

embedded



Voransichts- Version

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

②

Rafael Deliano
Steinbergstr.37
82110 Germering
Tel 089/8418317

j_r_d@t-online.de

V1.0 (Papier) 2. Dez 98
V2.0 (pdf) 22. Feb 01
V2.1 (pdf) 28. April 04
V2.2 (pdf) 14. Jan 07

READ . ME

Seit die ELRAD ihr Erscheinen eingestellt hat, gibt es in Deutschland leider keine Zeitschrift mehr die sich speziell mit Mikrocontrollern befaßt. In dieser Ausgabe sind nocheinmal alle meine in der ELRAD erschienen Beiträge zusammengefaßt.

Die Darstellung der Berechnung der CRC ist erweitert worden. Der in der ursprünglichen Ausgabe vorhandene Artikel zu Einstellregeln nach Ziegler & Nichols für PID-Regler enthielt diverse Fehler und erscheint deshalb hier nichtmehr. Zu Schrittmotoren und V24 folgen in den nächsten Heften weitere Beiträge.

Die Listings sind in nanoFORTH geschrieben. Für die Konvertierung in andere FORTH-Varianten finden sich Hinweise im nanoFORTH-Manual.

- 1 In
- Im
- 2 Ge
- 6 PID-Regler
- 10 Filter testen
- 11 Booth Multiplizierer
- 12 CCD-Zeilensensor
- 14 EEPROM am I2C-Bus
- 16 Multitasker WAIT
- 19 Schrittmotoren

Die Entwicklung der 8 Bit CPUs

Ein knapper Überblick, woher all die vielen Controller kommen. Die Darstellung kann nicht annähernd vollständig sein. Schwerpunkt liegt auf 8 Bit CPUs auf denen FORTH schonmal implementiert wurde. Einige wichtige Typen fehlen mangels Information. Besonders COP800 von NS, TMS1000 von TI und die Japaner, z.B. uPD78xx von NEC. Andererseits sind einige Typen die keine praktische Bedeutung hatten wegen ihrer historischen Wichtigkeit aufgeführt.

Am Markt liegen 4 Bit und 8 Bit Typen in Stückzahlen derzeit etwa gleichauf. 16/32-Bit-Controller und DSPs sind mit 10% - 20% noch relativ unbedeutend. Am stärksten wird das 8 Bit Segment wachsen, aber auch 16/32 Bit werden an Bedeutung gewinnen. Bei 4 Bit Typen liegen die Japaner vorn. Bei 8 Bit hält vermutlich Motorola (HC05, HC11) etwa 30% des Weltmarktes, und hat damit den 8051 verdrängt, der früher ähnliche Marktanteile hatte. Im 16/32 Bit Bereich versuchen viele neue Architekturen Fuß zu fassen.

Die Computer Terminal Corp. (CTC) beauftragte sowohl Intel wie auch Texas Instruments als second-source mit der Entwicklung des Chips. TI kündigte seinen Entwurf 1971 in einer Anzeige in einer Zeitschrift an, er wurde aber aufgrund technischer Probleme nie produziert. Nichtsdestotrotz beansprucht TI für sich den ersten 8-Bit Mikroprozessor entwickelt zu haben. CTC gab das Projekt schließlich auf.

Intel führte die Entwicklung jedoch weiter und produzierte den Chip schließlich für den freien Markt. Einige der ersten Bausätze für Hobbyisten haben ihn noch verwendet. Das IC kostete \$36.

Intel 8080

(1974 - 1977)
8 Bit Prozessor, NMOS
HLL: ja
Hersteller: Intel

Der erste herkömmliche 8 Bit Mikroprozessor. Für Anfangs 1974 und hatte 6.000 Transistoren, was für seine Leistung nicht zu halten konnte. Er war Standard in der Basisdaten für Betriebssysteme, aber auch die Bausätze der Heimcomputer waren.

Intel 8085

(1977 - ca. 1984)
8 Bit Prozessor, NMOS, CMOS
HLL: ja
Hersteller: Intel

Die Weiterentwicklung gegenüber dem 8080. Insofern verbessert sich Performance noch eine 5% Verbesserung bewerkstelligt. Das war allerdings die ersten 8 Bit Anwender davon anzubieten auf den 280 übergeben.

Intel 4004

(1971 — ?)
4 Bit Controller, NMOS,
2100 Transistoren
HLL: nein
Hersteller: Intel

Der erste Mikroprozessor. Im Auftrag der japanischen Tischrechnerfirma (Taschenrechner gabs damals noch nicht) Busicom von Intel entwickelt. Entworfen von Ted Hoff, realisiert von Fredrico Faggin und Masatoshi Shima. Der Chip war kommerziell aber nicht sehr erfolgreich, auch weil Intel sich auf komplexere Produkte konzentrierte.

Intel 8008

(1972 - 1974)
8 Bit Prozessor, NMOS
HLL: nein
Hersteller: Intel

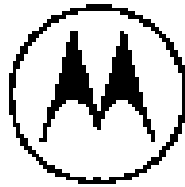
Bild1: Mit dieser Anzeige stellte Intel der Welt den ersten Mikroprozessor vor



Announcing a new era of integrated electronics

A micro-programmable computer on a chip!

intel delivers.



MOTOROLA

6800

1974 bis 1981
 8 Bit Prozessor, NMOS
 DIL-16
 Hersteller: Motorola, AMI, Toshiba, T-1011

Aus Motorola kam 1980 auf den Markt das erste Mikrocontroller 6800 der die heute Standard in elektronischen Geräten ist. Motorola kam Silicon-Battery-Struktur an den ersten achtbits Großunternehmen, besonders von der Billig-Lösung für kleine Industrie-Anwendungen eingeführt. Das wurde nicht die Ressourcen für eine CPU, sondern nur die Halbleitertechnik zu entwickeln. Der 8-Bit-Mikrocontroller 6800 besteht aus einem 8-Bit Registerdatei mit einem Adressregister und sein bestanden 18-Bit Busnummer. Beim Hersteller der 6800 8-Bit CPU war die CPU auch relativ neu. Sie war für kleine industrielle Anwendung mehr dem Zweck.

6809

1976-1981
 8 Bit Prozessor, NMOS
 16 Pin package
 Hersteller: Motorola, Texas AM

Die Entwicklung, die diese CPU Standard für die 8-Bit Homecomputer Generation wird bewährten sich nicht Sie nur an der Motorola entwickelte gleichzeitig der 6800. In schließlich im 1980 eingesetzt

wurde. Obwohl diese wird Motorola 8-Bit CPU sind Anwender sehr geschätzt wurde ihre Weiterentwicklung eingesetzt. Es diente der erste 8-Bit Mikrocontroller genannt sein, zu dem 6800 verfügbar wurde

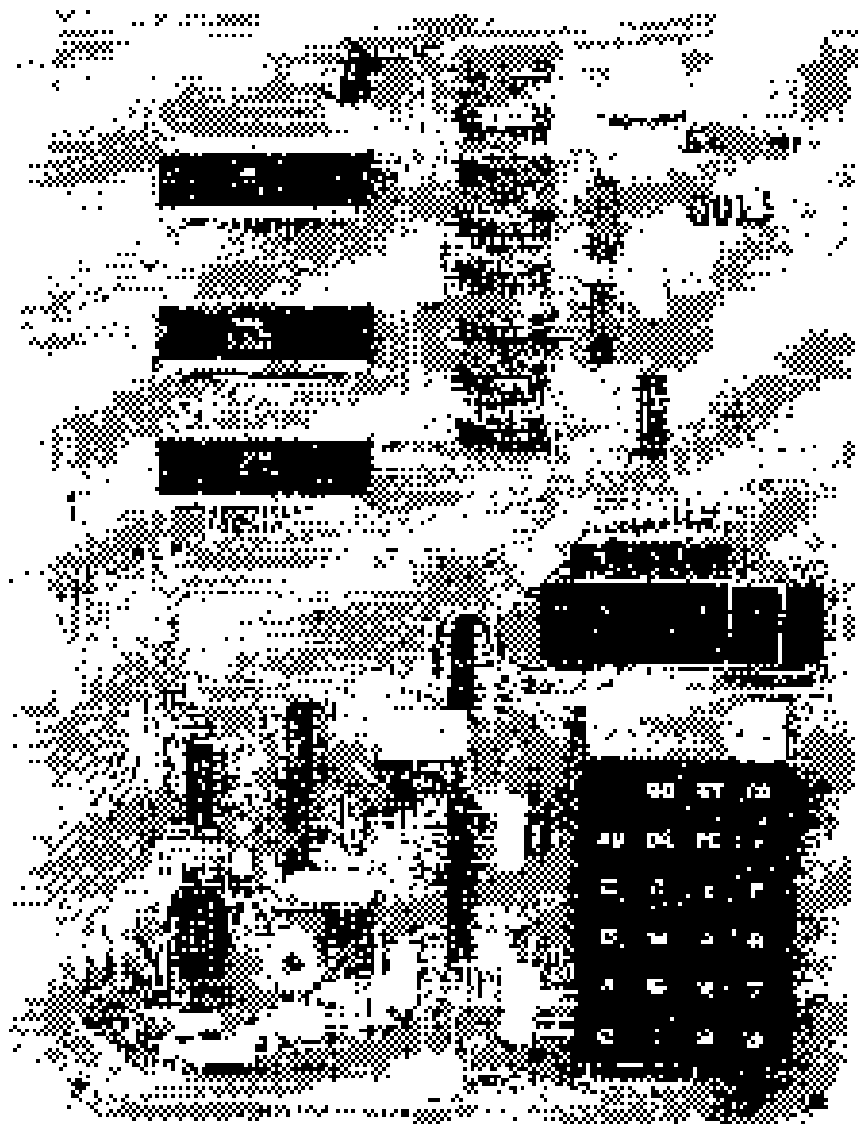
6805

1980
 8 Bit Controller, NMOS, CMOS
 DIP-16
 Hersteller: Motorola, RCA, Toshiba, Thomson
 Die damit verbundenen Mikrocontroller sind sehr klein, wenn auch in der Leistungsfähigkeit eingeschränkt, seine 8-Bit Architektur nur für einen 16-Bit-Register

68HC11

1984
 8 Bit Controller, CMOS
 DIP-16
 Hersteller: Motorola, Toshiba

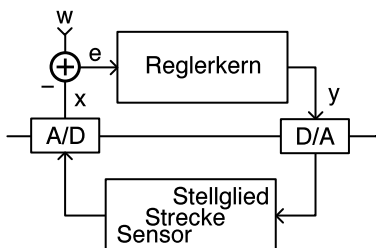
Das ist zweitens größte Nachfolger des 6800. Die zweite Architektur im den Vorgänger wurde durch Einführung eines zweiten 16-Bit-Registers und mehr 16-Bit-Register angeschlossen. Die CPU verfügt über Unterstützung von HILF. Es gibt in Teil und Leistung über ein kleiner Ende des 8-Bit-Bus.



PID-Regler auf Controller

Viele Mikrocontroller sind heute mit Multiplikationsbefehl sowie A/D- und D/A-Wandler ausgestattet. Damit kann man mit ihnen preiswert digitale Regler implementieren. Deren Dynamik und Geschwindigkeit ist zwar einem analogen Regler unterlegen. Man kann jedoch beim digitalen Regler sehr einfach die PID-Koeffizienten verändern und damit eine für den jeweiligen Betriebszustand optimale Reglercharakteristik nachladen. Dadurch ist ein digitaler Regler einem festeingestellten analogen Regler in manchen Anwendungen überlegen.

Bild 1: Blockschaltbild Regelkreis



Regler

Der digitale Regler (Bild 1) ist über A/D und D/A-Wandler mit seiner analogen Umwelt verbunden. Dort befindet sich die Regelstrecke (z.B. DC-Motor), die Aufbereitung des Stellsignals y (Leistungs-OP) und ein Sensor zur Bestimmung des Istwerts x (Tacho). Der Sollwert w ist eine Zahl die vom Hauptprogramm im RAM übergeben wird und damit variabel ist. Die eigentliche Reglersoftware läuft im Interrupt. Der Interrupt wird von einem freilaufenden Timer ausgelöst, der eine konstante Abtastfrequenz sicherstellt. Das Interruptprogramm liest den A/D-Wandler aus. Dann subtrahiert es diesen Wert von w und erzeugt damit das Fehlersignal e . Die Größe von e ist proportional zur Regelabweichung und die Polarität ist der Regelabweichung entgegengesetzt. Anhand von e be-

Bild 3: PID-Kern

Parallelimplementierung

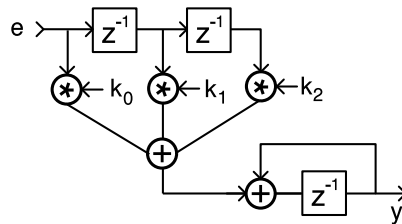
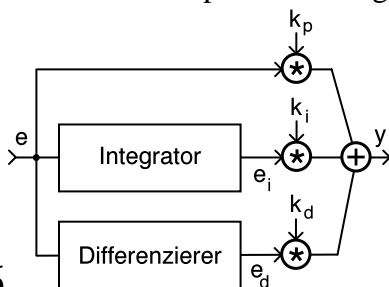


Bild 2: rekursiver Reglerkern

stimmt der Reglerkern den Stellwert y und schreibt diesen Wert in den D/A-Wandler. Damit endet der Interrupt.

PID-Kern

Neben der direkten Implementierung, die hier näher behandelt werden soll, ist auch die rekursive Form verbreitet. Ein Variante davon ist in Bild 2 dargestellt. In dieser Form wurde die Reglerfunktion in ein FIR-Filter umgewandelt. Das vermindert zwar die Zahl der benötigten Operationen. Leider sind dann aber die Reglerkoeffizienten nicht mehr unabhängig voneinander einstellbar. Das ist in der Praxis ein nicht zu unterschätzender Nachteil. Die Koeffizienten müssen normalerweise experimentell durch Versuch an der Regelstrecke bestimmt werden. Und dafür sollten sie getrennt veränderbar sein.

Direkte Implementierung

Üblich in Software ist die parallele Anordnung der drei Signalzweige (Bild 3). Man bewertet das Fehlersignal e mit den drei Anteilen und summiert das Ergebnis zum Stellsignal y . Beim P-Anteil genügt die Multiplikation mit dem Regler

koeffizienten k_p . Beim I- und D-Anteil muß das Signal vorher einen Integrator oder Differenzierer durchlaufen. Die drei Regleranteile können parallel die drei Anteile der gegebenen Reglercharakteristik realisieren. Die Charakteristik die für ein bestimmtes Regelobjekt am besten geeignet ist, kann man als Konstanten in einem Mikrocontroller speichern. Wenn die das Hauptprogramm im RAM überläßt, das Hauptprogramm kann ohne Anstoß vom Benutzer zum nächsten Schritt die Reglercharakteristik nachladen.

Das in den Anfangskapitel dargestellte 2. Schaltungsbeispiel bedeutet „Verzögerung um ein Sample“. Diese Funktion wird durch eine Verzögerung erreicht, aber keine Reduktion. Wenn werden Addition und Subtraktion benötigt, die über aus allen Operationen effizient verbunden sind. Bei Multiplikation, die einen problematischen ist. Multiplikation mit einer Konstante, wie sie in der Regelabrechnung und Differenzierung gefunden ist, kann auf Schritt zurückgeführt werden. Z.B. bei der Wert $0,5$ und x für x wie $1,5$ kann x einen Schritt und eine Addition reduziert werden.

Einfacher Integrator

Der Integrator summiert die Energie des Fehlers über die Vorgangzeit. Die diskretisierte Version einer kontinuierlichen Integrator. Die diskrete Sample e und die Abtastzeit T bilden eine Produkt $T \cdot e$, das die Energie im Sample umschließt. Da die Abtastzeit T konstant ist, kann die Wert T weggelassen werden und die Multiplikation in der Regel entfällt. Es genügt also das Fehlersignal e zu summieren wie im Blockdiagramm dargestellt. Diese einfache Integration enthält das 3. Problem, das so groß ist. Der Fehler kann jedoch reduziert werden, wenn man die Abtastfrequenz erhöht, also die Abtastzeit T verringert. Die Grenzfrequenz der Regelstrecke zählt.

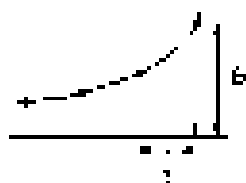
Verbessertes Integrator

Das T-Integrator (Bild 4) verbessert auch nur den Wert T .

$$\theta_1 = \theta_0 + (T \cdot \theta)$$



Bild 4:
Einfacher
Integrator

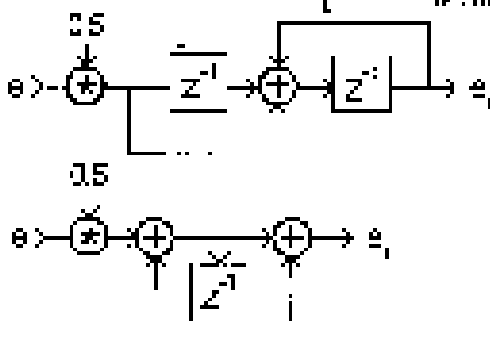
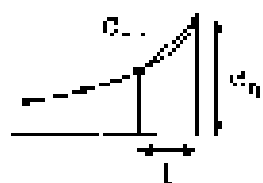


weisen Sinus mit ω ändert aus der Wert des in dem unmittelbar vorherigen Taktperiode. Das Problem ist zwar kompliziert, zu keine weitere Multiplikation benötigt wenn statt der Delta zu Berechnen nur die Differenz $\theta_1 - \theta_0$ welche die Taktperiode eine gute Näherung zum Rechteckvergnome ist. Neben der einfachen Implementierung ist die numerische Instabilität gegeben, die weniger stabilisiert wird.

In diesem Fall befindet sich der Ausgang θ_1 im Gegensatz zu dem Eingang θ im Bereich der Nullstelle. Dieses führt zu Sprüngen in der PD-Bewertung. In dem Fall, Normalisiert sich der Sprungsummand, verhält sich der Ausgang wieder. Dabei entstehen über längere Schaltzeiten kleine Amplituden. Deshalb muß man die Leistung des Integrators anpassen, solange sich θ in ständig belastes. Der Integrator soll eigentlich nur die Integration vornehmen, den der D Anteil nicht instabil kann. Man kann deshalb mal überlegen, es gibt ein mit dem Integrator genau stabilisiert, solange es keine sich der Bereich über der 5. Ebene erreicht. In der Praxis

$$\theta_n = \theta_{n-1} \cdot \left(1 - \frac{T \cdot \theta_{n-1}}{2}\right)$$

Bild 5:
Trapez-
Integrator



signifikantes Verhalten mit guter Regelleistung. Es handelt sich dann über den Wert um einen linearen Bezug

Einfacher Differenzierer

Bei der Berechnung $\theta_1 = \theta_1 - \theta_0$ soll die diskrete zeitliche Änderung des Fehlersignals bestimmt. Das heißt man normalisiert die Distanz zu den Zeitpunkten und die Sprunkebene muss in der Zeit nicht sein, weil die Wert der vorkünftigen Samples noch nicht bekannt sind. Stattdessen schenkt man die Tangente durch den Punkt an die man durch die beiden nächsten bekannten Samples aussteigt. Die Tangente entspricht konstant ist, kann man auch hier, den Wert θ_1 aus dem Wert θ_0 ableiten. In der Praxis ist die Fehler zu Fall. Auch der Fehler des Differenzierers wird eliminiert, wenn man eine möglichst hohe Abtastfrequenz wählt.

Wenn kein Filter gegeben oder Sprünge springerig vorhanden regiert der D-Glied mit einem kleinen Impuls hoher Amplitude. Bei Regelern mit niedriger Dynamik ($\omega < 1$ besonders 8 Bit D/A-Wandler) wird der Impuls nicht so klein, weil er gekoppelt wird (Bild 9). Ein Filterprogramm kann den gekoppelten Anteil des Pulses auf die folgenden Takte verteilen.

Verbessertes

Differenzierer

Man kann die Wertänderung des Pulses nicht durch eine IFF-Anforderung erreichen (Bild 7). Der Vergleich der Sprungantworten in der Zeitbereich zeigt die Änderung der Funktion.

Der einfache Differenzierer hat zusätzlich den Nachteil, daß bei hochfrequenten Rauschen das dem Sensorsignal überlagert ist, stark auf das Sinus signal einwirken. Der IFF begegnet dem durch Tiefpassfilterung. Ein anderer, aufwendige Weg besteht darin, die Integration des Delta als zwei Samples zu behandeln. Ein

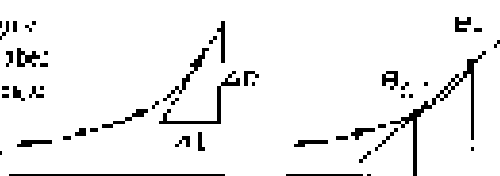


Bild 6:
Einfacher
Differenzierer

$$\theta_1 = \frac{\theta_1 - \theta_0}{T}$$

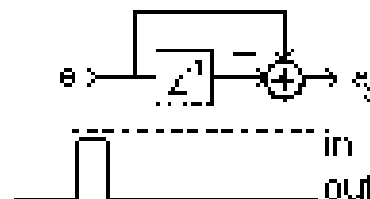


Bild 7: Vereinfachtes PD-Filter
mit IFF

Abmagerung

Abmagerung von der Regelstrecke kann nur bei Systemen des Typs I, II, III auf PD- und PD-Regler reduziert werden. Damit benötigt man weniger Regelzeit. Die Abmagerung kann durch eine Reduzierung der Regelzeit zu einer Verbesserung der Regelzeit.

Implementierung

Als Software-Beispiel als Beispiel: Lösung PID-Filter soll als Mittelwert-Filter mit einem 8 Bit A/D-Wandler mit 50 mV Auflösung, einer 5 Bit RZ-2V/A-Wandler, und einem 12 Bit PWM-Regler. Ein Multiplikationsbefehl der zwei Bytes zu einem 16 Bit-Ergebnis verarbeitet. Bei 8 Bit-Wert ist, zum Delta-Temporal

Lösung zeigt das geschätzte Fließglied mit der Wert, auf die Eingangsweite θ in Abhängigkeit

Bild 8: DDT-Definition

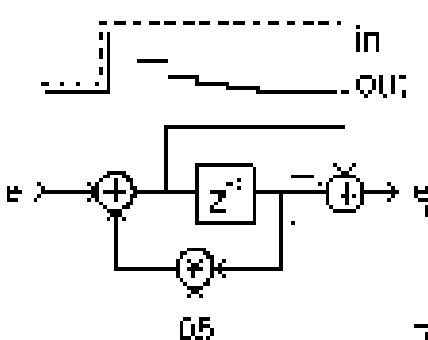


Bild 8: FIR-Filter als Differenzierendes

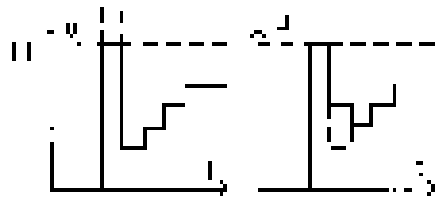
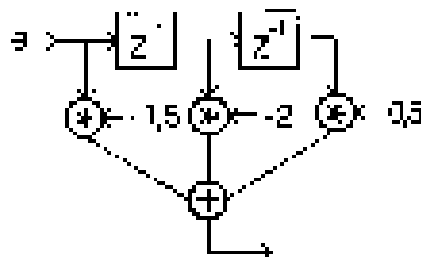


Bild 9

Speicher für geklappte Spitzer

Bei der Auswertung 16 Bit Auflösung, verwendet man den 20MHz

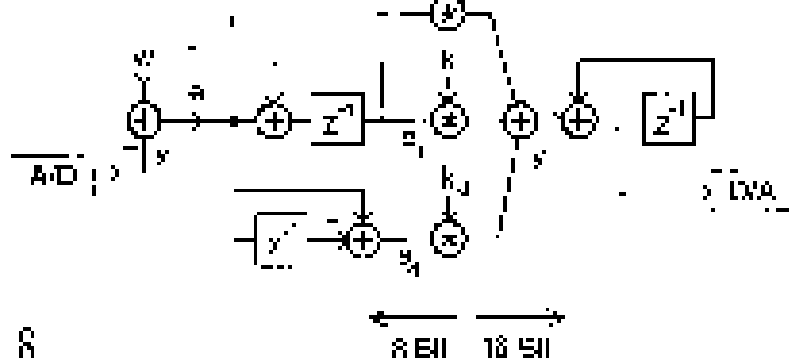
wenden alle 16 Bit zu speichern. Für den FIR-Filter werden die Koeffizienten der Impulsantwort gespeichert. Die Koeffizienten werden in die Speicher mit 16 Bit Auflösung. Damit wird die Auflösung verbessert. Um die Filterleistung des Reglers zu verbessern, wird das Signal immer noch nach der Zeitumkehr gegeben.

Kern

Die drei PLL-Kostenwerte und werden als Summe der 8 Bit Zahlen gegeben. Auch der PLL-Kern der PLL-Kern wird als der PLL-Kern gegeben. Die PLL-Kern wird als der PLL-Kern gegeben.

Das Filterband zu begrenzen, benötigt ein System. Die PLL-Kern wird als der PLL-Kern gegeben. Die PLL-Kern wird als der PLL-Kern gegeben.

Bild 10: Input-Verstärkung Beispiel



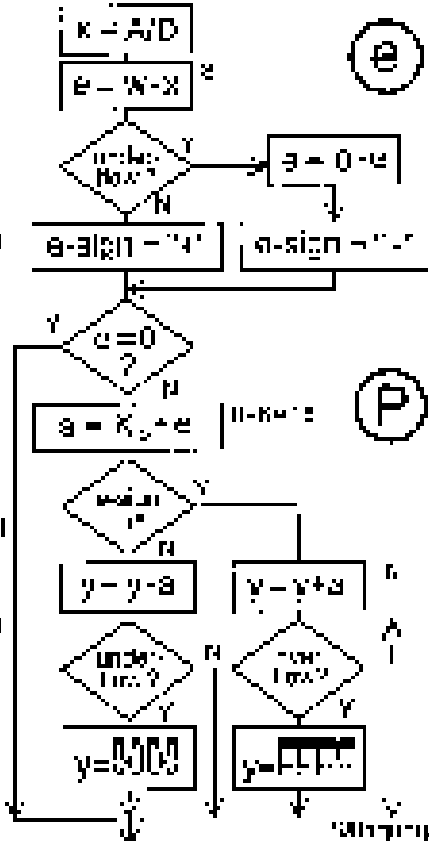
Bei der Verzögerung von einem in einem digitalen Filter abgelesenen Wert, kann man ein Bit zu einem und beschleunigt die Berechnung von Kern. Nach der Berechnung von e durch die Subtraktion von w und x (Bild 11) liegt allerdings bei negativen Werten die Zweikomponente von der man die den Schrittkern von Null macht, bei der Ableitung unberücksichtigt.

Im zweiten Abschnitt wird der Zustand beschreiben, wobei die Regelabweichung e Null ist, wenn man die Berechnung überträgt, um Regeln mit zu geben. Sowohl, für die auch der Betrag von e sind verschiedene Zahlen und können durch zwei Multiplikationsfaktoren (CFR) verändert werden. Danach wird die Area 15 Bit mit der Regelabweichung Addition bzw. Subtraktion kann eventuell ein Modulo

bedeutet ein Wert. Bei Regeln mit den diesen PLL eine Stützpunkt charakteristik vorgegeben, wenn man einen Kern, dann wird das Ausgangsden Wert auf null gesetzt. Das Ergebnis durch den PLL-Kern (PLL) berechnet, das den Wert Y wird zum Wert Y Wert gegeben wird, anstatt der Ausgangsintegrator

Der dritte Abschnitt hilft den PLL-Integrator aus. Hier wird über PLL, so das letzte Ausgangsregeln von PLL-Kern FF oder 01 war. In diesem Fall wird die Kernwertung des Integrators überprüften und sein Wert zum Eingabe. Die Möglichkeit zu vermeiden, ist gegeben, wenn man direkt durch das PLL-Befehl zu erfahren, wenn man die Verzögerung, diese Werte von 01 und 01 zu geben.

Der PLL-Integrator in einem Abschnitt wird ähnlich



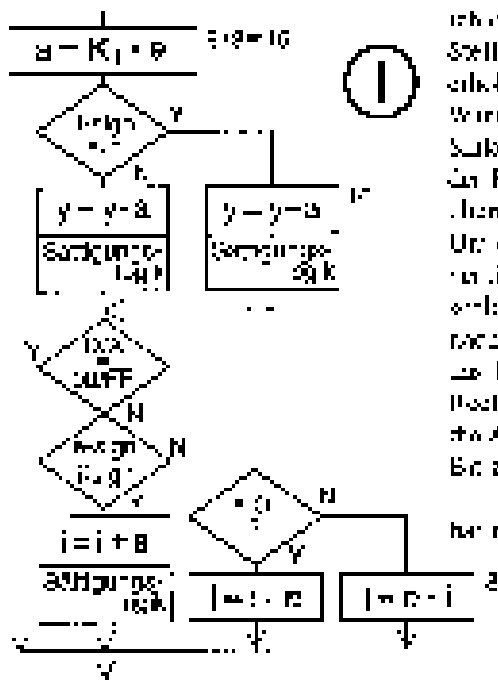
ausgegeben. Als Ergebnis wird das letzte Ergebnis von Y auf dem PLL-Kern wieder ausgegeben.

Das Systemprogramm wird basierend auf dem Mikrochip 6802 (1992) 440 Hz. Die Berechnung wird typisch. Wenn bei 1.5 MHz Bus-Auflösung. Damit kann man die Abtastzeit auf eine 1 kHz gegeben, was für zwei Berechnungen des Regelprogramms. Die PLL-Kernwertung der Regelregeln sollte zwischen 100 bis 1000 Hz gegeben.

Probleme

Wenn man die Kostenwerte gegeben, kann der Regelregeln geringen. Normalerweise kann man die anhaltende Phase des PLL-Kernwertung von der Phase des PLL-Kernwertung. Das wenn die Signalfrequenz sich um den PLL-Kernwertung, entspricht die Phase des PLL-Kernwertung. Selbstverständlich, ist es ein Problem, wenn man das PLL-Kernwertung, ist es ein Problem, wenn man das PLL-Kernwertung.

Die in Controller vorhandene PLL-Kernwertung für diese Zwecke



Wenn der Regler bei den Stellern am Ausgang geringfügig aussteigt, merkt ihn der 1. Ableiter. Wundert kein Steigen, sondern ein Sinken des Prozesswertes. Damit wird der Regler diese Verweirde nicht überpassen, wenn er langsam regelt. Um ein dynamisches Verhalten zu erreichen, sind bei Prozesswerten schnelleren Wänders sinnvoll sein. Nach dem Ausstrom des 2. Ableiters (z.B. 0) des Differenzwertes (denn ein Prozesswert parallel zu einem Wert die Auflösung dem Prozesswert auf 7 bis zu beschleunigen.

Auch ein Wert 1 Die Wandler hat nur 48M Dynamik und bei einer

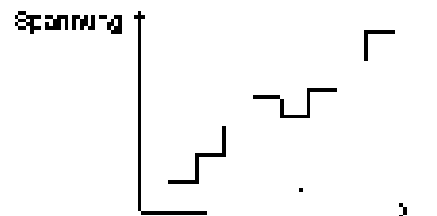
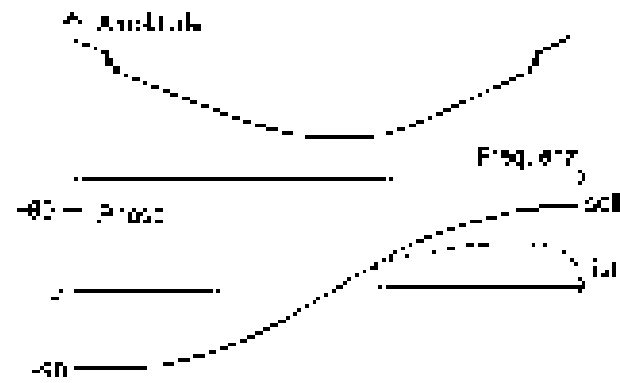
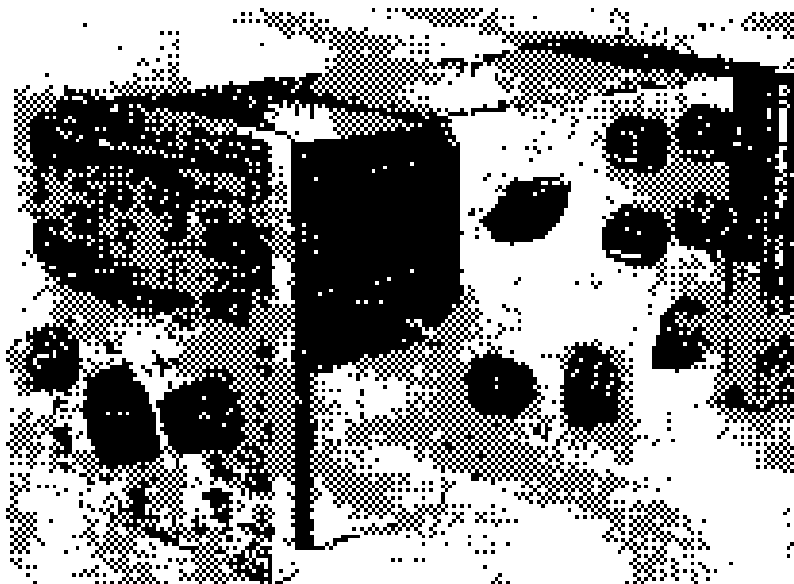
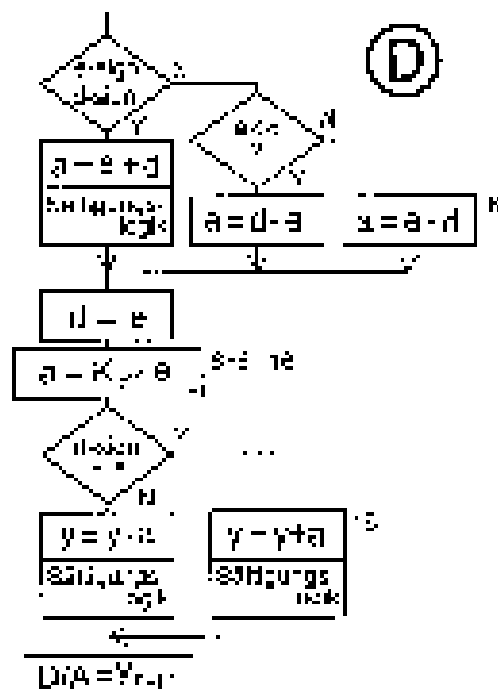


Bild 12: nichtstationäre Wandler
 Prozessspannung von 10V bei 10V die
 Schaltung mit einer 20mV. Nach
 10ms (ca. 10ms) kann nach 10ms
 1 Anteil des Reglers (mit
 Kompensiert werden. Aber zum
 Prozess Regelstrecke und dessen
 sich nur 10ms (10ms) beschleunigen,
 welche der Regler in der
 Praxis Kompensiert.

Bild 13: Frequenzgang eines PD-Regler



200g Kraft (ca. 100 N) zu bewirkt.
 Regelzeit 200ms. Sollte man die
 Grenzfrequenz des Eingriffsgerätes
 und nicht die Frequenz des Regelwertes.
 Die Wandler haben sich nicht mehr 100
 100ms (ca. 100ms) zu beschleunigen
 würde, sondern manchmal nur 10ms
 10ms. Im letzten Fall ist die Kontrolle
 der Wandler wahrscheinlich auch
 möglich, wenn der Wandler nicht
 tritt besonders bei Wandler die mit
 100ms (ca. 100ms) zu beschleunigen sind
 häufig auf eine nicht maximale
 beschleunigen. Bild 12 (folgt) bei Regel
 Zeit zu beschleunigen (ca. 10ms).



Digitale Filter testen

Auf einem Interpreter kann man das prinzipielle Verhalten der Teilkomponenten des digitalen PID-Reglers sehr leicht testen. Vorausgesetzt man verzichtet auf Grafik und Komfort, ist die benötigte Software sehr kompakt.

Syntax

Wesentlich fürs interaktive Arbeiten ist eine einfache Syntax. Um mit der Software im Listing FTST das FIR-Filter mit einem Puls anzuregen gibt man ein:

```
PULSE ` FIR RUN
```

Der Fliegendreck vor FIR ist ein „Tick“. Der Befehl bestimmt die Startadresse von FIR. Die eigentliche Ausführung des Befehls FIR erfolgt in RUN durch EXECUTE. PULSE muß in dieser Form die sequentiell erzeugten Werte ersteinmal irgendwo abspeichern. Da hier nur 10 Samples erzeugt werden, legt es sie einfach in umgekehrter Reihenfolge auf den Stack. Die Teilkomponenten haben alle einen Eingang und einen Aus-

gang. Die Eingangsdaten werden deshalb vom Stack übernommen, bearbeitet und auf dem Stack weitergegeben.

Datenformat

In der Signalverarbeitung arbeitet man mit einem Wertebereich der von +1 bis -1 reicht. Der Wert +1 wird hier auf 1000d skaliert. Dieser dezimale, an 100% angelehnte Wert ist gut fürs Vorstellungsvermögen. Da man noch weit von den Grenzen der 16 Bit Arithmetik entfernt ist, sieht man sofort den Überlauf, ohne daß dieser tatsächlich bereits eintritt.

Die meisten Teilkomponenten haben interne Speicher, hier als Z1 und Z2 bezeichnet. Diese müssen zu Beginn der Simulation auf Null initialisiert werden.

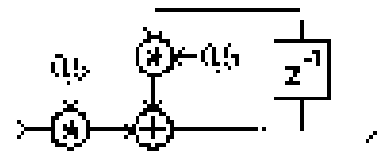


Bild 1: einfacher Differenzier

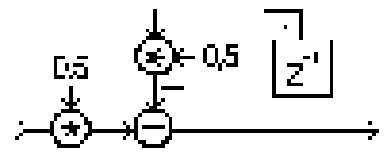


Bild 2: einfacher Integrator

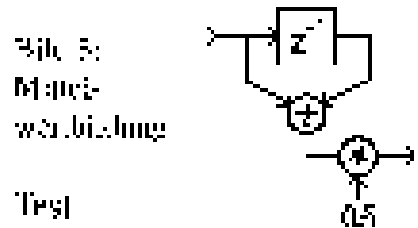
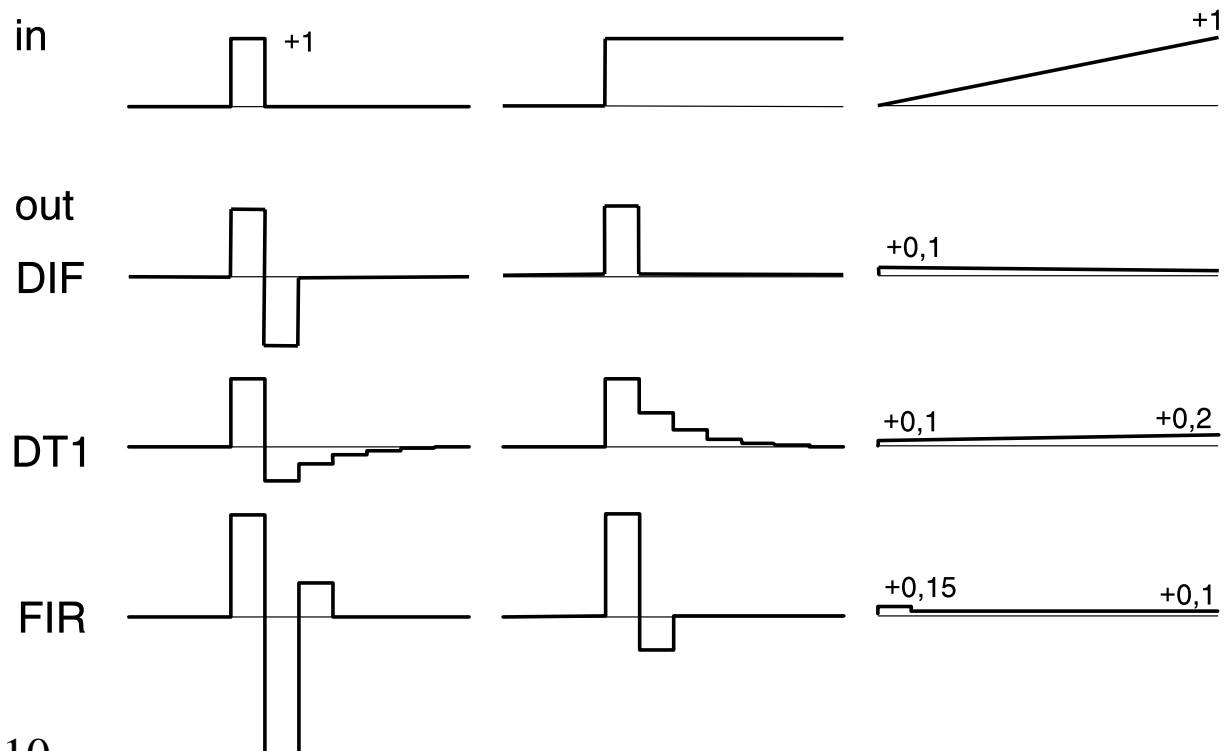


Bild 3: diskontinuierliche Integration

Für Integration ist neben FIR Filtern eine andere Möglichkeit: Die verschiedenen Z1'se an einer Stelle sind viel bei diesen ersten Punkt kann unterschiedliche Differenzier werden analoger Weise getestet.

Können die Funktionen für PLI getestet werden auch noch ein analoges Differenzier (Bild 1), Integrator (Bild 2) und ein diskontinuierliche (Bild 3) im Listing aussuchen:

Bild 1: 3 verschiedene Differenzierer getestet mit Puls, Sprung und Rampe



Booth-Multiplizierer

Die übliche Shift&Add Multiplikation kann nur auf vorzeichenlose Zahlen angewendet werden. Für 2er-Komplementarithmetik sind für sie deshalb umständliche Umwandlungen vor und nach der Operation nötig. Der Booth-Algorithmus verarbeitet 2er-Komplement direkt. Mit eingeschränkter Wortlänge eignet er sich auch für vorzeichenlose Zahlen.

Algorithmus

In Bild 1 ist das vereinfachte Flußdiagramm dargestellt. Bild 2 zeigt als Beispiel dazu die ersten Schritte einer Multiplikation von zwei 8 Bit Zahlen. Das Verfahren testet im Multiplikator Y von rechts nach links die Bitübergänge. Bei einem Übergang von 1 nach 0 wird der Multiplikand zum Produkt addiert. Bei einem Wechsel von 0 nach 1 wird er vom Produkt subtrahiert. In den beiden anderen Fällen ist keine Operation nötig. Anschließend wird in allen vier Fällen aus dem Produkt durch einen Arithmetik Shift Right ein Bit herausgeschoben.

Zur Initialisierung muß man das Produkt löschen. Für den ersten Bitübergang wird angenommen, daß das vorhergehende, fiktive Bit Null war. Das Ergebnis der Multiplikation von zwei 8 Bit Zahlen ist 16 Bit breit. Aber die beiden obersten Bit sind eigentlich nur Vorzeichen, denn man hat 7 Bit Daten multipliziert und kann deshalb nur 14 Bit Daten im Ergebnis haben.

Listing BOOTH1.F74 zeigt eine Implementierung für den 6502 die aus zwei 16 Bit Zahlen ein 32 Bit Ergebnis erzeugt. Die Daten werden dabei vom Stack übernommen und das Ergebnis liegt direkt auf dem Stack. Das vorhergehende Bit für die Prüfung des Bitübergangs wird in einer Bytevariable zwischengespeichert. Das neue Bit des Übergangs wird laufend durch einen Rechtsshift von Y in die Position von Bit 0 gebracht. Das Programm ist für Eingangswerte von +32767 bis -32767 verwendbar. Bei $X = -32768$ tritt jedoch interner Carryüberlauf bei Addition und Subtraktion auf und das Ergebnis wird verfälscht.

Multiplikation für 2er-Komplement kann man offensichtlich auch für vorzeichenlose Zahlen verwenden, wenn deren Wortlänge um ein Bit verkürzt angenommen wird. In diesem Fall also 15 Bit. Durch Erweiterung des Algorithmus auf 17 Bit kann man somit Varianten erzeugen, die entweder 2er-Komplement inklusive -32768 verarbeiten (Listing BOOTH2.F74) oder 16 Bit vorzeichenlose Zahlen multiplizieren (Listing BOOTH3.F74) können. Leider wird das Programm durch die internen 24 Bit Additionen und Subtraktionen deutlich langsamer.

Geschwindigkeit

Die Laufzeit ist offensichtlich abhängig vom Bitmuster des Multiplikators. Die ungünstigste Situation ist, wenn die Bits in Y laufend wechseln, z.B. bei AAAA. Optimal sind hingegen Werte wie FFFF. Für die 16 Bit Version in Listing 1 ergibt sich auf einem 2,5 MHz Mitsubishi-6502 eine Laufzeit von 500 - 700 usec. Die 17 Bit Version in Listing 2 und 3 braucht 600 - 900 usec. Der konventionelle Shift& Add-Algorithmus für vorzeichenlose Zahlen liegt bei 250 - 450 usec. Wenn man bei 2er-Komplement auf -32768 verzichten kann, ist der 16 Bit Booth-Algorithmus also konkurrenzfähig.

Größere Bitgruppen

Die hier beschriebene, grundlegende 1 Bit Variante betrachtet den Übergang zwischen 2 Bits. Es gibt auch 2 und 3 Bit Booth-Versionen die größere Bitgruppen auswerten um die Effizienz zu steigern. Diese werden

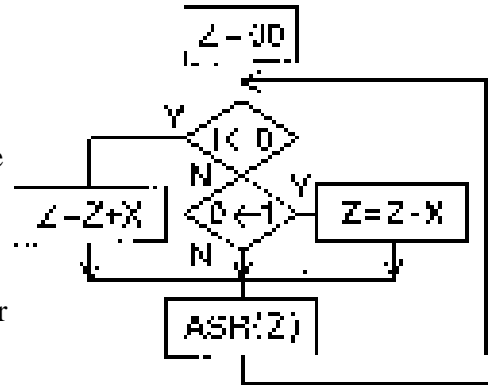


Bild 1: vereinfachtes Flußdiagramm

$$X \cdot Y = Z$$

$$-8 = 5 + -6$$

	X	-111101	2
	Y	1111011(0)	-5
1 ← 0	Z	0000000	
	-X	1111101	+(-8)
		0000011	3
	ASH	00000011	
1 ← 1	ASH	000000011	
0 ← 1	Z	00000003	
	-X	1111101	+(-8)
		11111011	
	ASH	111110011	
0 ← 1	Z	1111110	-2
	-X	1111101	+(-8)
		000000111	
	ASH	0000000111	

Bild 2: Die ersten 4 Teilschritte eines vereinfachten Booth

Verfahren-Algorithmus beschreiben. Eine generelle Darstellung findet sich in [2]. Bei Anwendung ist jedoch wegen der maximal möglichen Komplexität der Hardware vorzuziehen. Booth-Verfahren ist eine gute Alternative, die eine gewisse Vergrößerung der Bitlänge gegen sich haben kann.

- [1] Booth, R. A. Sign-Extended Binary Multiplication. *IBM Journal of Research and Development*, 1951.
- [2] Clock, George. *Designing and Building High-Speed Digital Apparatus*. McGraw-Hill, New York, 1959.

CCD-Zeilensensor

Im letzten Heft wurde diese Anwendung als Beispiel für die Nützlichkeit von linearen Programmen vorgestellt. Die Hardware wurde dabei allerdings etwas sehr knapp nur als Blockschaltbild abgehandelt. Das hat zu Rückfragen von interessierten Lesern geführt. Deshalb die Schaltung hier noch einmal detaillierter dargestellt wird.

Die Besonderheit am verwendeten Zeilensensor Sony ILX703 ist, daß er einen Shutter hat. Wie in Bild 1 dargestellt kann der Controller dadurch die Sensoren löschen, dann die Belichtung durch eine Verzögerungsschleife steuern, schließlich den Sensor auslesen. Bei üblichen CCD-Bildaufnehmern erfolgt das Löschen mit dem Auslesen, so daß man mit variabler und meist recht hoher Geschwindigkeit lesen muß. Der ILX703 ist andererseits etwas teurer (ca. 70DM) und nicht so gut lieferbar wie die Versionen ohne Shutter. Die Einsparung in der Beschaltung und die bessere Einstellbarkeit der Belichtung wiegt diesen Nachteil jedoch auf.

CCD

Der Bildaufnehmer (Bild 2) ist relativ einfach zu ansteuern,

weil seine Eingänge bereits 5V CMOS kompatibel sind. Ferner benötigt seine 9V Versorgung weniger als 15mA, so daß notfalls auch Versorgung aus 5V mit DC/DC- Wandler möglich ist. Hier wird das Vorhandensein einer Versorgungsspannung von etwa 11 - 24V angenommen, aus der der LM358 die 9V macht. Diese Spannung muß immer größer als die Spannung an 5V sein, sonst kann der Sensor zerstört werden. Der LM358 verwendet deshalb die 5V als Referenz. Dadurch werden die 9V auch im Powerup und Powerdown immer proportional größer als die 5V erzeugt. Zusätzlich ist eine dicke Schottkydiode SB120 vorgesehen. Kleinsignal-Schottkys bieten keinen Schutz, weil an ihnen bei einigen 10mA bereits 0,8V abfallen.

Die analoge Ausgangsspannung der CCD hat etwa 600 Ohm Ausgangsimpedanz und schwankt zwischen 5,5V (dunkel) und 0V (hell).

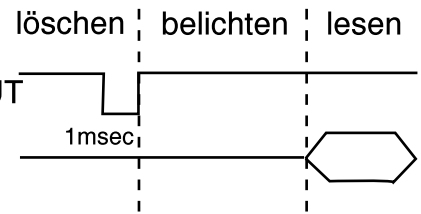


Bild 1: Ablauf einer Messung

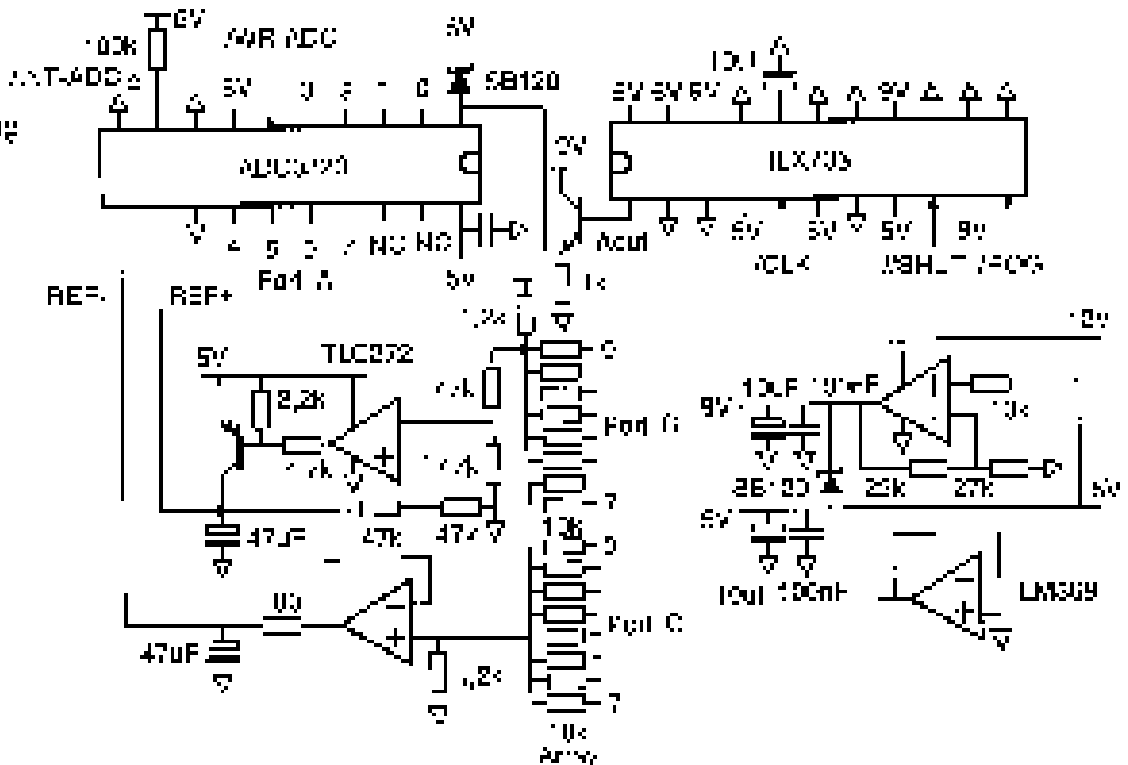
Der folgende Transistorimpedanzwandler verbessert den Ausgangswiderstand und schiebt den Pegel etwas nach unten. Das ist als Anpassung an den folgenden A/D- Wandler nötig, da dieser nur 0 - 5V digitalisieren kann.

Der 10uF Elko an der CCD muß eine Tantalperle sein, denn er ist der Speicher der internen Sample&Hold. Auch die übrigen Elkos der Schaltung sollten Tantal sein. Im Layout ist die Länge der Verbindung des Analogsignals von CCD zu Wandler minimieren. Auch die Versorgungsspannungen von CCD und Logik sollten sauber getrennt sein.

ADC

Als A/D-Wandler wurde der ADC0820 von National Semiconductor verwendet. Dieser schon recht angejahrte 8 Bit Halflash ist billig (30DM) und schnell genug. Es gibt

Bild 2: Analogschaltung





EEPROM am I2C-Bus

schreiben, die mit dem Master und der CPU verbunden sind. Die Logik ermöglicht den A/D-Wandler durch zugehörige Punkte von AVR-AD-Register, das die Daten nach einem Start & Master im I2C-Bus sendet. Das I2C gibt die nächsten verfügbaren Signal, jedoch die Adresse der Daten, die die I2C-Adresse des Master. Nach dem Empfang der Daten, die im Register I2C-Adresse enthält, die fallenden Punkte hochziehen kann, werden die neuen Daten im Ausgaberegister geladen. In diesem Zeitpunkt sollte der Controller der CPU, die Befehle ausführen, die er empfangt die Daten vom Port empfangt.

Beispiel

Halten Sie die 24-Bit-Byte im Speicher, mit dem Controller verbunden, so kann die Kommunikation zwischen den Daten über die Busleitung nicht möglich sind. Capacitance ist ein Maß für die Widerstand, die die Daten über die Busleitung übertragen werden können.

Die Widerstand ist, wenn die Pin der CPU, die Sensor, ist extrem empfindlich und kann die Beladung des Sensors entsprechend zu steuern. Wenn die Widerstand ist zu hoch, wird die Widerstand, die die CPU nicht die Daten empfangen kann. Die Widerstand ist, wenn die Pin der CPU, die Sensor, ist extrem empfindlich und kann die Beladung des Sensors entsprechend zu steuern. Wenn die Widerstand ist zu hoch, wird die Widerstand, die die CPU nicht die Daten empfangen kann.

Abgesehen von der Widerstand der CPU, die die Daten empfangen kann, ist die Widerstand der CPU, die die Daten empfangen kann, ist die Widerstand der CPU, die die Daten empfangen kann.

Solange es nicht auf Geschwindigkeit ankommt, genügt eine Nachbildung mit Portpins und Software. Damit werden beliebige I2C Peripherie ICs ansteuerbar. Am häufigsten benötigt werden serielle EEPROMs. Hier wird die Ansteuerung für den 256 Byte Typ beschrieben.

Schaltung

Die in Bild 1 gezeigte Ansteuerung verwendet 3 statt 2 Pins. Die Taktleitung SCL ist wie üblich ein Output. Die bidirektionale Datenleitung SDA erhält jedoch einen Input- und einen Outputpin. Das ist manchmal nützlich um Pins zu verwenden die nicht bidirektional sind. Statt eines Opencollectortreibers mit einem Transistor, ist hier die einfachere Schaltung mit einer Schottkydiode vorgesehen. Diese ist nötig, um einen gültigen low-Pegel zu erzeugen.

Der Pullupwiderstand sollte zugunsten eines niedrigen Stromverbrauchs möglichst hochohmig sein. Werden jedoch mehrere ICs an den Bus geschaltet steigt die Lastkapazität rasch an. Das Diagramm (Bild 2) zeigt die empfohlene Obergrenze der akzeptablen Widerstandswerte. Die Untergrenze liegt bei ca. 2k Ohm. Timing ist bei einem softwaregesteuerten I2C-Bus jedoch fast immer unkritisch, da er zwangsläufig weit unterhalb der zulässigen Grenzfrequenz läuft.

Die Ansteuerung von SDA mit nur einem bidirektionalen Portpin ist natürlich auch möglich. Der Pullupwiderstand wird jedoch weiterhin benötigt.

Hardwareunterschiede bei den ICs der verschiedenen Hersteller gibt es an Pin 7. Er muß entweder auf 5V oder auf GND gelegt werden. Beim CMOS-

8582 von Philips wird der Pin auf high gelegt, beim 24C02 auf low.

Byteübertragung

Die Übertragung (Bild 3) der Daten erfolgt seriell als Bytes, das oberste Bit zuerst. Als 9. Bit folgt „Acknowledge“ den Daten. Es dient zur Fehlersicherung und wird vom Empfänger gesendet. Low bedeutet „OK“.

Im Ruhezustand sind SDA und SCL high. Danach folgt „Start“: erst SDA und dann SCL gehen auf Low. Im Übergangszustand zwischen zwei Bytes sind beide Leitungen auf Low. Der Ruhezustand wird durch „Stop“ erreicht, wenn erst SCL und dann SDA wieder auf High gehen. Zur Übertragung der Bits erzeugt der Master, also der Controller, die Taktimpulse auf SCL. Änderungen des Pegels auf der Datenleitung SDA sind nur zulässig, solange SCL auf Low ist. Etwas verwirrend ist die Senderichtung, sie wechselt nämlich innerhalb einer Byteübertragung. Wenn der Master, der Controller, ein Byte sendet z.B. die „IC-Adresse“, dann muß der Slave, das Peripherie-IC, den „Acknowledge“ senden. Die CPU sendet hierzu einen Highpegel auf SDA, den das Peripherie-IC auf Low zieht. Wenn die CPU Daten empfängt, muß jedoch sie den „Acknowledge“ senden.

Bild 1: Schaltung

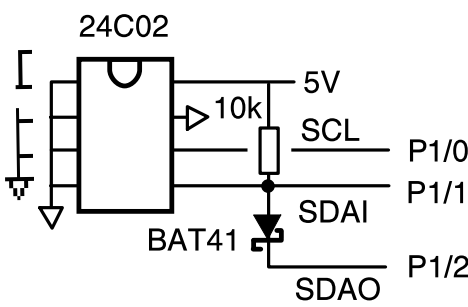
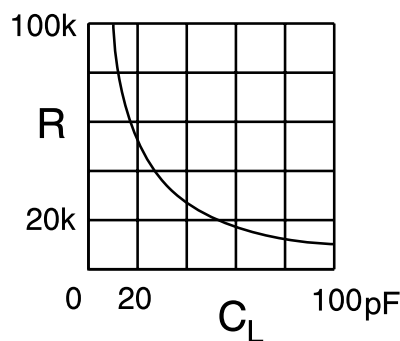


Bild 2: kapazitive Belastung



Simpler Zeitscheibenmultitasker

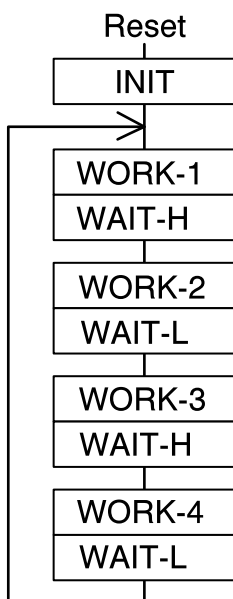
Die Anwendungen von Controllern erfordern oft, daß man unterschiedliche Abläufe gleichzeitig bearbeiten muß, Multitasking also. Will man kleine Controllern wie PIC 16C54 oder 68HC705J1A verwenden, sollte das gewählte Verfahren allerdings sehr einfach sein. Der 16C54 z.B hat keinen Interrupt. Generell ist sehr wenig EPROM und RAM verfügbar. Vorausgesetzt man programmiert in Assembler, ist die Verteilung der Software in eine Hauptschleife die in konstante Zeitscheiben unterteilt wird, eine wirksame Lösung. In den Listings wird hier durchwegs der 68HC05 verwendet, weil sein Assembler gut lesbar ist. Das Verfahren ist jedoch universell verwendbar.

Hauptschleife

Nach dem Reset erfolgt die übliche Initialisierung von CPU und I/O. Anschließend sollte man das gesamte RAM auf 00h initialisieren. Der Code dafür beansprucht meist weniger als 10 Bytes und nur einige Millisekunden Ausführungszeit. Man hat dann in allen Variablen klare und reproduzierbare Startverhältnisse und vermeidet so viele Fehler.

Die große Hauptschleife (Bild 1) ist in Abschnitte mit den Namen WAIT und WORK unterteilt die sich abwechseln und zusammen eine Zeitscheibe („timeslice“) ergeben (vgl Listing) In WORK werden kurze Teile des Anwendungsprogramms abgearbeitet. WAIT wartet anschließend bis ein Timer abgelaufen ist (Bild 2). Jede Zeitscheibe und damit die gesamte Schleife hat somit eine feste

Bild 1:
Schleife



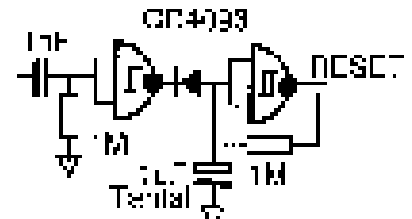
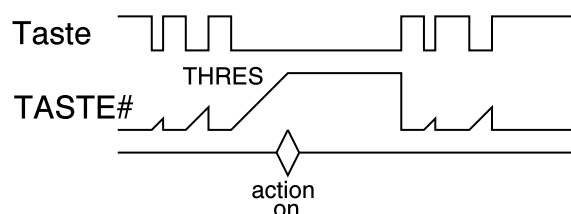
Durchlaufzeit. Wenn die Wartezeit 500usec beträgt, wird die Schleife in Bild 1 konstant alle 2msec durchlaufen werden. Unabhängig wieviele Opcodes in den WORK-Abschnitten tatsächlich ausgeführt werden.

WAIT

Für die Implementierung von WAIT ist es nötig, daß der Controller einen freilaufenden Timer hat. Man sucht sich in diesem ein Bit das den passenden Takt erzeugt und teilt WAIT in WAIT-H und WAIT-L auf. WAIT-H ist eigentlich nur ein Branchbefehl der auf sich selbst verzweigt, wenn das Bit gesetzt ist. Es hat sich in der Praxis jedoch als nützlich erwiesen, wenn man vor dem WAIT-Befehl einen TEST-Pin an einem Port der CPU setzt und anschließend löscht. Damit ist ein Unterprogramm WAIT-H sinnvoll, der Code sieht dann wie in Listing 1 aus.

Man kann an dem TEST-Pin nicht nur mit dem Oszilloskop feststellen, ob die Software sauber in der Schleife läuft. Man kann damit auch einen externen Watchdog ansteuern (Bild 3). Die Schaltung erzeugt als Reset einen sehr langsamen Takt,

Bild 2:
Timing der
Zeitscheiben



Ein Watchdog
kann durch eine Watchdog-
spannung an Eingang A und
ausgelöst werden

Wie lange

Man organisiert die WORK-
Abschnitte meist in Schritten zu
maximal 10 Bytes. Das heißt
Vorsicht daß man bei Programmierung
die Rückkehr in den kompletten Ab-
schnitt versteht. Diese Filterzeit
findet sich Fehler vermeiden. In
Assemblern sind die Anweisungen die
keine Befehlszyklen auf Zyklen
beruhen. Bei 1 MHz CPU-Takt kann
WORK 1000 Bytes maximal. Man
kann die WAIT-Zeit mit zwei Resistor-
anstellungen setzen. Das ist also
schon der korrekten Schaltungsteil. Man
kann sich auch Programmcode in die
die nicht unter 2550 Zyklen programmi-
erhalten sind. Das man nicht überlegt
kann Implementierung für die man
nicht was spekulieren in der Wartezeit
versuchen sollte. Typischerweise für
WAIT sind etwa 500usec oder
1msec. In Schichten kann man
ein 68HC05 wählen. Die Zeit der PC-
51 kann 1000msec. Bei 100usec für
Controller meist zu langsam

Wie viele

Für komplexe Applikationen
genügen oft schon 8 WAITs. Aber
auch mit 2-3 in geringerer Le-
istung kann man Applikationen bauen.
wobei mehr WAITs immer noch
Man kann immer genaue Auslegung
der WAITs auch kenne

