

# embedded



## Voransichts- Version

Für Bezug des Originals  
siehe FAQ auf  
[www.embeddedFORTH.de](http://www.embeddedFORTH.de)

- 1
- 2
- 4
- 5 Januarsener Tag
- 6 Unix-Zeit
- 8 LCD-Display
- 10 Touch Memory
- 12 Intel-Hex & Motorola S
- 13 HDLC-Rahmen
- 15 Linearer Code

1

Rafael Deliano  
Steinbergstr.37  
82110 Germering  
Tel 089/8418317

[j\\_r\\_d@t-online.de](mailto:j_r_d@t-online.de)

V1.0 ( Papier ) : 9. März 98  
V2.0 ( pdf ) : 18. Feb. 01  
V2.1 ( pdf ) : 28. April 04  
V2.2 ( pdf ) : 14. Jan 07

## READ . ME

In diesem Heft ist hauptsächlich Material zusammengefaßt das von Claus Vogts VD übriggeblieben ist, für sie nicht mehr rechtzeitig fertig wurde, oder für das nicht durchführbare VD-Sonderheft „embedded Systems“ gedacht war.

Die übrigen Beiträge bilden den Schwerpunkt dieses Heftes: „Zeit“. Dieses Thema ist damit nicht vollständig abgehandelt, weil z.B. eine Beschreibung von DCF77 fehlt. Kommt in einer späteren Ausgabe.

Man braucht Kalendersoftware sicher nicht oft auf Einplatinencomputern. Ich bin erst über das UNIX-Zeitformat gestolpert, als ich ZMODEM zu implementieren anfang. Irgendwann benötigt man solche Algorithmen also doch und man möchte dann gerne auf fertige und dokumentierte Routinen zurückgreifen.

Die Listings sind in nanoFORTH geschrieben und in der ursprünglichen Version von Heft 1 war deshalb ein Artikel enthalten, der die Konvertierung auf andere FORTHS beschreibt. Dieser Beitrag wurde überarbeitet ins nanoFORTH-Manual übernommen.







# Scaligers Julianischer Tag

Das Umrechnen von Datumsangaben mit dem gregorianischen Kalender ist extrem unhandlich. Scalingers Kalender ist eine Kette aus Tagen ohne Jahre und Tage, der sich auszeichnet für Rechnungen eignet. Obwohl er schon Jahrhunderte alt ist, findet man ihn immer noch in Chronologie, Astronomie und Raumfahrt.

Josef Scaliger lebte im 16. Jahrhundert und war einer der führenden Kalenderexperten seiner Zeit. Er hatte vergleichende Studien europäischer und arabischer Kalender gemacht und seine neue Darstellungsweise als wirksames Hilfsmittel dafür entwickelt. Die gregorianische Reform war von Protestanten wie Scaliger noch nicht akzeptiert worden. Das Jahr basiert bei ihm weiterhin auf dem einfacheren julianischen Kalender der Römer mit seinen 365,25 Tagen. Also wieder 3 Jahre zu 365 Tage, gefolgt von einem Schaltjahr zu 366 Tagen. Wobei jedes durch 4 teilbare Jahr das Schaltjahr ist.

Das System wird deshalb oft auch julianischer Kalender genannt. Es ist aber nicht mit dem historischen römischen Kalender identisch. Weshalb man die Bezeichnung tunlichst vermeiden sollte. Der Scaliger-Zyklus beginnt mit dem 1. Jan 4713 vChr historischer Zeitrechnung. Dabei ist erstens anzumerken, daß die historische Zeitrechnung das Jahr 0 nicht verwendet, was die mathematische Handhabung erschwert. Es ist deshalb für negative Jahre eine astronomische Zählweise entstanden, die mathematisch korrekt ist, aber damit von der gängigen Zeitrechnung um ein Jahr abweicht.

-4713 ... -3 -2 -1 +1 +2 historische Zr,  
-4712 ... -2 -1 0 +1 +2 astronomische

Zweitens wird im Zusammenhang mit Scaligers System vor der gregorianischen Kalenderreform normalerweise nach dem römischen julianischen Kalender gerechnet. Die Bezeichnung der Tage entspricht dabei jedoch dem heutigen Gebrauch und ist damit historisch nicht korrekt. Name und Dauer der Monate, auch des Februars mit seinem Schalttag sind richtig. Die alten Römer haben die Tage im Monat jedoch nicht einfach nur durchnummeriert und auch den Schalttag im Februar anders angeordnet. Der letzte Tag nach altem System ist der 4. Okt 1582. Auf



*So bin ich ein Astronomus/  
Erkenn zukünftig Finsternuß/  
An Sonn vnd Mond/durch das Gestirn  
Darauf kan ich denn practieirñ/  
Ob künfftig komm ein fruchtbar jar  
Oder Theurwung vnd Krieghsgefahr/  
Vnd sonst manicherley Kranckheit/  
Milesius den anfang geit.*

Der Astronomus. Holzschnitt von Jost Amman, Vers von Hans Sachs

ihn folgt als nächster Tag der 15. Okt 1582 nach gregorianischem System.

Der Scaligerzyklus endet nach 7980 julianischen Jahren mit dem 31. Dez 3267 nChr julianischer Zeitrechnung. Im Gregorianischen Kalender ist das der 22. Jan 3268. Da der Zyklus damit die gesamte interessante Zeit der Weltgeschichte abdeckt und einige rechentechnische Vorteile bietet, avancierte er zum bevorzugten System der Chronologie und Astronomie. Er ist deshalb heute immer noch in Gebrauch.

## Julianischer Tag

Tatsächlich rechnet man heute in diesem Kalender nicht mit Jahren, sondern sein Rückgrat ist der Julianische Tag, „Julian Date“, JD. Die Tage laufen ab der Stunde Null kontinuierlich hoch. Z.B.:

1. Jan 1900 = JD 2415020

Die Zahl wird ausgerechnet mit einem nicht übermäßig komplizierten Rechenverfahren, man muß hier die kleine Zahl des Julianischen Kalenders in Stunden, Minuten, Sekunden herunterrechnen, umrechnen, zu werden, dann multiplizieren als Formelstelle eingeben. Z.B.

2010 = Mitternacht des 15. Okt  
1950 = Mittag des 1. Sept

Es gibt aber die astronomischen Formeln hier an der Definition gleich. Sie haben die Stunden für die Verschiebung der Nullstelle verschoben, damit sie an einem bestimmten Beobachtungszeitraum keinen Holmswechsel haben. Umrechnung:

JD 2415020 + 4713 = 1582

Und es haben ihren eigenen JD hat, 2010 Weltzeit sind, was JD bedeutet. Sie kann daher auch angegeben, daß sich in JD heute immer ganz in der Literatur findet. Das wird dann auch, daß der Jahre Astronomie oder Chronologie, was man wissen kann.

## Modifizierter Julianischer Tag

Mit anderen Worten, die Verschiebung um die 4713 Stunden, was die JD hat, die Stunden und Minuten der Mitternacht, jetzt JD ist die Formel der Zahlen zu bestimmen, man kann auch mit dem 15. Nov 1858 00:00:00 als JD 2415020.

JD 2415020 + 4713 = 1582  
JD 2415020 - 4713 = 1950

Viele Angaben werden heute als JD publiziert. Es hat sich jedoch nicht allgemein durchgesetzt. Der JD ist ein, wenn unpräzise, weil sich die Beobachtungszeit nicht mit JD bezeichnen können. Es gibt eine abgeleitete Variante der Julianischen Zeitrechnung, daß in Form der Julianischen Tag, die Formel der Zahlen zu bestimmen, man kann auch mit dem 15. Nov 1858 00:00:00 als JD 2415020.

erweitert sein. Nullpunkt zum 1. 1970  
gelöst:

ANSI C Tag - 11: 24439735

### Implementierung

Bei der Implementierung für die Wandlung von Kalendertangaben in UTC wurde auf die unvollständigen Daten vor die 15. Stunde (Dauer von zwei oder drei Tagen) keine Rücksicht genommen.

Die Routine `FTO` und `FTOH` arbeiten mit dem angegebenen Datum, nicht mit dem Datum des kalendrischen Datums. Die Variable `DATE` verarbeitet

die Angaben auch gegenüber dem Kalender und ist für Datumangaben gültig. In 1.1.1970 zu veränderten. Die Variable `DATE` verarbeitet kalendrische Angaben für die Zeitdauer.

Die Variable `DATE` verarbeitet die Angaben an dem 1.1.1970. Die Anwendung, die nicht den kalendrischen Wert des Datums verarbeitet, kann in 1.1.1970 zu veränderten. Die Variable `DATE` verarbeitet kalendrische Angaben für die Zeitdauer.

Die Variable `DATE` verarbeitet die Angaben an dem 1.1.1970. Die Anwendung, die nicht den kalendrischen Wert des Datums verarbeitet, kann in 1.1.1970 zu veränderten. Die Variable `DATE` verarbeitet kalendrische Angaben für die Zeitdauer.

man im üblichen 16 Bit FORTH keine 64 Bit Divisionsbefehle. Zweitens haben die Jahre unterschiedliche Länge, Division wäre also unmittelbar nicht verwendbar. Man behilft sich mit einer Primitivdivision durch kontinuierliche Subtraktion. Für das Sekundenjahr das man jeweils abzieht, kann man so auch passend die Länge variieren, wenn es sich um ein Schaltjahr handelt. Nächstes Problem ist das Erkennen des Unterlaufs, wenn man in FORTH kein Carryflag hat. Für Jahre bis 2038 ist das oberste Bit des 32 Bit Wertes normalerweise Null. Man kann Unterlauf dann leicht erkennen, wenn man das oberste Bit nach der Subtraktion darauf testet, ob es gesetzt ist. Damit ist das Resultat negativ und diese Subtraktion war schon zuviel. Man verwirft das neue Resultat und der vorhergehende Wert ist das Endergebnis.

Die Variable `DATE` verarbeitet die Angaben an dem 1.1.1970. Die Anwendung, die nicht den kalendrischen Wert des Datums verarbeitet, kann in 1.1.1970 zu veränderten. Die Variable `DATE` verarbeitet kalendrische Angaben für die Zeitdauer.

## UNIX-Zeit

Dieses computerinterne Zeitformat dürfte nicht nur die größte Verbreitung haben, sondern es ist auch sehr effizient. Man muß nur einen 32 Bit Zähler jede Sekunde inkrementieren. Leider ist die Umwandlung daraus ins „menschliche“ Format etwas mühsam. Deshalb nicht günstig um daraus die Uhrzeit auf einem Display anzuzeigen. Aber sicher optimal für Zeitstempel.

Die 32 Bit Zahl gibt die Sekunden an, die seit dem 1. Jan 1970 00:00:00 UTC vergangen sind. Durch UTC wird man von Zeitzonen unabhängig. Dafür fehlt natürlich der sichere Bezug zur örtlichen Zeit. Gewisse Feinheiten wie Schaltsekunden werden dabei auch nicht berücksichtigt. Beispiel:

00000000h = 1. 1.1970 0: 0: 0  
30000000h = 9. 7.1995 16:12:48  
7FFFFFFFh = 19. 1.2038 3:14: 7  
FFFFFFFFh = 7. 2.2106 6:28:15

Vorteilhaft ist die Kompaktheit, man belegt nur 4 Bytes. Diese ist wünschenswert, wenn man im Kopf eines Files abspeichern will, wenn es das letztmal modifiziert wurde. Gleichzeitig kann man diese Zeitangabe sehr simpel erzeugen, wenn man als Systemtick 1sec wählt. Man muß nur einen 32 Bit Zähler inkrementieren. Trotz des platzsparenden Formats wird ein Zeitraum von über

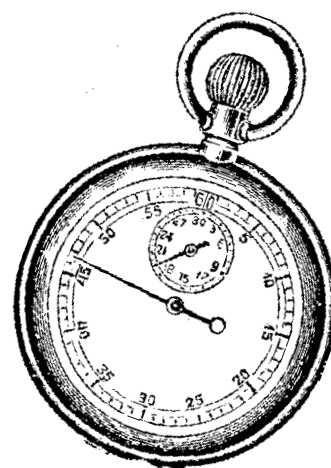
100 Jahren auf die Sekunde genau aufgelöst.

Gewisse Nachteile ergeben sich natürlich auch. Man kann kein Datum vor 1970 darstellen, denn negative Zeiten mittels des 32. Bits sollte man nicht definieren. Sonst reicht die Wortbreite nur bis zum Jahr 2038, und das ist heute schon etwas knapp. Hauptproblem ist aber vor allem die zähe Umwandlung ins „menschliche“ Format.

### Datum decodieren

Die Routine `DATE!` erhält die codierte Zahl auf dem Stack und legt das Ergebnis in den Variablen `DAY`, `MONTH`, `YEAR`, `HOURL`, `MIN`, `SEC` ab ( Listing UT1 ). Für diese genügt ein Byte. Nur `YEAR` muß 16 Bit breit sein.

Die Extraktion der Jahre erfolgt nominell durch „Division“ mit der Sekundendauer eines Jahres. Damit fangen dann schon die Probleme an. Erstens hat









# Treiber für Touch Memory

Die Firma Dallas Semiconductor hat einen seriellen Bus eingeführt mit dem man über nur zwei Kontakte auf Peripherie-ICs zugreifen kann. Auf Leiterplattenebene ergeben sich dabei kaum Vorteile gegenüber gängigen Verfahren wie dem I2C-Bus. Doch kann man damit ICs in den robusten Knopfzellegehäusen unterbringen, die von Batterien bekannt sind. Im einfachsten Fall enthält der Chip nur eine individuelle Seriennummer im ROM. Für komplexere Anwendung ist auch lithiumgepuffertes RAM verfügbar. Damit ist das Touch Memory ( TM ) als billiger Transponder einsetzbar, da es in vielen Fällen kein Nachteil ist, daß es mechanisch kontaktiert werden muß.

Hier wird die Software für die Ansteuerung der ältesten Version, den DS1991 dargestellt. Dieses TM verfügt über eine Seriennummer im ROM und 192 Byte lithiumgepuffertes RAM. Man kann mehrere TMs in einem Bussystem parallel auf eine Leitung schalten. Was jedoch die Software verkompliziert und von der Applikation meist nicht benötigt wird. Hier ist deshalb nur die Ansteuerung eines einzelnen TMs vorgesehen.

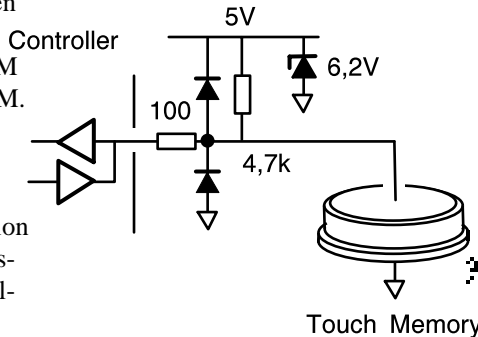
## Hardware

Man braucht am Controller einen Portpin den man bidirektional umschalten kann, sowie einen Pullup-Widerstand ( Bild 1 ). Da das TM jedoch vom Anwender wechselbar sein soll, ist normalerweise noch etwas ESD-Beschaltung benötigt ( Bild 2 ).

## Software

Auf unterster Ebene sind Befehle definiert die einen Reset auslösen, Bits lesen und schreiben. Im Gegensatz zum I2C-Bus muß hier ein festes Timing eingehalten werden, weshalb man die Verzögerungsschleifen auf den Takt der

Bild2: robustere Beschaltung

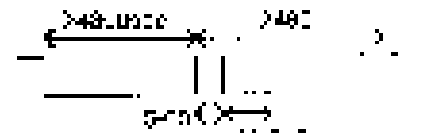


CPU abstimmen muß. Interrupts sind während dieser Befehle gegebenenfalls zu sperren. Das bedeutet eine Sperrung von 70 usec während eines Bits. Und nominell 1msec für den Reset, jedoch kommt man hier mit etwas komplizierterer Programmierung auf 80 usec herunter.

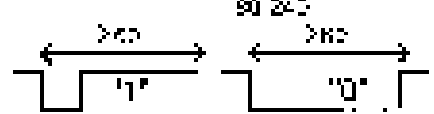
Der Reset des TM wird vom Controller durch einen langen Lowpuls ausgelöst ( Bild 3 ). Das TM antwortet mit einem kurzen Quittungspuls. Der Befehl RESET-TM liefert deshalb ein entsprechendes Flag zurück. Soll das Einstecken eines TM automatisch erkannt werden, muß der Controller die Leitung etwa alle 500msec mit einem

Bild3: Timing

Reset



Write Bit



Read Bit

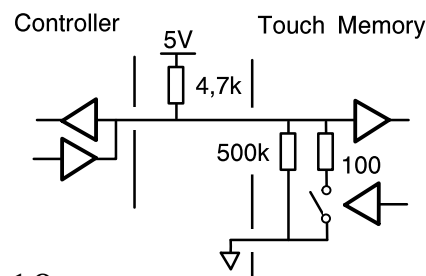


Bild1: simple Beschaltung

RESET-TM liefert und prüfen, ob eine Quittung kommt. Die Uhrzeitgleichzeit und durch einen kurzen Stampfstrom Controller kann gesehen. Zwischen den Bits können notwendig für die Berechnung existieren. Die von Frontfall herunter. Grundsätzlich ist ein einzelnes Bit ein Byte. Die von Frontfall herunter. Grundsätzlich ist ein einzelnes Bit ein Byte. Die von Frontfall herunter. Grundsätzlich ist ein einzelnes Bit ein Byte.

## Zugriff aufs ROM

Jede Operation beginnt mit einem Reset. Bei jedem Zugriff aufs ROM. Um einen manuell programmierten Teil (Serial ROM) oder auch über das I2C-Bus. Bei einem Zugriff aufs ROM. Um einen manuell programmierten Teil (Serial ROM) oder auch über das I2C-Bus. Bei einem Zugriff aufs ROM. Um einen manuell programmierten Teil (Serial ROM) oder auch über das I2C-Bus.

## Zugriff aufs RAM

Das RAM besteht aus 192 Bytes. Die ersten 16 Bytes sind für die Seriennummer reserviert. Die restlichen 176 Bytes sind für den Anwender verfügbar. Die ersten 16 Bytes sind für die Seriennummer reserviert. Die restlichen 176 Bytes sind für den Anwender verfügbar.





Um zu vermeiden, daß die Kommunikation bei Übertragung der Adressen nicht abgebrochen wird, muß in jedem Byte ein 1er- und ein 0er-Bit vorkommen. Dies wird durch „bit stuffing“ erreicht.

## Load

Das Listing DUMP zeigt das Ergebnis der Übertragung von 1000 Bytes. Die Übertragung wurde durch CTRL-C unterbrochen. Die Adresse 0000 bis 000F zeigt die Übertragung des Programms. Die Übertragung des Programms wurde durch CTRL-C unterbrochen. Die Übertragung des Programms wurde durch CTRL-C unterbrochen. Die Übertragung des Programms wurde durch CTRL-C unterbrochen.

## Motorola-S-Records

Type	Length	Address	Data	Checksum
31	4	0000	0000	0000
32	4	0004	0000	0000
33	4	0008	0000	0000

Das Listing zeigt die Übertragung des Programms. Die Übertragung des Programms wurde durch CTRL-C unterbrochen. Die Übertragung des Programms wurde durch CTRL-C unterbrochen. Die Übertragung des Programms wurde durch CTRL-C unterbrochen.

# HDLC-Rahmen in Software

„High Level Data Link Control“ ist ein sehr verbreitetes Format zur Übertragung von Datenpaketen. Es arbeitet auf synchronen Kanälen. Im Gegensatz zur asynchronen Übertragung, wie bei V24, sind hier keine Start- und Stopbits nötig, wodurch der Durchsatz höher liegt. Normalerweise findet die Auswertung von HDLC-Rahmen und CRC in einem Kommunikations-IC, wie z.B. der Z80 SIO, in Hardware statt. Bei hoher Übertragungsgeschwindigkeit und großen Datenmengen wird man immer auf solche ICs zurückgreifen müssen. Bei Anwendungen mit niedriger Datenrate, wie dem D-Kanal von ISDN mit 15kBit/sec, und ausreichender Rechenleistung des Controllers, kann man diese exotischen Bausteine vermeiden, indem man die Bits in Software verarbeitet.

könnte eine beliebig krumme Zahl von Bits lang sein. Es ist aber vorgeschrieben, daß die Zahl der Bits durch 8 teilbar ist, die kleinste Einheit sind also Bytes oder ein Vielfaches davon. Da das Ende durch ein Stopzeichen markiert wird, gibt für das Paket keine Längenbeschränkung. Meistens begrenzt man es auf maximal 4k Byte, weil die Sicherheit der Fehlererkennung durch die 16 Bit CRC sonst zusehr nachläßt. Außerdem ist die Wiederholung der Übertragung mit kürzeren Paketen effizienter.

## Flag

Start des Blocks ( Bild 1 ) ist das „Flag“, ein Byte mit dem Wert 01111110b. Es ist das heilige Zeichen des Protokolls und darf bis zum Ende des Blocks nicht mehr im Datenpaket auftreten. Das wird durch „bit stuffing“ zwangsweise sichergestellt. Bit stuffing bedeutet, daß wenn in den Daten mehr als 5 Einsen hintereinander vorkommen, der Sender nach der fünften Eins eine Null einfügt ( Bild 2 ). Der Empfänger entfernt dieses Bit wieder. Außerdem überprüft der Empfänger, ob 7 aufeinanderfolgende Einsen ankommen. Das ist offensichtlich illegal und wird in

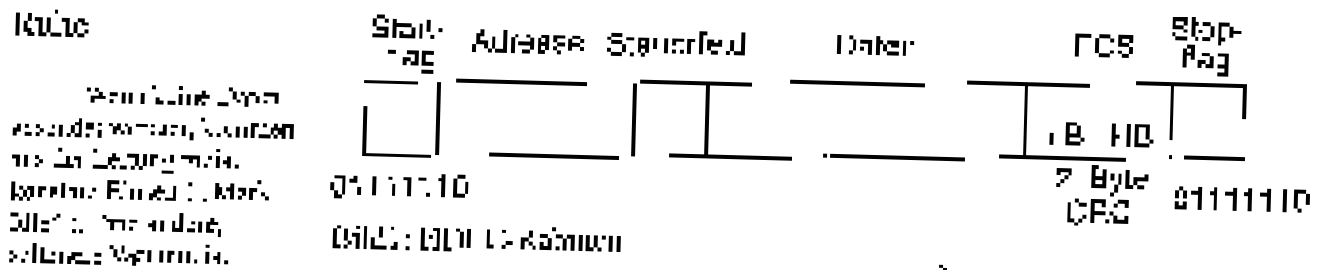
manchen Systemen als „abort sequence“ dazu verwendet, die Übertragung eines Pakets vorzeitig abbrechen. Das Ende-Flag eines Pakets kann gleichzeitig das Start-Flag des nächsten Blocks sein.

## Datenblock

Der Datenteil besteht nominell aus Adreß-, Steuer-, und Datenfeld. Deren Aufbau ist in den verschiedenen Anwendungsvarianten unterschiedlich. Weshalb dieser Teil auch nicht von den Hardwarecontrollern sondern der Software behandelt wird. Der Datenblock

## CRC

Die CRC gehört eigentlich in den Datenteil, wird aber traditionell vom Hardwarecontroller miterledigt. Weshalb man sie in der Praxis dem Rahmen zuordnet. Sie wird über alle Datenbytes, nicht aber das Start-Flag gebildet. Verwendet wird das CCITT-Polynom mit der Übertragung LSB zuerst. Eine genauere Beschreibung der Anomalien für HDLC findet sich in [1].



Wenn keine Daten zu senden, warten bis zur nächsten freien Einzel-1-Mark. Dann 1. 1-Mark, gefolgt von 10 Nullen.

Wenn 1-Mark Daten zu senden, wartet bis die 1-Mark 10 Nullen gefolgt sind. Dann 1-Mark, gefolgt von 10 Nullen.

**Code:**

System: Die Leitung hat in der Senderrichtung ein Data-Pol (ausgewiesen) und ein Data-Pol (ausgewiesen) und ein Data-Pol (ausgewiesen). Die Leitung hat in der Senderrichtung ein Data-Pol (ausgewiesen) und ein Data-Pol (ausgewiesen).

**Datensatz:**

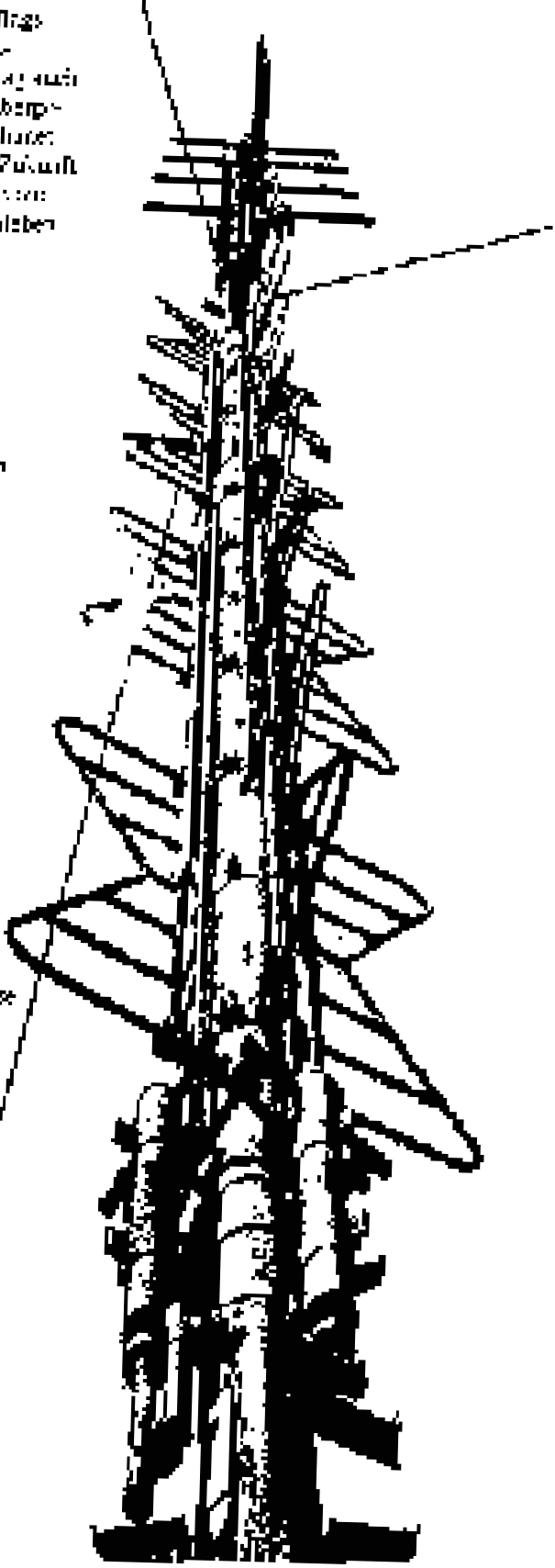
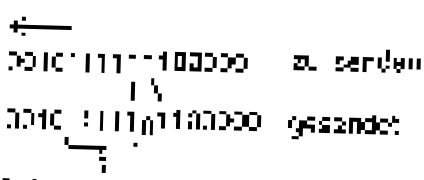
Die Daten sind in einem Datensatz (Data-Block) dargestellt. Die Daten sind in einem Datensatz (Data-Block) dargestellt.

Die Daten sind in einem Datensatz (Data-Block) dargestellt. Die Daten sind in einem Datensatz (Data-Block) dargestellt.

Die Daten sind in einem Datensatz (Data-Block) dargestellt. Die Daten sind in einem Datensatz (Data-Block) dargestellt.

Die Daten sind in einem Datensatz (Data-Block) dargestellt. Die Daten sind in einem Datensatz (Data-Block) dargestellt.

**Bit 2: Bit 2 (Flag)**



11. Antennenbau  
Final 200, S. 1-13

# Lineare Assemblerprogramme

In FORTHs die auf Einplatinencomputern laufen, sind meist Postfix-Assembler implementiert. Ihre unübliche Syntax ist nicht immer die Freude des Anwenders. Sie haben aber durch die enge Einbindung in die Programmiersprache auch Vorteile. Insbesondere vereinfachen sie das Schreiben von Programmgeneratoren. Mit diesen kann man einfach und übersichtlich linearen Code erzeugen. Dabei handelt es sich um ein sehr langes Assemblerprogramm das keine Verzweigungen hat. Seine hohe Geschwindigkeit und sein exaktes Timing prädestinieren es für Aufgaben in der Datenerfassung und Signalerzeugung. Dafür drei Beispiele.

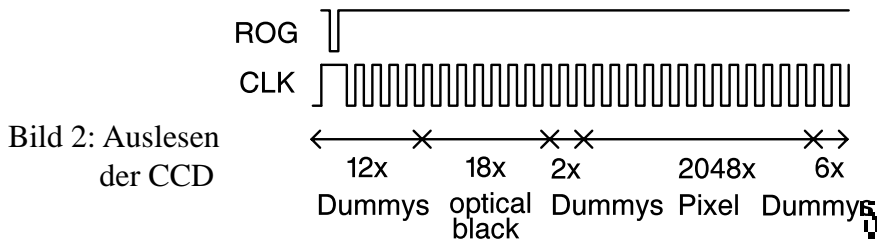


Bild 2: Auslesen der CCD

## CCD-Zeilensensor

Ein CCD-Zeilensensor (Bild 1) ist ein Bildaufnehmer wie er in Faxgeräten und Scannern verwendet wird. Hier dient als Beispiel der Typ ILX703 von Sony. Er enthält 2048 Sensoren von 14x14u Fläche. Sie arbeiten wie Integratoren und bauen abhängig von der einfallenden Lichtstärke Ladung auf. Die Belichtungszeit ist durch das externe Signal /SHUT von der Software steuerbar. Durch den externen Auslesepuls ROG werden die akkumulierten Ladungen in ein analoges Schieberegister, die eigentliche CCD, übernommen und dann seriell ausgelesen. Die Spannungen müssen dann extern von einem 8 Bit A/D-Wandler digitalisiert

und im RAM des Mikroprozessors abgelegt werden.

Dabei ergeben sich zwei Probleme. Erstens muß das Auslesen möglichst schnell erfolgen, da die Ladungen im Schieberegister rasch abklingen. Zweitens ist der Signalrahmen etwas unübersichtlich aufgebaut (Bild 2). Erst kommen 12 Dummy-Samples. Dann 18 Samples des Referenzsignals /

Das gesamte Programm zum Erzeugen des linearen Code ist in der Datei 'ccd.asm' zu finden. Es ist ein sehr langes Programm, das die gesamte CCD-Abtastung steuert. Die Belichtungszeit ist durch das Signal /SHUT steuerbar. Die Auslesepulse sind durch das Signal ROG steuerbar. Die Spannungen müssen dann extern von einem 8 Bit A/D-Wandler digitalisiert

und im RAM des Mikroprozessors abgelegt werden. Dabei ergeben sich zwei Probleme. Erstens muß das Auslesen möglichst schnell erfolgen, da die Ladungen im Schieberegister rasch abklingen. Zweitens ist der Signalrahmen etwas unübersichtlich aufgebaut (Bild 2). Erst kommen 12 Dummy-Samples. Dann 18 Samples des Referenzsignals /

Das gesamte Programm zum Erzeugen des linearen Code ist in der Datei 'ccd.asm' zu finden. Es ist ein sehr langes Programm, das die gesamte CCD-Abtastung steuert. Die Belichtungszeit ist durch das Signal /SHUT steuerbar. Die Auslesepulse sind durch das Signal ROG steuerbar. Die Spannungen müssen dann extern von einem 8 Bit A/D-Wandler digitalisiert

und im RAM des Mikroprozessors abgelegt werden. Dabei ergeben sich zwei Probleme. Erstens muß das Auslesen möglichst schnell erfolgen, da die Ladungen im Schieberegister rasch abklingen. Zweitens ist der Signalrahmen etwas unübersichtlich aufgebaut (Bild 2). Erst kommen 12 Dummy-Samples. Dann 18 Samples des Referenzsignals /

Das gesamte Programm zum Erzeugen des linearen Code ist in der Datei 'ccd.asm' zu finden. Es ist ein sehr langes Programm, das die gesamte CCD-Abtastung steuert. Die Belichtungszeit ist durch das Signal /SHUT steuerbar. Die Auslesepulse sind durch das Signal ROG steuerbar. Die Spannungen müssen dann extern von einem 8 Bit A/D-Wandler digitalisiert

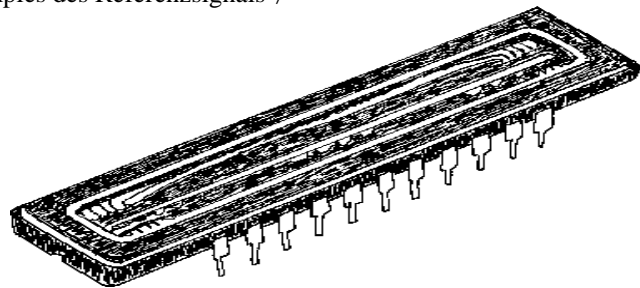


Bild 1: Prinzipschaltbild der CCD intern. 4 von 2048 Sensoren dargestellt

