf (Bild 2) sind steckbar zwei Glühlampen angebracht die mit regelbarer Lichtstärke das Papier beleuchten. Hinter einer etwa 4cm langen Optik befindet sich ein CCD-Bildaufnehmer mit 16x64 Pixel Auflösung. Aus der Optik wird über eine BPW34 Fotodiode die Lichtstärke ausgekoppelt und ins Steuergerät übertragen. Das grüne LED dient als Rückmeldung für den Benutzer, seine Steuerleitung wird aber auch in der Schaltung verwendet.

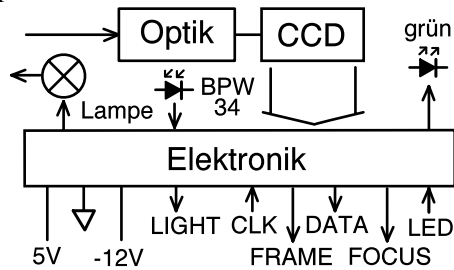


Bild 2:
Blockschaltbild



Bild 1:
Lesegerät

Verbindung zum Hauptgerät erfolgt über einen 15pol SubD-Stecker auf dem alle Signale Logikpegel haben (Tabelle 1). Die Stromaufnahme der 5V variiert stark mit der Beleuchtung durch die Glühlampen. Das Signal an Pin 6 zeigt über das in die CCD reflektierte Licht an ob die Pistole Papier im Blickfeld hat. Die eigentlichen Daten kommen mit TTL-Logikpegel an Pin 7 und der Rahmentakt an Pin 12. Takt wird vom Steuergerät an Pin 11 dazu geliefert.

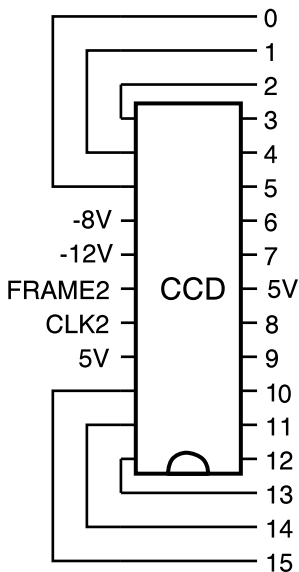
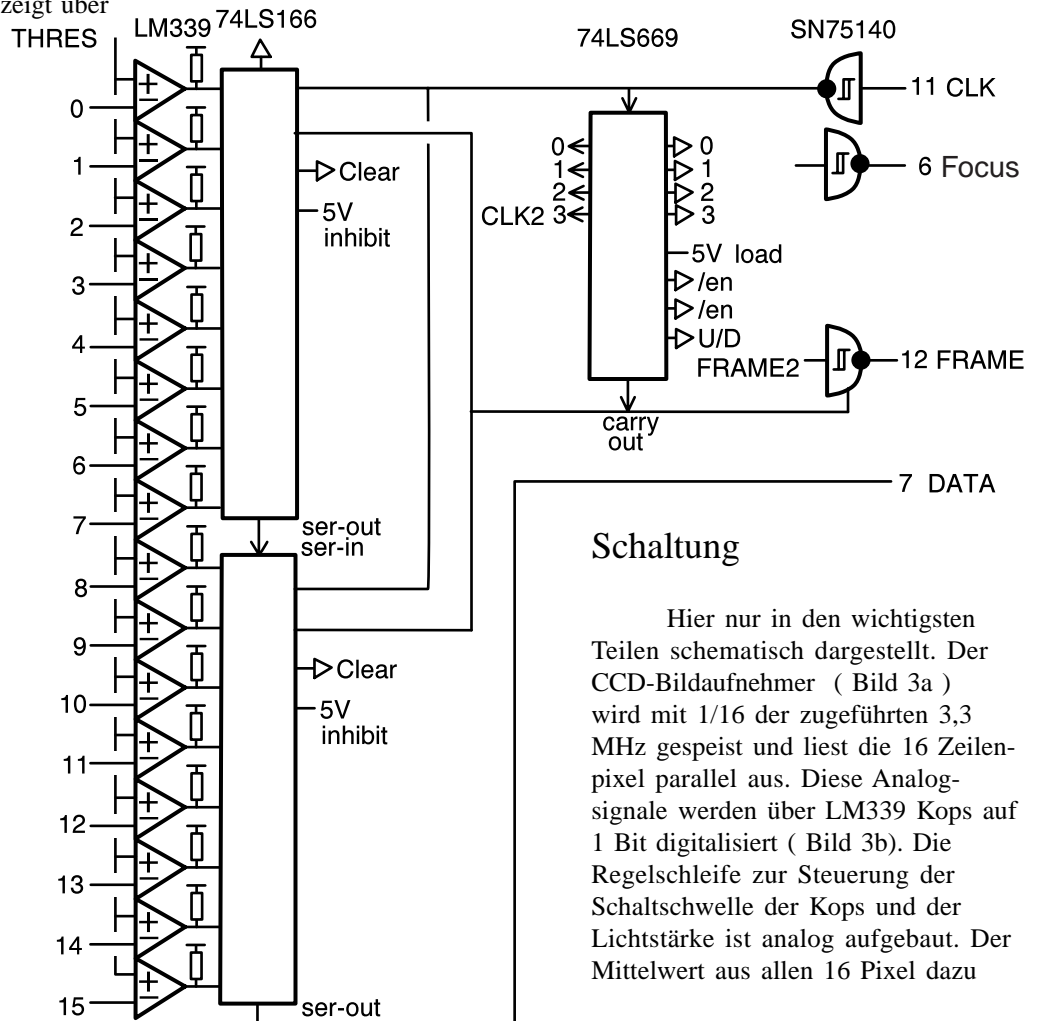


Bild3a: CCD

Bild 3b: Komparatoren & Schieberegister



Schaltung

Hier nur in den wichtigsten Teilen schematisch dargestellt. Der CCD-Bildaufnehmer (Bild 3a) wird mit 1/16 der zugeführten 3,3 MHz gespeist und liest die 16 Zeilenpixel parallel aus. Diese Analogsignale werden über LM339 Kops auf 1 Bit digitalisiert (Bild 3b). Die Regelschleife zur Steuerung der Schaltschwelle der Kops und der Lichtstärke ist analog aufgebaut. Der Mittelwert aus allen 16 Pixel dazu

Das ist die Schaltschleife der digitalen Steuerung.

```

1 000
2 000
3 20 32000 2000
4 00
5 -100 10000
6 00 FRAME : 8000000 ;
7 1 = auf 20 20000
8 0 = 10 50000
9 00 DATA : 8000000 ;
10 00
11 000 100 1000 20
12 1 = 100 20 ; 20000 ;
13 1 = 100 000
14 00 1000000 1000000
15 00 100 1000 1000
16 1000000 1000000
17 00
18 00 1000000
19 00 1000000 1000000
20 00 1000000
21 00

```

Die Zeit im Mikrosekundenbereich hat immer ein Jiffy. Das digitale 16 Bit Datenwort wird mit dem lokalen 74LS165 Schieberegister an einem gemeinsamen Bus übergeben. Der FRAME wird in die DCO erzeugt und pulst während der gesamten Zeit high (100 ns).

Adapter

Als Steuerung wird hier ein 5 Bit Turbinencomputer bestehend aus MC146818-002 mit 12KByte RAM verwendet. Der Kern stellt sich nicht den kontinuierlichen Datenstrom an, sondern, das einzelne Bits.

Die geteilten Daten vom Kabel werden über Schieberegister wieder in ein 16 Bit Datenwort gebracht (Bild 5) und dem Empfänger 74LS165 übergeben. Die Daten werden die Datenbreite von alle Bytes. Das ist am besten durch einen Controller zu bewerkstelligen. Dazu muß aber das Lesegerät mit dem Teil des Prozessors geteilt werden. Durch die Latenz der Dioden kann das nur ein Teilerschnitt sein. Der Empfänger wird die Adresse des Schieberegister in der Zeit, wo die Daten liegen, nutzen, dann jedes 5 µs für 16 Bytes, dann. Das ist ein "gerade" Wert der mit der ersten 16 Bytes des ersten Logik nachgefragt werden kann. Da aber keine zwei Bytes konstant frei sind, mussen für die letzten 16 Bytes über einen 74LS165 während des Einfusses

eines Bits des Empfängers werden. Die des 6,75 MHz des Datenkabels ergeben sich an der Frequenz von 200 MHz, das 200 weniger als im Original. Wenn die Leistung aber nicht zu klein ist, dann kann der Quell des Controllers auf 12,25 MHz erhöht, dann kann auf 9,75 MHz also 20%. Allerdings muß man diese die Einstellung der Hardware in der UMSI in der FORTS-Firmware prüfen.

Die des Kabel die Zeit sind an DATA, FRAME, CLK passende Terminierungen nötig. Trotzdem sind die Signal nicht optimal, weil 74LS165 und TTL Logik gemischt sind.

Die Schaltschleife der Logik wird durch die Signal 100 vom Grundgerät beeinflusst. Es ist ein Original und die Details eines Frames und 50% Intervall. Dabei wird die Länge der konstanter Länge in der Länge innerhalb des Frames hier und verglichen. Die Nachricht hier ist eine, wenn die Zeit ist erfolgreich, dann ist eine feste Einstellung durch Post. Das hat sich bei Verlegen mit einem Kontakt an unterschiedlich erweisen.



Bild 5: Ausdruck als ASCII

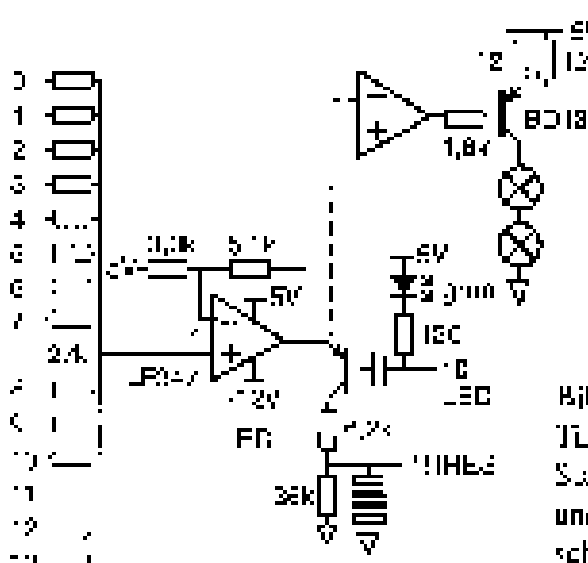


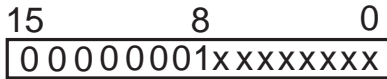
Bild 4: Teilung, Schmittstelle und Adapter-schaltung

Bild 3c: Schmittschwellenreglung

Einfacher Logarithmus

Speziell für Dynamikkompressoren sind oft nur sehr grobe Näherungen nötig. Da die Datenworte dabei sehr breit sind, empfehlen sich Tabellen wegen des Speicherverbrauchs nicht.

Bild1: scanbit



scanbit

Die simpelste Näherung ist sich bei einer positiven Zahl das höchste gesetzte Bit zu suchen (Bild 1). Die Funktion gibt es bei manchen 32 Bit RISC-Prozessoren im Befehlssatz, weil für Normalisierung von Floats nützlich. Dafür wird manchmal der Name „scanbit“ verwendet. Im ARM heißt er CLZ, „Count leading Zeros“.

Entsprechend gibt es auch den Befehl „spanbit“ der die höchste gesetzte Null findet. Für negative 2er-Komplementzahlen. Allerdings ist in Software wohl eine bitweise Invertierung des Datenworts gefolgt von scanbit angemessener.

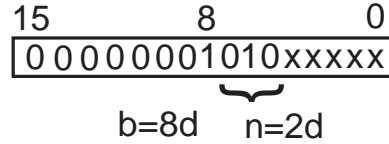
Die Werte von scanbit entsprechen der Funktion $y=lb(x)+1$. Wobei lb der binäre Logarithmus ist. Da Taschenrechner den manchmal nicht haben, ist die Umrechnungsformel über ln für Nachrechnen von Hand nützlich (Bild 2).

Etwas problematisch ist der hohe Datenverlust, wenn man 16 Bit auf 4 Bit staucht. (Listing SCNBIT.F74)

Tabelle 1:

x	y
0	0
1	0
2	1
4	2
8	3
16	4
32	5
...	...
16384	15

Bild 3: bitlog 16



$$y = 8 (b - 1) + n$$

Bild 2: Umrechnung ln auf lb

$$lb(x) = 1,44269041 * ln (x)$$

bitlog

Man kann das Verfahren natürlich verfeinern indem man mehr Bits auswertet. Unter dem Namen bitlog ist eine solche Variante in [1] für 16 Bit Integer angegeben (Bild 3). Dabei werden die 3 folgenden Bits hinter dem führenden Bit zur Veredelung verwendet. Die Berechnung entspricht relativ gut $y=8(lb(x)-1)$. Bei Eingangswerten 0 ... 7 treten allerdings Probleme auf so daß ein Patch nötig ist (Tabelle 2). Man beachte auch den Unterschied der Kennlinie zu echtem Logarithmus in diesem Bereich (Bild 4): letzterer läuft auf 1.

Die 16 Bit werden auf etwa 7 Bit gestaucht, der Maximalwert für y ist 119d. Das Verfahren läßt sich un schwer auf 32 und 64 Bit Datenworte skalieren (Tabelle 3). In Tabelle 4 sind Speicher und Laufzeit für Assemblerrou tinen auf Mitsubishi-6502 mit 2,54 MHz Busfrequenz angegeben (Listings BL16.F74 , BL32.F74 BL64.F74)

Tabelle 3: Skalierung von bitlog von 16 auf 32 und 64 Bit

Y=	b =	n =	bit	X<	Y=	Ymax =
$8(b-1)+n$	0 ... 15d	3	bit	X<8	$Y= X*2$	119d
$16(b-2)+n$	0 ... 31d	4	bit	X<16d	$Y= X*2$	479d
$32(b-3)+n$	0 ... 63d	5	bit	X<32d	$Y= X*2$	1951d

Tabelle 2: $y = lb(x)$

x	binär	lb(x)
0	00000000	0
1	00000001	0
2	00000010	1
3	00000011	1
4	00000100	2
5	00000101	2
6	00000110	2
7	00000111	2
8	00001000	3
9	00001001	3
10	00001010	3
11	00001011	3
12	00001100	3
13	00001101	3
14	00001110	3
15	00001111	3

Expander

Aus beiden Verfahren lassen sich auch die schrägen Kennlinien ableiten durch schlichte Exponentialfunktionen welche für diese sind wegen der kurzen Verzögerung von 1 oder 2 Bit-Berechnungen, wodurch man beliebige Kennlinien erzeugen kann

Listing 1: MATH-Block für 256x16-Programmierung CMB-Blocks 2002

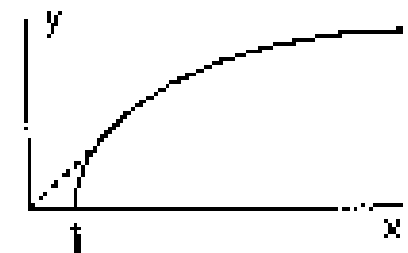


Bild 4: Kennlinien lb(x) & bitlog

Tabelle 4: bitlog Speicher & Laufzeit

Speicher	25600	15000
16384	120	120
8192	60	60
4096	30	30

Compact Flash an Controller

Der ideale Weg um einen Einplatinencomputer mit so etwas wie einer Floppy auszustatten. D.h. einen wechselbaren Speicher mit dem man Files von und zu PCs übertragen kann.

Die Vorzüge gegenüber einer Floppy sind erheblich. Die Schnittstelle paßt direkt an den Bus einer 8 Bit CPU. Die Bauform ist klein. Es ist sowohl 3,3V als auch 5V Versorgung möglich. Stromaufnahme ist mit ca. 50mA erträglich.

Die beiden Varianten CF I und CF II unterscheiden sich nur in der Bauhöhe. Die dünnere CF I paßt auch in CF II Stecker. 1994 mit 4 MByte Karten eingeführt denen 1996 die heute übliche kleinste Größe 8 MByte folgte. Übliche Speicherdichten für kleine Varianten sind 8, 16, 32 MByte.

Verfügbar, aber derzeit noch teuer, sind auch wesentlich größere Ausführungen. Da ursprünglich für Laptops gedacht paßt der 50pol Steckverbinder über einen passiven Adapter in PCMCIA-Karten. Neben dieser Anschaltung ist alternativ „True IDE“ möglich. In diesem Fall emuliert man genauso direkt eine Harddisk. Hier ist aber „hot plugging“, also Stecken und Entfernen der Karte bei laufendem Gerät nicht vorgesehen. Auf diese wünschenswerte Eigenschaft will man aber ungern verzichten. Die dritte Beschaltung „Common Memory“ ist

für Controller besonders geeignet. Man hängt direkt memory mapped am Bus der CPU.

- [1] www.compactflash.org
„CF+ and CompactFlash Specification Revision 2.0“
- [2] www.sandisk.com
„Compact Flash Memory Card Product Manual“
„Using SanDisk Flash ATA Components with an 8051 Microcontroller“
- [3] www.t13.org
„Working Draft X3T10/0948D“
ATA-2
- [4] www.renesas.com
„Hitachi Flash Cards User's Manual“
„Q&A on Hitachi CF and ATA Cards“

Hardware

Wegen des häufigsten RAMs ist ein Blindefachschalter sinnvoll. Der ist ein Flash-Controller. Die CF-Karte legt dann wie ein Hauptspeicher an Speicher. Wegen der Zugriffszeit von 100ns hat der Bus nicht zu schnell sein.

Der „hot plugging“ besteht in einem über Treiber die man instand schalten kann. Also sein I/O-Port. Das unterbindet sich der FSB-Schaltkreis. Die Treiber sind meist nicht, weil in der Schnittstelle nicht I/O sondern CS/CS-Pipe gefordert sind. Die Beschaltung Adapter ist komplex, benötigt werden typisch über 100k. Also könnte man sich auch auf CMOS legen. Neben der Bootschaltung sind drei weitere wichtige Punkte erwähnenswert.

Mit Flash schaltet die CPU die Versorgung des Speichers an.

Über den Pin „RESET“ kommt der Speicher zurücksetzen. Die Flash braucht nicht größer. Klare sein, so kann der Betrieb fortzusetzen wenn die Karte wieder benötigt. Ummittelbar nach PowerUp kann es sein 400ms dauern. Deutsche Zeiten schreien von 1000ms.

Zwei haben die Karten notwendig eine interne Reset-Schaltung. Über den Master wird man bei Anlegen der Versorgung. Also mit Schalter (mit einer Reset-Versorgung mit einem 10k Pullup). Wenn man den Einmenne anstrebt will verfahren, man ist fast mit CS/CS.

Der CS/CS-Pin ist an dem die Karte den Master herstellt und ist meist in der Pin-Liste nicht angegeben. Bei Pin, gelagert, man kann das Bit auch in einem Register abfragen.

CS/CS ist an einem mit CS/CS verbunden. Damit prüft die CPU ob eine Karte existiert. Sie wird dann die Versorgung anschalten und damit auch die Treiber freisetzen.

Schaltung

Der hier gezeigte Mittelteil CS/CS ist jeden Schaltungstyp als auf-mehrere-Zyklen aus. Das macht bei Vergleichsleistungen fast immer Probleme.

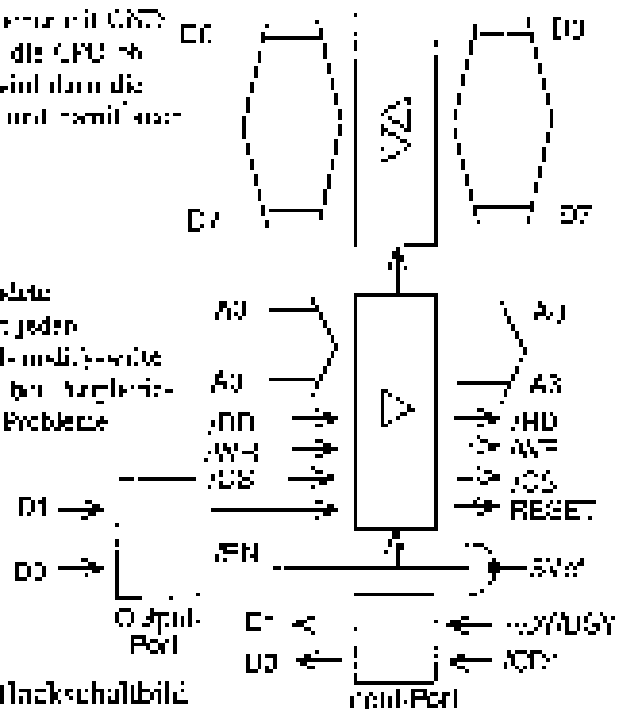


Bild : Blockschaltbild

sich CP-Zusten automatisch nicht schliessen, sondern erst wenn vorher Befehle über das Netz richtig abgeschickt werden. Voraussetzung ist, dass der künftige Empfänger dieser Daten aber auch unaktiv ist.

Hardwarekatalogen

Die Bedeutung findet man nicht überall, aber die Parameter Data Bits mit Error Indication sind nicht uninteressant. Frankweiler beschreibt in seinem zur PIC-Listung Hinzugefügten Text in Anlehnung an Hühner-Eindringler die Behandlung des 20-Bit-Error-Indicators. Der Anreger zum pathologischen Alter kann zu berücksichtigen sein, wenn aber nicht geschult ist.

(Fortsetzung von X20)

„Lassen Sie spekulieren“?

Diese Darstellung von 10^6 durch eine passende Anzahl Nullen (schon so normal wie auch Fehler in der Silberrunde mit den 10000, besonders unterstellten Punkten) besteht aus nachfolgender Form der Werte in der Spalte. Die Zahlen sind im Grunde auf 10 nach oben gerichtet, die Abweichung in den Spalten. Darauf aufbauend den maximalen Fehler an allen 4 Punkten berechnet. Ist ein größeres geworden, ist ein geprüfter Wert, den aber kein Nachprüfung, sondern die Gleichheit, sondern. Als alternative Alternative wurde geprüft, ob die Vorzeichen der Fehlerwechsel. Es ist ein Wert an der Stelle, wenn es von erwartet hat, dann ist es selbst ein Wert, dann die Fehler an allen Werten, nicht sein soll: Lösung PLINLE741.

Man sollte sich nicht an die 10-Bits stellen als Ergebnis der Fehler, die Vorzeichen Punkte (1) (Tabelle 1). Bei der Funktion X25 ist der relative Fehler anfangs klein, nach 6m Länge der Tabelle kann er steigen. Ein für die tatsächlichen Anweisung, wenn Werte unterhalb 500 und oberhalb 2000 weniger gute. Behandlung 8000000 andere Operationen über 1000000.

Tabelle 1: Test IDENTIFY-QUIVE

— — —	IF	→	set the power of
— — —	IDENTIFY-QUIVE	→	read data
	IDENTIFY-QUIVE	→	print data
	general configuration table		8480
	default number of cylinders		1016
	default number of heads		1008
	number of unformatted bytes per track		1000
	number of unformatted bytes per cylinder		1000
	default number of sectors per track		1000
	number of sectors per card		1000 3000
	static address		80000
	10000 10000 100		
	number type + data number	→	1000 1000
	number size in 512 byte increments		1000
	total 100 bytes passed in with flow		1000
	format number		Rev 2.00
	model number		
		Hardware	3000 2.0
	Maximum of 1 sector on 10M multiple		001
	write word not supported	→	1000 0000
	DMA not supported, DMA not supported	→	1000 0000
	DMA data transfer cycle timing not supported	→	1000 0100
	DMA data transfer cycle time not supported	→	0000
	load validity		0000
	current number of cylinders		0000
	current number of heads		0000
	current number of tracks		0000
	current number of sectors		0000
	current number of cylinders		0000 1000
	multiple sector setting is valid	→	0000 0100
	total number of sectors addressable in 10Mbytes		0000 1000
	supported PIO modes supported	→	0000 0000
	supported DMA transfer scheme + low number	→	0000 0000
	supported PIO transfer with IOCY flow control		0000 0000
	DMA → power down		

(Fortsetzung von X21, „Lassen Sie spekulieren“?)

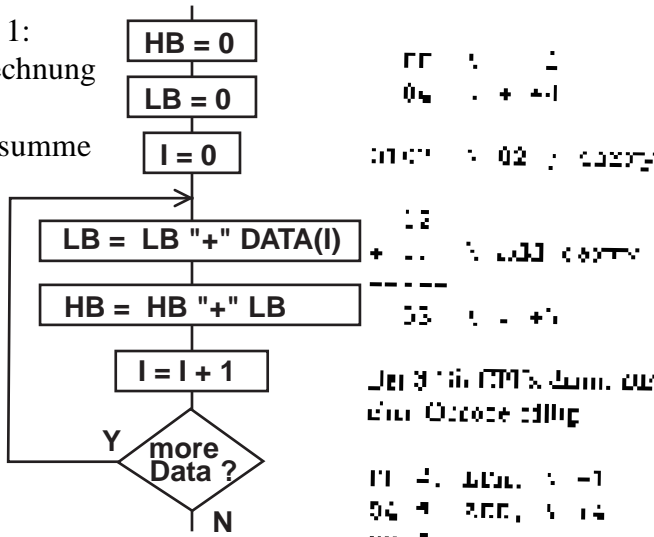
Adler X2
 In X25 sind sich es von 16 in Ueber eine notwendige Variante die bei etwas geringerer Fehler rate immer noch existieren, was 10000 in Spalte sein soll.
 The plotter was the mistake because of the lack of accumulation ignored. Man kann den Inhalt der Akkumulation nach als Rest einer Division durch 10000 interpretieren. Der Rest einer Division durch die Formel 10000 - angegeben, in 1000 durch den Betrag 10000 an der Rest als Ergebnis dargestellt. Durch passende Auslegung des Programms ist es wohl

- möglich die Division mit 10000 Bytes ausführen zu lassen. Trotzdem scheint das Verfahren für kleine Computer nicht zu passen.
- (1) "Plotter, An Anthracite Case" and "The Social Transmission" TFF Trans. on Communications Jan 1982
- (2) "Schweizer Anweisung für die Benutzung der OSI-Checksum Calculation" AFN Computer Kommunikation Review Okt. 1980

Fletcher Prüfsumme

In Software ohne Tabellen schnell berechenbar und fast so wirksam wie CRC.

Bild 1: Berechnung der Prüfsumme



Erste breitere Anwendung erfolgte in OSI-Protokollen. Da Pakete aus Bytes bestehen, wurde hier als Wortbreite der Daten 8 Bit verwendet. Wie auch in [1] als gängigste Variante angenommen. Bild 1 zeigt schematisch den Ablauf. Es gibt zwei Akkumulatoren LB und HB deren Länge dem Datenwort, also einem Byte entspricht. Nach Ende der Berechnung werden die Akkumulatoren zur Prüfsumme, hier also mit 16 Bit Länge zusammengesetzt. Typisch wird diese little-endian dem Paket angehängt.

Es sind nur Additionen erforderlich, aber Berechnung in 1er-Komplement ist besser als in 2er-Komplement und deshalb üblich [1]. In Bild 1 ist diese etwas andere Addition durch „+“ gekennzeichnet.

1er-Komplement

Eine recht ungebrauchliche Variante der Zahlendarstellung (Bild 2), die Arithmetik üblicher CPUs verarbeitet 2er-Komplement direkt. Für 1er-Komplement ist hier die nachträgliche Addition des Carrybits nötig. Beispiel:

$$FE \setminus -1$$

Bild 4: Adler-Variante

$$A = (LB + DATA(I)) \bmod FFF1h$$

$$B = (LB + HB) \bmod FFE1h$$

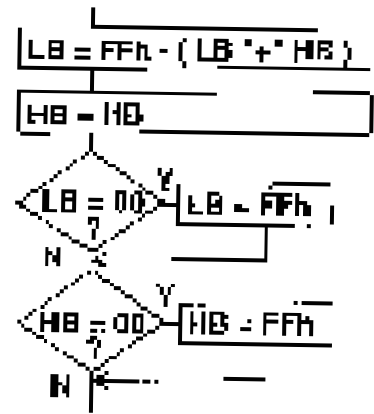


Bild 3: Modifikation des Erzwertes

Nur bei 1er-Komplement die Fletcher Prüfsumme. Auch dagegen gibt es Einwände, so soll die Fletcher Prüfsumme nicht in 16 auf 16 Bit sondern in 32 Bit dargestellt sein. Die 32 Bit Darstellung ist in der Tabelle dargestellt. Nur die 8-Bit Darstellung der Prüfsumme ist in der Tabelle dargestellt.

Die Fletcher Prüfsumme ist in der Tabelle dargestellt. Die 8-Bit Darstellung der Prüfsumme ist in der Tabelle dargestellt.

Fletcher-16

Für Internetanwendungen z.B. RDP 146 wird die Fletcher Prüfsumme in 16 Bit dargestellt. Die 16 Bit Darstellung der Fletcher Prüfsumme ist in der Tabelle dargestellt. Die 8-Bit Darstellung der Fletcher Prüfsumme ist in der Tabelle dargestellt. Die 16 Bit Darstellung der Fletcher Prüfsumme ist in der Tabelle dargestellt.

Bild 2: 2er Komplement

	2er Komplement	1er Komplement
1er/2er +127	7F	7F
Komplement 8 Bit	01	01
+0	00	00
-0	FF	FF
-1	FE	FE
-2	FD	FD
-127	80	80
-128	80	80

FF 1 1
04 1 + 4
0100 1 02 1 0000
12
+ 11 1 000 0000
133 1 1 + 1
Der 3 in CPU dann nur ein mal die eine Operation nötig
11 1 100 1 1
04 1 100 1 1
011 1 100 1 1

Da man in CPU mit 16 Bit Operatoren rechnen kann man sich keine Sorgen machen. Die Fletcher Prüfsumme ist in der Tabelle dargestellt. Die 8-Bit Darstellung der Fletcher Prüfsumme ist in der Tabelle dargestellt.

Null

Wenn man bei CRC mit Startwert Null alle Bytes eines Pakets mit der Fletcher Prüfsumme addiert, so ist das Ergebnis Null. Es hat sich eingebürgert dieses Verhalten mit anderen Prüfsummen aufzuweisen. Dann wird man die Fletcher Prüfsumme addieren, bevor man sie zum Paket sendet. Als 0 wird dann vom Empfänger bei Mittelwertwert in Sender annull 0, wenn man ein geringeres A. wird in Einzelfall 0 ist. Dann ein Paket wird nur einmal gesendet aber beim Weg durch ein Netzwerk war die Prüfsumme oft überprüfbar.

Wenn alle Bytes des Pakets

Hashing für RFIDs

RFIDs sind über Funk abfragbare Seriennummer-ICs. Hashing ist ein Verfahren für effizientes Suchen in Listen.

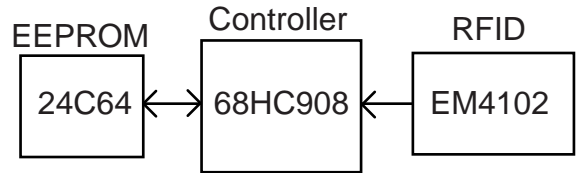


Bild 1: Lesegerät

Bild 2: Adreßgenerierung

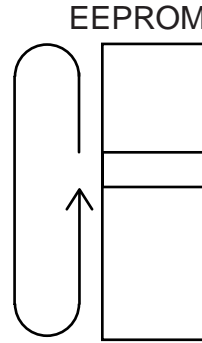
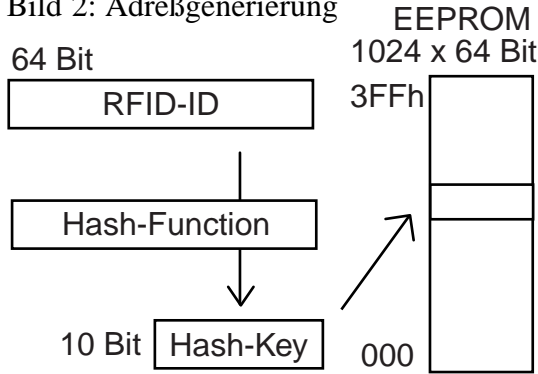


Bild 3: Suche

Stück linear suchen. Beim Einprogrammieren eines Transponders sucht man die nächste leere Speicherzelle, also alle 64 Bits gesetzt. Dort programmiert man die RFID-ID ein. Beim Suchen vergleicht man die RFID-ID des Transponders mit dem Inhalt der Speicherzelle auf Identität. Bzw. ob die Speicherzelle leer ist. In letzterem Fall ist die ID nicht im EEPROM.

Das Verfahren reduziert die Suche in einer langen Liste auf die Suche in vielen kurzen Listen. Diese entstehen zwangsläufig, da Hash-Kollisionen, also mehrere IDs zeigen auf eine Teilliste, nicht vermeidbar sind.

Weiter entwickelte Hash-Verfahren führen statt linearer Suche noch einen 2. Hash-Schritt durch, aber das ist hier zu kompliziert. Besonders weil man aus dem seriellen EEPROM nach der Startadresse die folgenden Bytes kontinuierlich lesen kann, während man für Sprünge nochmal Kopf und Adresse schreiben muß.

Effizienz

Die Wirksamkeit des Verfahrens hängt von der Statistik der Eingangsdaten und der Anpassung der Hash-Funktion an diese ab. Ferner davon wie voll der Speicher ist.

Die preiswertesten 125kHz Transponder (RFID) haben eine feste Nummer die vom Hersteller z.B. per Laser eingeschrieben wird. Damit dieser trotz kontinuierlicher Fertigung eine „einzigartige“ Seriennummer garantieren kann, ist diese ziemlich lang, typisch 64 Bit. Da darin auch Prüfsummen enthalten sind, ist die effektive Länge kürzer. Um Portierung zwischen verschiedenen Anbietern zu vereinfachen ist es für ein Lesegerät trotzdem günstiger mit der vollen Länge von 64 Bit zu arbeiten.

Bei Anwendung in größeren Gebäuden und Hotels kann es erforderlich sein, daß das Lesegerät hunderte von Schlüsseln erkennen soll. Wenn diese in einem externen seriellen EEPROM gespeichert sind, ist der Zugriff langsam (Bild 1). Um die

Reaktionszeit des Lesegeräts kurz zu halten muß die Suche optimiert werden.

Hashing

Das Suchen in ungeordneten langen Listen ist ein altes Problem der Datenverarbeitung und eine bekannte Lösung heißt Hashing. Durch eine geeignete Hash-Funktion wird aus dem 64 Bit Datensatz des Transponders ein 10 Bit Wert berechnet, der als Zeiger in die Liste mit den 1024 x 64 Bit Worten dient (Bild 2). Dort sucht man nun linear aufwärts weiter. Wenn oberhalb 3FF ein Moduloüberlauf eintritt, geht die Suche ab Adresse 000 weiter (Bild 3). Der Endpunkt ist der ursprüngliche Einsprungpunkt auf den der Hash-Key zeigt. Normalerweise muß man aber nur ein kurzes

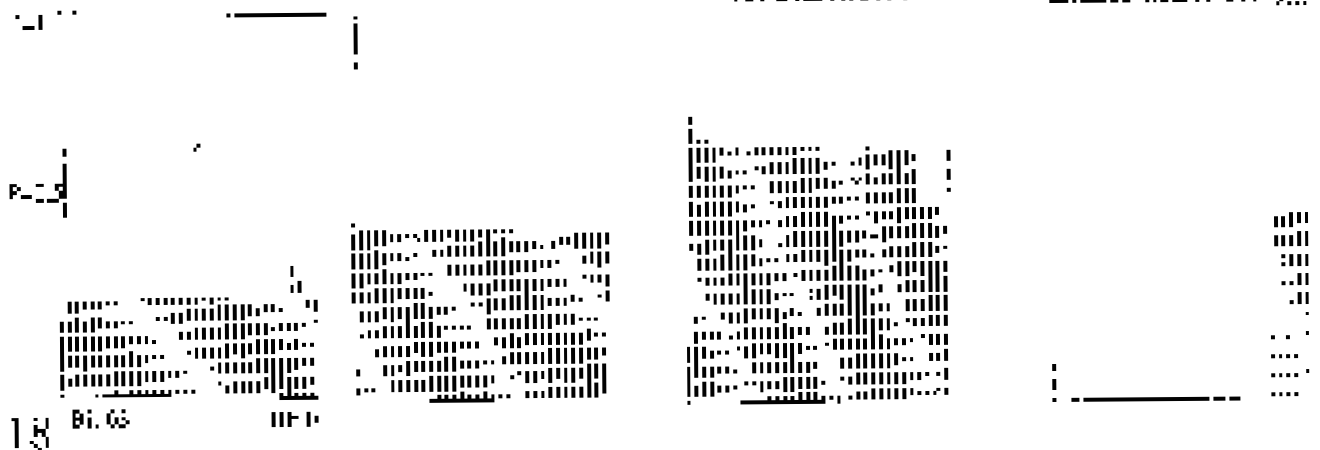
HA = US-Gesetzgeber:

LFZR Startwert 1 ...

LFZR Startwert A

LFZR Startwert F ...

LFZR Startwert 9...



Werte 10 Bit benötigt wird. Einfache Lösung ist aus jeweils 32 Bit zwei 8 Bit Teilworte zu bilden und diese durch XOR zu verknüpfen (Bild 5).

Test

Man läßt den Generator wieder 1024 Muster erzeugen die dann über die Hash-Funktion auf 10 Bit Adressen komprimiert werden. Anhand dieser inkrementiert man jeweils einen von 1024 16 Bit Zählern die die Speicherzellen des EEPROMs darstellen. Im Idealfall enthält jeder Zähler nach Ende des Tests den Wert 1. Jede RFID-ID wäre dann einer anderen

Adresse zugeordnet worden. Praktisch kommt es aber zu Kollisionen, also enthalten manche Zähler Werte grösser 1. Dementsprechend werden einige Adressen überhaupt nicht an-gesprochen und enthalten den Wert 0.

Tabelle 1 zeigt die Ergebnisse für 8 Bit CRC, Fletcher und XOR. Mit Testdaten erzeugt vom Binärzähler und LFSR, letzterer mit 3 verschiedenen Startwerten (vgl auch Bild 4). Die Spalte „1“ enthält die guten Fälle, rechts davon sind die Kollisionsfehler. Um den Vergleich zu vereinfachen ist am Ende noch eine gewichtete Fehlersumme aufgeführt, die

ermittelt. Die Kollisionsraten sind 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100.

- Die CRC enthält zwei bit-reverse Werte für den Zähler
- Aber das Ergebnis von Fletcher ist ungeachtet XOR ist sehr
- wichtig, man beachte die Vielfach-Kollisionswerte nach unten in der Tabelle
-
- [1] „PN-Stampen“
- © 2008
- [2] „A Concise and Usable Scheme for Address Lookup in Computer Networks“ IEEE Trans. on Com. 1979, 27, 1151-1157

Lineare Interpolation in Tabellen

Ein simples, wohlbekanntes Verfahren wird genauer wenn man die Tabellen optimiert.

Hier anhand der Berechnung

$$y = x^{2,5}$$

dargestellt. Wobei für x Werte von 0 ... 400h berechnet werden sollen. Eine direkte 32 Bit Tabelle würde 4kByte Speicher benötigen. Die hier gewählte Alternative ist, eine verkürzte Tabelle zu verwenden die nur

Tabelle 1: Fehler

unoptimierte Tabelle:

X	Ref Y	Fehler = Approx - Ref
0000	0000	0000 <-
0001	0000	0007
0002	0000	000A
0003	0000	0010
0004	0000	0020 <-
0005	0000	0038
0006	0000	0058
0007	0000	0082
0008	0000	00B5 <-
...		
03F8	01F6 0EFB	0000 <-
03F9	01F7 4B79	00B3
03FA	01F8 886E	00EF
03FB	01F9 C5DB	00B3
03FC	01FB 03BF	0000 <-
03FD	01FC 421C	00B3
03FE	01FD 80F0	00EF
03FF	01FE C03C	00B3
0400	0200 0000	0000 <-

jede 4. Stützstelle enthält und damit nur 1kByte belegt. Die Berechnung der 3 anderen Punkte durch lineare Interpolation (Bild 1) erfordert nur Additionen und Shiftbefehle und ist damit auch auf Controllern sehr leicht durchführbar. Dabei ist eine Sprungzieltabelle die von den untersten beiden Bits von x gesteuert wird eine geeignete Art der Implementierung (Listing LINT.F74)

optimierte Tabelle:

X	Ref Y	Fehler = Approx - Ref
0000	0000	0000 + 0000 <-
0001	0000	0001 + 0005
0002	0000	0006 + 0007
0003	0000	0010 + 0003
0004	0000	0020 - 0006 <-
0005	0000	0038 + 0006
0006	0000	0058 + 000A
0007	0000	0082 + 0004
0008	0000	00B5 - 000A <-
...		
03F8	01F6 0EFB	- 0077 <-
03F9	01F7 4B79	+ 003C
03FA	01F8 886E	+ 0078
03FB	01F9 C5DB	+ 003C
03FC	01FB 03BF	- 0077 <-
03FD	01FC 421C	+ 003C
03FE	01FD 80F0	+ 0078
03FF	01FE C03C	+ 003C
0400	0200 0000	- 0078 <-

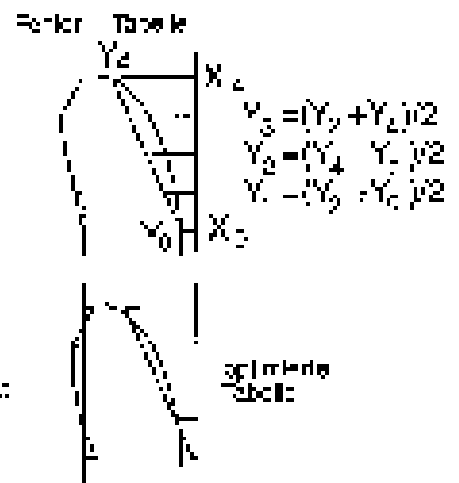


Bild 1: Interpolation, korrekt & optimiert.

Optimierung

Wie in Bild 1 noch dargestellt, sollte man nicht einfach nur gespeicherte Werte der 4kByte Tabelle verwenden. Man hätte zum einen in den Stützstellen permanent 16 Bit oder in den interpolierten Punkten nur 8 Bit zu bewahren. Wirtschaftlicher ist eine gleichzeitige kleine Abweichung an allen Punkten. Wie in Bild 2 dargestellt, dadurch erreicht, daß man nur die 4kByte Tabelle eines verschreibt

(Übersetzung J.76)