Unterlagen mit Kontrollfragen für das Selbststudium im Fach Rechnertechnik I

15.04.2021

Erstellt von:
Stefan Lehmann

0 Aktuelle Änderungen

	Todo:
15.04.2021	Literaturliste um einen Eintrag ergänzt
19.03.2021	Anhang: Hexadezimalsystem ergänzt
11.03.2021	Unterkapitel über Adressierungsarten hinzugefügt
15.02.2021	Kapitel über Automaten hinzugefügt Unterkapitel über unvollständige Dekodierung hinzugefügt
23.11.2020	Speichertechnologie um PROM und EPROM erweitert Handshake und Duplex bei RS232 ergänzt.
05.10.2020	Unterkapitel über Stack hinzugefügt.
19.06.2020	SPI, I^2C, Minidrucker und LCD ergänzt Fragen zur Selbstkontrolle vervollständigt.
24.04.2020	Danke an Herrn Fey für wichtige Korrekturen und Formulierungsvorschläge.
14.04.2020	Aufmerksamen Studenten aus AI2 und AI4 sind Fehler in den Abbildungen 29 und 36 aufgefallen. Danke für die Hinweise.

1 Vorwort

Diese Unterlagen sind kein Ersatz für den Besuch der Vorlesung. Nein, sie sollen zur Wiederholung und Vertiefung des Vorgetragenen dienen. Darüber hinaus sind hier Kapitel aufgenommen, die in der Vorlesung nicht vorkommen oder besten Falls am Rande erwähnt werden. Dennoch sind diese Kapitel Inhalte für das Selbststudium. Statt mühsam im Internet recherchieren zu müssen, sind die wichtigsten Punkte hier kompakt und anschaulich dargestellt.

Der Fragenkatalog im Anhang soll ihnen zur Orientierung dienen, was in der Klausur abgefragt werden kann[1].

1 Es werden natürlich nicht alle Fragen in der Klausur gestellt, sondern nur ein kleiner Teil.

Inhaltsverzeichnis

2 Automaten

Automaten sind Maschinen, die Eingangssignale mit internen Signalen miteinander verknüpfen können um daraus neue Ausgangssignale zu erzeugen. Sie werden auch als Zustandsmaschinen bezeichnet. Dabei unterscheidet man zwischen den endlichen (finite state machine) und unendlichen (infinite state machine) Zustandsmaschinen. Bei den endlichen Zustandsmaschinen gibt es eine endliche Anzahl von Verknüpfungsergebnissen. Hierunter fallen alle üblichen Rechner und deren Programme, bei denen das Ergebnis vorher bestimmbar ist.

Automaten bestehen aus einem Verknüpfungsfeld und einem getakteten Speicher. Der Takt bestimmt den Zeitpunkt, an dem die Daten aus dem Verknüpfungsfeld in den Speicher übernommen werden. Das Verknüpfungsfeld kann sowohl die Eingangssignale als auch die Ausgänge des Speichers logisch verknüpfen. Damit ist der Ausgangsvektor nicht nur von den Eingangsvektoren sondern auch vom aktuellen internen Zustand abhängig.

Man unterscheidet drei Typen von Automaten. Der einfachste ist der Medwedev-Automat. Hier ist der äußere Ausgangszustand (= Ausgangsvariablen, Ausgangssignale) gleich dem internen Zustand.

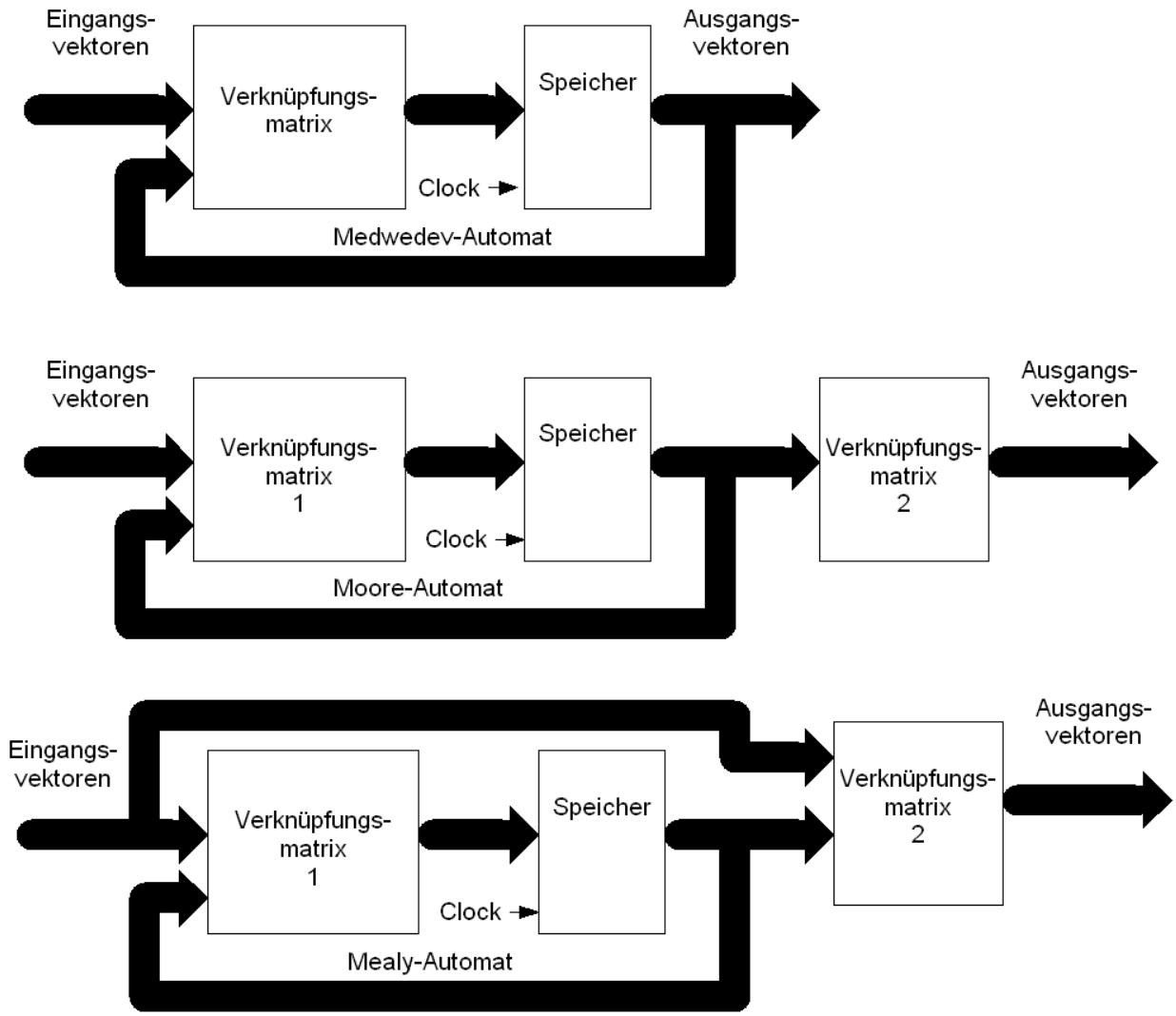


Abbildung 1: Medwedev-, Moore- und Mealy-Automat

Ergänzt man den Medwedev-Automat um ein weiteres Verknüpfungsfeld am Ausgang bekommt man den Moore-Automat. Hier unterscheiden sich die Ausgangsvektoren von den internen Zuständen.

Der Medwedev- und der Moore-Automat sind reine synchrone Maschinen. Im Gegensatz dazu ist der Mealy-Automat nicht mehr synchron. Er benötigt auch ein Taktsignal um die internen Zustände zu speichern aber die Eingangsvariablen können direkt auf das zweite Verknüpfungsfeld geführt werden und so die Ausgangsvektoren unabhängig vom Takt verändern.

2.1 Zustandsdiagramm

Der Vorteil solcher Automaten ist die einfache Darstellung und Umsetzung von zeitlich ablaufenden Vorgängen. Die Darstellung erfolgt mittels Zustandsdiagrammen. Diese bestehen aus den Knoten und den Kanten. Die Knoten stellen die einzelnen Zustände dar, während die Kanten die Übergangsbedingungen von einem Zustand in einen anderen definieren. Abbildung 2 zeigt mittels

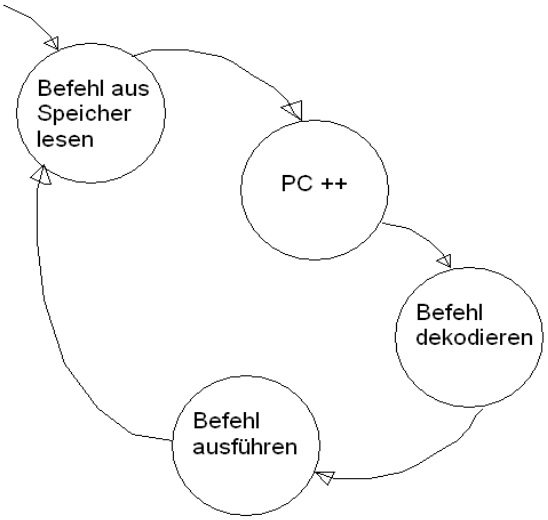


Abbildung 2: Beispiel eines Zustandsdiagramms

einfachem Zustandsdiagramm wie die Ausführung eines einfachen Befehls in einem Mikroprozessor funktioniert.

Wie kann eine solche Zustandsmaschine programmiert werden? Dazu gibt es viele Ansätze. Hier ein sehr einfaches aber überschaubares Beispiel in einem Pseudocode:

```
zustand1:
        zustand = 1                 //aktuelle Zustand merken
        read inputs                 //lese Eingangsvariablen
        cp alte_inputs              //vergleiche mit der alten Eingangsvariablen
        jz zustand1                 //wenn sich nichts geändert hat, warten
        alte_inputs = inputs        //aktuelle Eingangsvariablen merken
        if inputs = 2 then goto zustand2
        if inputs = 3 then goto zustand3
        if inputs = 4 then goto zustand4
        goto zustand1               //andere Eingangsvariablen sind nicht relevant
```

3 Rechnerarchitekturen

Rechner arbeiten nach dem Prinzip Eingabe - Verarbeitung - Ausgabe (EVA). Im Laufe der Zeit haben sich viele Konzepte für die Realisierung von Rechnern entwickelt.

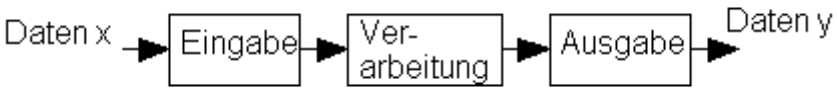


Abbildung 3: Das EVA-Prinzip

Konrad Zuse entwickelte sehr früh einen programmgesteuerten Rechner mittels elektromechanischen Relais. Das Programm war in Form eines Lochstreifens (alter Kinofilm) bei dem das Lochmuster die Befehle und Daten darstellten. Der Rechner konnte noch keine Verzweigungen und war deshalb nur für einfache Berechnungen nutzbar.

Kurze Zeit später entstand in den USA der erste, mit Elektronenröhren aufgebaute Rechner. Die Zeiten in der dieser Rechner repariert werden musste, waren viel länger als die Zeiten, in der er funktionstüchtig war.

Zu dieser Zeit entwarf der Ungar John von-Neumann in den USA die erste theoretische Abhandlung für den Bau eines Rechners, der vollkommen frei anwendbar war und damit eine kostengünstigere Alternative zu den spezifischen Maschinensteuerungen war. Denn diese ist das Programm nicht als Software sondern in fest verdrahteter Logik realisiert. Änderungen im Ablauf bedeutete die Änderung der Hardware. Der Entwurf von Von-Neumann trennte die Funktionslogik von der Hardware indem Befehle die Funktion der Hardware beeinflussten.

Sein Konzept basierte darauf, dass immer ein Befehl und ein Datum bearbeitet wird. Dass es dazu auch andere Alternativen gibt, zeigt die Klassifizierung von Rechnern nach Flynn.

3.1 Klassifizierung nach Flynn

SISD	Single Instruction Stream, Single Data Stream
SIMD	Single Instruction Stream, Multiple Data Stream
MISD	Multiple Instruction Stream, Single Data Stream
MIMD	Multiple Instruction Stream, Multiple Data Stream

3.1.1 SISD

Der typische Universalrechner wie z.B. Von-Neumann oder Harvard, fallen in die Klasse SISD. Den Befehls- und Datenfluss bei einem SISD-Rechner zeigt die Abbildung 4.

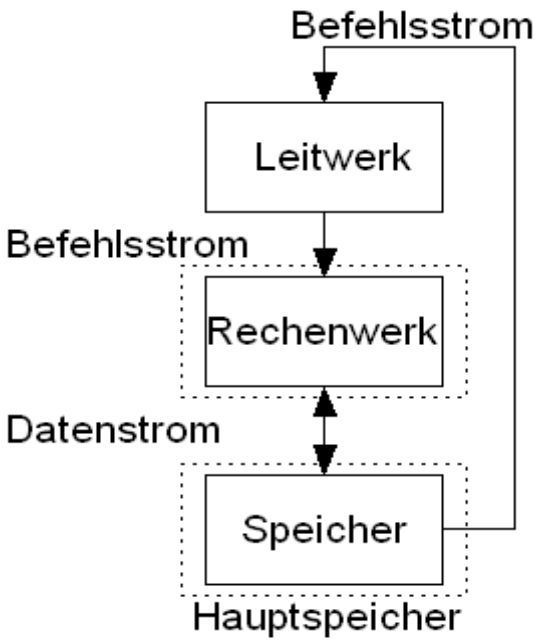


Abbildung 4:
Informationsfluss beim SISD-Rechner

3.1.2 SIMD

Beim SIMD-Rechner handelt es sich um eine Erweiterung eines Von-Neumann-Rechners. Darin existiert ein Leitwerk aber mehrere Rechenwerke und Speichermodule für die Daten. Für das Programm liegt dagegen nur ein Speichermodul zur Verfügung. Abbildung 5 verdeutlicht diesen Aufbau. Zu den wichtigsten SIMD-Rechner zählt der sogenannte Feldrechner.

3.1.3 MISD

Diese Klasse kann nur schwer den bekannten Rechnersystemen zugeordnet werden.

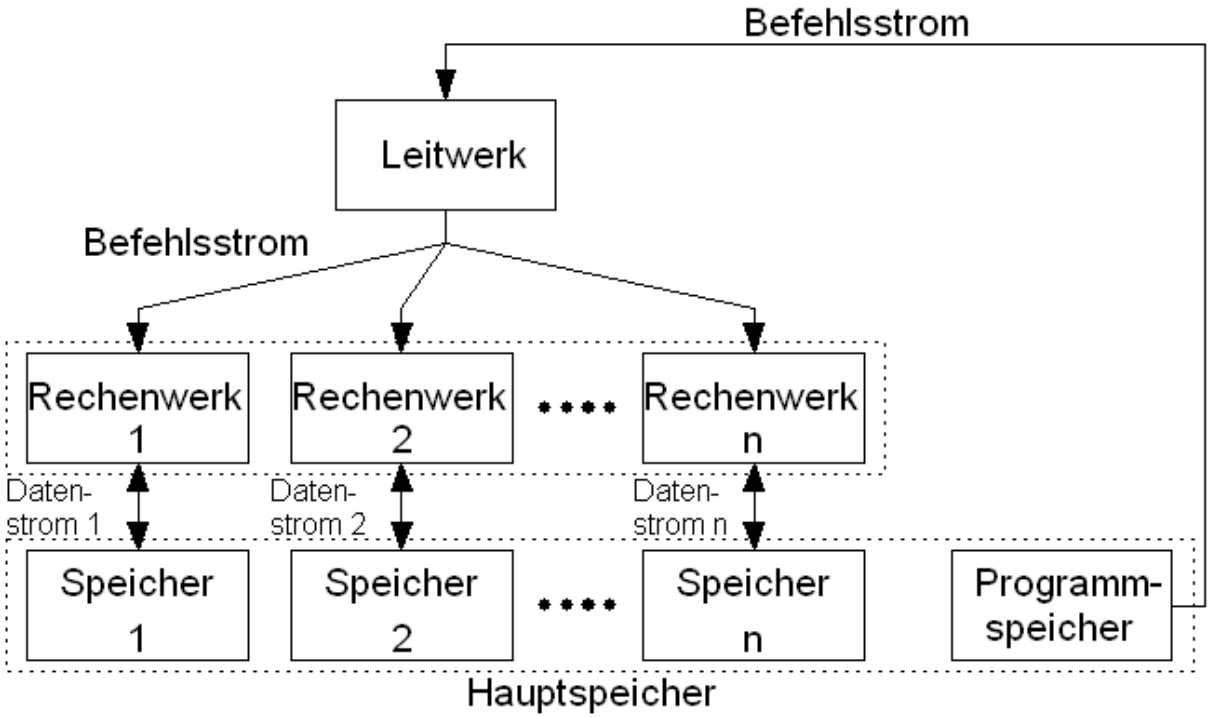


Abbildung 5: Informationsfluss beim SIMD-Rechner

3.1.4 MIMD

Im Prinzip sind dies Rechner mit mehreren unabhängigen Prozessoren. Diese einzelnen Prozessoren haben ein eigenes Leit- und Rechenwerk sowie ein eigenes Speichermodul für Daten. Das Programm kann in einem gemeinsamen Programmspeicher oder in einem der Speicher, die den einzelnen Prozessoren zugeordnet sind.

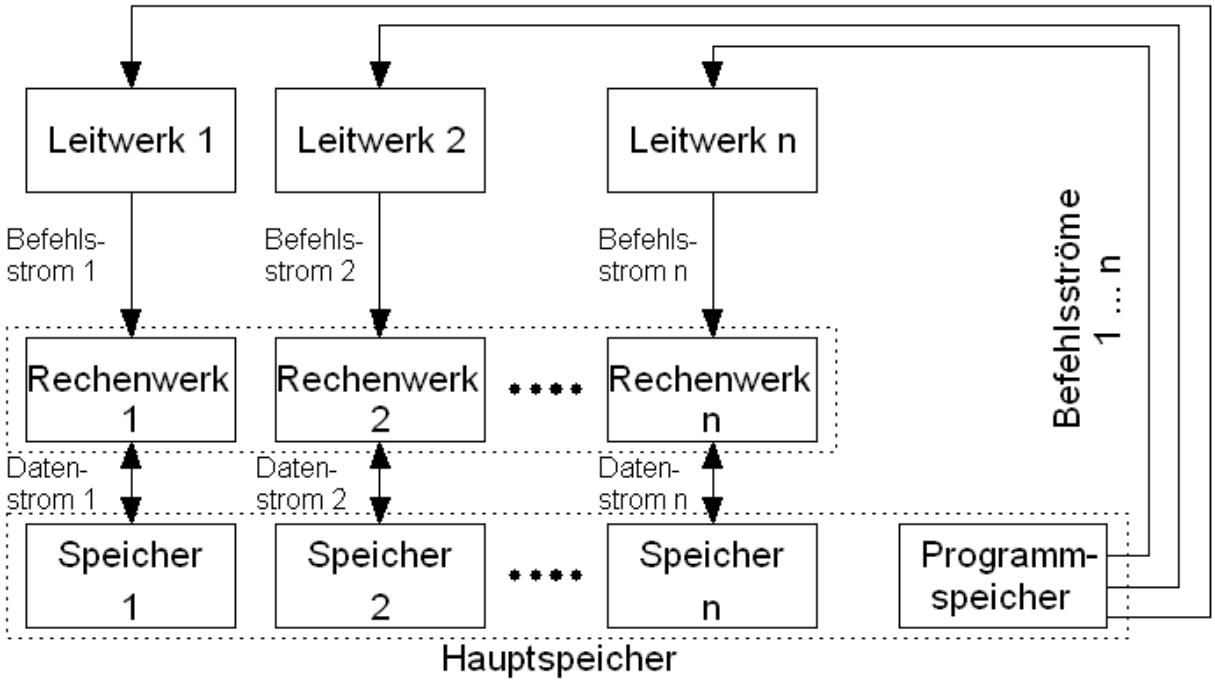


Abbildung 6: Aufbau eines MIMD-Rechners

3.2 Einteilung nach Befehlsformaten

Man unterscheidet zwischen 0-, 1-, 2- und 3-Adress-Maschinen. Dabei geht es darum, wie auf die einzelnen Operanten vor und nach der Verknüpfung zugegriffen wird.

Maximal sind vier Adressen: Zwei für die beiden zu verknüpfenden Operanten, eine für die Zieladresse, wo das Ergebnis abgelegt wird, sowie die Adresse an welcher der nächste auszuführenden Befehl steht, notwendig. Geht man davon aus, dass auf den aktuell ausgeführten Befehl der nächste Befehl folgt (sequentielle Anordnung), kann die letzte Adressangabe entfallen, was dann der 3-Adress-Maschine entspricht.

3.2.1 1-Adress-Maschine

Hier steht ein Operant im einem speziellen Register, dem Akkumulator, und der andere im Speicher. Auf diesen Speicher muss der Prozessor zuerst zugreifen um diesen zweiten Operanten zu holen um ihn dann zu verarbeiten. Das Ergebnis kommt in das Spezialregister Akkumulator.

Abbildung 7: Eine 1-Adress-Maschine

Es gibt in diesem Fall noch eine kleine Modifikation. Ein zusätzliches Bit entscheidet, ob das Ergebnis in den Akkumulator kommt oder den zweiten Operanten im Speicher überschreibt. In diesem letzten Fall bleibt der Akkuinhalt unverändert. Solche Prozessoren werden auch als 1 ½ - Adress-Maschinen bezeichnet. Ein typischer Vertreter dieser Gattung ist der PIC Mikrocontroller.

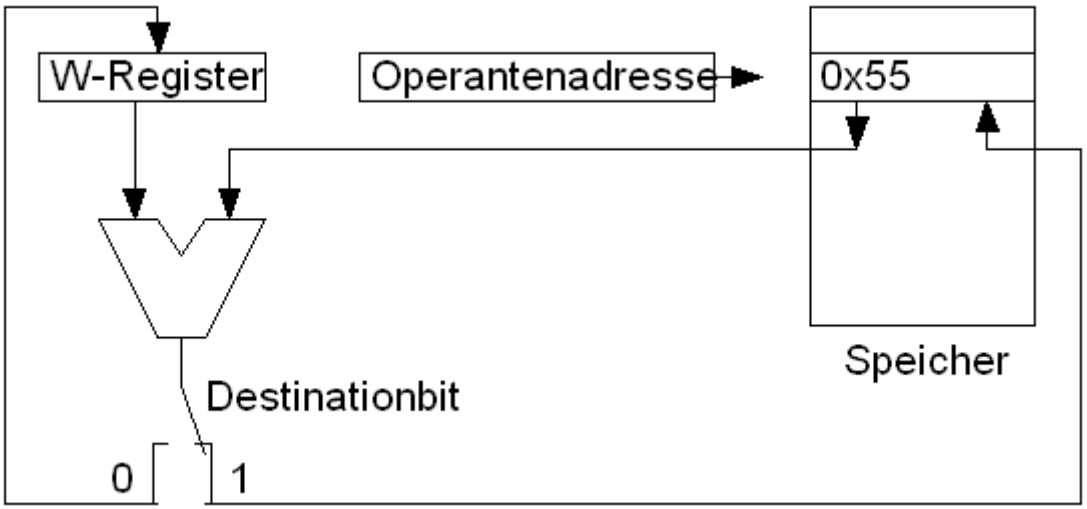


Abbildung 8: Die 1 1/2 -Adress-Maschine mit dem Destinationbit

3.2.2 2-Adress-Maschine

Die 2-Adress-Maschine verzichtet auf die Angabe der Zieladresse. Somit wird in eine der beiden Operantenadressen das Ergebnis abgelegt.

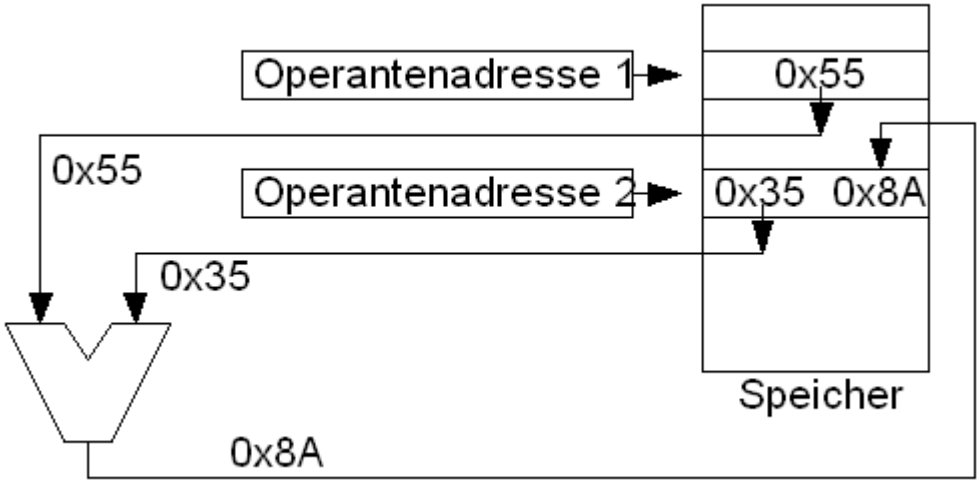


Abbildung 9: Eine 2-Adress-Maschine

3.2.3 3-Adress-Maschinen

Hier werden die Operatoren und das Ergebnis an verschiedenen Stellen im Speicher adressiert. Das erlaubt eine sehr flexible Programmierung, die man sich allerdings mit einen sehr langen Befehl erkauft.

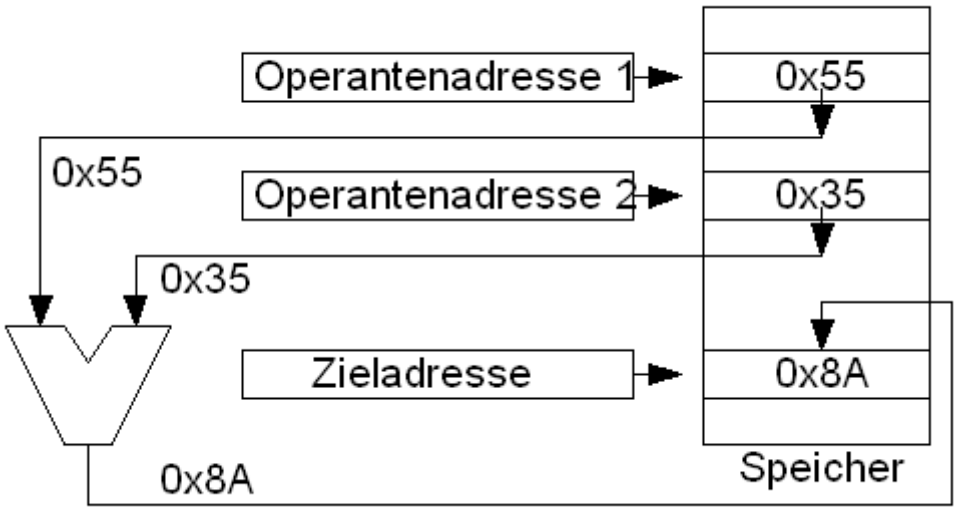


Abbildung 10: Eine 3-Adress-Maschine

3.2.4 0-Adress-Maschine

Hier zeigt keine Adresse auf einen im Speicher liegenden Operanten. Alle Operanten liegen in einer internen Struktur vor, die ohne Adressierung auskommt. Dies ist z.B. bei der Stack Architektur der Fall (siehe Kapitel 3.3.5 auf der Seite 19).

3.3 Befehlssatzarchitektur

Die nachfolgende Unterscheidung beruht darauf, woher die Operanten für eine arithmetische oder logische Operation stammen. Die Befehle mit denen die Operanten zuerst "in Position gebracht" werden müssen, bleiben ohne Bedeutung.

Es gibt auch Mischformen, bei denen die eindeutige Zuordnung zu einer dieser Kategorien nicht immer leicht ist.

3.3.1 Akkumulator-Architektur

Diese Architektur stellt den Akkumulator (kurz: Akku) in den Fokus. Dieses zentrale Register hält immer ein zu verknüpfendes Argument bereits. Außerdem wird das Ergebnis der logischen oder arithmetischen Operation hier abgelegt. Das Ziel "Akkumulator-Register" steht somit implizit im Befehl und muss nicht als Argument beim Befehl mit angegeben werden.

Typische Befehle sind z.B.:

```
ADD     B       ;Akku + Inhalt Reg. B, Ergebnis in Akku
MOV     D       ;kopiere Akkuinhalt nach Reg. D
```

Ein typischer Vertreter ist der 68HC12 und dessen Vorgänger 68HC11. Abbildung 11 zeigt die internen Register, die der 68HC12 zur Verfügung stellt.

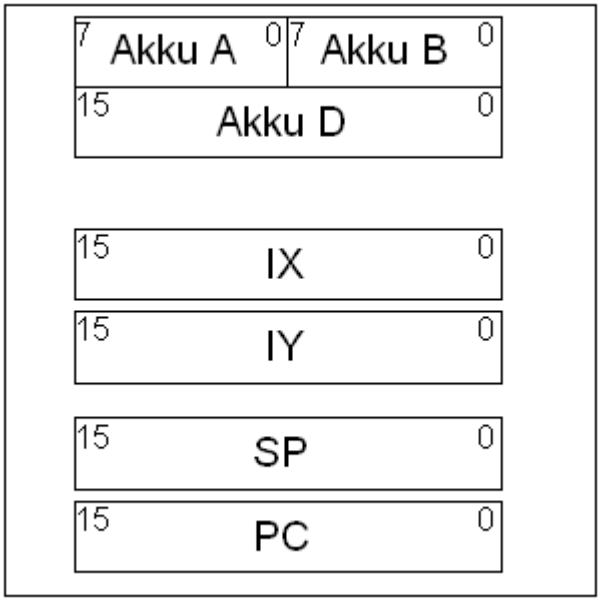


Abbildung 11: Interner Registersatz des 68HC12

Beim 68HC12 wird zuerst eines der beiden Indexregister IX oder IY mit der Adresse geladen, deren Inhalt mit dem Inhalt des Akkus verknüpft werden soll. Das Ergebnis steht anschließend im Akku und muss durch einen zusätzlichen Befehl in den RAM-Speicher "gerettet" werden. Der Akku kann sowohl als 16-Bit Register, als auch als zwei getrennte 8-Bit Register verwendet werden.

3.3.2 (GP-)Register-Architektur

Es gibt bei dieser Architektur Register, auf die das Steuerwerk[2] in einem Taktzyklus zugreifen kann. Damit ist der Zugriff sehr schnell. Die Anzahl dieser Register schwankt zwischen 32 und 512. Diese Tatsache ist vor allem für Compiler von großem Interesse. Sie können wichtige und oft genutzte Variablen in diesen Registern ablegen. Der Zugriff auf Variablen im RAM-Bereich ist entsprechend langsamer.

3.3.3 Register-Speicher-Architektur

Hier gibt es eine kleinere Anzahl von Registern mit denen sehr effektiv gearbeitet werden kann. Sind jedoch viele Variablen oder Daten zu verarbeiten, muss auch der RAM-Speicher für die Datenzugriffe genutzt werden. Diese Architektur wird meist als 1-Adress-Maschine aufgebaut. Die Operanten können zwei Register sein oder ein Register und eine Speicheradresse.

2 mehr dazu in Kapitel 4 ab Seite 20.

Die Intel-Prozessoren 80x86 fallen in diese Kategorie.

Beispiel:

```
ADD     AL, (adr1)      ;Ergebnis steht in AL
ADD     SI, CX          ;Ergebnis steht in SI
```

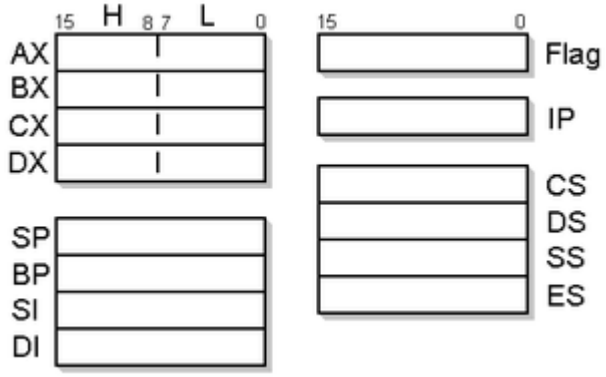


Abbildung 12: Der Registersatz des 8086

Mehr zu diesem Prozessor wird in Kapitel 8 erörtet.

3.3.4 Speicher-Speicher-Architektur

Diese Architektur ist mit die Universellste. Die Operanten können sowohl aus den Registern als auch aus dem RAM-Speicher kommen. Das Ergebnis kann in einem Register oder im RAM abgelegt werden, je nach dem wie viele Adressen abgegeben werden können. Im Maximalfall sind dies drei, zwei für die Operanten und eine dritte für das Ergebnis. Man spricht dann von einer 3-Adress-Maschine.

Beispiel:

ADD (adr1), (adr2), (adr3)[3]

	Adresse	vor der Addition	nach der Addition
adr1	0x1234	0x38	0x38
adr2	0x1235	0x4F	0x4F
adr3	0x1236	??	0x87

```
ADD     reg1, reg2, reg3        ;addiert Inhalte von reg1 und reg2,
                                ;das Ergebnis kommt in reg3
```

3 (adr) bedeutet es wird der Inhalt der Speicherzelle mit der Adresse adr verwendet (indirekte Adressierung).

3.3.5 Stack-Architektur

Bei der Stack-Architektur handelt es sich um eine sogenannte 0-Adress-Maschine. D.h. die Argumente für eine Verknüpfung stammen nie direkt aus einer Speicheradresse sondern liegen auf einem Stack. Von dort werden immer die zwei obersten Einträge miteinander verknüpft, das Ergebnis überschreibt den obersten Eintrag.

Diese Art zu Rechnen wurde bei früheren HP-Taschenrechnern implementiert. Auch die Programmiersprache FORTH orientiert sich an der sogenannten UPN (Umgekehrte Polnische Notation). Der Vorteil war der, dass man keine Klammerebenen benötigt. Viele Compiler lösen die Gleichungen so auf, dass deren Operanten auf einen Stack gelegt werden können, um dann mittels UPN abgearbeitet zu werden.

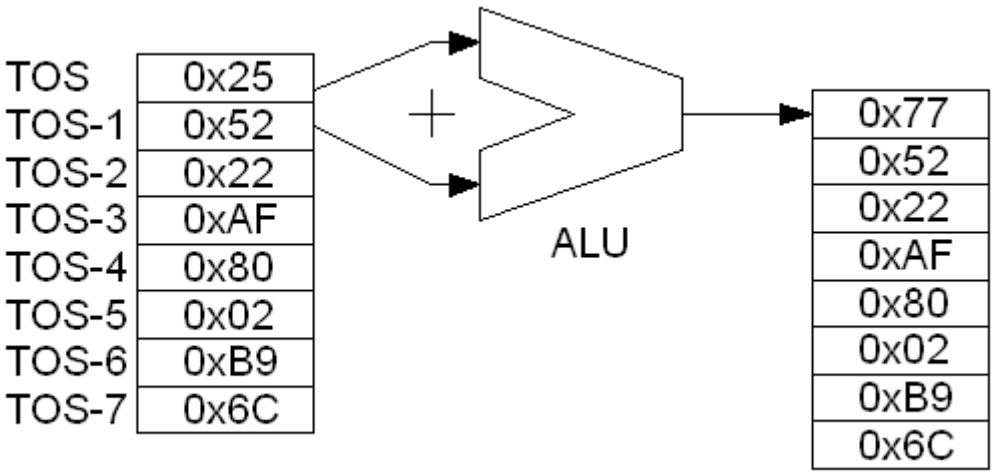


Abbildung 13: Eine Stack-Architektur addiert zwei Zahlen

4 Die Von-Neumann-Architektur

Sie ist die am meisten eingesetzte Art und Weise um Mikroprozessoren und Mikrocontroller zu realisieren. Auch die sehr leistungsstarken Pentium-CPUs sind so aufgebaut.

4.1 Grundeigenschaften

Eine Von-Neumann-Architektur besteht aus folgenden Rechnerteilwerken:

- Speicherwerk (kurz: Speicher)
- Rechenwerk (ALU - Arithmetic Logic Unit)
- Leit- oder Steuerwerk (CU - Control Unit)
- Ein- / Ausgabewerk
- Bus, der die obigen Werke miteinander verbindet

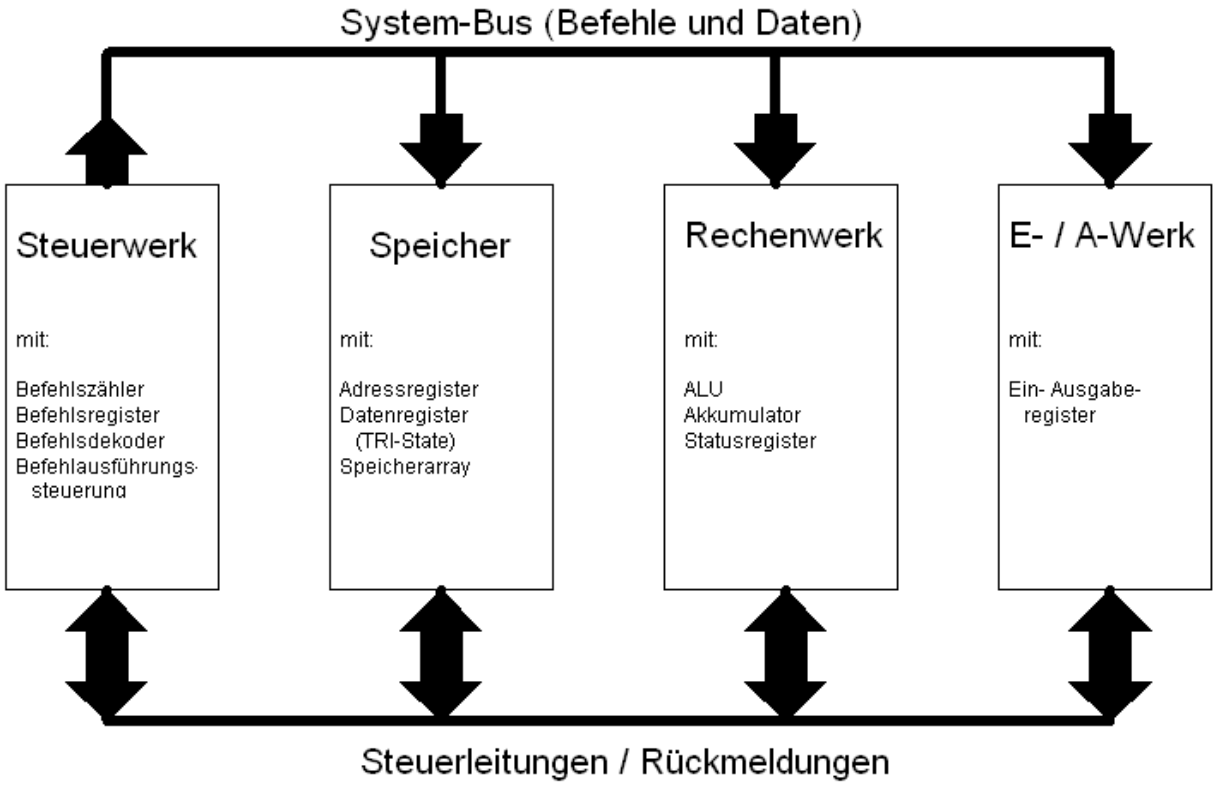


Abbildung 14: Struktur eines Von-Neumann-Rechners

Das Speicherwerk dient zum Abspeichern von Programmen und Daten. Die Adressierung von Befehlen und deren Argumente erfolgt durch den Programmzähler, der auf den nächsten abzuarbeitenden Befehl zeigt. Sollen Daten ge-

schrieben oder gelesen werden, erfolgt der Zugriff entweder direkt auf die Adresse, die im Befehl als Argument hinterlegt ist oder indirekt, wo ein Register die eigentliche Zugriffsadresse enthält.

Das Rechenwerk berechnet aus zwei Argumenten mittels arithmetischer oder logischer Verknüpfung ein Ergebnis. Weiterhin werden hier Schiebe- und Rotationsbefehle abgearbeitet. In einfachen 1-Adress-Maschinen ist ein Argument im Akkumulator gespeichert, das andere entweder aus einem anderen Register, direkt oder indirekt aus dem Speicher oder besteht aus einer Konstanten (Literal). Das Ergebnis kommt in hier in den Akkumulator.

Das Steuerwerk (auch Leitwerk) beinhaltet den Befehlszähler (auch Programmzähler oder Instruction Pointer), das Befehlsregister, den Befehlsdekoder und die Befehlsausführungssteuerung. Der Programmzähler zeigt auf den nächsten abzuarbeitenden Befehl oder auf Argumente, die als Konstante direkt hinter dem Befehlscode abgespeichert sind.
Im Befehlsdekoder werden die ins Befehlsregister eingelesenen Befehle analysiert und dann von der Ausführungssteuerung abgearbeitet. Diese Abarbeitung ist mal mehr, mal weniger aufwendig, je nach der Komplexität des Befehls.

Das Ein- / Ausgabewerk ist die Schnittstelle zu externen Speichermedien und Bedienungs- sowie Aktoreneinheiten. Zu den Bedienungseinheiten zählen u.a. Tastaturen, aber auch einfache Schalter oder Sensoren. Die Aktoreneinheiten sind Bildschirme, Drucker, Relais, Motoren usw.

Der Bus verbindet all diese Einheiten, so dass diese untereinander kommunizieren können. Man unterteilt den Bus in drei Teilbusse: den Daten-, Adress- und Steuerbus. Einheiten, die an einem Bus angeschlossen sind, müssen i.d.R. sogenannte TRI-State-Treiber besitzen. Auf diese Weise können sie sich, in dem sie sich in den hochohmigen Zustand versetzen, vom Bus abkoppeln.

Das Blockschaltbild in Abbildung 15 zeigt ein typischen Aufbau eines einfachen Mikroprozessorsystems mit den entsprechenden, auf dem Markt erhältlichen Bauteilen. Der Programmspeicher ist hier als EPROM[4]-, der Datenspeicher als RAM-Baustein realisiert. Obwohl es mehrere Bauteile sind, liegen alle Speicheradressen in einem Adressraum, was dem obigen Speicherwerk entspricht. Die Selektion der anzusprechenden Bausteine wird über den externen Adressdekoder erledigt.
Der CPU-Baustein beinhaltet neben dem Rechenwerk auch das Steuerwerk.

4 Erasable Programmable Read Only Memory

Das E- / A-Werk ist hier in Form eins PIO- (Parallel Input Output) und SIO-Bausteins (Serial Input Output) realisiert.

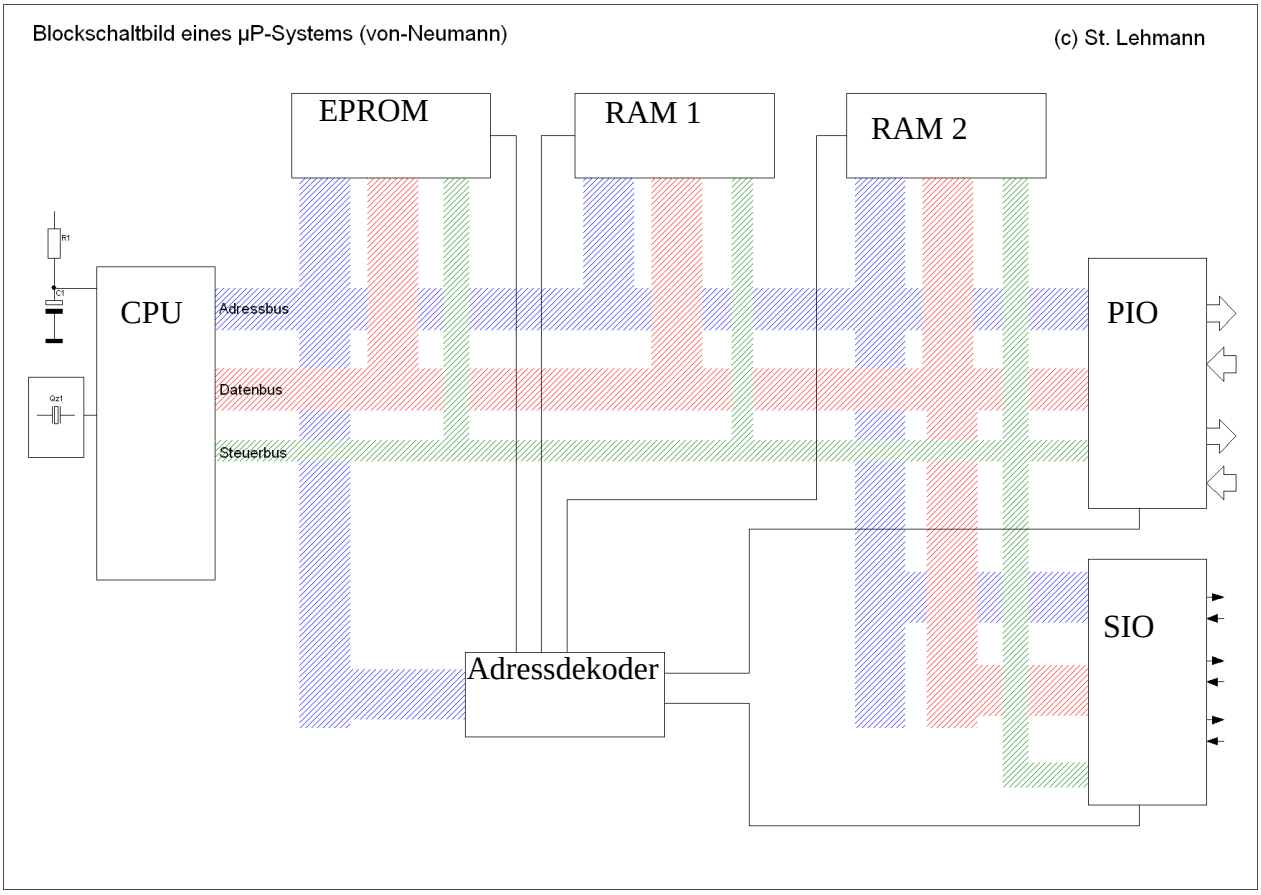


Abbildung 15: Blockschaltbild eines typischen Mikroprozessorsystems (Von-Neumann)

4.2 EPROM-Baustein

Das EPROM ist ein Speicher, dessen Inhalt von dem Prozessor nur gelesen werden kann. Sein Inhalt lässt sich durch UV-Licht löschen. Dazu besitzt er ein Quarzfenster, durch das das UV-Licht auf den Die scheinen kann.

In Abbildung 16 ist die Anschlussbelegung eines 32k x 8 Byte großen EPROMs dargestellt.

Man erkennt insgesamt 15 Adressleitungen (A0 bis A14). Damit kann dieser Baustein 32768 Speicherplätze adressieren. Die 8 Datenleitungen bedeuten einen 8-Bit breiten Datenbus. Somit besitzt jede Speicheradresse 8 Bits, die parallel /gleichzeitig) ausgegeben werden.

Der Vpp-Anschluss wird für die Programmierung dieses Bausteins in einem

speziellen Programmiergerät benötigt.

Die Ausgangstreiber der Datenleitungen (O0 bis O7) werden durch den $\overline{OE}$ ($\overline{Output\ Enable}$) aktiv geschaltet. Ist dieser Pin auf High, befinden sich die Datenleitungen in einem hochohmigen Zustand, bei dem der Baustein die Daten einlesen kann. Soll er dagegen die Daten auf den Bus ausgeben, müssen die Ausgangstreiber durch ein Low am $\overline{OE}$-Pin aktiv geschaltet werden.

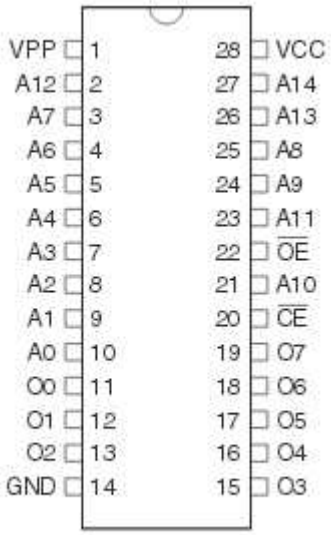


Abbildung 16:
Anschlussbelegung des 27C256

Der CE-Anschluss (Chip Enable) aktiviert den Baustein. Nur in diesem Fall kann er die Signale auf dem Adress- und Datenbus verarbeiten.

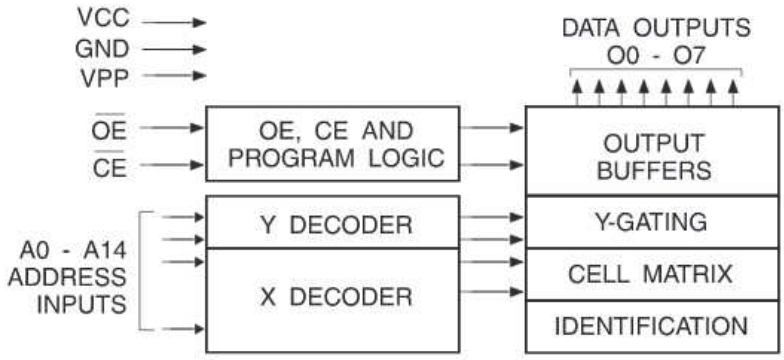


Abbildung 17: Blockschaltbild der internen Logik des 27C256

Der Speicher ist als dreidimensionales Array organisiert (Abbildung 17). Einige Adressleitungen werden an die X-Spalten, die übrigen an die Y-Spalte des Array

gelegt. Im Kreuzungspunkt befinden sich der Ort für die 8 Bits dieser Speicheradresse.

4.2.1 Speicherelemente bei Festwertspeicher

Die ursprünglichen ROM-Bausteine erhielten ihre Informationen während des Fertigungsprozesses. Die entsprechenden Herstellungsmasken waren ausschließlich für diese Anwendung nutzbar. Damit war die Herstellung der Bauteile nur in großen Mengen wirtschaftlich. Wurde ein Programmierfehler entdeckt, waren diese Masken nur noch Müll.

Um diesen teuren Fertigungsaufwand zu umgehen, wurden die PROMs entwickelt. Diese Bausteine waren nach der Fertigung noch ohne Informationsinhalt. Dieser wurde erst zu einem späteren Zeitpunkt meist durch den Kunden selbst in diese Bausteine gebracht. Im Prinzip wurde eine kleine Verbindung mittels Stromimpuls unterbrochen, so wie sich eine Sicherung durchbrennen lässt. Daher auch die Bezeichnung ein "Baustein brennen". Das Unterbrechen dieser Verbindung war allerdings endgültig und unumkehrbar. Falsch programmierte Bausteine waren ebenfalls Müll.

Im nächsten Entwicklungsschritt wurde diese Verbindung durch einen Feldeffekttransistor mit einem sogenannten Floating-Gate ersetzt. Im gelöschten Zustand ist das Floating-Gate frei von Ladungen und der Transistor sperrt. An seinem Ausgang liegt ein H-Signal an. Beim Programmieren des Bausteins werden nun auf diese Floating-Gates Ladungen aufgebracht, die den Transistor durchschalten. Dadurch entsteht an dessen Ausgang ein Low-Signal. Die Anforderungen an dieses Floating-Gate sind sehr hoch, denn im Prinzip bildet es einen Kondensator, der geladen wird. Damit die Information des Bausteins über lange Zeit - mind. zehn Jahre - hält, muss der Leckstrom sehr, sehr klein sein. Somit muss zwischen dem Gate und seiner Umgebung ein extrem hoher Isolationswiderstand vorliegen.
Zum Löschen muss dieser Isolationswiderstand verringert werden, damit die Ladungen vom Gate abfließen können. Das erreicht man z.B. mit hoch energiereichem Licht (UV-Licht). Trifft es durch das eingebaute Quarzfrenter auf diesen Bereich, wird dieser schwach leitend, was ausreicht, um die Ladungen abfließen zu lassen. Anschließend sperren alle Transistoren wieder und ein neuer Programmiervorgang kann durchgeführt werden.

Da das Löschen relativ viel Zeit braucht und dazu i.d.R. der Baustein aus der Schaltung ausgebaut werden muss wurde das EEPROM entwickelt. Bei diesem Bauteil wird der Löschvorgang nicht durch UV-Bestrahlung sondern durch ei-

nen elektronischen Schalter realisiert. Deshalb fehlt bei diesen Bauteilen das Quarzfenster. Neuere Typen führen den Löschvorgang sogar selbstständig durch, sobald der Baustein neu programmiert wird.

Eine weitere Verbesserung wurde durch die Einführung der Flash-Technologie gemacht. Bisher dauerte das Programmieren einer Speicherzelle ca. 5 ms bis 10 ms. Bei der Flash-Technologie nur noch 100 µs bis max 1 ms. Zusätzlich werden nicht nur einzelne Bytes programmiert, sondern ganze Blöcke von z.B. 64 Bytes. Dazu werden die 64 Bytes in einen Puffer geladen und dann dieser Puffer en bloc programmiert.

4.3 RAM-Bausteine

RAM bedeutet Random Access Memory. Man kann Daten sowohl auslesen als auch einschreiben. Das Random bezieht sich auf die Freiheit jeden beliebigen Bereich direkt anwählen zu können, im Gegensatz zum sequentiellen Lesen und Schreiben. In ist ein 2k x 8 und in ein 8k x 8 großer RAM-Speicher abgebildet.

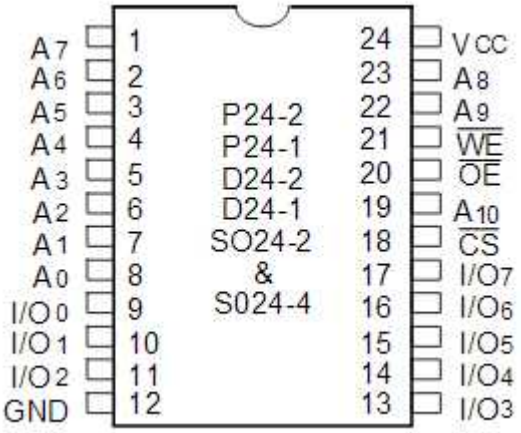


Abbildung 18: RAM mit 2k x 8 Bit

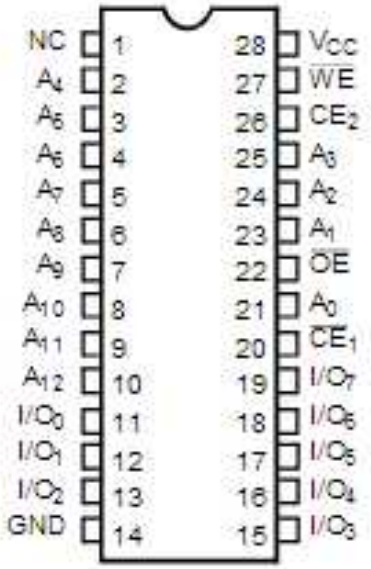


Abbildung 19: RAM mit 8k x 8 Bit

Im Gegensatz zu den ROM-Bausteinen gibt es hier keinen Vpp-, dafür einen WE-Anschluss (Write Enable). Dieser WE-Pin signalisiert dem Baustein, dass die am Datenbus anstehenden Daten eingelesen werden sollen. Die Richtung des Datentransportes wird immer aus Sicht der CPU definiert.

Das Blockschaltbild der internen Funktionsgruppen des 8k-RAM-Baustein ist

in Abbildung 20 zu sehen. Man erkennt das Speicherarray, das in 32 Spalten à 256 Zeilen aufgeteilt ist. An jeder Adresse sind 8 Bits untergebracht.

Die Anschlusspins sind zum einen mit den Ausgangstreiber (TRI-State) und zum anderen mit den Eingangspuffer verbunden. Die Ausgangstreiber werden aktiv, wenn das $\overline{OE}$ und $\overline{CE_1}$ auf Low und das CE_2 auf High liegen. Die Daten gelangen in die Eingangpuffer, wenn $\overline{WE}$ und $\overline{CE_1}$ auf Low und $\overline{CE_2}$ auf High liegen.

Liegt $\overline{CE_1}$ jedoch auf High, wird über die Power-Down-Logik der Baustein in den Stromsparmodus und damit in einen inaktiven Zustand versetzt.

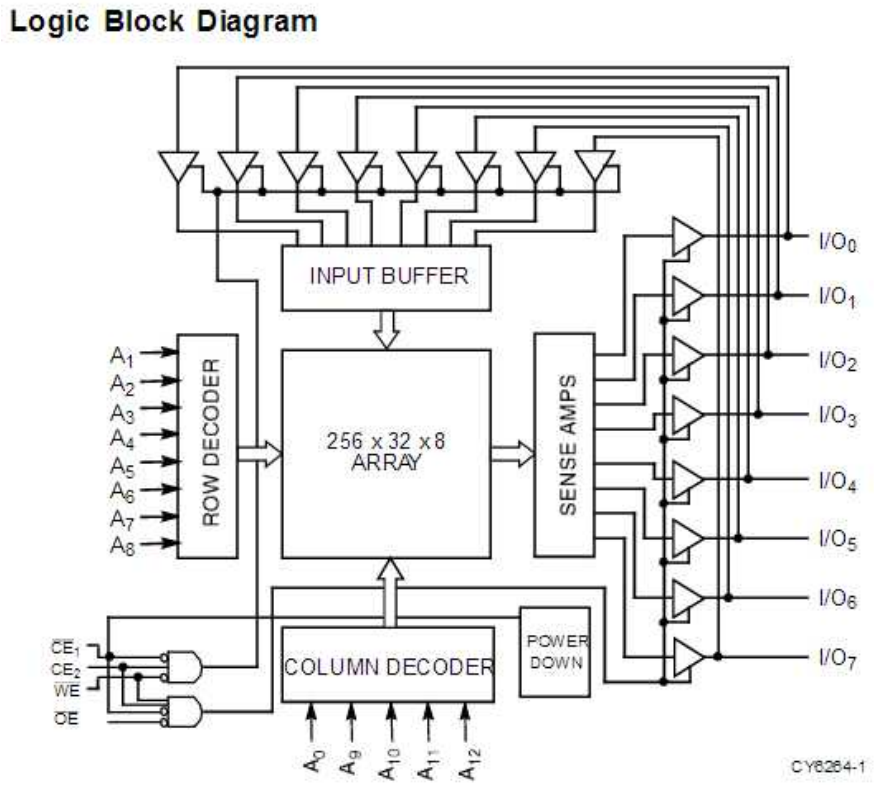


Abbildung 20: Blockschaltbild es 8k-RAM

4.3.1 Speicherelemente bei den Schreib- / Lesespeicher

Schreib- / Lesespeicher sind entweder als statische Speicher mit einem Flip-Flop oder als dynamische Speicher mit einem Kondensator als Speicherelement aufgebaut.

Flip-Flop

Ein Flip-Flop ist ein bistabiles Kippglied und besteht aus zwei Transistoren.

Früher wurden diese aus bipolaren Transistoren gefertigt. Heute werden MOS-Feldeffekt-Transistoren verwendet, die einen deutlich geringeren Stromverbrauch haben. Nähere Funktionsbeschreibung ist im Anhang 17.3 auf Seite 115 zu finden.

4.4 Das Bussystem

Das Bussystem eines Von-Neumann-Rechners besteht aus dem Adress-, dem Daten- und dem Steuer- oder Kontrollbus.

4.4.1 Der Adressbus

Seine Aufgabe des Adressbusses besteht darin, die Information des Programmzählers an die Speicherbausteine zu führen. Alle diese Speicherbausteine werden parallel angeschlossen. Der Anschluss A0 des EPROMs wird mit dem A0 des RAM 1 und des RAM 2, dem A0 von PIO und SIO sowie dem A0 der CPU verbunden. Das Gleiche gilt für die übrigen Adressleitungen. Die CPU besitzt in diesem Fall insgesamt 16 Adressleitungen. Das EPROM, die RAM-Bausteine, die PIO und SIO benötigen jedoch weniger Adressleitungen.

Der Adressbus ist i.d.R. unidirektional und wird nur von der CPU angesteuert. Allerdings kann die CPU diesen Bus in den inaktiven Zustand (hochohmig) versetzten. Dies ist z.B. bei DMA-Betrieb (Direct Memory Access) notwendig.

4.4.2 Der Datenbus

Hier handelt es sich um einen bidirektionalen Bus, d.h. die Daten können entweder von der CPU gesendet oder empfangen werden. Auch hier gilt, das die D0-Leitung die D0-Anschlüsse aller Bausteine verbindet. Entsprechendes gilt für die übrigen Datenleitungen.

Die Breite des Datenbusses bestimmt, ob es sich um eine 8-Bit, 16-Bit, 32-Bit oder 64-Bit CPU handelt.

4.4.3 Der Steuerbus

Im Steuerbus sind viele Signalleitungen unterschiedlichster Art zusammengefasst. Neben den $\overline{\text{RD}}$- (Read) und $\overline{\text{WR}}$-Signalen (Write) gibt es noch $\overline{\text{MREQ}}$ ($\overline{\text{Memory Request}}$), $\overline{\text{IORQ}}$ ($\overline{\text{IO-Request}}$), M1 (M1-Zyklus, Fetch), $\overline{\text{INT}}$ ($\overline{\text{Interrupt}}$), $\overline{\text{INTA}}$ (Interrupt Acknowledge), $\overline{\text{NMI}}$ (Nicht Maskierbarer Interrupt), $\overline{\text{RESET}}$ etc. dazu. Zu dieser Auszählung ist anzumerken, dass es sich um die Be-

zeichnungen von Intel-Prozessoren bzw. deren Abkömmlingen handelt. Andere Hersteller verwenden teilweise andere Begriffe.

Die beiden Leitungen $\overline{RD}$ und $\overline{WR}$ sind bei einigen Herstellern zu einer R/$\overline{W}$-Leitung zusammengefasst.

4.5 Der Adressraum

Der Adressraum definiert die Größe des adressierbaren Speicherbereichs, also die niedrigste und die höchste Adresse die das Steuerwerk verwalten kann. Diese Größe legt auch die Anzahl der notwendigen Adressleitungen fest.

Beispiel:
Ein Adressbus mit 16 Leitungen (A0 bis A15) kann 2^{16} = 65536 Speicherstellen adressieren. Das bedeutet es liegt ein 64k großer Adressraum vor.
Bei 24 Adressleitungen ergibt sich ein Adressraum von 16 M (≙ 16.777.216 Speicherstellen).

Bei den nachfolgenden Beispielen und Aufgaben wird von einem 64 k großem Adressraum und einer Datenbusbreite von 8 Bit ausgegangen.

Beim Von-Neumann-Rechner liegt sowohl der Programm- als auch der Datenspeicher in diesem Adressraum. Das bedeutet, der Rechner kann von sich aus nicht zwischen Programmcode und Daten unterscheiden, was von Vor- aber auch von Nachteil ist.

4.5.1 Zwei Adressräume

Manche Prozessorhersteller, wie z.B. Intel, spendieren einen zweiten Adressraum. Dieser ist nicht für Speicherbausteine sondern für die Ein- / Ausgabebausteine vorgesehen. Um die unterschiedlichen Adressräume eindeutig ansprechen zu können, bedarf es zusätzlicher Befehle. Speicherbefehle sind u.a. MOVE- oder LOAD-Befehle, während der IO-Bereich mit IN- und OUT-Befehlen angesprochen wird. Dieser zusätzliche Adressraum ist deutlich kleiner. In diesem Fall umfasst er 256 Adressen, was durch die Adressleitungen A0 bis A7 abgedeckt wird.

4.6 Der Adressdekoder

Die Speicherbausteine EPROM, RAM 1 und RAM 2 in Abbildung 15 finden

sich somit in diesem Adressraum. Angenommen RAM 1 und RAM 2 sind baugleich und haben somit die gleiche Anzahl von Speicherzellen. Es stellt sich nun die Frage, wie es sich arrangieren lässt, dass in RAM 1 andere Daten gespeichert werden können als in RAM 2 wo doch deren Adress- und Datenleitungen direkt verbunden sind. Um dieses Problem zu lösen nutzt man die $\overline{CS}$- bzw. $\overline{CE}$-Anschlüsse der Bausteine. Mit ihnen können die Bausteine individuell aktiviert werden. Auf diese Weise ist gewährleistet, dass immer nur einer dieser Speicherbausteine arbeitet.

Man ordnet die Speicherbausteine so im Adressraum des Prozessors an, dass es keine Adressenkonflikte gibt.

Das EPROM wird so angeschlossen, dass es den Adressbereich von 0x0000 bis 0x7FFF (32k) umfasst. Da bei einem RESET diese CPU ihren Programmzähler auf 0x0000 setzt, muss auch an dieser Stelle ein gültiger Befehl stehen, da der erste Fetchzugriff auf diese Adresse erfolgt. Bei anderen Prozessoren kann dies durchaus anders sein.
Falls es sich um 2k-RAM handelt, kommt das RAM 1 an die Stelle 0x8000 bis 0x87FF und RAM 2 an 0x8800 bis 0x8FFF. Wären es 8k-RAMs würden sie die Adressen 0x8000 bis 0x9FFF und 0xA000 bis 0xBFFF belegen.

Die Peripheriebausteine PIO und SIO besitzen normalerweise viel weniger Adressleitungen, da bei ihnen nur wenige Register (≙Speicheradressen) vorhanden sind. Üblich sind zwischen zwei und sechzehn Register. Für diesen Fall nehmen wir für die PIO und SIO jeweils 4 Register an. Somit könnte die PIO die Adressen 0x00 bis 0x03 und die SIO die Adressen 0x04 bis 0x07 im IO-Adressraum belegen.

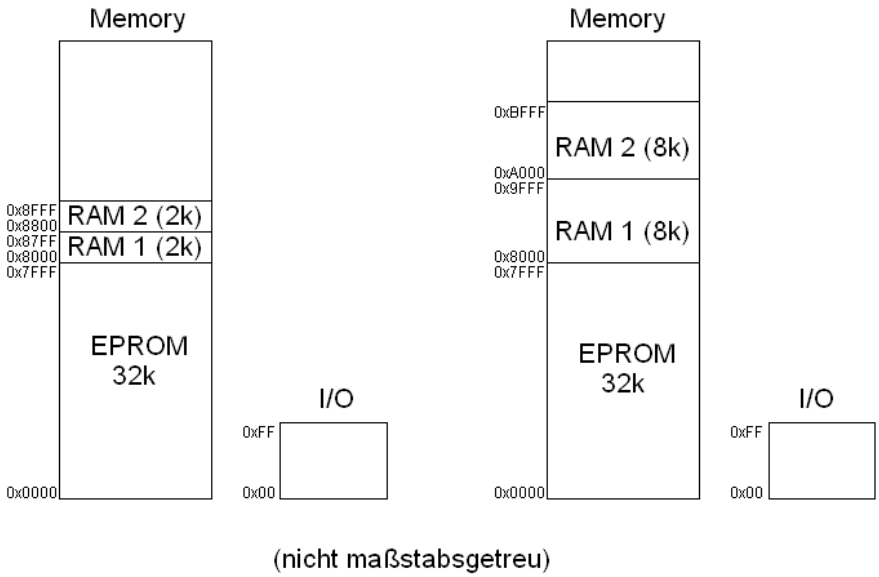


Abbildung 21: Verteilung der Speicherbausteine im Adressraum

Welcher Baustein jeweils aktiviert werden muss, hängt somit von der Adresse, auf die man zugreifen will, ab. Um den entsprechenden Baustein zu aktivieren, muss dessen $\overline{CE}$-Anschluss mit Low angesteuert werden. Diese Arbeit übernimmt der Adressdekoder.

Will die CPU auf eine Adresse im Bereich 0x0000 bis 0x7FFF zugreifen, muss der Adressdekoder seinen Ausgang, der mit dem $\overline{CE}$-Eingang des EPROMs verbunden ist, aktiv und damit auf Low schalten. Für die linke Anordnung in Abbildung 21 gilt: Greift die CPU auf Adressen zwischen 0x8000 und 0x87FF aktiviert der Adressdekoder die $\overline{CE}$-Leitung an RAM 1, beim Zugriff auf 0x8800 bis 0x8FFF die $\overline{CE}$-Leitung an RAM 2.

Um diese Aufgabe ausführen zu können, benötigt der Adressdekoder entsprechende Informationen. Dazu werden diejenigen Adressleitungen an ihn angeschlossen, die nicht an den Bausteinen verschaltet sind. Für das EPROM bedeutet dies, die Leitung A15 kommt an den Adressdekoder, da A0 bis A14 direkt angeschlossen sind. Bei den beiden RAMs sind A0 bis A10 direkt angeschlossen. Deshalb müssen A11 bis A15 an den Adressdekoder. Da A15 bereits wegen des EPROMs hier liegt, kommen noch A11 bis A14 hinzu. Außerdem muss das $\overline{MREQ}$-Signal an den Adressdekoder geführt werden, damit dieser zwischen Speicher und IO unterscheiden kann.

Für die PIO und SIO müssen A2 bis A7 zum Adressdekoder geführt werden, da A0 und A1 direkt an PIO und SIO angeschlossen sind. Damit diese Signale

auch den IO-Adressraum ansprechen können, muss die $\overline{\text{IORQ}}$-Leitung am Adressdekoder angeschlossen sein.

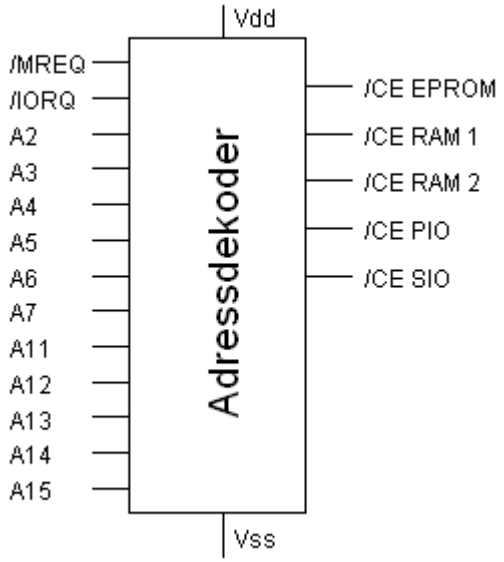


Abbildung 22: Der Adressdekoder

4.6.1 Ein spezieller Adressdekoder wird entwickelt

Solch ein Adressdekoder kann nicht einfach mittels eines simplen Demultiplexer realisiert werden, denn die zu selektierenden Adressblöcke sind, wie aus Abbildung 21 zu erkennen ist, unterschiedlich groß. Deshalb werden sogenannte Programmierbare Logikbausteine (PAL, GAL o.ä.) verwendet. Bei diesen Bausteinen lassen sich logische Verknüpfungen einprogrammieren.

Die Erstellung der Programmierdaten erfolgt mit entsprechenden Hilfsprogrammen z.B. ABEL. Hier können z.B. die Boolschen Gleichungen eingegeben werden. Daraus erstellt dann das Programm die passenden Daten. Somit benötigt man die passenden Gleichungen.

Zuerst erstellt man eine Tabelle (siehe Tabelle 1):
In diejenigen Adressspalten, deren Adresse direkt mit dem Speicherbaustein verbunden ist, wird ein x (≙ don't care). Für die Adressen, die mit dem Adressdekoder verbunden sind gilt: die Startadresse für diesen Speicherbereich kommt in binärer Schreibweise in die obere Zeile, darunter die Endadresse des Bereichs.

Position 1 in der Tabelle zeigt den Zustand, wenn sowohl $\overline{\text{MREQ}}$ als auch

$\overline{\text{IORQ}}$ auf High sind. In diesem Fall wird weder der Speicher- noch der IO-Adressraum angesteuert. Deshalb kommen in alle Adressspalten ein x. Auf der Ausgangsseite sind alle $\overline{\text{CS}}$-Signale auf 1 (high).

Pro Baustein werden zwei Zeilen verwendet. Die Einträge für das EPROM befinden sich an Position 2 und 3. Dessen Startadresse beginnt bei 0x0000, die Endadresse lautet 0x7FFF. Man erkennt, dass in beiden Zeilen nur bei der Adresse 15 die Werte gleich sind. Alle übrigen sind oben 0 und unten 1. Man kann sie deshalb auch durch ein x markieren, um die Tabelle etwas zu vereinfachen.
Auf der Ausgangsseite wird in der Spalte /CE_EPROM ($\hat{=}$ $\overline{\text{CE\_EPROM}}$) eine 0 (Baustein ist aktiv), bei den restlichen Spalten eine 1 eingetragen.

	Steuer-bus		**Eingangsseite Adressbus**																	**Ausgangsseite Selektleitungen**				
Position	**/MRQE**	**/IORQ**	**A15**	**A14**	**A13**	**A12**	**A11**	**A10**	**A9**	**A8**	**A7**	**A6**	**A5**	**A4**	**A3**	**A2**	**A1**	**A0**		**/CE_EPROM**	**/CE_RAM 1**	**/CE_RAM2**	**/CE_PIO**	**/CE_SIO**
1	1	1	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x		1	1	1	1	1
2	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		0	1	1	1	1
3	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1		0	1	1	1	1
4	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		1	0	1	1	1
5	0	1	1	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1		1	0	1	1	1
6	0	1	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0		1	1	0	1	1
7	0	1	1	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1		1	1	0	1	1
8	1	0	x	x	x	x	x	x	x	x	0	0	0	0	0	0	0	0		1	1	1	0	1
9	1	0	x	x	x	x	x	x	x	x	0	0	0	0	0	0	1	1		1	1	1	0	1
10	1	0	x	x	x	x	x	x	x	x	0	0	0	0	0	1	0	0		1	1	1	1	0
11	1	0	x	x	x	x	x	x	x	x	0	0	0	0	0	1	1	1		1	1	1	1	0

Tabelle 1: Ausführliches Beispiel einer Wahrheitstabelle für einen Adressdekoder

Position 4 und 5 gilt für das RAM 1. Dessen Bereich geht von 0x8000 bis 0x87FF. Auch hier wird die Startadresse binär an Position 4, die Endadresse an Position 5 geschrieben. Ausgangsseitig wird eine 0 in Spalte /CE_RAM1 eingetragen.

Die gleiche Prozedur gilt für das RAM 1 unter der Position 6 und 7.
Bei den IO-Bausteinen haben die Adressen A8 bis A15 von vorne herein keine Bedeutung und werden deshalb mit einem x gekennzeichnet.

Pro Baustein werden normalerweise 2 Zeilen benötigt. Lediglich dort, wo die Zuordnung sehr einfach und überschaubar ist, reicht eine Zeile. Das ist in diesem Fall die Position 2 für das EPROM. Der Adressbereich, den dieses Bauteil abdeckt reicht von 0x0000 bis 0x7FFF. Das bedeutet, lediglich das Signal an A15 entscheidet, ob das EPROM aktiv ist oder nicht. Deshalb wird bei A15 eine 0 eingetragen, der Rest der Adressspalten bekommt ein x.

Sofern man den Überblick hat, kann diese Vereinfachung auch für die übrigen Bauteile gemacht werden. Es gilt: sobald für das gleiche Bauteil sowohl eine 1 aus auch eine 0 an einer Adresse geschrieben werden kann (gelb unterlegter Bereich), ist diese Adresse don't care.

Tabelle 2 zeigt eine solche vereinfachte Tabelle.

	Steuer-bus		**Eingangsseite Adressbus**																	**Ausgangsseite Selektleitungen**				
Position	**/MRQE**	**/IORQ**	**A15**	**A14**	**A13**	**A12**	**A11**	**A10**	**A9**	**A8**	**A7**	**A6**	**A5**	**A4**	**A3**	**A2**	**A1**	**A0**		**/CE_EPROM**	**/CE_RAM 1**	**/CE_RAM2**	**/CE_PIO**	**/CE_SIO**
1	1	1	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x		1	1	1	1	1
2	0	1	0	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x		0	1	1	1	1
4	0	1	1	0	0	0	0	x	x	x	x	x	x	x	x	x	x	x		1	0	1	1	1
6	0	1	1	0	0	0	1	x	x	x	x	x	x	x	x	x	x	x		1	1	0	1	1
8	1	0	x	x	x	x	x	x	x	x	0	0	0	0	0	0	x	x		1	1	1	0	1
10	1	0	x	x	x	x	x	x	x	x	0	0	0	0	0	1	x	x		1	1	1	1	0

Tabelle 2: Kompakte Wahrheitstabelle für obigen Adressdekoder

Aus der kompakten Tabelle lassen sich nun sehr einfach die Boolschen Gleichungen ableiten. Benötigt werden lediglich die blauen und grünen Bereiche.

$\overline{CE\_EPROM} = \overline{MREQ} \wedge \overline{A15}$
$\overline{CE\_RAM1} = \overline{MREQ} \wedge A15 \wedge \overline{A14} \wedge \overline{A13} \wedge \overline{A12} \wedge \overline{A11}$
$\overline{CE\_RAM2} = \overline{MREQ} \wedge A15 \wedge \overline{A14} \wedge \overline{A13} \wedge \overline{A12} \wedge A11$
$\overline{CE\_PIO} = \overline{IORQ} \wedge \overline{A7} \wedge \overline{A6} \wedge \overline{A5} \wedge \overline{A4} \wedge \overline{A3} \wedge \overline{A2}$

$\overline{CE\_SIO}$ = $\overline{IORQ} \wedge \overline{A7} \wedge \overline{A6} \wedge \overline{A5} \wedge \overline{A4} \wedge \overline{A3} \wedge A2$
Die Eingabe in das entsprechende Umsetzungsprogramm (PAL-Assembler) muss die Schreibweise etwas angepasst werden. Statt dem ∧-Zeichen wird oft ein Multiplikationspunkt verwendet.

/CE_EPROM = /MREQ ∧ /A15
/CE_RAM1 = /MREQ ∧ A15 ∧ /A14 ∧ /A13 ∧ /A12 ∧ /A11
/CE_RAM2 = /MREQ ∧ A15 ∧ /A14 ∧ /A13 ∧ /A12 ∧ A11
/CE_PIO = /IORQ ∧ /A7 ∧ /A6 ∧ /A5 ∧ /A4 ∧ /A3 ∧ /A2
/CE_SIO = /IORQ ∧ /A7 ∧ /A6 ∧ /A5 ∧ /A4 ∧ /A3 ∧ A2

4.6.2 Unvollständige Dekodierung / Adressspiegelung

Wird in einem Mikroprozessorsystem der Adressraum nicht komplett benötigt, kann die Dekodierung vereinfacht werden. Im Extremfall ist ein absolutes Minimalsystem ausreichend wie hier in abgebildet.

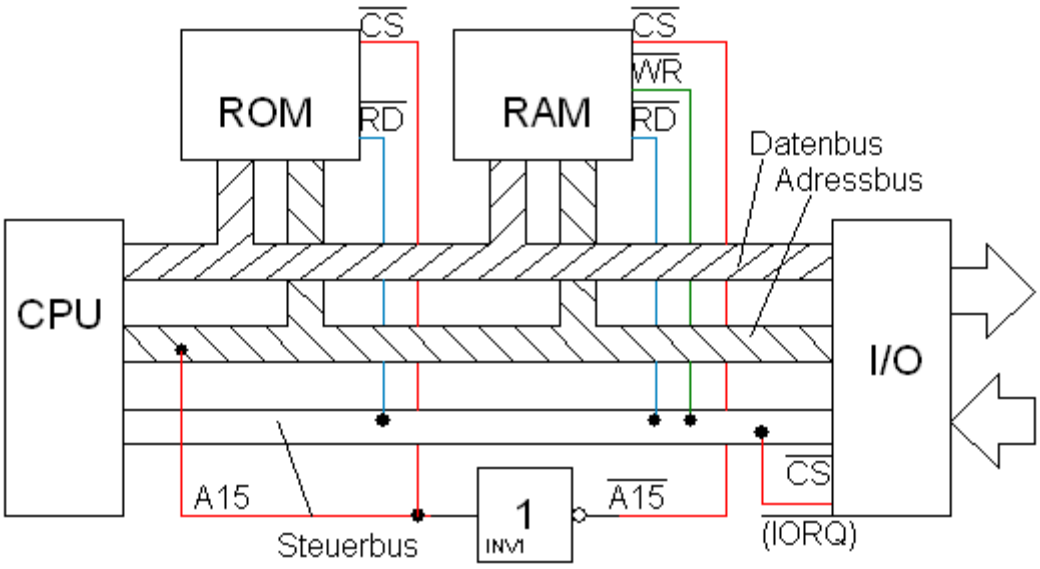


Abbildung 23: Minimalsystem (von-Neumann)

Hier besteht der Adressdekoder aus einem einzigen Inverter. Die Adressleitung führt direkt auf den $\overline{CS}$-Anschluss des ROMs. Das invertierte A15-Signal steuert den RAM-Baustein an. Somit sind alle Adressen von 0000H bis 7FFFH für das ROM, die Adressen 8000H bis FFFFH für das RAM. Das ROM darf einem maximale Größe von 32k haben, ebenso das RAM. Bei 32k großen Bausteinen sind die Adressleitungen A0 bis A14 am Baustein angeschlossen. Wird jedoch nur ein Baustein mit einer Speicherkapazität von 16k eingebaut, fehlt diesem

der A14-Anschluss. Wenn A14 nicht für den Dekoder zur Verfügung steht, dann entsteht eine unvollständige Dekodierung. Der 16k Baustein, dessen Adressbereich von 0000H bis 3FFFH reicht, wird in den Adressbereich 4000H bis 7FFFH gespiegelt. Für den Prozessor ist es dann gleichgültig, ob er z.B. auf die Adresse 0000H oder 4000H zugreift. Physikalisch werden immer die gleichen Speicherzellen angesprochen.

Wird ein 8k-ROM eingesetzt, wird dessen Speicherbereich sogar 4 mal gespiegelt: 0000H – 1FFFH, 2000H – 3FFFH, 4000H – 5FFFH und 6000H – 7FFFH.

Dasselbe gilt auch für das RAM und dessen Adressbereich ab 8000H.

4.6.3 Ein GAL als Adressdekoder

Ein GAL ist ein PAL (Programmable Array Logic) von der Firma Lattice. Es besteht aus einer Matrix wo Ein- und Ausgangssignale auf fast jede Art logisch verknüpft werden können. Eine OLMC (Output Logic Macro Cell) beinhaltet neben einem D-Flipflop auch Tristate-Ausgangstreiber.

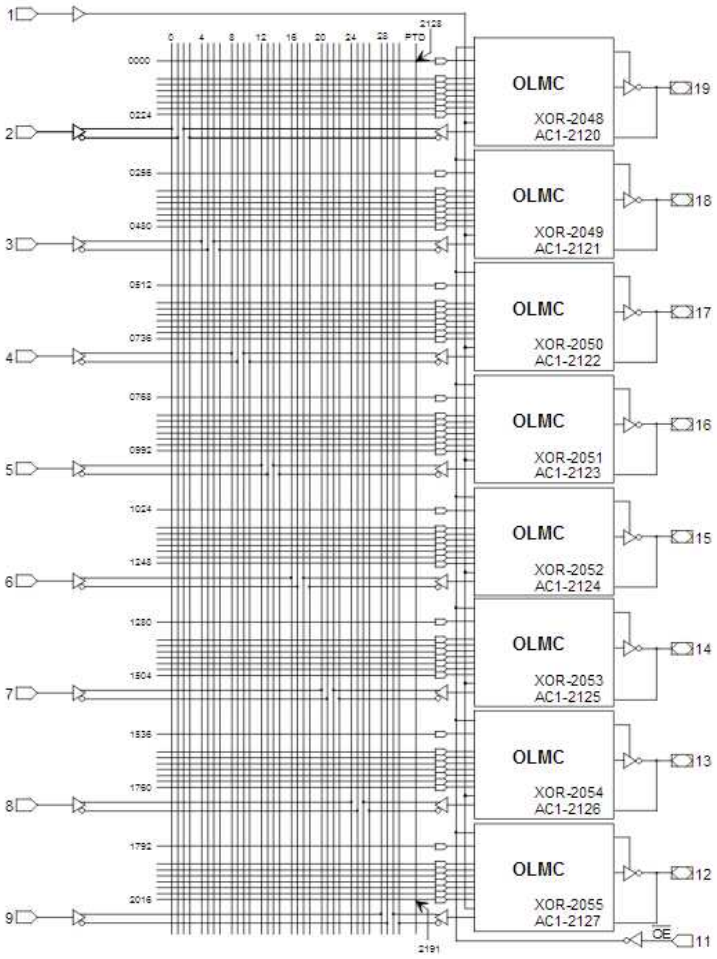


Abbildung 24: Interna des GALs 16V8

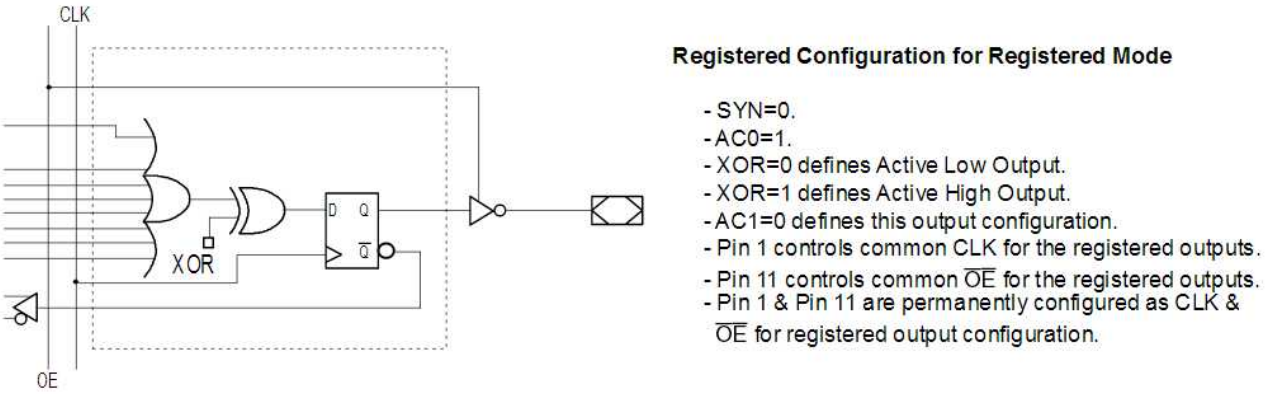


Abbildung 25: Die Ausgangs-Makrozelle des GAL 16V8

Heute werden diese GAL-Bausteine durch leistungsfähigere Bauteile, z.B. FPGAs, ersetzt.

5 Die Harvard-Architektur

Der Von-Neumann-Rechner hat leider einen sogenannten Flaschenhals. Prinzipbedingt kann er den nächsten Befehl erst laden, wenn der vorherige Befehl vollständig abgearbeitet ist. Das liegt daran, dass er Programmcode und Daten im gleichen Adressraum ablegt. Die Busleitungen werden beim Befehl einlesen und beim Daten lesen oder Daten schreiben in gleicher Weise benutzt.

Die Harvard-Architektur (Abbildung 26) besitzt einen vom Programmspeicher physikalisch getrennten Datenspeicher. Deshalb besitzt er auch zwei von einander unabhängige Bussysteme und umgeht so diesen Flaschenhals. Während der Prozessor Daten im Datenspeicher ablegt oder holt, kann es gleichzeitig aus dem Programmspeicher den nächsten Befehl laden.

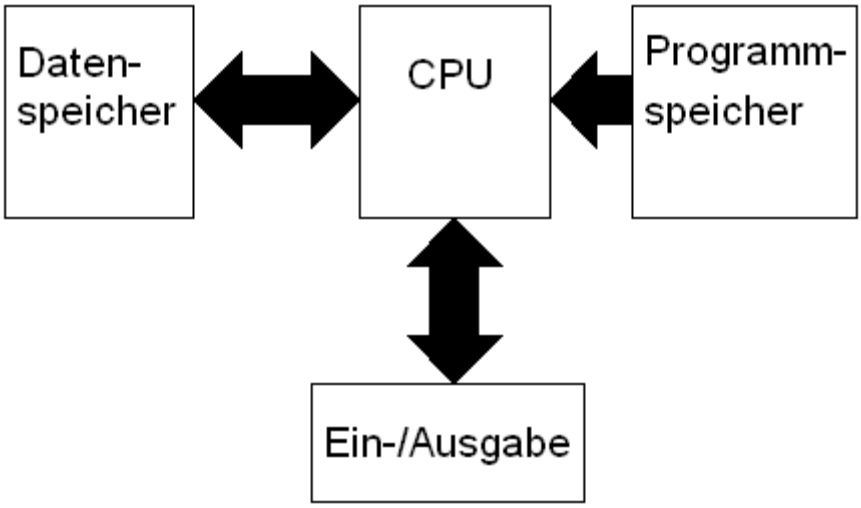


Abbildung 26: Blockschaltbild eines Harvard-Rechners

Die Fetch- und die Executephase überlappen sich hier. Das bedeutet eine praktische Verdoppelung der Ausführungsgeschwindigkeit. In Abbildung 27 kommt diese Verdoppelung nicht so richtig zur Geltung, da der Executeblock nicht zeitmaßstäblich dargestellt ist.

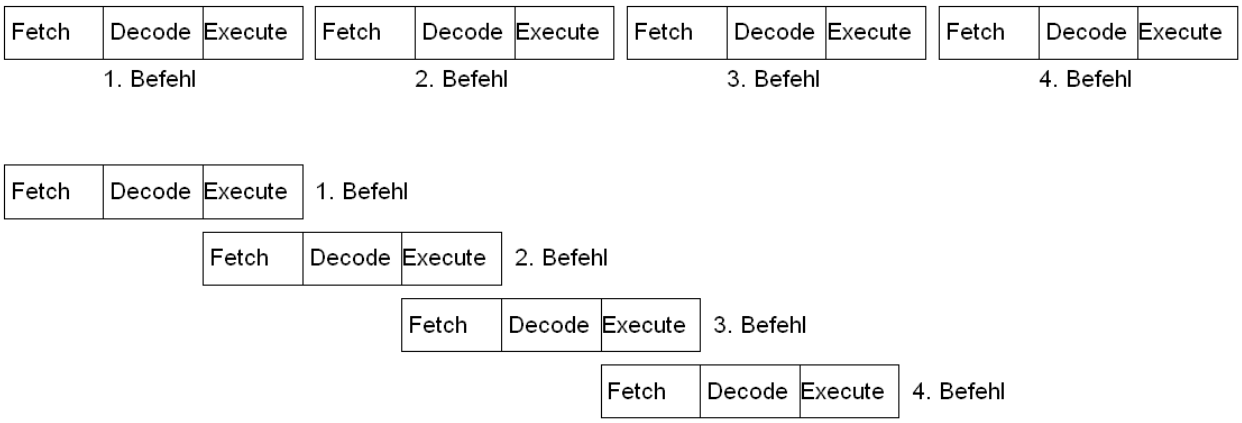


Abbildung 27: Befehlsausführung bei Von-Neumann und Harvard

Ein weiterer Vorteil den die getrennten Speicherbereiche bieten ist die Möglichkeit die Busbreiten für Programm- und Datenspeicher unterschiedlich breit auszulegen. So kann man den Datenbus für den Datenspeicher z.B. 8-Bit, den für den Programmbus 14-Bit[5] breit machen. Damit lassen sich Argumente, die beim Von-Neumann-Rechner nur mittels zusätzlichem Speicherzugriff ladbar sind, gleich beim Fetchzyklus mit laden. Das reduziert die Speicherzugriffe deutlich, was im Schnitt eine weitere Verdopplung der Ausführungsgeschwindigkeit zur Folge hat. Damit ist der Harvard-Rechner prinzipbedingt etwa vier Mal so schnell wie der Von-Neumann-Rechner.

Natürlich hat auch der Harvard-Rechner Nachteile gegenüber des Von-Neumann-Rechners. So lassen sich z.B. Konstantentabellen bei Harvard nur über Umwege im Programmspeicher, und damit dauerhaft, ablegen. Während der Von-Neumann-Rechner problemlos über direkte oder indirekte Speicherzugriffe die Daten lesen kann, muss beim Harvard z.B. Befehle wie RETLW[6] implementieren um auch Konstantentabellen, zumindest in einer einfachen Form, nutzen zu können.

5 Als Vorbild dient hier der PIC 16Fxx Mikrocontroller.

6 Den Befehl RETLW wird beim PIC dazu verwendet, um ein Unterprogramm (CALL) zu beenden und gleichzeitig das W-Registers mit einer Konstanten (Literal) zu laden.

6 Die Befehlsabarbeitung

Der Von-Neumann-Rechner und der Harvard-Rechner zählen zu den klassischen Universalrechnern mit einer dreiphasigen Befehlsabarbeitung.

Befehlholphase	bzw.	fetch cycle
Dekodierphase	bzw.	decode cycle
Ausführungsphase	bzw.	execute cycle

6.1 RESET-Zustand

Bevor der erste Befehl abgearbeitet wird, muss der Prozessor nach dem Einschalten als erstes in einen definierten Zustand versetzt werden. Hier werden einige Register mit vorgegebenen Werten initialisiert. Das sind u.a.:

- Programmzähler auf 0x0000 stellen[7]
- Ein- / Ausgangsleitungen auf Eingang (hochohmig) stellen
- Interrupts sperren

Alle anderen Register- und Speicherinhalte bleiben unverändert bzw. haben zufällige Inhalte.

6.2 Fetch

In der Befehlholphase legt der Prozessor den Inhalt des Programmzählers (Adressregister, Instruction Pointer) auf den Adressbus. Durch aktivieren der RD-Leitung und ggf. des MREQ wird eine Adresse im Adressraum selektiert und deren Inhalt wird über den Datenbus zum Befehlsdekoder (Befehlsregister) gebracht. Sobald der Speicherzugriff abgeschlossen ist, wird der Programmzähler um eins erhöht.

6.3 Decode

In der Dekodierphase wird der Befehl analysiert. Dabei müssen u.U. weitere Daten, z.B. zu ladende Konstanten bzw. Literale oder Sprungadressen, aus dem Programmspeicher gelesen werden. Bei jedem weiteren Zugriff auf den Programmspeicher, der durch das Lesen von diesen weiteren Daten verursacht wird, wird der Programmzähler inkrementiert.

7 Es gibt Prozessoren, die an einer anderen Adresse mit der Befehlsausführung beginnen.

6.4 Execute

In der Ausführungsphase wird der erkannte Befehl ausgeführt. Dazu werden die entsprechenden Register durch das Steuerwerk angesprochen und z.B. in der ALU verknüpft. Das Ergebnis wird danach gespeichert.

6.5 Adressierungsarten

Die Operanden, die ein Befehl benötigt, werden auf unterschiedliche Art und Weise implementiert. Die Beispiele beziehen sich auf den Befehlssatz des 80x86. Deshalb ist der Inhalt des Codesegmentregisters mit zu berücksichtigen.

6.5.1 Unmittelbare Adressierung (immediate)

Von einer unmittelbaren Adressierung spricht man, wenn eine Konstante direkt (unmittelbar) hinter dem Befehlscode steht. Damit können u.a. auch Register mit speziellen Startwerten beschrieben werden.
Bsp:
Mnemonic ADD AX,5 Addiere zu AX die Konstante[8] 5

ip	0x04	Opcode für 8-Bit ADD
ip + 1	0x05	Konstante

6.5.2 Absolute Adressierung (direct, absolute)

Auf den Operand wird durch eine absolute Adresse gezeigt.
Bsp:
Mnemonic MOV AX,mem Lade AX mit dem Wort an mem

ip	0xA0	Opcode für 16-Bit Ladebefehl
ip + 1	kk	Lowbyte der Zieladresse[9]
ip + 2	jj	Highbyte der Zieladresse
...	...	...
(CS * 4) + jjkk	0x12	
(CS * 4) + jjkk + 1	0x34	

In AX steht am Ende des Befehls 0x1234.

8 Konstanten werden oft auch als Literale bezeichnet
9 Der 80x86 arbeitet nach der Little Endian Zuordnung (siehe auch Kapitel 6.6)

6.5.3 Registeradressierung (register direct)

Wenn der Operand in einem Register steht, dann wird direkt im Befehlscode dieses Register adressiert. Damit unterscheiden sich die Befehlscodes mit der Wahl des gewünschten Registers.
Bsp:

Mnemonic INC CX erhöhe den Inhalt von CX um 1

	0x41	

Mnemonic INC BX erhöhe den Inhalt von BX um 1

	0x43	

6.5.4 Registerindirekte Adressierung (register indirect)

In einem Register steht die Speicheradresse, wo sich der Operand befindet oder wohin er geschrieben werden soll. Der Vorteil einer solchen Adressierung ist, dass man die Adresse berechnen kann und so z.B. Tabellen einfach bearbeiten kann.
Bsp:

Mnemonic MOV AL,[SI] Lädt in AL Inhalt von Adresse [SI]

ip	0x8A	
ip + 1	0x04	

Steht in SI die Adresse 0x1000 und dort der Wert 0x55, dann steht nach der Befehlsausführung in AL der Wert 0x55.

6.5.5 Registerindirekte Adressierung mit Postinkrement

Hier wird nach der Befehlsausführung das Adressregister, z.B. SI inkrementiert und zwar so, dass bei einem Bytezugriff um 1 erhöht wird, bei einem Wortzugriff um 2 und bei einem Langwort um 4.

6.5.6 Registerindirekte Adressierung mit Displacement

Hier wird zum Adresszeiger ein konstanter Wert addiert, bevor auf den Speicher zugegriffen wird.
Bsp:

Mnemonic MOV DX,[SI + 0x10]

ip	0x8B	
ip + 1	0x54	

ip + 2	0x10	

6.5.7 Indizierte Adressierung

Hier wird kein fester Offset als Displacement addiert, sondern der Inhalt eines weiteren Registers.
Bsp:
Mnemonic MOV CX,[SI + BX]

ip	0x8b	
ip + 1	0x08	

Steht in SI die Adresse 0x1000 und in BX der Offset 0x0010 und an der Speicheradresse 0x1010 der Wert 0x34 und an 0x1011 der Wert 0x12, steht nach Ausführung des Befehls der Wert 0x1234 im CX Register.

6.5.8 Implizierte Adressierung

Bei manchen Befehlen muss beispielsweise das Zielregister nicht angegeben werden, weil der Akku immer das Ziel ist. Auch bei PUSH und POP steht die Zieladresse grundsätzlich im Stackpointer. Dementsprechend darf diese Adressinformation im Befehlscode fehlen.
Bsp:
Mnemonic MUL BX Multipl. AX mit BX, Ergebnis in AX

6.5.9 PC-relative Adressierungen

Wenn man auf den aktuellen Programmzählerstand eine Konstante oder den Inhalt eines Registers vorzeichenrichtig addiert, lassen sich relative Sprünge ausführen. Das Programm springt dann beispielsweise bis zu 128 Adressen vor oder bis zu 126 Adressen zurück.
Bsp:
Mnemonic JMP label

ip	0xEB	
ip + 1	0x10	
...	...	
ip + 0x12		hier geht es weiter

6.5.10 Weitere Adressierungsarten

Je nach Prozessortyp können weitere Adressierungsarten vorkommen. Dies sind

teilweise Kombinationen aus den bekannten oder aber auf den Prozessor bezogene Arten.

6.6 Little Endian / Big Endian

Muss im Speicher einArgument in mehreren aufeinanderfolgenden Speicheradressen abgelegt werden, weil der Wert größer als die Datenbusbreite ist, dann lässt sich dies auf zwei grundsätzliche Arten erledigen. Wird direkt auf den Befehlscode das Low.Byte oder Low-Word abgelegt, spricht man vom Little-Endian-Format. Wird dagegen der höherwertige Teil des Arguments direkt auf den Befehlscode folgend abgelegt, handelt es sich um das Big-Endian-Format.

Bsp.

Befehl + Argument	Big Endian	Little Endian
CALL 0x1234	Befehlscode	Befehlscode
	0x12	0x34
	0x34	0x12

6.7 Befehlstypen

Man teilt die Befehle in unterschiedliche Kategorien ein.

6.7.1 Lade- bzw. Move-Befehle

Sie transportieren (eigentlich ist es nur ein Kopiervorgang) Daten von Register zu Register, von Register zum Speicher oder vom Speicher zum Register. Bei komplexeren Prozessoren können auch Daten direkt von einer Speicheradresse zu einer anderen übertragen werden (2- oder 3-Adress-Maschine).

Beispiele (µP 8080):

```
MOV     B, A        ;kopiert den Inhalt von A nach B
MOV     A,(HL)      ;lädt den Inhalt, dessen Adresse in HL
                    ;steht in den Akku.
MVI     BC, 1234H   ;lädt 0x1234 ins Doppelregister BC
```

6.7.2 Arithmetische Befehle

Hier werden zwei Werte in der ALU durch Addition oder Subtraktion verknüpft.

Größere Prozessoren haben auch Multiplikations- und Divisionsbefehle. Der Datentyp ist normalerweise ein Integer. Sollen Fließkommazahlen verarbeitet werden, sind dazu Bibliotheken notwendig. Falls diese Berechnungen zu zeitaufwändig werden, kann man auch sogenannte Floating-Point-Units (FPU) einsetzen. Sie berechnen die Aufgaben unabhängig von dem Prozessor (Co-Prozessor).

Es gibt neben den logischen Schiebe- auch arithmetische Schiebebefehle denn deren Wirkung kann auch als Multiplikation mit 2 und als Divison durch 2 genutzt werden.

Beispiel:

```
              a b c d e f g h i j k l m n o p
      SAR     a a b c d e f g h i j k l m n o
```

Das `p` an Bitposition 0 geht verloren oder kommt ins Carryflag. Die höchstwertige Bitposition wird verdoppelt.

```
              a b c d e f g h i j k l m n o p
      SAL     b c d e f g h i j k l m n o p 0
```

Das `a` an der höchsten Bitposition geht verloren oder kommt ins Carryflag.

6.7.3 Logische Befehle

Die logischen Befehle umfassen die UND, ODER, XOR und Negation. Oft ist auch ein Komplementbefehl vorhanden, was aber eine einfache XOR-Verknüpfung mit 0xFF ist.

Logische Schiebe- und die Rotationsbefehle zählt man ebenfalls zu dieser Gruppe. Dabei muss allerdings peinlichst darauf geachtet werden, ob und welche Rolle das Carry-Flag bei diesen Befehlen spielt.

```
            Cy
                a b c d e f g h i j k l m n o p
SHL         a   b c d e f g h i j k l m n o p 0

            Cy
            q   a b c d e f g h i j k l m n o p
ROTL        a   b c d e f g h i j k l m n o p q
```

Hier erfolgt die Rotation durch das Carryflag.

6.7.4 Stackbefehle

Der Stack ist eine wichtige und effektive Methode irgendwelche Werte kurzfristig zwischen zu speichern. Außerdem erlaubt ein Stack die sehr einfache Implementierung eines Mechanismus für Unterprogrammaufrufe.

Ein Stack ist ein Bereich des normalen Hauptspeichers. Die Adressierung erfolgt i.d.R. über spezielle Befehle wie z.B. PUSH, POP, oder CALL und RETURN. Darüber hinaus kann man auf diesen Bereich auch mit "normalen" Befehlen zugreifen, sofern man darf.

Erfolgt der Zugriff mittels PUSH, POP, CALL oder RETURN, dann übernimmt der sogenannte Stackpointer die Adressierung. In diesem Stackpointer steht, ähnlich wie beim Programmzähler, eine Adresse, auf die zugegriffen wird. Im Gegensatz zum Programmzähler, der nach jedem Speicherzugriff automatisch um 1 erhöht wird, wird der Stackpointer nach jedem Zugriff um 1 erniedrigt. Genauer: Ein Speicherzugriff über den Stackpointer ist i.d.R. ein Wort- oder Doppelwort (oder größer) Zugriff. Ein Wort wird im Speicher eines Byte orientierten Speichers unter zwei Adressen abgelegt und dementsprechend wird auch der Stackpointer um zwei Adressen verstellt.

Der Stack arbeitet immer nach dem sogenannten LIFO-Prinzip (Last-in, first-out). Derjenige Eintrag der zuletzt auch dem Stack abgelegt wurde, wird auch als erster vom Stack entnommen. Damit ist der Stack ein typischer sequentieller Speicher.

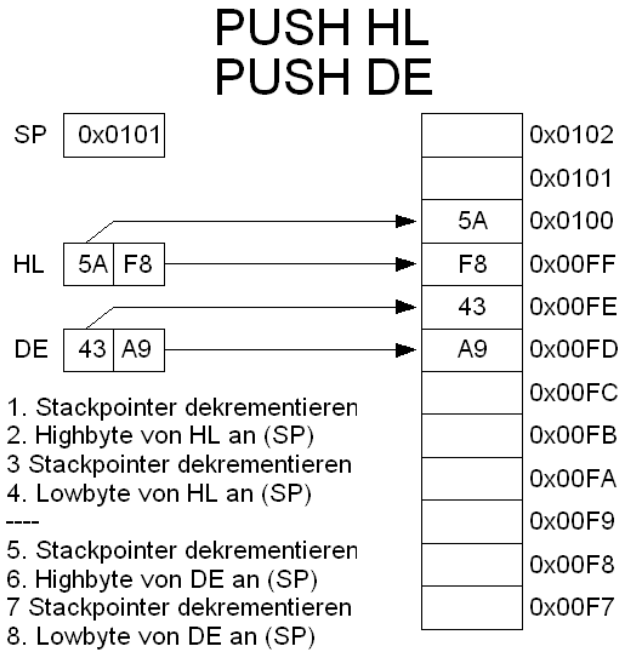


Abbildung 28: Registerpaar HL und DE auf Stack legen

Bei den meisten Prozessoren lässt sich der Stackpointer mittels einfachem Ladebefehl mit einem Wert belegen. Damit obliegt es dem Programmierer den Speicherbereich für die Stackdaten an der richtigen Stelle im Adressraum des Prozessors zu definieren.

Ein CALL-Befehl arbeitet ähnlich einem Sprungbefehl (Verzweigungsbefehl siehe dort).
Bei PUSH- und POP-Befehlen können Registerinhalte, oft auch von Registerpaaren, sehr schnell auf dem Stack zwischengespeichert werden. Dies wird i.d.R. bei Interrupts genutzt.

6.7.5 Verzweigungsbefehle

Mit diesen Befehlen wird die Programmabarbeitung an einer anderen Stelle fortgeführt. Dazu gibt es den einfachen Sprungbefehl (GOTO oder JMP) und Unterprogrammaufrufe mittels CALL. Teilweise sind die Verzweigungsbefehle auch an Bedingungen geknüpft. Dann erfolgt ein Sprung zu der anderen Adresse nur dann, wenn z.B. das Zeroflag gesetzt ist, d.h. bei einer vorangegangenen Operation war das Ergebnis 0.

Während ein GOTO die aktuelle Programmstelle für immer verlässt, sorgt ein CALL und das dazugehörende RETURN dafür, dass der Programmablauf später wieder an der alten Stelle weitermacht. Dazu wird die aktuelle Adresse im

Programmzähler - sie zeigt auf den Befehl nach dem CALL - auf dem Stack zwischengespeichert. Dann erfolgt der Sprung zur neuen Programmadresse. Sobald der RETURN-Befehl kommt, holt sich der Prozessor die zwischengespeicherte Adresse vom Stack und schreibt sie in den Programmzähler.

GOTO- und CALL-Befehle können die absoluten Zieladressen als Argumente mitbringen. Es sind dann absolute Verzweigungen. Die relativen Verzweigungen haben als Argument einen Offset dabei. Dieser Wert wird zum Programmzähler vorzeichenrichtig addiert. Bei indirekten Verzweigungen steht die Zieladresse in einem normalen Datenregister, von wo aus es in den Programmzähler verschoben wird.

6.7.6 Weitere Befehle

Es gibt weitere Befehle die nicht in die obigen Kategorien passen. Dazu gehören vor allem die Steuerbefehle. So schaltet der DI-Befehl (Diable Interrupts) die Interrupts komplett aus, der EI-Befehl (Enable Interrupts) wieder ein.

7 Interrupts

Interrupts sind zeitlich nicht vorhersagbare Unterbrechungen des laufenden Programm durch interne oder externe Ereignisse. Sie ermöglichen eine schnelle Reaktion auf unvorhersagbare Ereignisse bzw. deren Eintreffen.
Interne Ereignisse werden durch Funktionsgruppen wie beispielsweise Timer, Zähler (Capture / Compare), UART, I²C, SPI usw. generiert. Externe Hardwaremodule melden einen Interrupt über einen speziellen Interrupteingang an der CPU.

Tritt ein Ereignis ein, das einen Interrupt auslöst, muss der Prozessor den aktuellen Befehl komplett beenden. Dann erwartet er die Information, wohin im Programm verzweigt werden soll. An dieser Stelle steht die Interrupt Service Routine (ISR) mit dem der Programmierer auf dieses Ereignis reagiert. Wie zu dieser ISR verzweigt wird, hängt von der Interruptlogik ab.

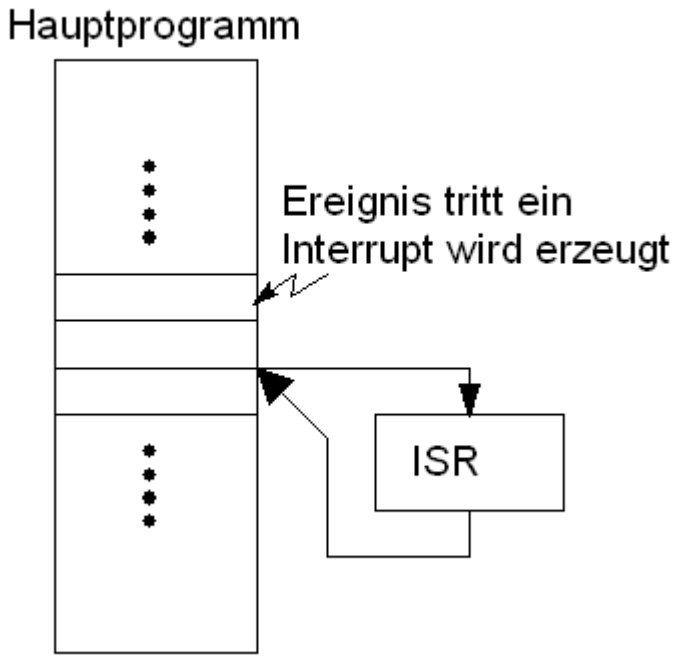


Abbildung 29: Prinzip eines Interrupts

Die Interruptlogik kann sehr einfach gestaltet sein. Dann sind die Möglichkeiten wie man sie einsetzt auch mehr oder weniger begrenzt. Sind die Möglichkeiten der Interruptnutzung sehr flexibel, zeigt sich dies auch in einer entsprechend komplex aufgebauten Logik.

7.1 Interruptkonzepte

Es gibt im Grunde vier grundlegende Interruptkonzepte.

a) Sprung an eine feste Programmadresse
b) Jede Interruptquelle hat ihre eigene feste Interrupt-Einsprungadresse
c) Jede Interruptquelle zeigt auf einen Tabelleneintrag mit der entsprechenden Einsprungadresse (Vektorinterrupt)
d) Bei einem Interrupt wird ein spezieller Befehl eingelesen[10]

7.1.1 Sprung an eine feste Adresse

Das einfachste ist der Sprung an eine fest vorgegebene Adresse. Egal welcher Interrupt ausgelöst wird, das Programm verzweigt immer nur an diese eine bestimmte Adresse. Dort muss dann herausgefunden werden welches Funktionsmodul den Interrupt generiert hat. Dieses Konzept sieht eine Verschachtelung von Interrupts nicht vor.

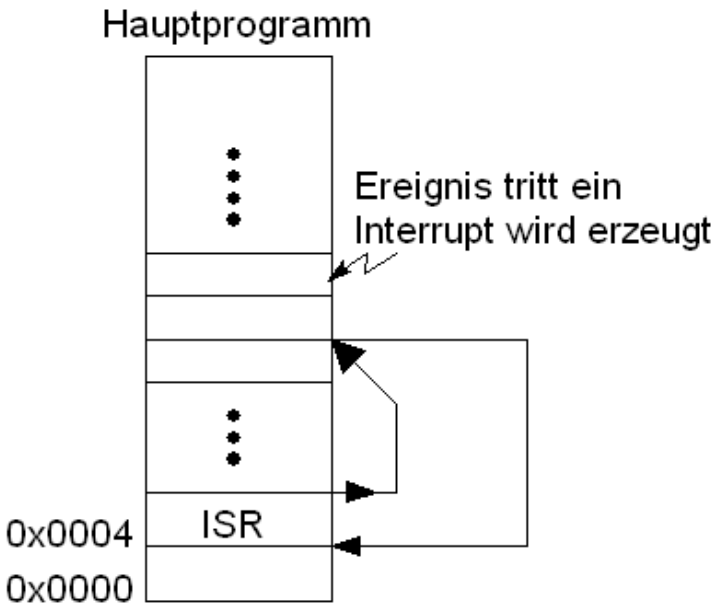


Abbildung 30: Einfachste Interruptlogik

7.1.2 Jede Interruptquelle hat eine eigene feste Einsprungadresse

Um sich das Suchen der auslösenden Interruptquelle in der ISR zu umgehen, sind bei diesem Konzept den Interruptquelle unterschiedliche Einsprungadres-

10 Dieses Konzept ist sehr alt und nur mit Hardwareaufwand realisierbar. Es wird deshalb heute nur selten eingesetzt. Fußnotentrennlinie über Format → Seite → Fußnote änderbar

sen zugeordnet. Eine weitere Erhöhung der Flexibilität erreicht man, indem man die Einsprungadressen frei wählen kann. Hier können auch Interrupts verschachtelt werden. Ein Interrupt mit hoher Priorität kann einen mit niedrigerer unterbrechen. Das setzt auch einen entsprechend großen Stackspeicher voraus.

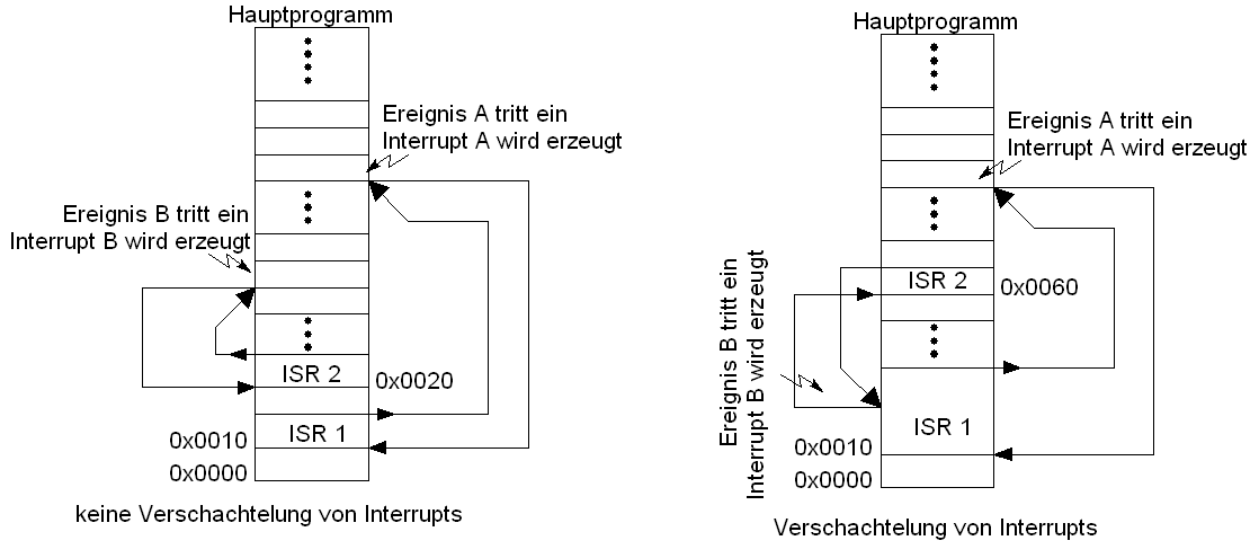


Abbildung 31: Unterschiedliche Interrupt Einsprungadressen

7.1.3 Vektorinterrupts

Eine weitere Steigerung der Flexibilität wird bei den Vektorinterrupts erreicht. Hier steht im Datenspeicher eine Tabelle mit den Adressen der unterschiedlichen Interrupt Service Routinen. In den Interrupt auslösenden Funktionsmodulen wird der entsprechende Zeiger auf die Tabelleneinträge hinterlegt. Im Falle eines Interrupts meldet die CPU ein INTA sobald sie den Interrupt erkannt und den aktuellen Befehl abgearbeitet hat. Dieses Hardwaresignal INTA veranlasst die Interruptquelle den Zeiger auf den Datenbus zu legen, von wo ihn die CPU einliest. Die CPU greift über diesen Zeiger auf die Tabelle und holt von dort die eigentliche Einsprungadresse für die ISR. Danach führt sie den Sprung aus. Dieser Sprung wird in Form eines Unterprogrammaufrufs (CALL) durchgeführt, da die Adresse, an der sich die aktuelle Programmausführung befindet, auf den Stack abgelegt werden muss.

Bild

7.1.4 Interruptaufruf mit RST-Befehl

In den Anfangszeiten der Mikroprozessoren wurde bei einem Interrupt-Acknowledge (INTA) vom Sender ein Befehl auf den Datenbus gelegt. Die CPU machte dann einfach einen neuen Fetchzyklus um den nächsten Befehl auszu-

führen. Dieser kam aber nicht aus dem Programmspeicher sondern eben vom Interruptsender. Um dies kompakt zu realisieren zu können, hat man spezielle kurze CALL-Befehle entwickelt. Diese hießen beispielsweise RESTART-Befehle (RSTx). So gab es den RST0 der einen CALL auf Adresse 0 durchführte oder einen RST10 der zur Adresse 0x0010 verzweigte. Diese RST-Befehle hatten die Zieladresse implizit im Befehlscode enthalten. Weitere Adressangaben konnten so entfallen.

Bild:

7.2 Die ISR

Die Interrupt Service Routine muss natürlich den Umstand berücksichtigen, dass deren Aufruf zu irgendeinem Zeitpunkt und von irgendeiner Adresse des Programms aufgerufen wird. Es kann somit durchaus vorkommen, dass das Hauptprogramm gerade eine komplizierte Formel o.ä. bearbeitet. Daher sind die Register und Hilfsvariablen mit Daten belegt, die nach der Abarbeitung des Interrupts wieder zur Verfügung stehen müssen. Daher werden beim Eintritt in die ISR alle diejenigen Register gesichert, die die ISR selbst verändert. Diese Registersicherung erfolgt i.d.R. durch deren Ablegen auf dem Stack. Wo kein genügend großer Stack zur Verfügung steht, müssen die Werte in einem dafür vorgesehenen Datenspeicher gesichert werden.

Verschachtelte Interrupts
Bei verschachtelten Interrupts müssen Prioritäten definiert werden. Auf diese Weise kann sichergestellt werden, dass ein wichtiger Interrupt auch dann sofort bearbeitet wird, wenn sich gerade ein weniger wichtiger in Bearbeitung befindet. Z.B. Unterbricht ein Timer-Interrupt einen UART-Interrupt. Auf diese Weise ist eine exakt laufende Uhr gewährleistet, während Daten in der UART nur „langsam“ einlaufen.

Die Prioritäten zu definieren ist nicht immer einfach und erfordert eine genaue Abschätzung wie oft und wie lange die einzelnen Interrupts in ihrer Bearbeitung sind. Oft lassen sie kurze, schnelle ISR öfters abarbeiten wie lange, die sehr oft kommen. In jedem Fall muss man vermeiden, dass bei der Abarbeitung eines Interrupts dieses Ereignis nochmals eintritt bevor die ISR beendet wird. Sonst gehen u.U. einzelne Ereignisse verloren.

Wichtig!
Der Einsatz von Interrupt sollte nicht ausufern. Jeder Interrupt der ausgelöst

wird sorgt wohl dafür, dass das Ereignis zeitnah bedient wird, aber bremst dadurch das Hauptprogramm aus. Dabei wird nicht nur die Zykluszeit beeinflusst sondern auch das Sichern und Zurücksichern der im ISR verwendeten Register und Variablen Code vergrößert. Ereignisse, die langsam, im Vergleich zum Programmdurchlauf, können i.d.R. per Polling erfasst werden.
Auch sollten Interrupt Service Routinen so kurz wie möglich gehalten werden. Das verhindert u.a. dass ein weiterer Interrupt eintrifft, während der alte noch nicht abgearbeitet ist. Der neue geht dann i.d.R. verloren.

8 Die Intel-CPU 8088 / 8086

8.1 Registersatz des 8086

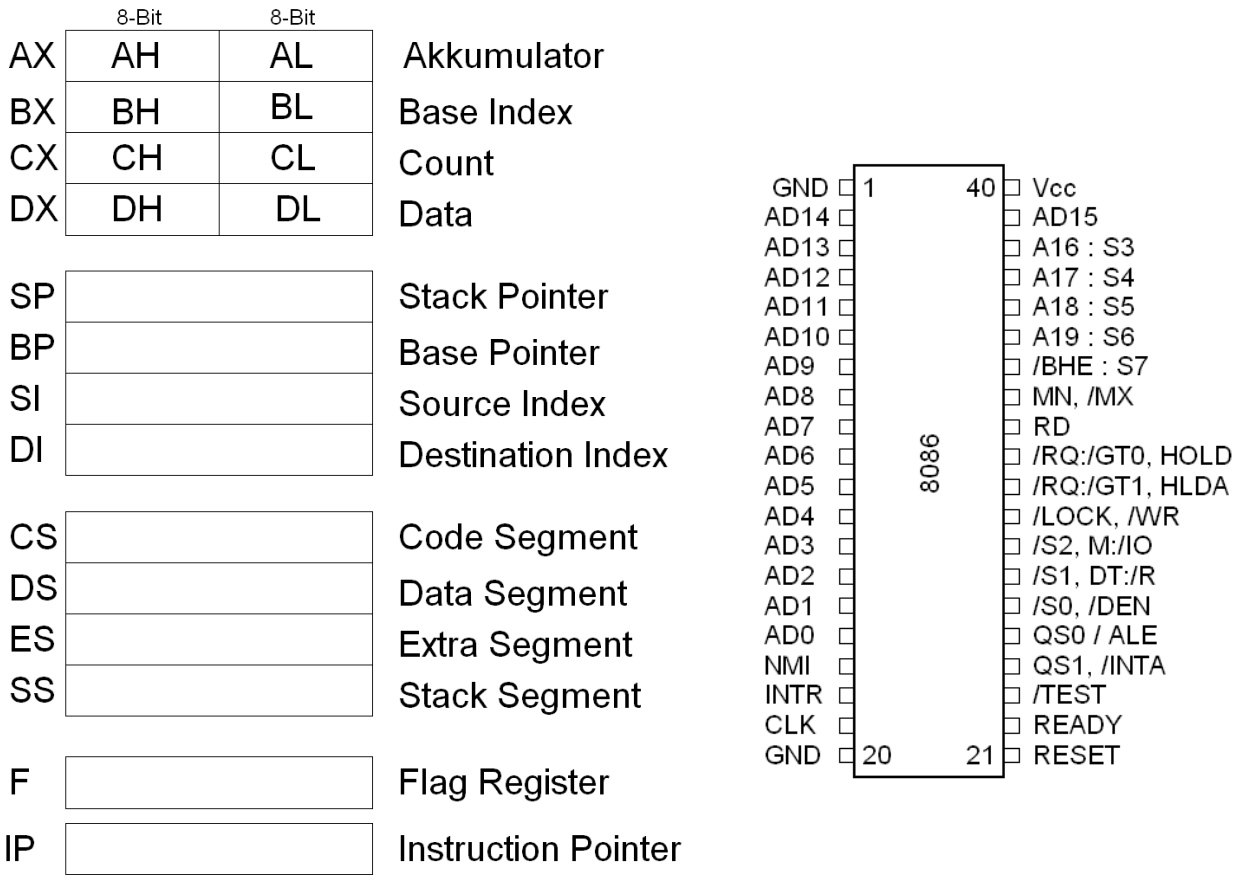


Abbildung 32: Registermodell des 8086 / 8088

Der 8088 hat die gleiche Registerstruktur wie der 8086, hat aber nur einen 8-Bit breiten Datenbus nach außen. Damit konnten einfache 8-Bit breite Speicherbausteine verwendet werden. Ein 16-Bit Zugriff erfolgte unter zwei Schritten. Der Adress- und Datenbus sind auf die gleichen Pins herausgeführt. Dieses Multiplexen hat einen zusätzlichen Hardwareaufwand zur Folge. Beim 8086 wird D0 bis D15 mit A0 bis A15 gemultiplext.
Beim 8088 ist es nur D0 bis D7 mit A0 bis A7.

Mit dem Eingang MN/$\overline{MX}$ kann der Prozessor zwischen einem Minimum- und einem Maximummodus umgeschaltet werden. Dies ist für Mehrprozessorsysteme von Interesse.

Die CPU besitzt interne Register. Inhalte dieser Register können in der ALU

arithemtisch und logisch verknüpft werden. Andere dienen als Zeiger um bestimmte Speicherstellen zu adressieren.

Die 16-Bit Register AX, BX, CX und DX sind aus jeweils zwei 8-Bit Register zusammengesetzt. Das AX besteht aus AH und AL (AH = obere 8 Bit von AX, AL = untere 8 Bit von AX). Das Gleiche gilt für BX (BH, BL), CX (CH, CL) und DX (DH, DL).

Die Index- und Zeigerregister sind 16-Bit breit. Es sind dies der Stackpointer SP, der Basiszeiger BP, der Quellindex SI (Source Index) und der Zielindex DI (Destination Index). Mit diesen Befehlen lassen sich vor allem Schleifen zum Zählen, Vergleichen und Verschieben sehr einfach realisieren.

Beispiel:
Es soll die Länge eines Strings (Zeichenkette) ermittelt werden.

```
          MOV       AH,DELIMITER   ;Endezeichen laden
          MOV       SI, STRG_ANF   ;Anfang der Zeichenkette
          MOV       DX,0FFFFH      ;Zählregister (Ergebnis)
LOOP      INC       DX
          MOV       AL,[SI]        ;Zeichen aus String holen (indirekt)
          CMP       AH,AL          ;beide Zeichen Vergleichen
          JNZ       LOOP           ;zu LOOP, wenn nicht gefunden
          RET
```

Die Segmentregister dienen zur Positionierung des 64k großen Speichersegments innerhalb des 1 M großen Adressraums (= 20 Adressleitungen). Es sind dies das Codesegment CS, Datensegment DS, Extrasegment ES und das Stacksegment SS. Da mit den internen 16-Bit Register nur 64 k direkt angesprochen werden können, der 8086 aber 20 Adressleitungen besitzt und damit 1 M ansprechen kann, wird die tatsächliche Adresse berechnet. Dazu wird z.B. der Inhalt des CS mit 16 multipliziert und dann zum Inhalt des Programmzählers hinzuaddiert.

Beispiel:
IP = 1234H
CS = 0005H

angesprochene Adresse = 16 x 0005H + 1234H = 1284H

Das 64 k große Segment lässt sich somit in Schritten von 16 Adressen verschieben.

Wenn beispielsweise das Segmentregister die Adresse 1234hex und das Offset-Register die Adresse 5678hex enthält, was meist als 1234:5678 geschrieben wird, dann wird im Speicher auf die Adresse 1234hex × 10hex + 5678hex = 12340hex + 5678hex = 179B8hex zugegriffen.

Als Kontrollregister sind beim 8086 der Befehlszähler bzw. Instruction Pointer IP und das Statusregister vorhanden. Der Inhalt von IP wird aber immer mit dem Inhalt von CS verrechnet bevor der eigentliche Speicherzugriff erfolgt. IP kann nur mit Befehlen wie JMP, CALL oder anderen Sprungbefehlen verändert werden.

Im Statusregister (Processor Status Word PSW) sind folgende Flags vorhanden:

Statusbits	**Flags**	**Bedeutung**
15	-	
14	-	
13	-	
12	-	
11	OF	Overflow
10	DF	Direction
9	IF	Interrupt Enable
8	TF	Trap
7	SF	Sign
6	ZF	Zero
5	-	
4	AF	AL-Carry
3	-	
2	PF	Parity
1	-	
0	CF	Carry

CF, PF, AF, ZF, SF und OF werden durch arithmetische und logische Operationen beeinflusst. Die übrigen Bits können durch Assembleranweisungen gesetzt oder gelöscht werden. Sie dienen zur Steuerung der Interrupts, für die Nutzung des Testmodus und zur Richtungsvorgabe bei der Verarbeitung von Strings.

8.2 Befehlsatz

Wie bereits erwähnt besitzt der 80x86 Prozessor nicht nur Befehle für die Verarbeitung von 16-Bit breiten Daten[11] sondern aus Gründen der Abwärtskompatibilität auch Befehle für 8-Bit Daten. Dabei werden die Register in ein High- und Lowbyte aufgeteilt. Zum Beispiel besteht der 16-Bit Akkumulator AX aus den beiden 8-Bit Akkus AH und AL.

8.2.1 8-Bit Befehle

Sie dienen zur Abwärtskompatibilität. Software aus alten 8-Bit Zeiten brauchten nur noch neu übersetzt werden. Auf diese Weise konnte bei der Markteinführung des 8086 auf ein großes Repertoire an Software zurückgegriffen werden.

Beispiel:

```
MOV AL,1        ;lade den unteren Teil von AX mit 1
MOV AH,2        ;lade den oberen Teil von AX mit 2
```

8.2.2 16-Bit Befehlen

Die obige Befehlssequenz lässt sich auch durch

```
MOV AX,0102H
```

ersetzen.

11 Inzwischen hat man die Datenbusbreite auf 32- bzw. 64-Bit erweitert.

9 Speicherorganisation

9.1 Segmentierter Speicher

Die Größe des physikalische Adressraums wird durch die Anzahl der Adressleitungen bestimmt. Bei 16 Adressleitungen sind somit 65536 (≙ 64k) Speicheradressen ansprechbar, bei 32 Leitungen ist der Adressraum 4.294.967.296 groß, was 4 GB entspricht. Beim 8086 ist der Adressraum 1 MB groß.

Dieser Adressraum kann man in Segmente einteilen. Dazu dienen sogenannte Segmentregister, die die Basisadresse für das entsprechende Segment beinhalten. Der Programmzähler und das Codesegment ergeben dann zusammen die absolute Zugriffsadresse.

Durch die Segmentierung erreicht man eine einfache Trennung zwischen unterschiedlichen Task bzw. Benutzern. Dabei wird jeder Task / User ein bestimmtes Segment zur Verfügung gestellt, das ausschließlich für ihn bestimmt ist. Es ist auch möglich, die Segmente teilweise zu überlappen, um so einen gemeinsamen Speicherbereich für die einzelnen Tasks bereit zu stellen. Das Umschalten zwischen den Tasks erfolgt indem man die entsprechenden Segmentregister mit den Basisadressen beschreibt.

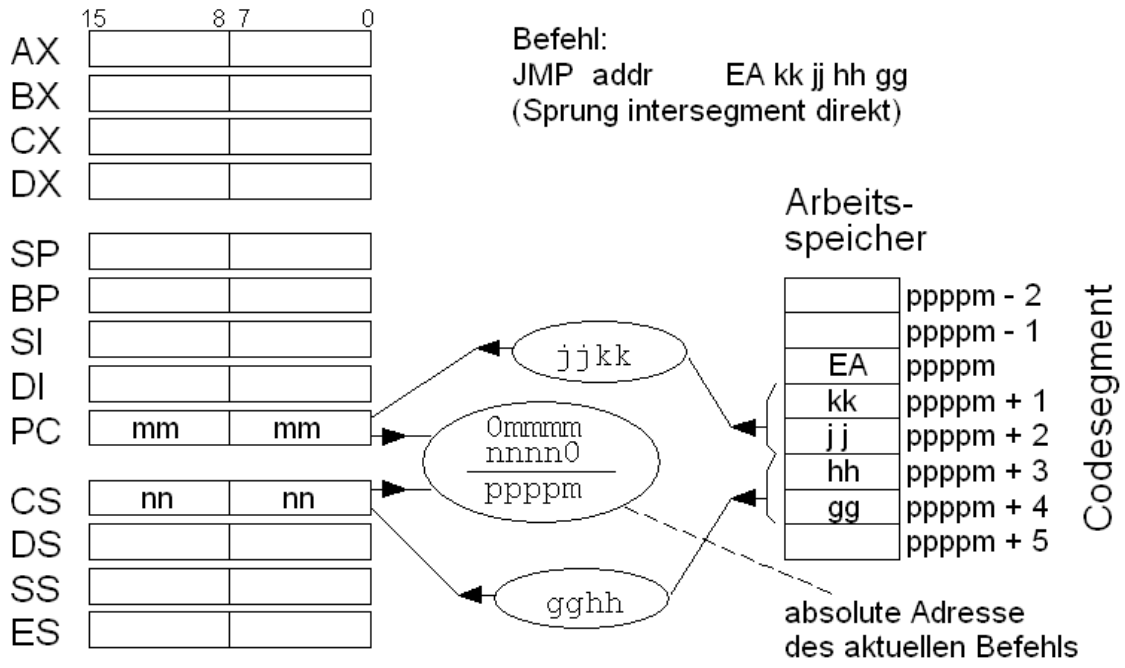


Abbildung 33: Ein einfacher Sprungbefehl beim 8086-Prozessor

Abbildung 33 zeigt einen einfachen, absoluten Sprungbefehl des 8086-Prozessors. Die aktuelle Adresse des Sprungbefehls 0xEA wird durch das Codesegment und den Programmzähler bestimmt. Dabei wird das Codesegment um vier Bits nach links verschoben (nnnn0). Zu diesem Wert wird der Programmzähler (0mmmm) addiert. So entsteht eine 20-Bit breite Adresse die auf das Befehlsbyte zeigt. Zwei der nachfolgenden Argumente überschreiben den Inhalt des Programmzählers, die nächsten zwei kommen in das Codesegment. Mit diesem Sprungbefehl kann somit der gesamte Adressraum dieses Prozessors erreicht werden.

9.2 Speicherbank

Soll ein größerer Speicher, als der physikalische Adressraum umfasst, verwaltet werden, greift man zur Bankumschaltung. Dazu gibt es, neben dem eigentlichen Programmzähler, ein sogenanntes Bank-Selekt-Register (BSR).

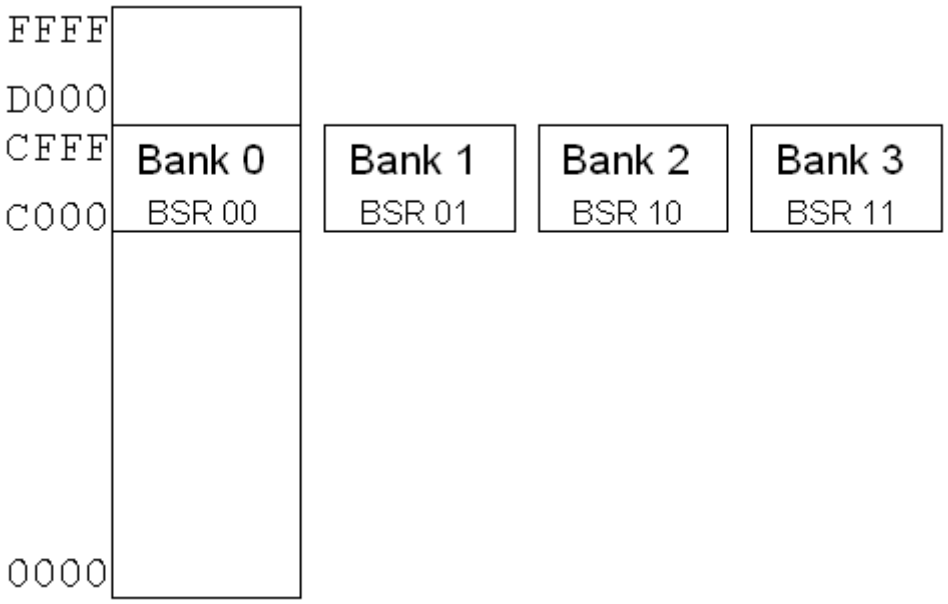


Abbildung 34: Speichererweiterung durch Bankumschaltung

Abbildung 34 zeigt wie der 4k große Speicherbereich zwischen 0xC000 und 0xCFFF durch vier verschiedene Speicherbänke ausgewählt werden kann. Solche Techniken wurden bei den älteren PCs für den Bildschirmspeicher angewendet.

Das Segmentregister definiert die Adresse, an der der Adressraum des Programmzählers "eingeblendet" wird. Bei manchen Prozessoren muss die Um-

schaltung vom Programmierer bzw. dem Betriebssystem aktiviert und auch deaktiviert werden.

Ein besonders einfaches Banking liegt beim PIC 16F84 Mikrocontroller vor. Hier ist der Datenspeicher in 2 Bänke aufgeteilt. Man kann entweder auf Bank 0 oder Bank 1 zugreifen. Welche Bank angesprochen werden kann, entscheidet das RP0-Bit im Statusregister. Ist es 0, ist Bank 0 aktiv, ist es dagegen 1, erfolgt der Zugriff auf Bank 1. Die Adressen sind in beiden Fällen 0x00 bis 0x7F. Wird Adresse 6 adressiert, wird bei RP0 = 0 das Port B, bei RP0 = 1 das Tris B angesprochen.

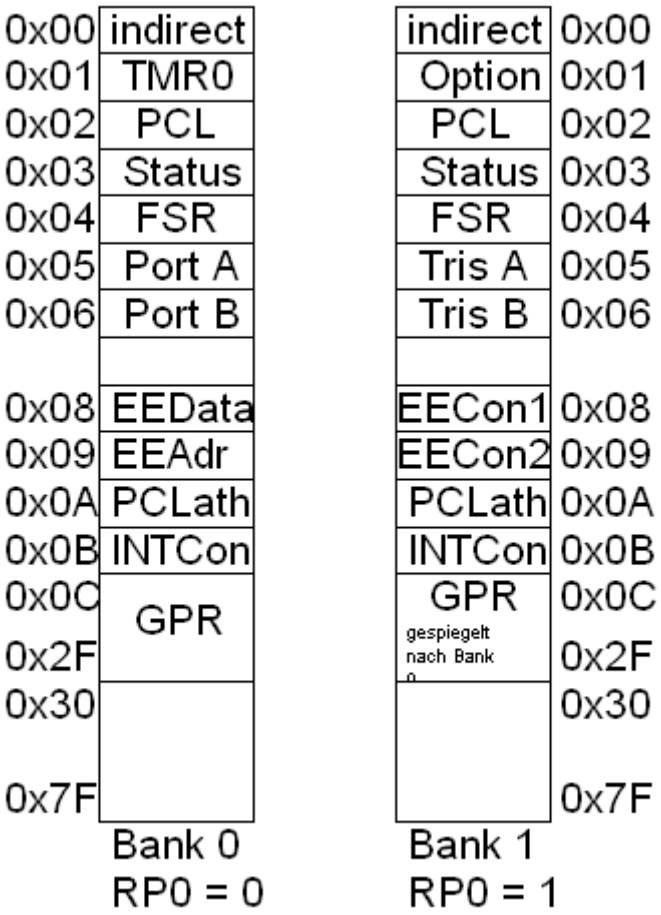


Abbildung 35: Die Registerbänke des PIC 16F84

9.3 Paging

Das Paging wird für das Umschalten bei Multitasking-Systemen genutzt. Im Prinzip ist auch hier der Speicher in entsprechende Blöcke aufgeteilt. Bei einem Kontextwechsel (Task Umschaltung) werden dann nur noch die entsprechenden

Register umgeladen.

Segmentierung und Paging können von einer MMU (Memory Management Unit) automatisch durchgeführt werden. Sie übernimmt die Umsetzung von logischen in physikalische Adressen.

9.4 Cache

Der Cachespeicher ist ein schneller, direkt an der CPU platzierter Speicher. Greift die CPU direkt auf den externen Speicher zu, so müssen die Signale über lange Verbindungsleitungen vom Speicher-IC über den Frontside-Bus zur CPU transportiert werden. Je größer die Entfernung, desto länger dauert dieser Vorgang.
Der Cache ist so aufgebaut, dass ihn die CPU nicht als solchen Zusatzspeicher erkennt. Er ist für die CPU völlig transparent. Die Verwaltung übernimmt eine übergeordnete Logik.

Greift die CPU auf eine Adresse zu, prüft diese Logik, ob diese Adresse bereits im Cache steht. Falls nicht - man spricht dann von einem MISS-Zugriff -, wird der entsprechende Speicherbereich in den Cache geladen. Ist der Bereich im Cache vorhanden, hat man einen HIT vorliegen.

In einer Tabelle werden die Speicherbereiche die sich im Cache befinden verwaltet.
...

10 Pipeline

Um die Geschwindigkeit er Prozessoren weiter zu erhöhen, wurden u.a. Konzepte zur Parallelisierung eingeführt. Dabei schafft man die Möglichkeit, zwei oder mehrere Befehl gleichzeitig abarbeiten zu lassen. Eine Möglichkeit sind Pipelines. Es handelt sich hier um komplette Funktionsgruppen aus Speicher, Dekoder und Ausführungseinheit. Eine übergeordnete Instanz versucht nun die Befehle so auf die Pipelines zu verteilen, dass sie möglichst gleichmäßig und ohne gegenseitige Abhängigkeiten arbeiten können.

Die Befehle für die Pipelines stammen aus dem Cache. Aber auch die Daten, die verarbeitet bzw. danach wieder abgespeichert werden, liegen hier.

Bei der 80x86-Prozessorfamilie gibt es dafür eine U- und eine V-Pipeline. Da man erkannte, dass bestimmte, vor allem komplexe Befehle weniger oft ver-

wendet werden als die einfachen Grundbefehle, hat man sich entschlossen das Konzept der vollständigen Ausführung der Befehle mittels Mikrosequenzer wieder zu verlassen. Ein Grund war auch die etwas geringere Ausführungsgeschwindigkeit dieser Grundbefehle verglichen mit einer fest verdrahteten Logik. Nun können diese Grundbefehle sowohl mittels Mikrosequenzer der ALU der U-Pipeline als auch in der V-Pipeline verarbeitet werden. In der V-Pipeline steckt somit keine Mikrosequenzer mehr, sondern die „alte“ fest verdrahtete Logik. Damit ergibt sich für die Befehlsabarbeitung folgendes Schema:

- einfache Befehle kommen in die U- oder V-Pipeline
- komplexe Befehle müssen in die U-Pipeline

Somit kann es zu unterschiedlich langen Ausführungszeiten des ein und des selben Befehls kommen, je nach dem in welcher Pipeline er abgearbeitet wird. Die Logik für die Befüllung der Pipeline ist aber so ausgerichtet, dass die jeweilige Pipeline optimiert beladen wird.

10.1 Prefetch

Während die CPU einen Befehl abarbeitet, wird der nächste Befehl in den Puffer oder die Pipeline geladen (Prefetch: vorholen). Für den Fall, dass der aktuelle Befehl einen Sprung auslöst, werden die Daten des nächsten Befehls verworfen. Das Einlesen des nächsten Befehls war nutzlos. Da Sprunganweisungen in einem Programm im Vergleich zu anderen Befehlen eher selten vorkommen verbessert dieser Vorgriff die Performance.

10.2 Sprungvorhersage

Nach einem bestimmten Algorithmus und mit einer „Branch History Table“ versucht der Prozessor vorherzusagen, ob ein bedingter Sprung ausgeführt wird oder nicht. Die Abarbeitung kann dann evtl. schon entsprechend spekulativ fortgesetzt werden. Eine gute „Branch Prediction“ ist bei mehreren Pipelinestufen essentiell, weil bei falscher Vorhersage die gesamte Pipeline gelöscht und neu gefüllt werden muss. Es geht nur um bedingte Sprünge z.B. in Schleifen. Wenn es nicht im Cache bereits liegt, sorgt er dafür , dass es im Vorfeld nachgeliefert wird.

Branch History Table
Diese Tabelle dient der Branch Prediction zur Vorhersage der Sprünge. Hier werden Vorhersagen und Ergebnisse gespeichert und für die nächsten Vorhersa-

gen ausgewertet. Wenn ein Sprung durchgeführt wurde, dann wird die Sprungadresse in der Branch-History Table abgelegt. Bei einer erneuten Sprungvorhersage prüft er erst seine Branch-History-Table. Dabei kann er einen Treffer landen (d.h. er findet die Infos für den Sprung, da dieser evtl. schon einmal ausgeführt wurde (HIT=Treffer) oder eben nicht (MISS=kein Eintrag)

Hit: Sprungadresse in Tabelle (Branch-History-Table)
Miss: Adresse steht nicht drin→ muss im Programm selber prüfen, wohin der Sprung gehen soll, dann trägt er die Sprungadresse in die Tabelle ein (überschreibt bei Bedarf ältere Adressen, wenn die Tabelle überfüllt werden würde!)
→ Vorteil Tabelle: wesentlich näher → daher schneller

11 DMA

Sollen große Datenmengen z.B. von einem externen Speicher in den RAM-Speicher eines Rechners geladen werden, so wäre das Lesen von Byte für Byte durch die CPU und das anschließende Abspeichern ein sehr zeitaufwändiges Verfahren. Deshalb hat man eine Methode entwickelt, in der das externe Gerät direkt auf den Hauptspeicher der CPU zugreifen kann. Diese Methode nennt sich Direct Memory Access (direkter Speicherzugriff).

Hat beispielsweise die Festplatte Daten für die CPU, dann wird dies der CPU mitgeteilt. Daraufhin gibt die CPU den Adress-, Daten- und Steuerbus für den Zugriff durch die Festplatte frei. Die CPU ist i.d.R. dann mit internen Abläufen, ohne Speicherzugriff, beschäftigt. Die Festplatte adressiert nun den entsprechenden Speicher (Pufferbereich) und schreibt die Daten direkt in den Speicher der CPU.

12 Die Nachfolger der 8086-CPU

12.1 Die INTEL-CPU 80286

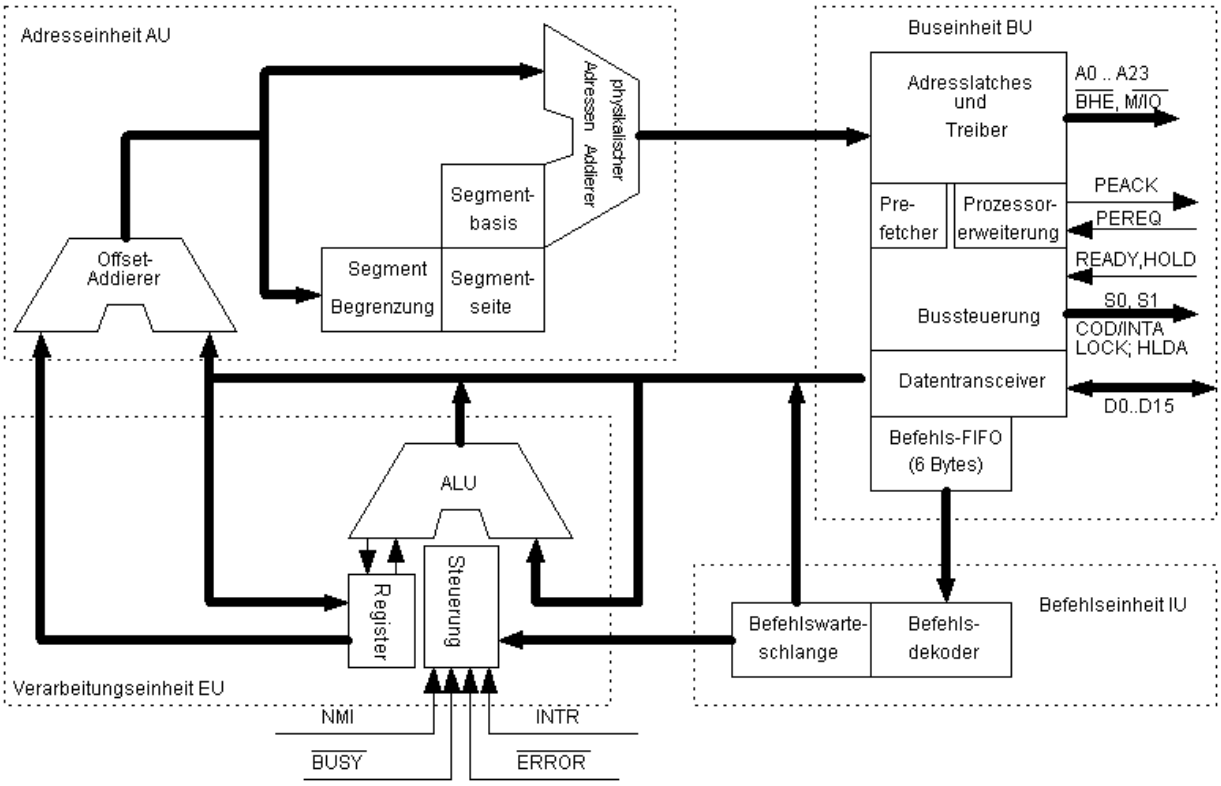


Abbildung 36: Blockschaltbild der CPU 80286

Grundlegende Unterschiede in der Architektur zum 8086:

- kein gemultiplexter Adress/Datenbus mehr (8086: Daten- und Adressbus gemultiplext)
- Schnellere Speicherzugriffe: 2 Takte (8086: 4 Takte)
- zusätzlicher Addierer für Adressberechnung (8086: normale ALU wurde zur Adressberechnung genutzt, 4 Takte zusätzlich bei einfachen Adressberechnungen `[BP+SI]`, 8 Takte bei doppelten `[BP+SI+nn]`)
- Hardware-Multiplizierer, 16 Bit-Multiplikation in 21 Takten (8086: 110–120 Takte, Z80: 750 Takte in SW)
- schnellerer Shifter (aber noch keinen Barrel-Shifter), 1 Takt pro Verschiebung (8086: 4 Takte)
- Zusätzliche Befehle: Shift immediate, MUL immediate, PUSH immediate, PUSHA/POPA, ENTER/LEAVE, INS/OUTS, BOUND
- Protected Mode
- 4 weitere Adressleitungen und damit einen Adressraum von 16 M.

- Implementierung eines virtuellen Speichers mit 1 G.
- Befehle für den 80287 (Floatingpoint) erfordern keine Synchronisation mittels `FWAIT` mehr.
- Zwei Arbeitsmodi: Real Address Mode (Real Mode) und Protected-address Mode (Protected Mode)

12.1.1 Real-Mode

Nach dem Reset arbeitet der 80286 im Real-Mode. Das entspricht der Arbeitsweise eines schnellen 8086. Es besteht Objektcodekompatibilität. Die Befehle entsprechen dem des 8086 (Grundbefehlssatz). Der Adressraum ist hier, wie beim 8086, nur 1 M groß.

12.1.2 Protected-Mode

Mittels Befehl konnte man vom Real-Mode in den Protected-Mode umschalten. Zurück ging es erst ab dem 80386. Damit konnte man die virtuelle Adressierung, ein erweitertes Speichermanagement (u.a. Speicherschutz) und Multiprozessing nutzen. Die Begrenzung der Segmente auf 64 k bleibt noch bestehen.

12.1.3 Buseinheit BU

Die Buseinheit ist das Interface zur Außenwelt. Hier werden die bidirektionalen Datenleitungen (16 Bit), den 24-Bit breiten unidirektionalen Adressbus und Signale des Steuerbusses.Außerdem erlaubt eine Prefetch-Warteschlange das vorausschauende Laden von 6 Byte Befehlscode.

12.1.4 Befehlseinheit IU

Die Befehlseinheit holt aus der Prefetch-Warteschlange Byte für Byte und dekodiert einen Befehl. Dann wird daraus in der Befehlswarteschlange ein 69-Bit breites Befehlswort erzeugt das von der Verarbeitungseinheit ausgeführt werden kann.

12.1.5 Verarbeitungseinheit EU

Neben der ALU und den Registern beinhaltet die Verarbeitungseinheit auch einen sogenannten Mikrocodesequenzer. Dies ist ein kleiner Prozessor der die ankommenden Befehle interpretiert. Er zerlegt dabei die von der Befehlswarteschlange kommenden Befehle in eine Reihe von Mikrobefehlen und führt diese aus. Durch diesen Mikrocodesequenzer lässt sich die bisher fest verdrahtete Lo-

gik durch Software ersetzen[12].

12.1.6 Adresseinheit AE

Hier sind das Speichermanagement und der Speicherschutz realisiert. Der Speicherschutz umfasst u.a. die Prüfung der Zugriffsrechte für den entsprechenden Speicherbereich und die Überprüfung, ob die Segmentgrenzen überschritten werden. Das Speichermanagement berechnet aus den virtuellen Adressen die physische Adressen auf die letztendlich zugegriffen werden kann.

12.1.7 Registersatz des 80286

Der Registersatz wurde vor allem um die Register für das Speichermanagement erweitert. Die eigentlichen Arbeitsregister sind mit dem des 8086 identisch und nur 16-Bit breit.

Neu hinzugekommen sind:
Global Descriptor Table Register GDTR
Es umfasst 40 Bit und beinhaltet die Startadresse der globalen Deskriptortabelle im Hauptspeicher (obere 3 Bits). Die unteren 2 Bytes enthalten die Größe der Tabelle. Diese globale Tabelle beinhaltet alle wichtigen Angaben für den gesamten Speicherbereich. Der globale Speicher ist die Systemresource die von allen Softwarekomponenten genutzt wird. Dies erfolgt aus Sicherheitsgründen aber nur unter der Kontrolle des Betriebssystems.

Interrupt Descriptor Table Register IDTR
Diese Tabelle erlaubt dem System einen sehr schnellen Zugriff auf die Serviceroutinen für die Interrupt- und Ausnahmebehandlungen.

Machine Status Word MSW
In diesen 16 Bits werden wichtige Informationen über den Zustand der CPU verwaltet.

12 Das geht soweit, dass bei anderen Prozessortypen dieses Mikroprogramm nachträglich umprogrammiert werden kann. Auf diese Weise lassen sich Fehler nachträglich korrigieren. Auch die Implementierung weiterer Befehle wird dadurch möglich.

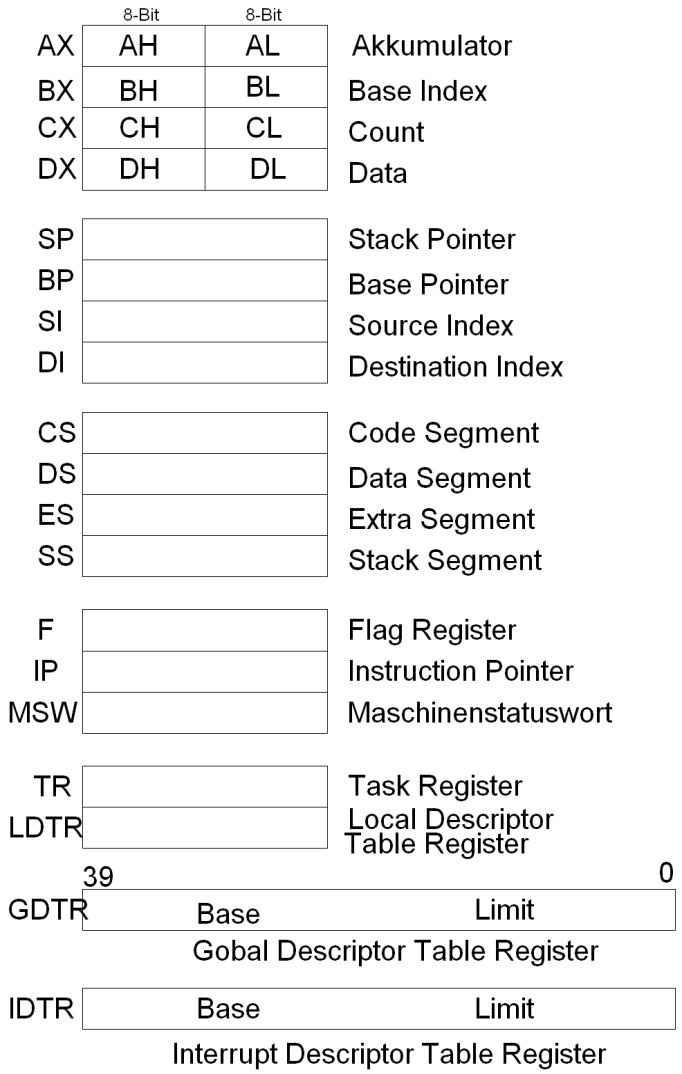


Abbildung 37: Registermodell des 80286

Task Register TR
Im Protected Mode wird hier der Selektor verwaltet, der angibt, welcher Prozess gerade ausgeführt wird (Context Switch).

Local Descriptor Table Register LDTR
Dieses Register verweist auf eine Deskriptortabelle, die den lokalen Speicher des entsprechenden Prozesses verwaltet.

12.1.8 Speicherzugriff

Im Protected Modus kann ein Programm nicht mehr feststellen, wo es sich im Speicher befindet.

Jeder Speicherzugriff erfolgt über Tabellen. Der Programmierer muss Sorge dafür tragen, dass diese Tabelle vor dem Umschalten in den Protected Mode richtig initialisiert werden.

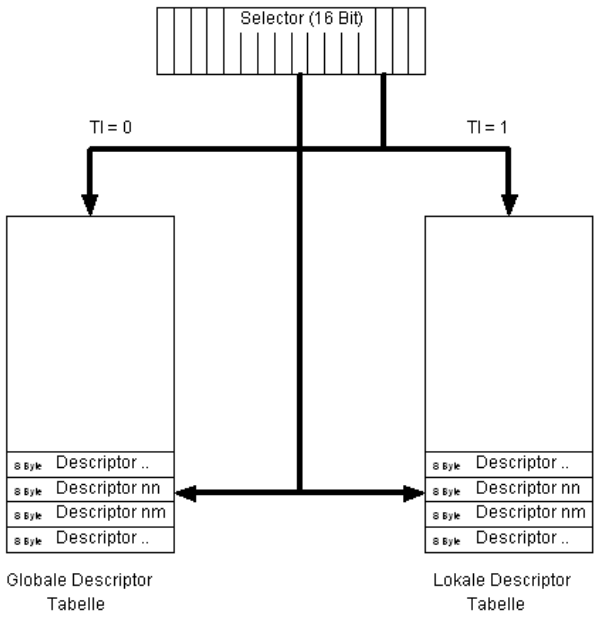


Abbildung 38: Zugriff auf Descriptoren

Deskriptoren:
Sie umfassen einen Block von 8 Byte und enthalten neben der Basisadresse im realen Speicher (24 Bit) und der Länge des Speicherblocks von max. 64 k (16 Bit) noch weitere Kennzeichnungen des Speichersegments, auf das zugegriffen werden kann.
Wird ein Selektor geladen, holt sich die CPU aus der GDT die Basisadresse und das Limit des Segments sowie deren Eigenschaften (Privilegstufe, Art des Segments usw.).

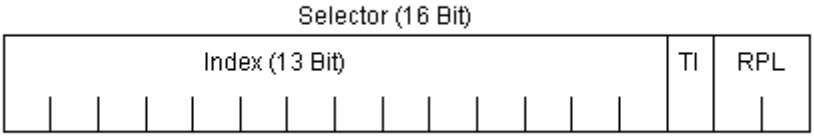


Abbildung 39: Aufbau eines Selektors

RPL = Privilegstufe für diesen Selektor (0 – 3)
TI = 0 → verwende Globale Descriptor Tabellen
TI = 1 → verwende Lokale Descriptor Tabellen

Index = Ausahl des gewünschten Descriptors aus der Tabellen

12.2 Die INTEL-CPU 80386

Der 80386 ist die echte 32-Bit Version des 80286.

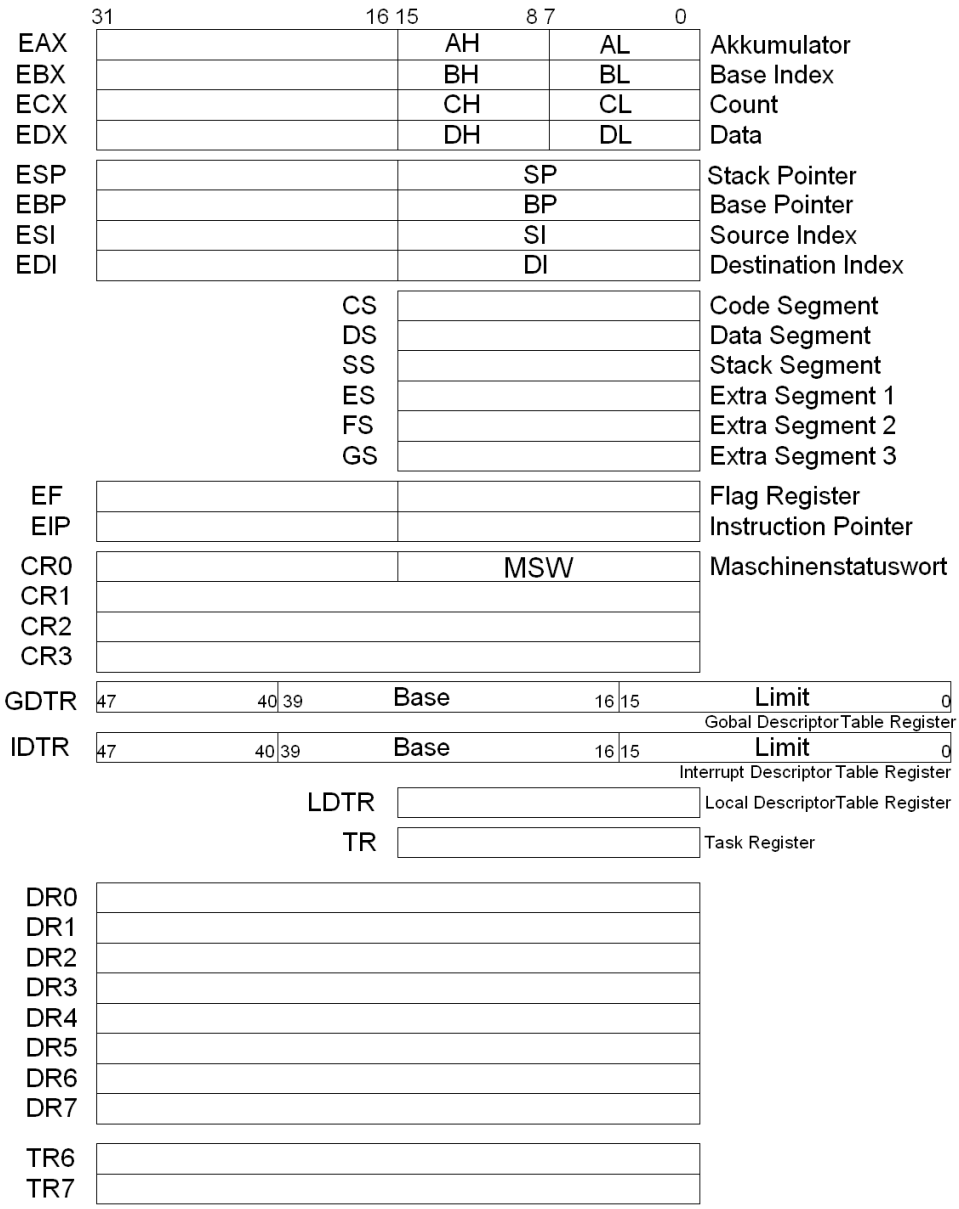


Abbildung 40: Das Registermodell des 80386

Die Register CR0 bis CR3 sind Control-Register. Hier wird all die Maschinenstati gespeichert die alle Tasks im System betreffen. In CR0 ist auch das MSW untergebracht. Hier sind die Bits PE (Protection Enable), MP (Math Present), EM (Emulate Coprocessor), TS (Task Switch), ET (Extention Type) und PG

(Paging) untergebracht.

Der 80386 kennt drei Arbeitsmodi:

- Real Mode
- Protected Mode
- Virtueller 8086 Mode V86

Im V86 Modus kann der 80386 mehrere, für den Realmode geschriebene 8086 Programme ausführen.

12.3 Die INTEL-CPU 80486

Dieser Prozessor ist eine Erweiterung des 80386 Prozessors. Er besitzt eine 8 kByte großen Codecache und einen verbesserten Numerik-Coprozessor auf dem gleichen Chip. Das Registermodell entspricht dem des 80386, jedoch um 8 Gleitkomma Datenregister erweitert.

12.4 Die INTEL-CPU 80586 (Pentium)

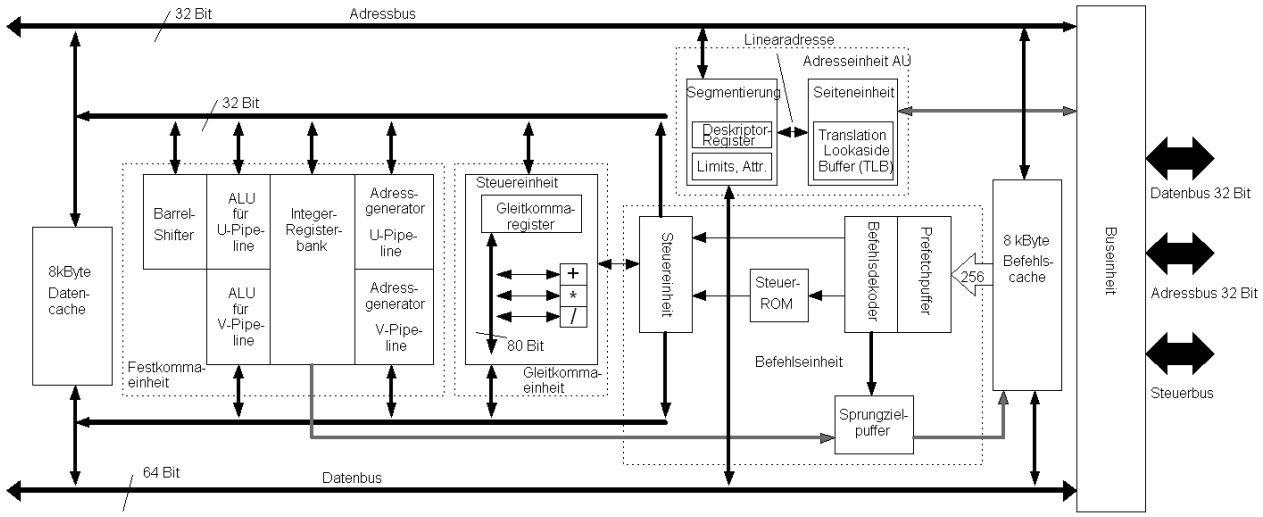


Abbildung 41: Blockschaltbild des Pentiums

Es sind sowohl für die Daten als auch für den Code ein jeweils 8 kByte großer Cache vorhanden.

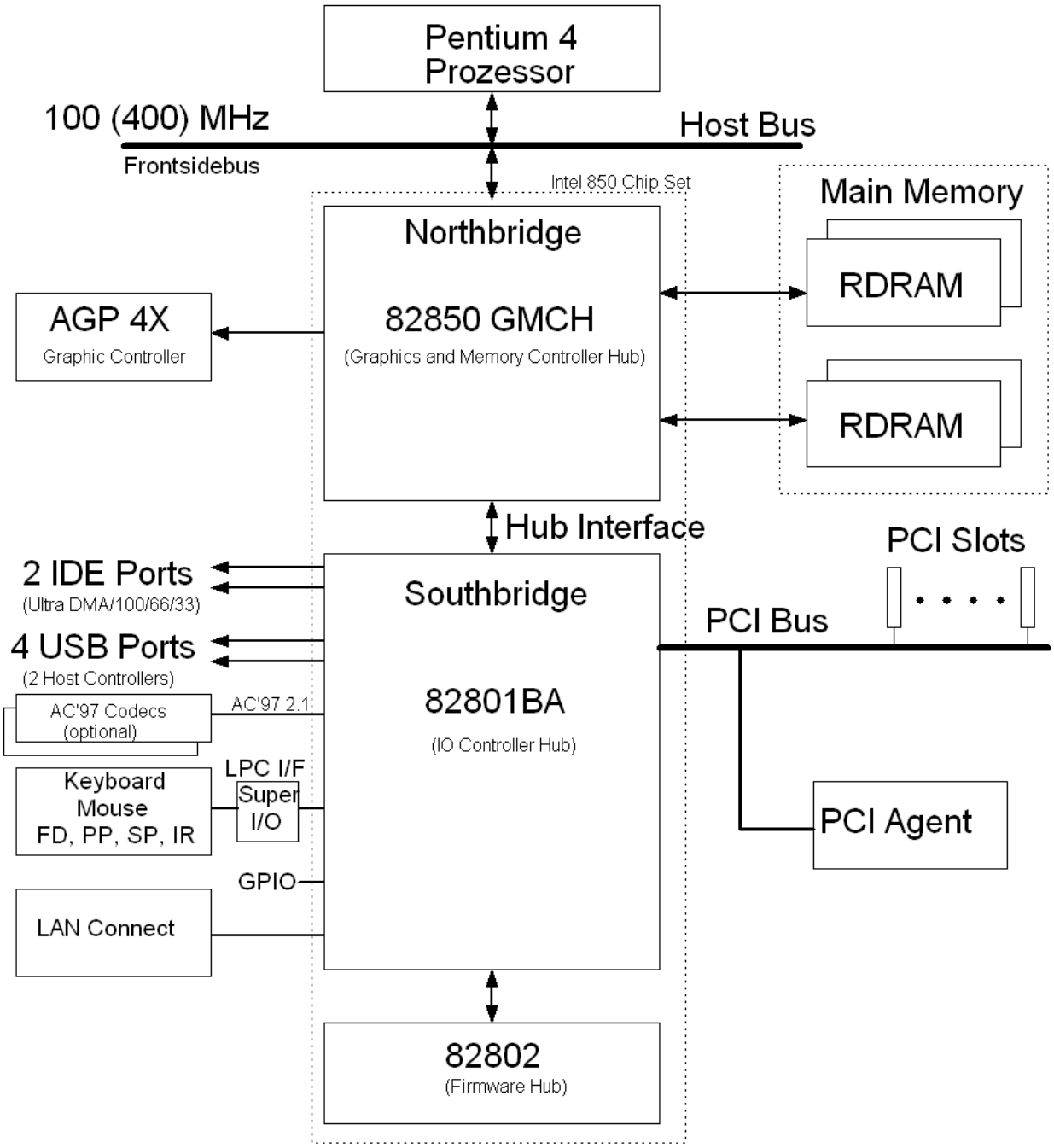
Es gibt eine U- und eine V-Pipeline. Da man erkannte, dass bestimmte, vor allem komplexe Befehle weniger oft verwendet werden als die einfachen Grund-

befehle, hat man sich entschlossen das Konzept der vollständigen Ausführung der Befehle mittels Mikrosequenzer wieder zu verlassen. Ein Grund war auch die etwas geringere Ausführungsgeschwindigkeit dieser Grundbefehle verglichen mit einer fest verdrahteten Logik. Nun können diese Grundbefehle sowohl mittels Mikrosequenzer der ALU der U-Pipeline als auch in der V-Pipeline verarbeitet werden. In der V-Pipeline steckt somit keine Mikrosequenzer mehr, sondern die „alte“ fest verdrahtete Logik. Damit ergibt sich für die Befehlsabarbeitung folgendes Schema:

- einfache Befehle kommen in die U- oder V-Pipeline
- komplexe Befehle müssen in die U-Pipeline

Somit kann es zu unterschiedlich langen Ausführungszeiten des ein und des selben Befehls kommen, je nach dem in welcher Pipeline er abgearbeitet wird. Die Logik für die Befüllung der Pipeline ist aber so ausgerichtet, dass die jeweilige Pipeline optimiert beladen wird.

12.5 Blockschaltbild eines PCs



12.5.1 RAM-Speicher

Der Hauptspeicher eines PCs besteht aus dynamischen Speicherelementen. Dabei ist das eigentliche Speicherelement ein Kondensator. Ist der Kondensator geladen, verliert er innerhalb einer bestimmten Zeit seine Ladung. Damit die Information nicht verloren geht, muss dieser Typ von Speicher regelmäßig aufgefrischt werden. Dazu muss der Speicher gelesen und wieder zurückgespeichert werden.

Früher musste dieser Refreshzyklus von außen angestoßen werden. Damit verbunden war ein zusätzlicher Aufwand an Hardware. Neue Bausteine haben diesen Refresh Automatismus intern, so dass sie, von außen betrachtet, wie statische Speicher erscheinen.

An die CPU werden diese Speicher über den Frontside-Bus angeschlossen.

12.5.2 North- und Southbridge

Betrachtet man ältere Mainboards so sind diese zum Einen sehr groß und zum anderen voll von ICs. Dabei fallen einem die größeren Bausteine wie CPU, EPROM oder Interruptcontroller sofort ins Auge und auch die gleichmäßige Anordnung der RAM-Speicher ist nicht zu übersehen. Der Rest der Platine ist voll von kleineren ICs und vielen passiven Bauteilen wie z.B. Widerstände und Kondensatoren.

Abbildung zeigt das Blockschaltbild und den Bestückungsplan eines solchen frühen PCs. Im Anhang sind die ausführlichen Schaltpläne zu sehen.

Um diese Menge an "kleinen" ICs zu reduzieren hat Intel zwei Chipsätze entwickelt, die die meiste Funktionalität übernehmen. Aufgrund ihrer üblichen Anordnung im Schaltplan werden sie als North- und Southbridge bezeichnet.

In der Northbridge findet die Anpassung zwischen CPU und externem RAM-Speicher statt. Auch die AGP-Schnittstelle für die Grafikkarte wird von diesem Chip bedient. Die Funktionsblöcke sind der Speichercontroller, Schnittstelle zur Grafikkarte für AGP und PEG, die Schnittstelle zum Prozessor über den sogenannten Frontside-Bus (FSB) und eine Verbindung zur Southbridge. Diese Verbindung ist als PCI-Schnittstelle ausgelegt. Somit ist die Southbridge wie eine "normale" PCI-Steckkarte ansprechbar.

Die Southbridge ist für die Steuerung der Peripherie zuständig. Dies sind u.a. COM und LPT-Schnittstellen, PS/2 (Tastatur und Maus), USB und ggf. ISA-Steckkarten. Auch der Flashspeicher für das BIOS ist hier angebunden.

Der Trend steht auf USB. So werden u.a. die Tastatur und Maus bei Notebooks nicht mehr über interne PS/2-Schnittstellen bedient, sondern über USB. Auch

die Drucker- und COM-Schnittstellen verschwinden zu Gunsten von USB immer mehr. Dort wo diese unbedingt gebraucht werden, sind sogenannte Schnittstellenkonverter im Einsatz.

12.5.3 Frontsidebus

Der Frontside-Bus verbindet die CPU mit der Northbridge. Die Hardwareschnitttselle ist nichts anderes als der Sockel für die CPU.
Die wichtigste Aufgabe des FSB ist die Anpassung zwischen der CPU- und der Speicherbausteingeschwindigkeit bzw. der I/O-Module. In der Regel läuft die CPU mit einer deutlich höheren (internen) Geschwindigkeit als dies über einen externen Bus mit seinen doch relativ langen Verbindungsleitungen möglich ist. Um die CPU durch die langsamen Zugriffe auf den externen RAM-Speicher nicht "auszubremsen" nutzt man schnelle Cache-Speicher innerhalb der CPU.

Man kann den Frontside-Bus auch als Taktvermittler zwischen den einzelnen Funktionsgruppen betrachten. Diese Taktentkopplung macht den üblichen Systembus überflüssig. Somit arbeiten aber CPU und die angeschlossenen Funktionsgruppen nicht mehr synchron.

Um die Anzahl der Leitungen, die den Prozessor mit dem Chipsatz verbindet, zu reduzieren, ist AMD bereits vor Intel zu einer seriellen Verbindungstechnik übergegangen. AMD bezeichnet dies als HyperTransport und ist ein offener Standard.

Bei Intel nennt sich diese Verbindung QPI (Quick Path Interconnect) und ist eine Mischung aus HyperTransport und PCI-Express.

Inzwischen sind bei modernen Prozessoren neben dem eigentlichen Prozessor auch der Grafikprozessor und der Speicher-Controller auf dem Chip integriert. Dadurch verliert auch der Chipsatz immer mehr an Bedeutung. Er ist aber immer noch für die Anbindung der Peripherie z.B. Festplatten, USB etc. notwendig.

12.5.4 PCI-Steckplätze

Diese Steckplätze erlauben auch weiterhin einen PC mit Fremdkarten zu erweitern. Was früher unbedingt notwendig war, die Funktionen des PCs waren z.T. sehr bescheiden, wird immer weniger genutzt. Nur sehr spezielle Anwendungen die viel eigene Intelligenz benötigen kommen noch vor. Auch hier hat der USB-Anschluss eine Erweiterung um nicht vorhandene Standardinterfaces der PCI-

Steckkarte den Rang abgelaufen.

PCI bedeutet Peripheral Component Interconnect und ist ein synchroner Bus mit einem maximalen Takt von 33,33 MHz oder 66,66 MHz. Die Signale sind bei steigender Flanke auf der Taktleitung gültig. Als Master kann eine Karte auch die Kontrolle über den Bus übernehmen um so z.B. einen schnellen Datentransfer (Netzwerkkarten oder Festplatten) durchzuführen. Im Falle von Festplatten werden große Datenblöcke direkt über eine Host-Bridge in den RAM-Speicher transferiert (DMA-Betrieb).

Bei PCI ist die Position an der die Karte gesteckt wird nicht relevant. Bei Systemstart wird jeder Steckplatz geprüft und der vorhandenen Karte ein Adressbereich zugeteilt. Beim älteren ISA-Bussystem musste der Karte eine individuelle Adresse (meist über Dipschalter) eingestellt werden. Nur unter dieser Adresse konnte die Karte angesprochen werden.

In der Regel beendet der Master den Transfer. Allerdings kann ein Slave auch über ein STOP-Signal das Ende erzwingen.

Ein Master kann über ein REQ den Bus anfordern. Die aktuelle Übertragung muss nach einer bestimmten Zeit abgebrochen werden, damit der neue Master Zugriff bekommt.

Der PCI-Bus hat vier Interruptleitungen die sich die Karten, falls sie nicht ausreichen sollte, auch teilen können.

13 Weitere Funktionsgruppen

13.1 D/A- und A/D-Wandler

Ein Analog- / Digitalwandler setzt ein Zeit und Werte kontinuierliches Signal in ein Werte diskretes Signal um. Aus einem linearen Signalanstieg wird ein treppenförmiges Signal. Es ist, solange keine Lücken vorliegen, immer noch ein Zeit kontinuierliches Signal (Abbildung 42 b).

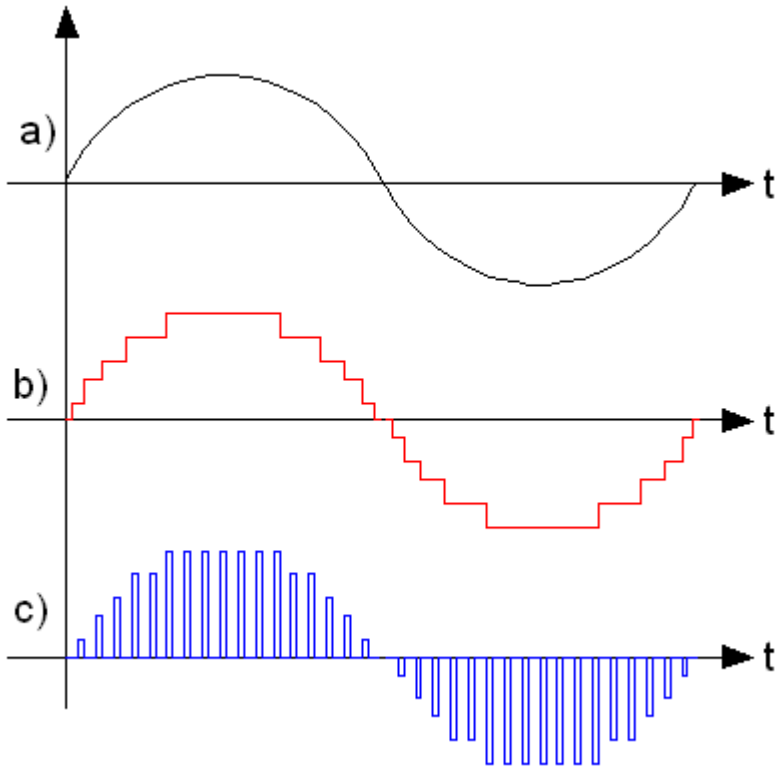


Abbildung 42: Kontinuierliche und diskrete Signale

Ein Zeit und Werte diskretes Signal zeigt Abbildung 42 c.

Subtrahiert man das Signal a von Signal b ergeben sich Spannungsreste die aufgrund der Digitalisierung entstehen.

SNR .....

13.2 Wichtige Grundbegriffe

13.2.1 Quantisierungsfehler

Dieser Fehler beschreibt die Differenz zwischen dem Eingangssignal und dem erzeugten Digitalwert. Da sich die Digitalwerte nur stufenweise ändern können wird in den meisten Fällen der kontinuierliche Eingangswert nicht exakt auf den Digitalwert umsetzen lassen. Der Digitalwert muss entweder ab- oder aufgerundet werden.

Beispiel:
Ein A/D-Wandler hat eine Auflösung von 8 Bit. Die Referenzspannung beträgt 5,0 V. Damit ergibt sich eine Schrittweite[13] von 5,0V / 256 = 19,53125 mV. Die zu messende Spannung hat einen Wert von 1,900 V. Beim Umsetzen liegen für diesen Spannungswert ein unterer Vergleichswert von 1,89453125 V (Stufe 97) und ein oberer Wert von 1,9140625 V (Stufe 98) zur Verfügung. Damit liefert der Wandler ein Ergebnis, das um 0,0140625V zu klein oder um 0,00546875 V zu groß ist.

Der maximale Quantisierungsfehler ist somit 0,5 der Schrittweite bzw. 0,5 LSB. Er kann sowohl positive als auch negative Werte annehmen. Dieser Fehler ist also systembedingt und zählt zu den sogenannten zufälligen Fehler. Diese Fehlerart kann man auch als Rauschen bezeichnen.

13.2.2 Linearitätsfehler

Bei einem D/A-Wandler mit R2R-Netzwerk bestimmt die relative Genauigkeit der Widerstandswerte zueinander in wie weit der Soll- vom Istwert abweicht.

Beispiel:
Man erwartet bei einem 8-Bit Wandler mit einer Referenzspannung von 5 V und einem Digitalwert von 128 (0x80) einen Sollwert von 2,500 V. Ist aber die Ausgangsspannung 2,507 V, dann ist dafür der Linearitätsfehler verantwortlich. Er beträgt in diesem Fall (2,507 V – 2,500 V) / 2,500 V * 100 % = 0,28 %.

13.3 D/A-Wandler

Ein Digital- / Analogwandler setzt einen digitalen Zahlenwert in eine analoge Spannung um. Es entsteht beim Versuch ein sinusförmiges Signal zu generieren das Signal b aus Abbildung 42. Da die einzelnen Stufen sehr steile Flanken be-

13 Man formuliert das häufig auch als Wertigkeit eines Bits.

sitzen, ist der Oberwellenanteil entsprechend hoch. Ein reiner Sinus besitzt nach Fourier keine Oberwellen.

Diese Oberwellen lassen sich durch entsprechende Tiefpassfilter mehr oder weniger stark dämpfen.

Wie groß die Maximalspannung wird hängt von der angelegten Referenzspannung U_{Ref} ab. In der Praxis sind vor allem das PWM-Verfahren und die Umsetzung mit einem R2R-Netzwerk weit verbreitet.

13.3.1 Puls-Weiten-Modulation-Verfahren (PWM)

Hier erzeugt der Mikroprozessor oder Mikrocontroller ein Rechteck förmiges Ausgangssignal mit einer festen Frequenz aber einem variablen Puls-Pausen-Verhältnis. Dieses Signal wird auf ein Integrator Glied geführt, der das arithmetische Mittel des Puls-Pausen-Verhältnis bildet (Abbildung 43).

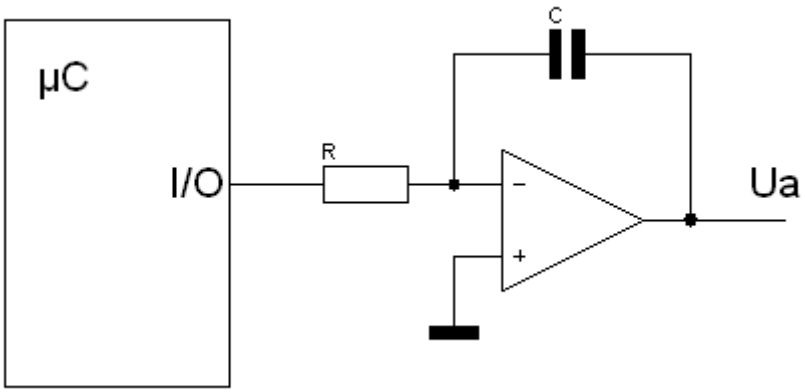


Abbildung 43: PWM-Signal des µCs wird integriert

Erzeugt der Mikrocontroller ein Signal nach Abbildung 44 a, dann ist der arithmetische Mittelwert 1/2 und damit die Ausgangsspannung $U_a = 0{,}5\ U_{Ref}$.

Wird die Pause größer, sinkt der High-Anteil des Signals und der arithmetische Mittelwert wird kleiner. In Abbildung 44 b ist das Puls-Pausen-Verhältnis 1:3. 25% Ein- und 75% Auszeit damit wird die Ausgangsspannung nur noch 25% des Referenzwertes betragen.

Im dritten Fall ist es umgekehrt. Hier ist der Impuls drei Mal so lang wie die Pause, so dass eine Ausgangsspannung von 75% des Referenzwertes erreicht.

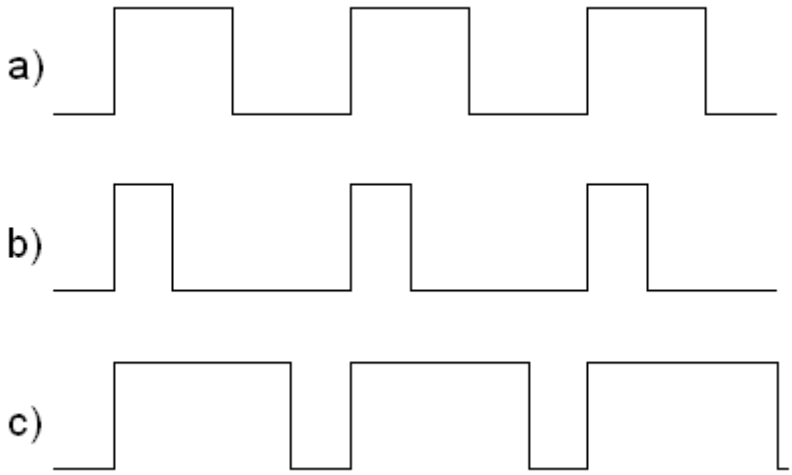


Abbildung 44: Verschiedene Puls-Pausen-Verhältnisse

Durch das Integrier-Glied, das wie ein Tiefpass wirkt, sind die Oberwellen des Ausgangssignal nicht stark ausgeprägt. Auf der anderen Seite werden sehr schnelle Änderungen u.U. nicht sichtbar. Es kommt immer darauf an, wie stark die Filterwirkung des Tiefpasses ist.

Anwendungen für diese Art der D/A-Wandlung sind u.a. LED-Leuchten, deren Helligkeit von dunkel bis hell gesteuert werden kann. Ist die Grundfrequenz genügend hoch, dass sie das menschliche Auge nicht mehr erfassen kann, kann der Tiefpass entfallen, da die LED selbst als Integrierer arbeitet.

Auf diese Weise lässt sich auch die Drehzahl von Gleichstrommotoren einstellen. Auch hier ist der Motor bzw. die internen Induktivitäten ein passender Integrierer.

13.3.2 R2R-Netzwerk

Abbildung 45 zeigt ein R2R-Netzwerk. Das Besondere daran ist, dass der Gesamtwiderstand unabhängig von der Anzahl der Stufen konstant bleibt. Damit ist die Belastung für die Signalquelle ebenfalls immer konstant.

Nehmen wir dann, S1, S2 und S3 sind mit 0V (GND) verbunden, S4 dagegen mit U_{Ref}. Welche Spannung erscheint in diesem Fall am Ausgang?
Der untere linke Widerstand liegt somit mit seinem linken Anschluss auf 0V und ist mit seinem rechten Anschluss mit dem untersten Widerstand verbunden. Dabei bildet sich eine Parallelschaltung mit einem Gesamtwiderstand von R.

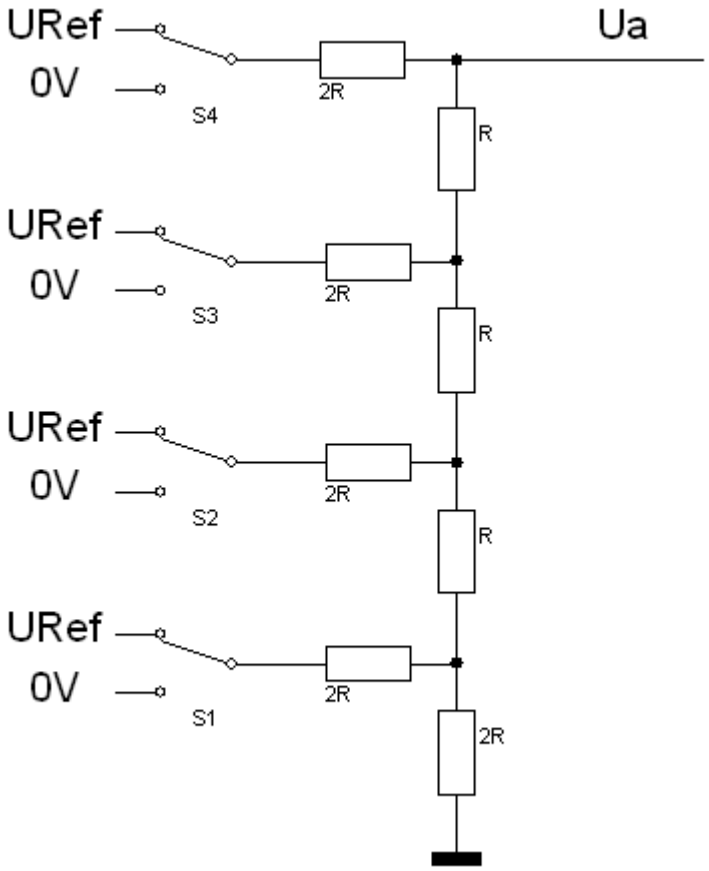


Abbildung 45: R2R-Netzwerk

Dieser liegt mit dem darüber liegenden in Reihe was einen Ersatzwiderstand von 2R ergibt. Der Widerstand bei S2 liegt jedoch auch auf Masse und bildet damit mit dem eben ermittelten Ersatzwiderstand eine Parallelschaltung die widerum einen Gesamtwiderstand von R aufweist. Das lässt sich nun für jede Stufe so weiterverfolgen. Der obere linke Widerstand liegt aber an U_{Ref}. Die darunter liegenden Widerstände haben, wie eben gezeigt, einen Wert von 2R. Somit ergibt sich aus Sicht des Ausgangs ein Spannungsteiler der U_{Ref} halbiert.

Die Wertigkeit für den Schalter S4 ist somit 0,5 U_{Ref}. S3 hat die Hälfte von S4 und somit 0,25 U_{Ref}. S2 kommt dann auf 0,125 U_{Ref} und S1 auf 0,0625 U_{Ref}. Mit dieser 4-Bit Anordnung ergibt sich eine Auflösung von 0,0625 U_{Ref}. Bei einem U_{Ref} von 5,0 V wäre dann die Auflösung (Stufenhöhe) 0,0625 x 5,0 V = 0,3125 V.

Solche R2R-Netzwerke lassen sich auch sehr einfach diskret aufbauen. In ist ein PIC-Mikrocontroller dargestellt, an dessen IO-Leitungen RB0 bis RB3 ein

R2R-Netzwerk angeschlossen ist.

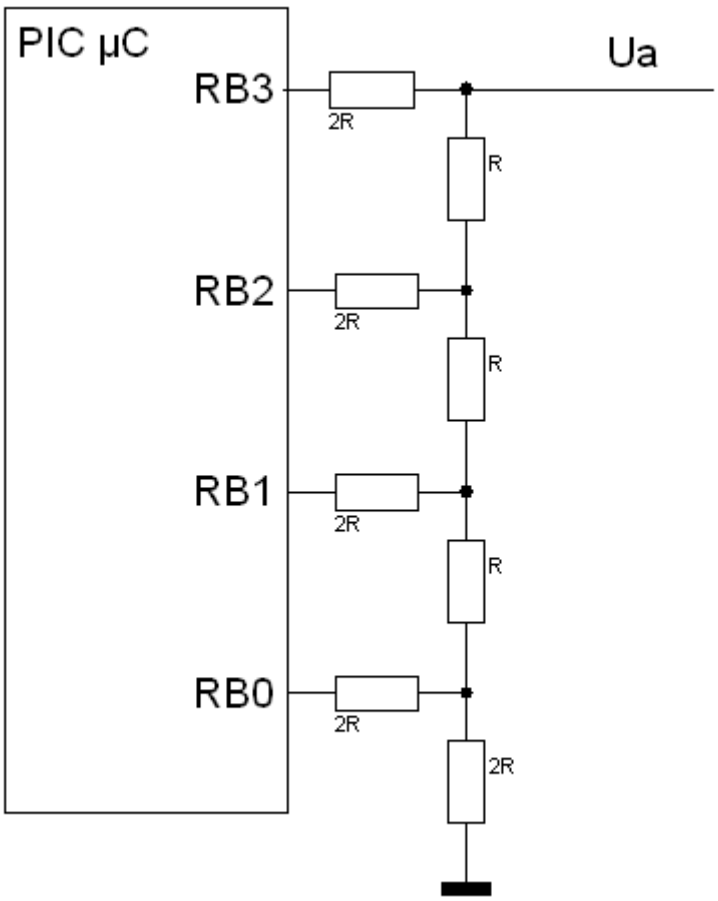


Abbildung 46: Ein diskretes R2R-Netzwerk

Auf diese Weise lassen sich sehr einfach beliebige Kurvenformen erzeugen. Allerdings sind diese Werte immer diskret.

13.4 A/D-Wandler

Hier sind, je nach Einsatzgebiet und Anforderungen, unterschiedliche Verfahren üblich. Die wichtigsten sind:

Zählverfahren
Wägeverfahren (ist eine Weiterentwicklung des Zählverfahrens)
Parallelwandlung (Flash-Wandler)
Dual Slope Wandler
Spannungs-Frequenz-Wandler
Delta-Sigma-Wandler

13.4.1 Zählverfahren

Hier wird mit einem Komparator das Eingangssignal mit einem Vergleichssignal verglichen. Das Vergleichssignal wird mit Hilfe eines Binärzählers und einem R2R-Netzwerk erzeugt. Der Zähler startet bei 0 und erhöht seinen Wert

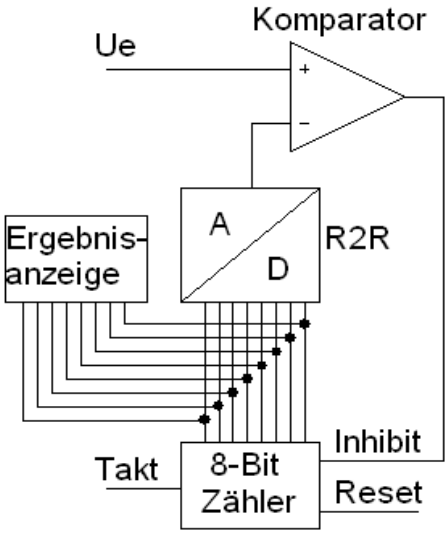


Abbildung 47: AD-Wandler nach dem Zählverfahren

immer um eins. Damit entsteht am Ausgang des R2R-Netzwerkes eine Spannungsrampe. Erreicht nun diese Vergleichsspannung die Höhe der Eingangsspannung, zeigt der Komparator dies an, der Zähler stoppt und liefert auch direkt das Ergebnis.

Nachteil dieser Methode ist die Abhängigkeit der Umsetzungszeit von der Höhe des Eingangssignals. Denn je größer die Spannung ist, desto länger muss der Zähler hoch laufen.

13.4.2 Wägeverfahren, sukzessive Approximation

Es ist eine Weiterentwicklung des Zählverfahrens. Man beschleunigt das Annähern indem man nicht bei Null beginnt, sondern bei der halben Maximalspannung (=Referenzspannung). Zeigt der Komparator ein niedrigeres Eingangssignal an, dann wird der nächste Wert auf ein Viertel gesetzt. Liegt nun die Eingangsspannung über dem neuen Referenzwert, kommt zu diesem Viertel noch ein Achtel hinzu.

Auf diese Weise wird, bei einem 8-Bit-Wandler, die Eingangsspannung in 8 Schritten umgesetzt, bei einem 10-Bit-Wandler werden 10 Schritte benötigt.

Abbildung 48 zeigt einen 6-Bit Umsetzer. Beim ersten Schritt bleibt die generierte Referenzspannung unterhalb der Eingangsspannung. Deshalb ist dieser Schritt gültig und die höchstwertige Stelle der Ergebnis wir 1. Im nächsten Schritt wird zu 0,5 U_{Ref} noch 0,25 U_{Ref} addiert. Dieses Mal ist der neue Referenzwert (0,75 U_{Ref}) größer als die Eingangsspannung. Deshalb wird dieser Schritt nicht gemacht und das entsprechende Ergebnisbit auf 0 gesetzt. Im nächsten Schritt beträgt U_{Ref} = 0,625 (0,5 + 0,125) und ist wieder höher als Ue. Deshalb wird auch dieses Ergebnisbit zu 0. Im vierten Schritt beträgt die generierte Referenzspannung 0,5 U_{Ref} + 0,0625 U_{Ref} = 0,5625 U_{Ref}. Dieser Wert liegt unterhalb Ue und somit ist dieser Schritt gültig, das passende Ergebnisbit ist 1.

Im fünften Schritt kommt wieder zu viel dazu, das Bit wird 0. Im letzten Schritt liegt die neue Referenzspannung im Gültigkeitsbereich es Eingangssignal, das Bit wird 1 und die Wandlung beendet.

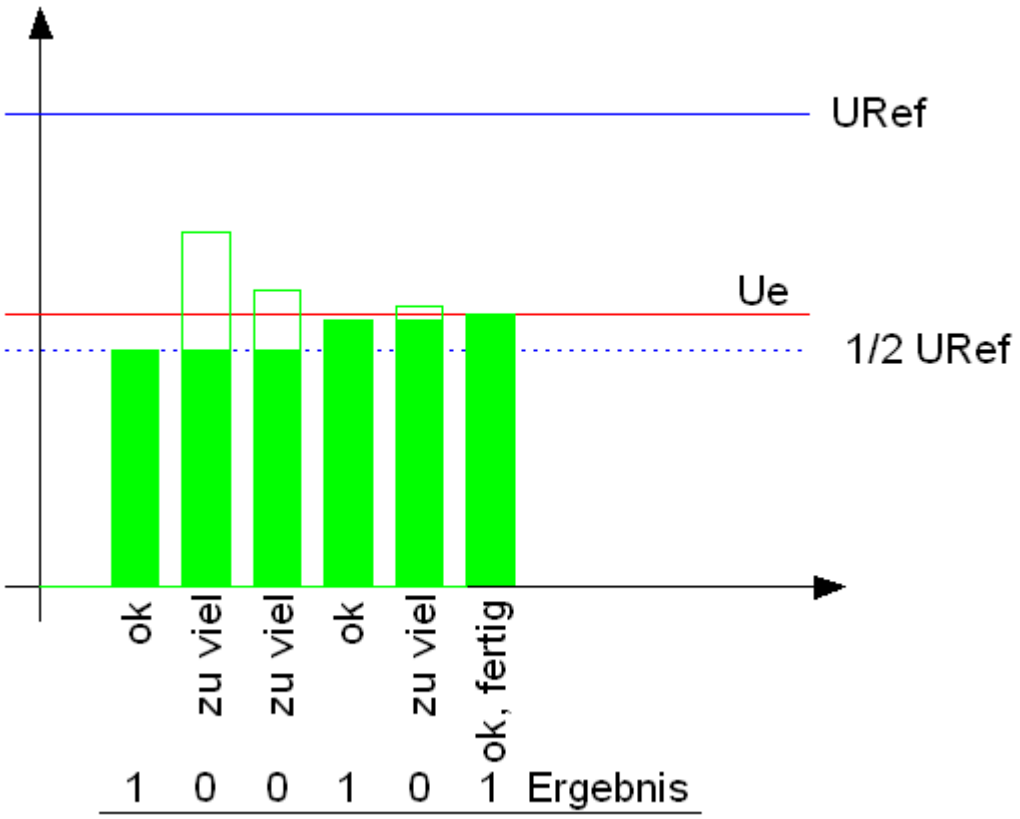


Abbildung 48: 6-Bit Wandler mit sukzessiver Approximation

Die Umsetzgeschwindigkeit liegt bei diesem Wandlertyp bei ca. 1 µs bis 20 µs. Damit können Eingangssignale mit bis zu einer Frequenz von 500 kHz umge-

setzt werden, ohne das Nyquist-Kriterium (siehe Seite 113) zu verletzen.

13.4.3 Parallelwandler

Sie sind die schnellsten Wandler überhaupt, aber der Aufwand ist enorm. Um einen 8-Bit-Wandler dieses Typs aufzubauen, benötigt man 256 Komparatoren und entsprechend viele Referenzspannungen. Die einzelnen Referenzspannungen lassen sich wohl von einer einzigen mittels Spannungsteiler ableiten, jedoch wird an die Genauigkeit der verwendeten Widerstände hohe Ansprüche gestellt.

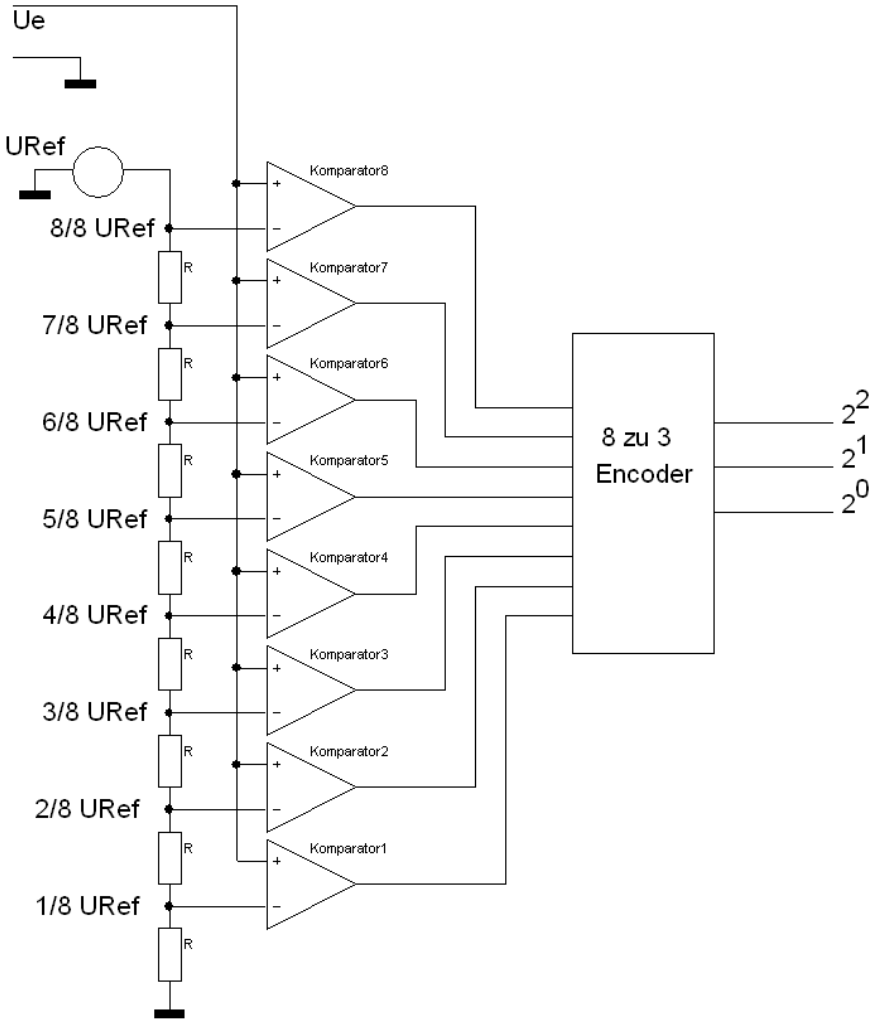


Abbildung 49: 3-Bit Flash-Wandler

Um die hohe Anzahl von Komparatoren zu reduzieren, kann man einen Trick anwenden. Dazu wird mittels 16 Komparatoren die oberen 4 Bits ermittelt. Dann wird dieses Ergebnis auf ein R2R-Netzwerk gegeben und daraus wieder ein Analogwert erzeugt. Diesen Wert subtrahiert man vom Eingangssignal und

erhält so die Spannung für die unteren 4 Bits. Diese Differenzspannung wird durch weitere 16 Komparatoren zu den unteren 4 Bits umgesetzt. Damit hat man einen 2-stufigen Flash-Wandler vorliegen.

Die Umsetzgeschwindigkeit ist etwas langsamer als bei einem einstufigen Flash-Wandler.

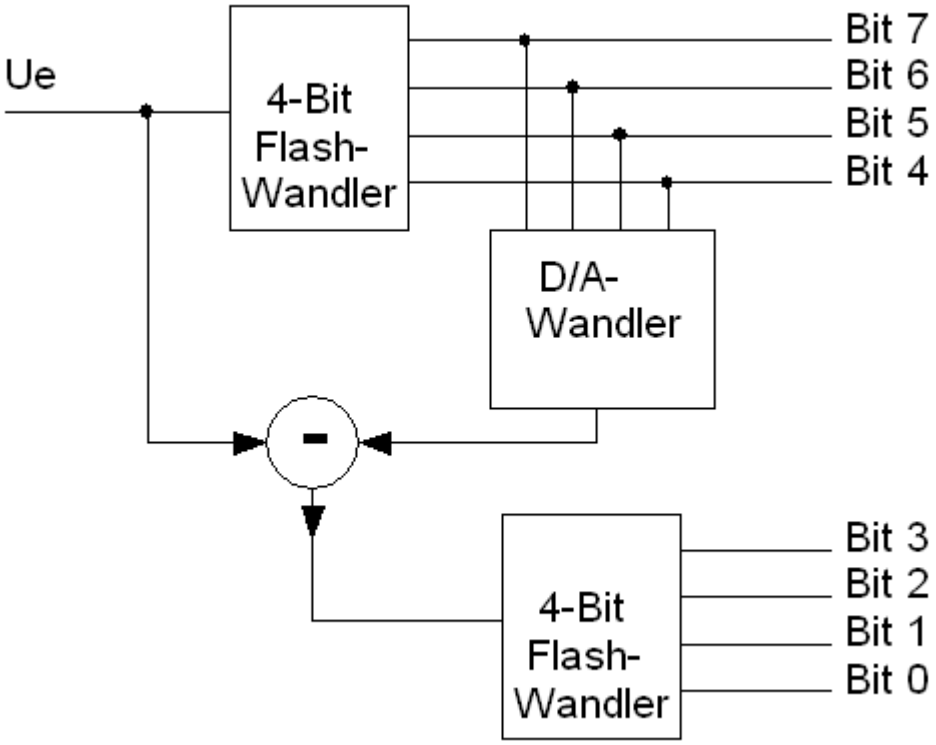


Abbildung 50: 2-stufiger 8-Bit Flash-Wandler

13.4.4 Dual-Slope-Wandler

Die unbekannte Eingangsspannung wird für eine bestimmte Zeit integriert. Dadurch lädt sich der Kondensator des Integrators auf einem, von der Eingangsspannung abhängigen Wert auf. Danach wird der Kondensator mit einem Konstantstrom entladen. Damit dauert der Entladevorgang, je nach Ladezustand, unterschiedlich lang. Diese Entladezeit ist somit proportional der Höhe der Eingangsspannung, $t_1 \sim Ue1$ bzw. $t_2 \sim Ue2$.

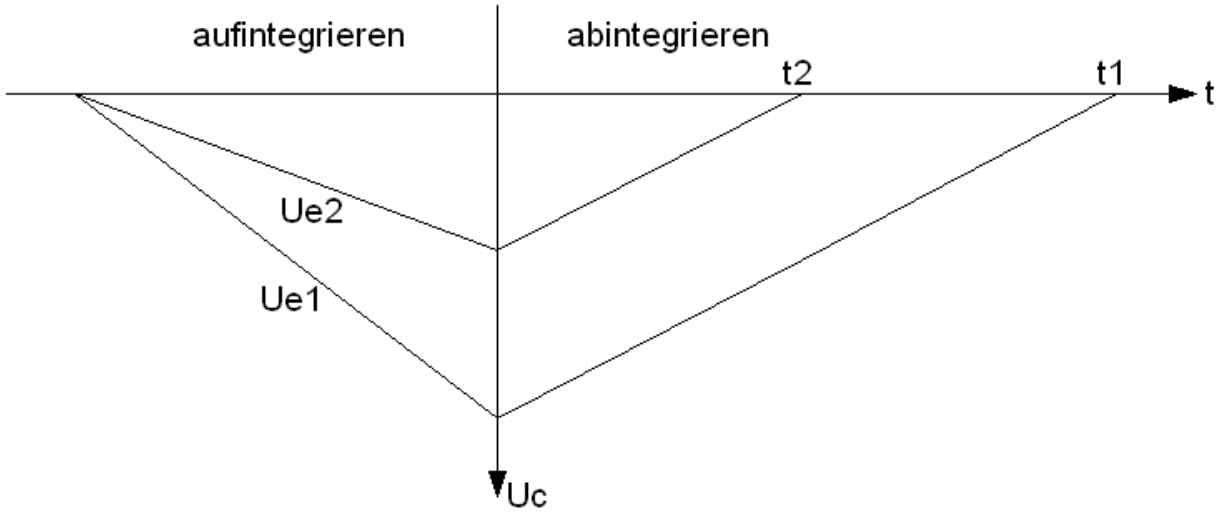


Abbildung 51: Dual-Slope-Wandler

Durch den Integrator können kurzzeitige Störungen nicht ins Ergebnis einfließen allerdings ist dadurch auch die Abtasterate auf ca. 10 begrenzt. Das Haupteinsatzgebiet sind Multimeter (DMM).

13.4.5 Spannungs-Frequenz-Wandler

Das Eingangssignal wird in eine proportionale Frequenz umgesetzt. Diese kann dann einfach mittels Frequenzzähler angezeigt werden. Ein spannungsgesteuerter Oszillator (VCO) ist das Herzstück dieses Wandlers.

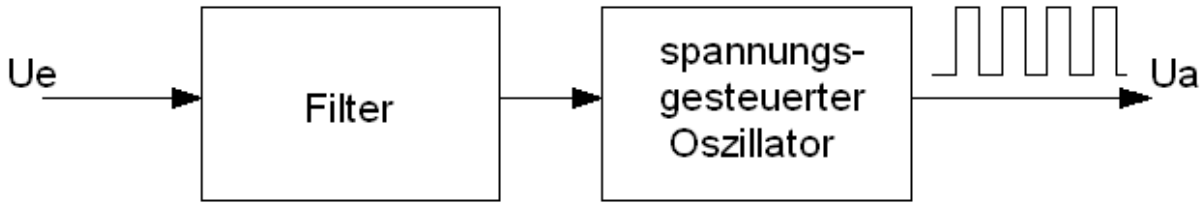


Abbildung 52: Spannungs-Frequenz-Umsetzer

13.5 Sigma-Delta-Verfahren

Ein Sigma-Delta-Wandler ist grundsätzlich von den bisher beschriebenen Verfahren verschieden.

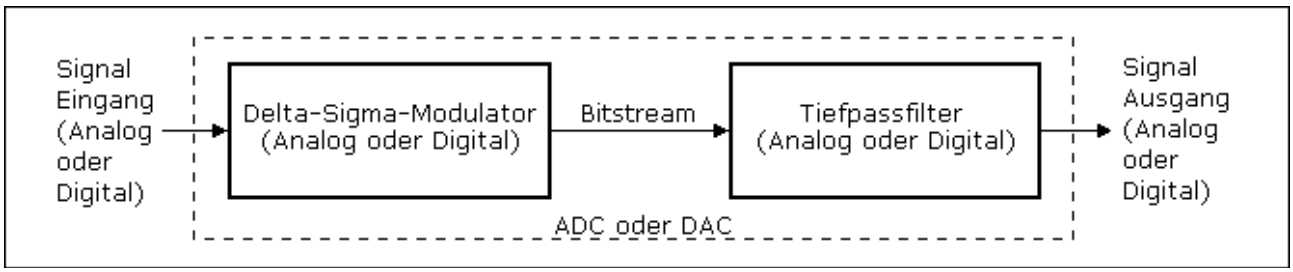


Abbildung 53: Blockschaltbild eines Sigma-Delta-Wandlers

Ein Sigma-Delta-Wandler besteht aus einem Subtrahierer. Er bildet die Differenz zwischen dem analogen Eingangssignal und dem analogen Ausgangssignal des DA-Wandlers. Diese Differenz wird integriert und auf einen Komparator (Hystereseschalter) geführt. Das Ausgangssignal des Komparators gelangt auf den Dateneingang des D-FF. Ein Taktsignal übernimmt nun immer diesen Pegel und sorgt so für ein serielles Ausgangssignal, dessen Verhältnis von Null und Einsen den analogen Eingangswert repräsentiert.

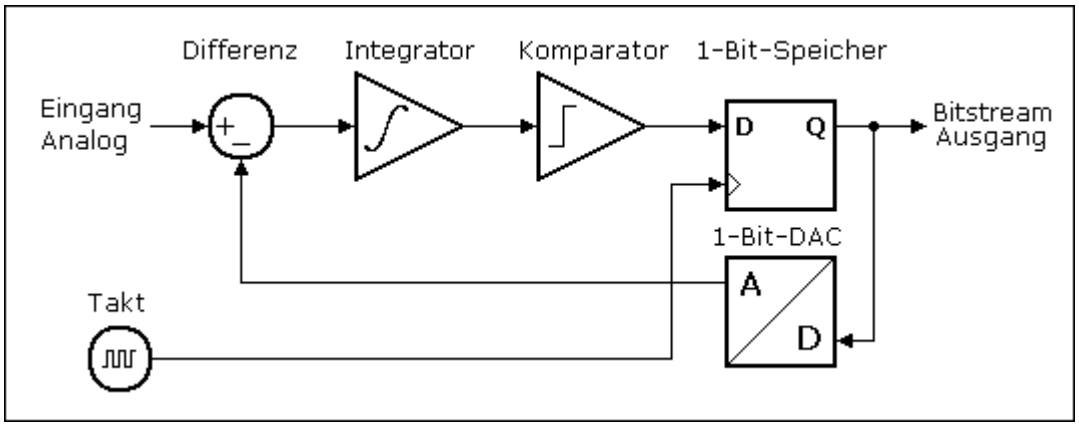


Abbildung 54: Sigma-Delta-Wandler erzeugt einen Bitstrom

Bei einer Eingangsspannung von 0 V ergibt sich folgendes Szenario: Der Ausgang des Intergators sei auf + 1 mV. Dann liegt am Ausgang des Komparators und damit am Eingang des D-FF eine Eins. Diese 1 erzeugt am analogen Ausgang des DA-Wandlers eine Spannung von + 1 V welche auf das Subtrahierglied gelangt. Damit wird dessen Ausgang aber – 1 V. Nun integriert der Integrierer gegen die ursprünglichen + 1 mV. Nach der Integration führt dieser Ausgang eine Spannung von - 1 mV. Das entspricht aber der unteren Umschaltschwelle des Komparators, der daraufhin eine 0 augibt. Diese Null erzeugt am DA-Wandler eine Spannung von – 1 V die über den Subtrahierer eine Spannung

von + 1 V erzeugt. und vom Integrierer wieder auf + 1 mV gebracht wird.

Bei einer Integrationsrate des Integrieres von 10 V/s ergibt sich eine Integrationszeit von:

$$t_{\int} = \frac{2\,mV}{10\frac{V}{s}} = 0{,}2\,ms$$

(2 mV = 1 mV + |-1 mV|)

Der Integrator integriert nun dauern von –1 mV nach +1 mV in 0,2 ms. Es entsteht ein serieller Datenstrom das einer Rechteckfrequenz mit 2,5 kHz und 50% Tastverhältnis entspricht.

Betrachtet man das Ganze bei einer Eingangsspannung von 0,5 V. Auch jetzt sei der Ausgang des Integrators auf + 10 mV. Damit bekommt der –-Eingang des Subtrahierers - 1 V und sein Ausgang hat nun (0,5 V) – (1 V) = -0,5 V. Diesen Spannungswert integriert der Integrierer mit einer Integrationsrate von 10 V/s gegen – 1 mV, was der unteren Umschaltschwelle des Komparators entspricht. Damit ergibt sich eine Integrationszeit von:

$$t_{\int} = \frac{2\,mV}{5\frac{V}{s}} = 0{,}4\,ms$$

Daraufhin erscheint am D-Eingang eine Null, die am DA-Wandlerausgang eine Spannung von + 1 V ergibt. Dieser Wert wird vom Subtrahierer von der Eingangsspannung abgezogen. Der neue Wert für den Integrator ist nun (0,5 V) – (–1 V) = 1,5 V. Die Integrationsrate wird damit 15 V/s und die Integrationszeit 0,133 ms. Das Ausgangssignal ist nun ein Rechteck mit 75% Tastverhältnis. Damit wird das gemittelte Ausgangssignal

1 V * 0,75 + (– 1 V) * 0,25 = 0,5 V.

Die eigentliche Wandlung besteht nun darin, dass man die Einsen und Nullen des entstehenden Rechtecksignals zählt. Lässt man den Zähler bei einer Eins hoch- und bei einer Null herunter zählen, dann ist der Zählerstand der gewandelte Analogwert.

Der Bereich der Eingangsspannung wird durch die Ausgangsspannung des DA-Wandlers bestimmt. In diesem Fall beträgt diese – 1 V bis + 1 V.

Eingangsspannung	Anzahl der Einsen	Anzahl der Nullen	Zählerstand
0 V	500	500	0
0,5 V	750	250	500
- 0,5 V	250	750	-500

Auf diese Weise sind hohe Auflösungen sehr leicht machbar. Um die Auflösung weiter zu erhöhen, braucht man nur die Zählzeit entsprechend verlängern.

Vorteil des Sigma-Delta-Wandlers:

- Es gibt kein Aliasing bei der Umwandlung in den Bitstrom, denn hier erfolgt keine Abtastung
- damit gibt es keine fehlerhafte Werte (die Sample-Hold-Schaltung kann entfallen)
- Das Wandlerverhalten ist absolut monoton und linear
- Der Wandler ist gegen steile Flanken, Rauschen und hochfrequente Störungen unempfindlich

13.6 Sample & Hold

Bei einer Umsetzung eines analogen Wertes in sein digitales Äquivalent benötigt der Wandler i.d.R. eine gewisse Zeit. Man muss für die Zeit der Wandlung eine konstante Eingangsspannung gewährleisten. Das ist bei sich schnell ändernden Signalen aber nicht möglich. Deshalb wurde die Sample&Hold-Schaltung entwickelt (Abbildung 55). Sie besteht im Prinzip aus einem Kondensator,

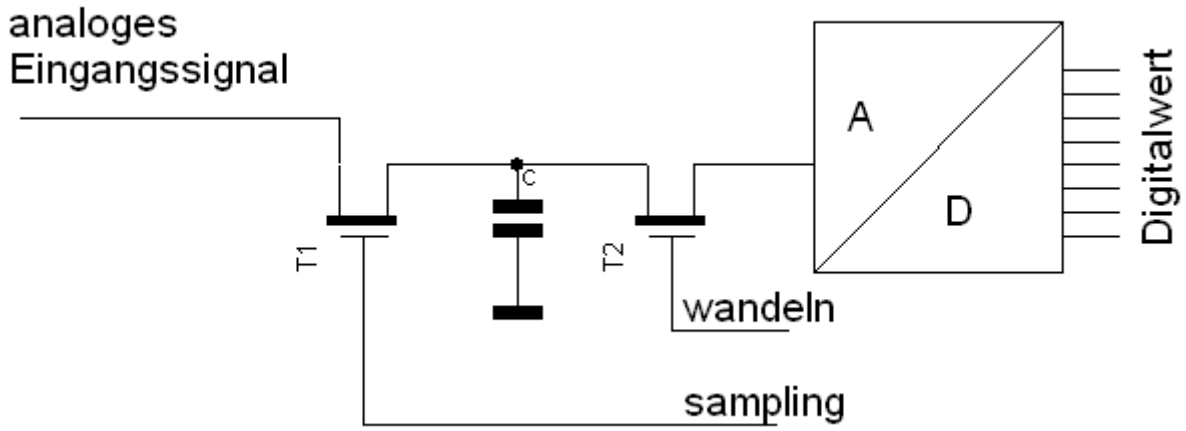


Abbildung 55: Sample & Hold für sich schnell ändernde Eingangssignale

der für einen kurzen Augenblick mit der analogen Eingangsspannung verbunden wird. Danach wird diese Verbindung unterbrochen und der Kondensator mit dem Wandler verbunden. Nun hat der Wandler für die Umsetzung genügend Zeit.

14 Schnittstellen (Punkt-zu-Punkt-Verbindungen)

14.1 Parallele Schnittstellen

Parallele Schnittstellen werden durch schnelle serielle Schnittstellen immer stärker verdrängt. Früher hatten parallele Schnittstellen vor allem den Vorteil der hohen Geschwindigkeit. Nachteilig waren die hohe Anzahl von Leitungen.

14.1.1 Druckerschnittstelle

Sie ist auch als „Centronics-Schnittstelle“ bekannt. Ursprünglich nur mit je einem 36pol. Amphenolstecker realisiert, hat IBM an ihrem PC eine 25pol. Cannon-Buchse eingebaut. Auf Druckerseite bleib die ursprüngliche Buchse bestehen. Es konnten Leitungslängen von typisch 2 Meter bis maximal 5 Meter verwendet werden. Die Signale waren 5V TTL Pegel. Sie hat folgende Signalleitungen:
Das Zeitdiagramm des Zugriffs auf den Drucker ist in Abbildung 56 dargestellt.

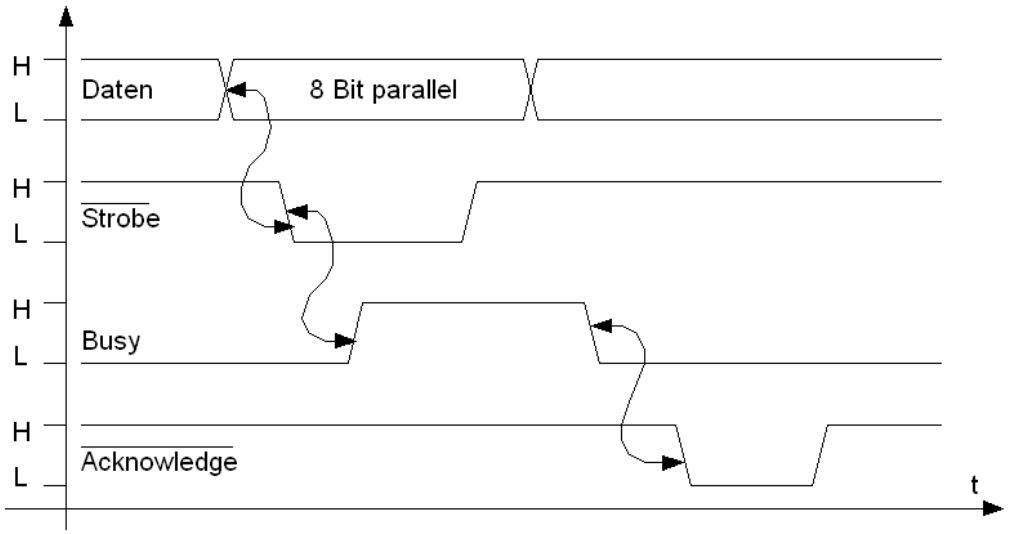


Abbildung 56: Zeitdiagramm einer Datenübertragung mit Handshake

Da für den Handshake drei Leitungen ($\overline{\text{Strobe}}$, Busy und $\overline{\text{Acknowledge}}$) verwendet werden, spricht man von einem 3-Leiter-Handshake.
Manche kleine Protokolldrucker verwenden einen vereinfachten Handshake mit nur zwei Signalleitungen, die $\overline{\text{Acknowledge}}$-Leitung entfällt.

14.1.2 Protokolldrucker M150

Abbildung 57: Normalpapier-Nadeldrucker mit 4 Nadeln

Dieser Drucker besitzt nur 4 Druckernadeln die horizontal angeordnet sind. Jede Nadel ist für den Druck von 6 (maximal 10) Zeichen zuständig. Z.B. 1. Nadel für die Zeichen 1 bis 6, 2. Nadel für die Zeichen 7 bis 12 usw. Das Druckbild wird komplett per Software realisiert, dadurch sind auch Bilder darstellbar. Die Graustufen müssen dabei durch das Weglassen von Punkte realisiert werden.

In Abbildung 59 oben rechts ist die Anschlussbelegung für diesen Drucker zu sehen. Der Reset Detektor erkennt die sogenannte Home-Position des Fahrwagens. Diese Erkennung erlaubt das Drucken eines sauberen linken Randes, während der Timing-Detektor die Geschwindigkeit des Fahrwagens erfasst. Diese Geschwindigkeit kann stark variieren. Gründe dafür sind u.a. die Versorgungsspannung, die Umgebungstemperatur und die Gängigkeit der Lager. Der Motor treibt sowohl den Papiertransport als auch den Fahrwagen an. Die Spulen der Druckernadeln werden über eigene Treibertransistoren mit Spannung versorgt, das der Impulsstrom einer dieser Spulen über 1 A betragen kann.

Das Druckbild ist in Abbildung 58 zu sehen. Man erkennt deutlich die Matrixstruktur der Zeichen. In diesem Fall sind die Zeichen aus einer 5 x 7 Matrix aufgebaut, was keine Unterlängen zulässt.

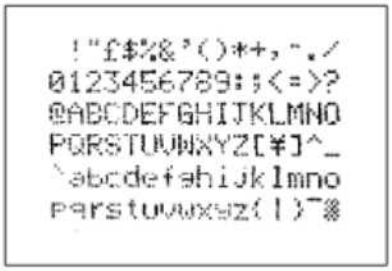


Abbildung 58: Probeausdruck

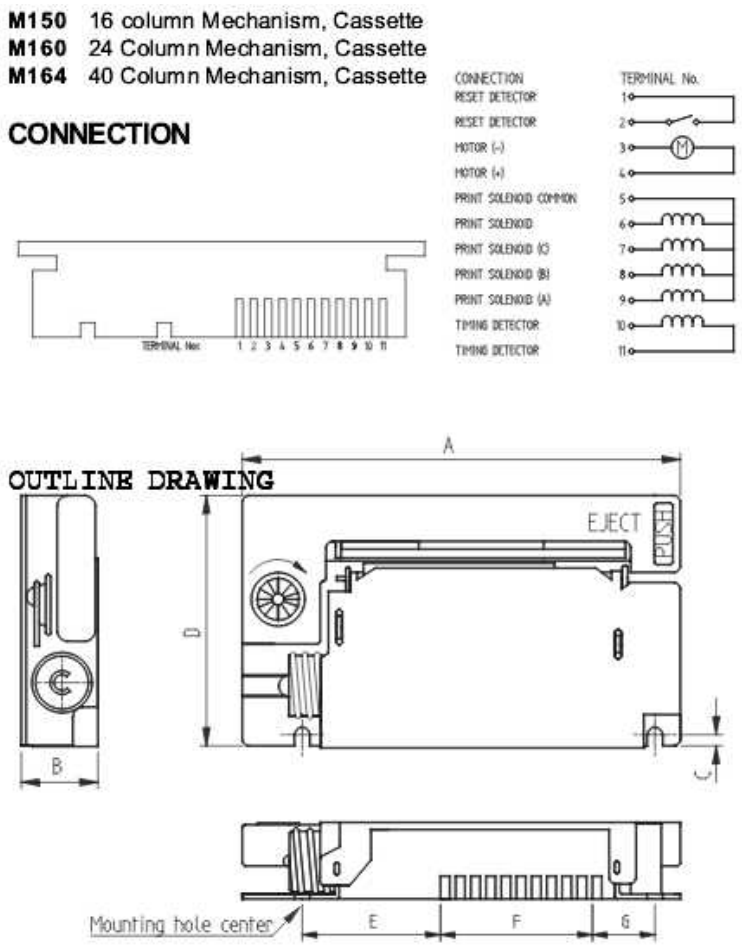


Abbildung 59: Gehäuseabmessungen und Anschlussbelegung

Ein Schaltbild für diesen Druckertyp befindet sich im Anhang auf Seite 76.

14.1.3 Alphanumerische LCD

Bei vielen Kleingeräten oder Automaten besteht die Bedieneranzeige aus einer einfachen alphanumerischen LC-Anzeige. Diese sind äußerst robust und gut lesbar. Ihre Schnittstelle ist oft als Parallelschnittstelle ausgeführt, inzwischen sind auch etwas intelligentere Typen auf dem Markt, die entweder eine SPI- oder I^2C-Schnittstelle besitzen.

Abbildung 60: Alphanumerische LCD

Die Schnittstelle, der in dargestellten LCD ist eine Parallelschnittstelle. Die Kontaktierung ist links vorne erkennbar. Sie besteht aus den 8 Datenleitungen, der Umschaltung zwischen Daten und Steuerzeichen (R/S), der Umschaltung zwischen Lesen und Schreiben (R/W) sowie der Enable-Leitung (E).

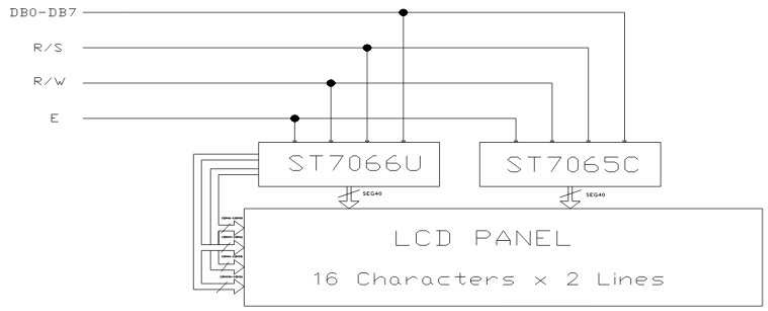


Abbildung 61: Blockschaltbild der LCD

Um I/O-Pins seitens des µC zu sparen, lässt sich die Anzeige auch mit einem 4-Bit-Datenbus ansteuern. In diesem Fall benötigt man zwei Zugriffe auf die LCD.

No.	Symbol	Level	Function	
1	Vss	--	0V	Power Supply
2	Vdd	--	+5V	
3	V0	--	for LCD	
4	RS	H/L	Register Select: H:Data Input L:Instruction Input	
5	R/W	H/L	H--Read L--Write	
6	E	H,H-L	Enable Signal	
7	DB0	H/L	Data bus used in 8 bit transfer	
8	DB1	H/L		
9	DB2	H/L		
10	DB3	H/L		
11	DB4	H/L	Data bus for both 4 and 8 bit transfer	
12	DB5	H/L		
13	DB6	H/L		
14	DB7	H/L		
15	BLA	--	BLACKLIGHT +5V	
16	BLK	--	BLACKLIGHT 0V-	

Abbildung 62: Anschlussbelegung der LCD

Der Schreibzugriff ist in Abbildung 63 dargestellt, der Lesezugriff in Abbildung 64 .

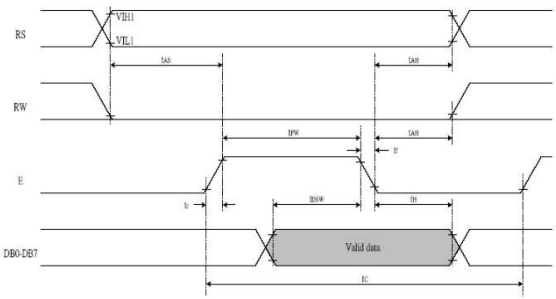


Abbildung 63: Schreiben auf die LCD

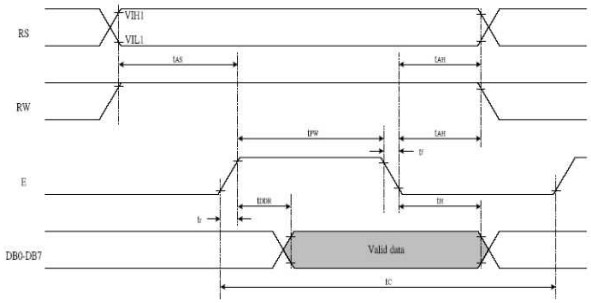


Abbildung 64: Lesen von der LCD

Die LCD kann in verschiedene Betriebszustände versetzt werden. Dazu muss sie entsprechend initialisiert werden. Die nachfolgende Tabelle (Abbildung 65) zeigt die dazu notwendigen Befehlsmuster.

Instruction Table:

Instruction	Instruction Code										Description	Description Time (270KHz)
	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0		
Clear Display	0	0	0	0	0	0	0	0	0	1	Write "20H" to DDRAM, and set DDRAM address to "00H" from AC	1.52 ms
Return Home	0	0	0	0	0	0	0	0	1	x	Set DDRAM address to "00H" from AC and return cursor to its original position if shifted. The contents of DDRAM are not changed.	1.52 ms
Entry Mode Set	0	0	0	0	0	0	0	1	I/D	S	Sets cursor move direction and specifies display shift. These operations are performed during data write and read.	37 us
Display ON/OFF	0	0	0	0	0	0	1	D	C	B	D=1:entire display on C=1:cursor on B=1:cursor position on	37 us
Cursor or Display Shift	0	0	0	0	0	1	S/C	R/L	x	x	Set cursor moving and display shift control bit, and the direction, without changing DDRAM data.	37 us
Function Set	0	0	0	0	1	DL	N	F	x	x	DL:interface data is 8/4 bits N:number of line is 2/1 F:font size is 5x11/5x8	37 us
Set CGRAM address	0	0	0	1	AC5	AC4	AC3	AC2	AC1	AC0	Set CGRAM address in address counter	37 us
Set DDRAM address	0	0	1	AC6	AC5	AC4	AC3	AC2	AC1	AC0	Set DDRAM address in address counter	37 us
Read Busy flag and address	0	1	BF	AC6	AC5	AC4	AC3	AC2	AC1	AC0	Whether during internal operation or not can be known by reading BF. The contents of address counter can also be read.	0 us
Write data to RAM	1	0	D7	D6	D5	D4	D3	D2	D1	D0	Write data into internal RAM (DDRAM/CGRAM)	37 us
Read data from RAM	1	1	D7	D6	D5	D4	D3	D2	D1	D0	Read data from internal RAM (DDRAM/CGRAM)	37 us

Note:
Be sure the ST7066U is not in the busy state (BF = 0) before sending an instruction from the MPU to the ST7066U. If an instruction is sent without checking the busy flag, the time between the first instruction and next instruction will take much longer than the instruction time itself. Refer to Instruction Table for the list of each instruction execution time.

Abbildung 65: Initialisierungsbefehle für die LCD

15 Serielle Schnittstellen

15.1 Serielle Schnittstellen

15.1.1 Synchron / asynchron

Daten lassen sich synchron oder asynchron übertragen. Bei der synchronen Übertragungen wird immer noch ein Taktsignal mitgeliefert. Ob diesem Signal eine eigene Leitung zur Verfügung gestellt wird oder nicht, ist weniger von Belang. Falls eine separate Taktleitung fehlt, wird der Takt aus dem Datenstrom hergeleitet. Das erfolgt mit einer sogenannten Taktrückgewinnungsschaltung, meist in Form einer PLL (Phase Locked Loop).

Bei asynchroner Übertragung muss der Empfänger den Beginn einer Übertragung mitbekommen. Deshalb wird ein Byte mit einem Übertragungsrahmen versehen. Dieser Rahmen besteht aus einem Startbit, einem oder zwei Stopbits und ggf. einem Paritätsbit. Im Ruhezustand ist der Pegel auf High. Daher ist das Startbit immer eine „0“. Es folgen die Datenbits mit D0 als erstes Bit. Sind alle Datenbits übertragen folgt ggf. das Paritätsbit und das Stopbit welches immer „1“ ist.

Bei der Parität unterscheidet man zwischen einer geraden (even) und einer ungeraden (odd) Parität. Das Paritätsbit wird nun immer so gesetzt, dass die Anzahl der Einsen zusammen immer die entsprechende Parität ergibt.

Beispiel:

	gerade Parität:		ungerade Parität
Datenbits	Paritätsbit		Paritätsbit
1010 1010	0		1
0001 1100	1		0

Tabelle 3: Ermittlung des Paritätbits

Der zeitliche Verlauf einer solchen Übertragung ist in Abbildung 66 dargestellt.

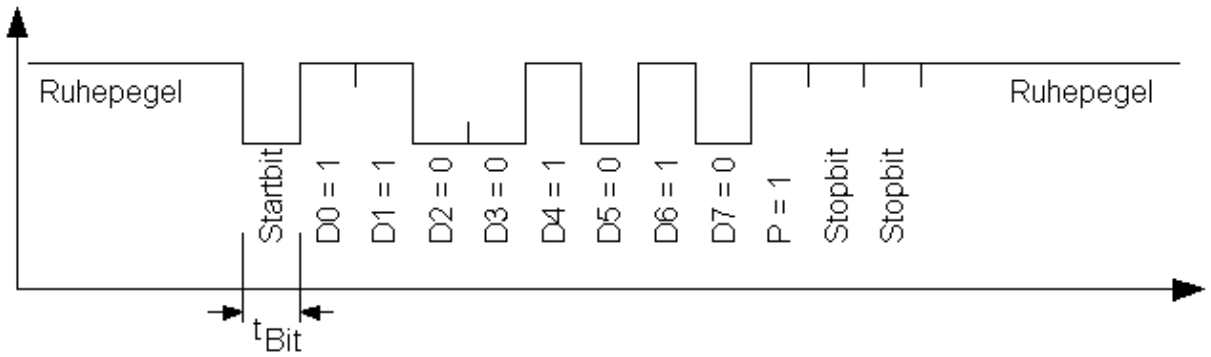


Abbildung 66: Zeitdiagramm einer seriellen Übertragung

Die Zeit t_{Bit} beträgt 208 µs bei 4800 Bit/s bzw. 104 µs bei 9600 Bit/s.

15.1.2 RS-232 (V.24)

Diese Schnittstelle gehört mit zu den ältesten. Dennoch ist sie auch heute noch wegen ihrer Einfachheit sehr beliebt. Die Signale werden als Masse bezogene Spannungen übertragen. Eine logische „1“ wird durch -15 V, eine logische „0“ durch + 15 V (max. +/- 25 V) repräsentiert. Wird durch große Leitungslängen und/oder geringe Leiterquerschnitte die Spannung kleiner als +/- 3V, sind die Signale nicht mehr definiert. Somit gilt dieser Spannungsbereich als unerlaubter Bereich.

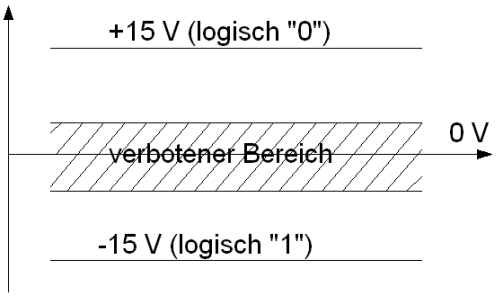


Abbildung 67: RS232 Pegeldefinition

Diese Schnittstelle wurde ursprünglich für die Kommunikation mittels Modem (analoge Telefonverbindung) entwickelt. Deshalb sind viele Signale speziell auf den Modembetrieb zugeschnitten.

Die Signale liegen üblicherweise an einem 25pol. Canon-Stecker an. IBM hat hier einen 9pol. Canon-Stecker eingeführt. Deshalb sind hier nur wenige Signale – i.d.R. die Notwendigen – aufgelegt.

Pin am 25pol. Stecker	**Pin am 9pol. Stecker**	**Bezeichnung**	**Bedeutung**	**Abkürzung**	**Signal-richtung**
1		Common Ground	Abschirmung		
2	3	Transmit Data	Sendeleitung des DTE	TXD	DTE → DCE
3	2	Receive Data	Empfangsleitung des DTE	RXD	DCE → DTE
4	7	Request to Send	Sendeanforderung (DTE will Daten senden)	RTS	DTE → DCE
5	8	Clear to Send	Sendeerlaubnis (DCE ist empfangsbereit)	CTS	DCE → DTE
6	6	Data Set Ready	DCE bereit, nicht unbedingt empfangsbereit	DSR	DCE → DTE
7	5	Ground (Bezug)	Signalbezugspegel	GND	
8	1	Data Carrier Detect	Trägersignal erkannt (Modulationsträger-erkennung	DCD	DCE → DTE
20	4	Data Terminal Ready	DTE signalisiert Betriebsbereitschaft	DTR	DTE → DCE
22	9	Ring Indicator	Zeigt an, dass ein „Anruf“ kommt	RI	DCE → DTE

Tabelle 4: Pinbelegung bei der RS232 Schnittstelle

Ein PC wird als DTE (Data Terminal Equipment) bezeichnet. Da Modem dagegen als DCE (Datenendgerät).

Erläuterung der Kürzel:
Rechner *(DTE = data terminal equipment)* (meistens mit Stecker)
Modem *(DCE = data circuit-terminating equipment)* meistens mit Buchse

15.1.2.1 Nullmodem-Kabel

Werden zwei PCs (DTEs) miteinander verbunden muss ein sogenanntes Nullmodemkabel verwendet werden. Hier sind einige Signale gekreuzt.

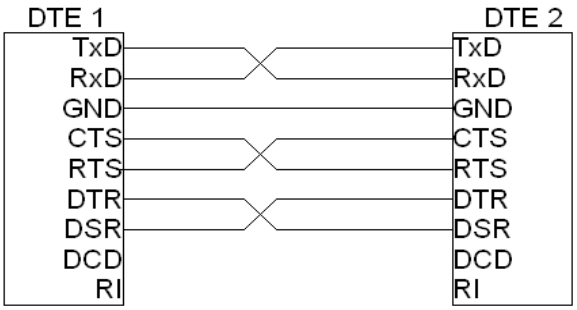


Abbildung 68: Typischer Aufbau eines Nullmodemkabels

Die Leitungen DTR und DSR können u.U. im Stecker direkt gebrückt werden, ohne dass diese Leitungen zum anderen Gerät geführt werden müssen.

15.1.2.2 Hardware-Handshake

Werden sehr hohe Datenraten und große Datenvolumen übertragen, kann es passieren, dass der Empfangsbuffer droht überzulaufen. Damit das nicht passiert kann man zum Handshake greifen. Dazu werden die Leitungen RTS (ready to send) und CTS (clear to send) genutzt.

15.1.2.3 Software-Handshake

Sind nur die TxD- und RxD-Leitung vorhanden, muss sich der Empfänger per Meldung beim Sender melden um die Datenübertragung anzuhalten. Das erfolgt mit den beiden ASCII-Zeichen Xon (11H) und Xoff (13H). Deshalb nennt man diese Art der Kommunikation auch XON/XOFF-Protokoll.

15.1.2.4 Voll- und Halbduplex

Beim Vollduplex können beide Geräte unabhängig voneinander senden und empfangen. Somit sind sowohl auf der TxD- als auch der RxD-Leitung gleichzeitig Daten unterwegs.
Beim Halbduplex hingegen darf immer nur einer Daten senden, der andere nur die Daten empfangen. Soll auch hier eine Kommunikation in beide Richtungen erfolgen, muss zuerst der eine fertig senden, damit der andere seine Antwort zurücksenden kann. Totzdem ist hier i.d.R. auch ein XON/XOFF möglich, da es sich nur um jeweils 1 Zeichen handelt und damit kein Buffer notwendig ist.

15.1.2.5 Normgerechte RS232-Pegel

Um die geforderten Spannungspegel der RS232 zu erzeugen, sind spezielle Bausteine entwickelt worden. Sie erlauben es, Signale direkt aus einem Mikrocontroller um zusetzen. Ein typischer Vertreter dieser Bausteine ist der MAX

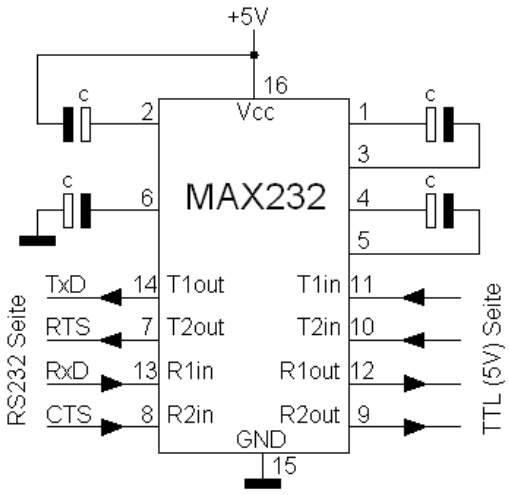


Abbildung 69: Pegelwandler MAX232

232 (Abbildung 69). Um die hohe positive Spannung von mind. 9 V und die negative Spannung von min.
- 9 V zu erzeugen, werden Ladungspumpen verwendet (Konensatoren an Pin 1-3, 4-5, 2-16, 6-15).

15.1.3 RS422A

Die Masse bezogenen Spannungen der RS232 Schnittstelle begrenzen die Kabellänge auf ca. 10 Meter (20 feet). Um größere Entfernungen ohne zusätzliche Repeater zu überbrücken wurde der RS422 Standard geschaffen. Er verwendet differentielle Signale, d.h. die Information steckt nicht mehr in der Amplitude sondern in der Polarität des Signals. Die erreichbaren Leitungslängen sind von der Übertragungsgeschwindigkeit abhängig.

1 MBit / s	13 m
100 kBit / s	1300 m

Die RS422 Schnittstelle wurde zum Bus fähigen RS485 Standard weiterentwickelt. Mehr dazu in Kapitel 16.2.1 auf Seite 106.

15.1.4 SPI

Die SPI-Schnittstelle ist im Prinzip "nur" ein Schieberegister, in das die Daten am MOSI-Eingang mittels Takt SCK hinein- und am anderen Ende herausgeschoben werden.

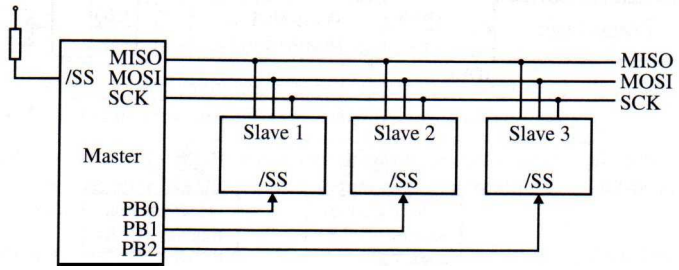


Abbildung 70: SPI-Bausteine an einem µC

Die Daten werden über MOSI vom Master (out) an den Slave (in) gesendet. Der Slave sendet über MISO die Daten weiter, entweder an nachfolgende Slave (Reihenschaltung) oder zurück an den Master. Werden die Slaves parallel betrieben (Sternanordnung), muss wie in das jeweilige Freigabesignal $\overline{SS}$ an dem gewünschten Baustein aktiviert werden. Im Anhang auf Seite 120 ist das Blockschaltbild der SPI-Schnittstelle (Master / Slave) beim PIC abgebildet.

Leider ist diese Schnittstelle nicht konsequent normiert worden, so dass heute unterschiedliche Varianten existieren. Einer dieser Unterschiede ist die Anzahl der Bits die übertragen werden müssen, ein anderer die Übernahmeflanke. Letzteres lässt sich auf dem Zeitdiagramm in an den vier verschiedenen SCK-Zeilen erkennen. Es gilt dabei immer nur eine dieser Zeilen, anhängig vom angeschlossenen Baustein. Erkennbar ist auch der zeitliche Versatz der SDI-Signale im Bezug auf SCK.

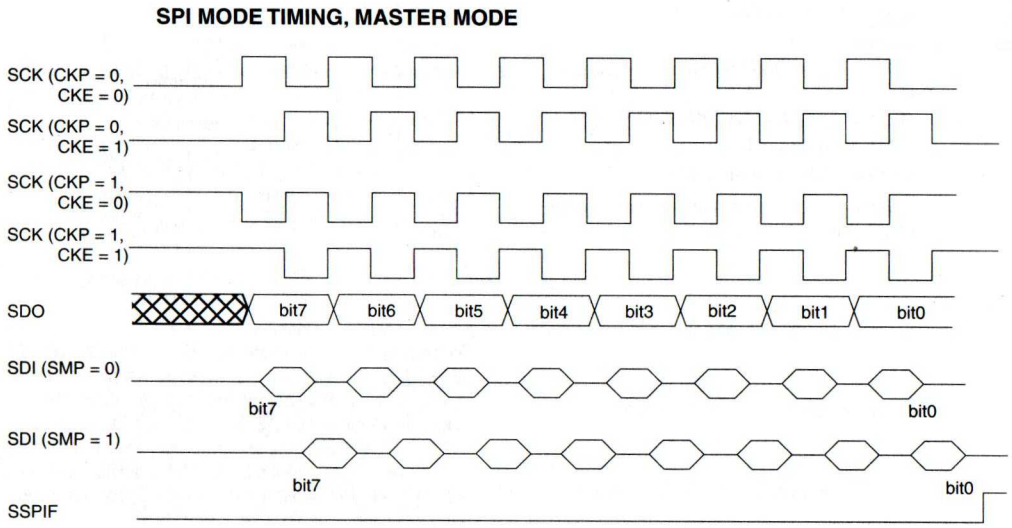


Abbildung 71: Das Zeitdiagramm der SPI-Schnittstelle (Masterbetrieb)

Für den Slave-Modus gilt das Zeitdiagramm in Abbildung 72.

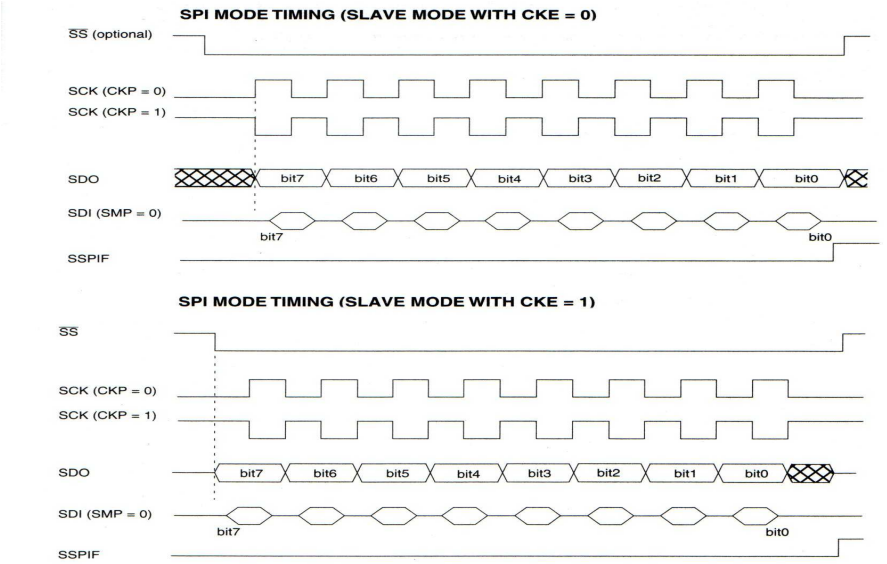


Abbildung 72: Zeitdiagramm der SPI-Schnittstelle (Slave-Modus)

16 Periphere Bussysteme

Anfangs waren die Peripheriegeräte in einer Punkt-zu-Punkt-Verbindung mit dem Rechner verbunden. Dann wollte man mehrere Geräte des gleichen Typs (z.B. Festplatten) anschließen. Dadurch entstand auch der Wunsch, diese einfach an einen Bus anschließen zu können.

Es wurden zwei Arten dieser peripheren Bussysteme entwickelt. Das sogenannte Daisy-Chain und die Party-Line. Dabei spielt es keine Rolle, ob es sich um einen parallelen oder seriellen Bus handelt. Abbildung 73 zeigt die prinzipiellen Unterschiede.

Eine typische Daisy-Chain Anordnung ist das ARCNET-Netzwerk. Dort wird der Token von Teilnehmer zu Teilnehmer geschickt um am Ende wieder beim Sender anzukommen. Dieser kann die ankommenden mit den gesendeten Daten vergleichen und im Fehlerfall sofort reagieren. Können Teilnehmer ausfallen, muss ein Mechanismus dafür sorgen, dass die Verbindung nicht dauerhaft unterbrochen ist.

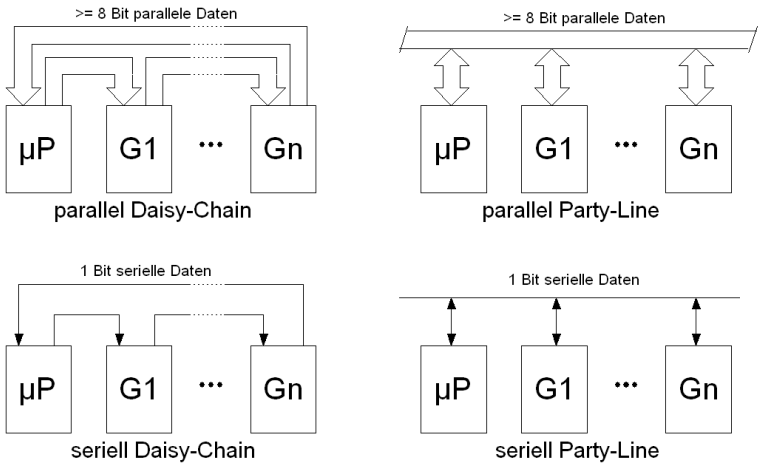


Abbildung 73: Prinzipaufbau von Bussystemen

16.1 Parallele Busse

16.1.1 SCSI-Bus

Der Small-Computer-System-Interface Bus ist ein sehr schneller Bus für hohe Datentransfers z.B. bei Festplatten. Die Datenraten sind so hoch, dass eine Leitungsterminierung unbedingt notwendig ist. Es kann eine aktive Terminierung oder eine passive verwendet werden. Letztere wird bevorzugt eingesetzt.

- Standard-SCSI überträgt die Daten asynchron mit Handshake.
- Fast-SCSI überträgt die Daten synchron und ohne Handshake.
- Ultra-SCSI ist ein Fast-SCSI mit höherem Takt.

16.1.2 IEC-Bus

Er wir auch General Purpose Interface Bus (GPIB) genannt. Er dient vor allem zur Kopplung vom Messgeräten und Messeinrichtungen. Entwickelt von HP um denen Messgeräte in automatisierte Messaufbauten zu integrieren. Damit konnten Langzeitmessungen durch Computer gesteuert und die Ergebnisse protokolliert werden. In Abbildung 74 ist ein typischer Messaufbau dargestellt. Man unterscheidet bei den angeschlossenen Geräten zwischen TALKER. Diese können Daten nur senden. Der Controller adressiert dabei das Messgerät, das dann seine Messwerte sendet. Ein LISTENER kann Daten nur empfangen. Das ist z.B. ein Drucker, auf dem die Messwerte, Uhrzeit oder sonstige Vorkommnisse protokolliert werden.

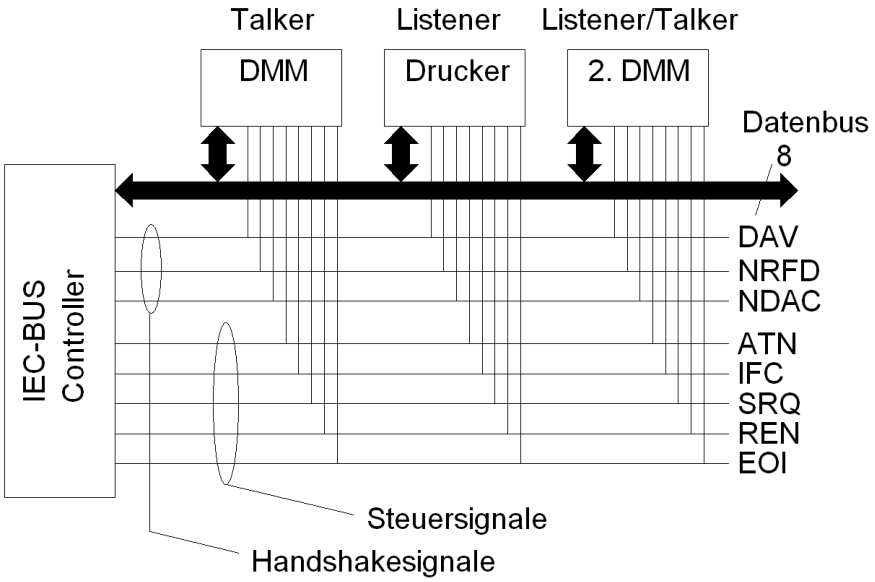


Abbildung 74: Messaufbau durch IEC steuerbar

Das DMM2 kann sowohl als TALKER als auch LISTENER arbeiten. Damit kann mittel Befehl der Messbereich umgeschaltet werden um genauere Ergebnisse zu erhalten.

DAV	DATA VALID	Daten gültig
NRFD	NOT READY FOR DATA	Nicht bereit, weil z.B. gerade gemessen wird
NDAC	NOT DATA ACCEPTED	Daten nicht erkannt
ATN	ATTENTION	leitet einen Datenaustausch ein
IFC	INTERFACE CLEAR	Geräte zurücksetzen
SRQ	SERVICE REQUEST	Talker / Listener meldet sich beim Controller
REN	REMOTE ENABLE	ein Gerät kann fernbedient werden.
EOI	END OF IDENTIFY	zeigt das Ende eines Datenblocks an.

Abbildung 75 zeigt den 3-Leiter-Handshake des IEC-Busses.

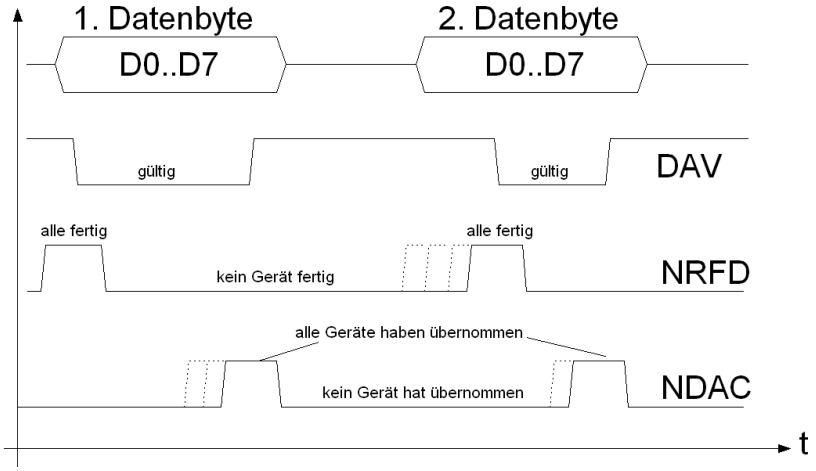


Abbildung 75: Handshake des IEC-Busses

16.2 Serielle Busse

16.2.1 RS485

Der RS485 baut auf der Schnittstelle RS422 auf. Während dort nur eine Punkt-zuPunkt-Verbindung möglich ist, erlaubt RS485 einen Betrieb als Bus. Dazu kann man bei den Treiberbausteinen die Datenrichtung umschalten.

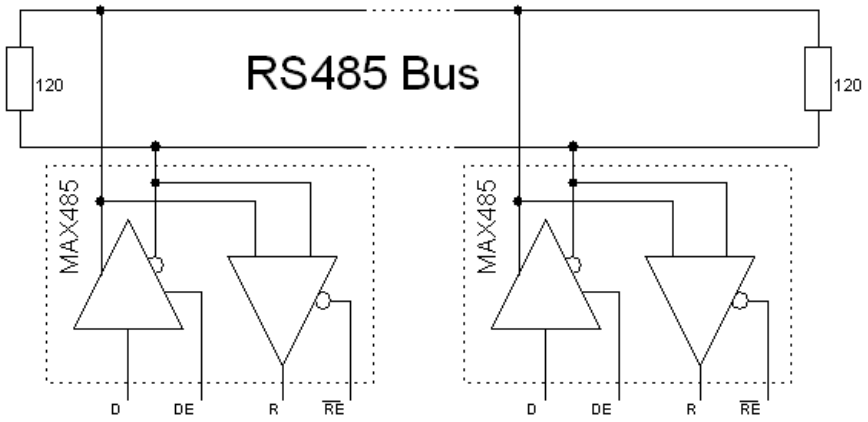


Abbildung 76: Treiberbausteine an einem differentiellen Bus

D	= Datensendeleitung	$\underline{R}$	= Datenempfangsleitung
DE	= High → Senderausgang aktiv	$\overline{RE}$	= Low → Empfangsleitung aktiv

16.2.2 I²C

Beim I2C werden die einzelnen Bausteine mittels eindeutiger Adresse über den Bus aktiviert. Der Bus selbst besteht nur aus zwei Leitungen, der SDA für die Daten und die SCL für die Taktleitung. Wichtig ist in diesem Zusammenhang, dass beide Leitungen über einen eigenen Pullup-Widerstand auf High liegen. Dieser High-Zustand kann nicht von den Busteilnehmern generiert werden sondern ausschließlich durch diese externen Widerstände. Im Gegensatz dazu können die Bustteilnehmer den Pegel auf den Leitungen aktiv auf Low ziehen. Man spricht in diesem Zusammenhang von einem dominanten Low und einem rezessiven High. Somit überschreibt ein Low immer ein High, nicht jedoch umgekehrt.

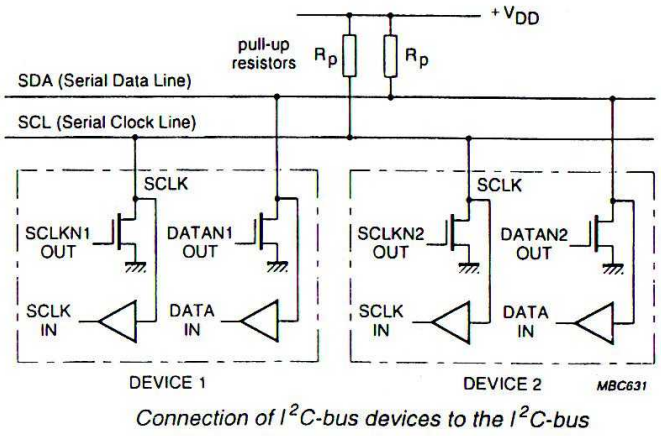


Abbildung 77: Ankopplung der Bausteine an den I²C-Bus

Der gewünschte Baustein muss über den Bus adressiert werden. Dazu besitzt jeder Busteilnehmer eine eindeutige Adresse. Die Empfangslogik vergleicht die eintreffende Adresse mit der eigenen und aktiviert den Funktionsblock. Stimmt die Adresse jedoch nicht überein, wartet der Slave auf die nächste Adresse.

Diese Vorgehensweise setzt jedoch voraus, dass sich eine Adresse eindeutig von den Daten unterscheidet. Weiterhin benötigt auch diese Kommunikation ein eindeutiges Signal, wann die Übertragung durch den Master eingeleitet wird. Die wird durch die sogenannte Start-Condition bewerkstelligt. Eine solche Start-Condition besteht aus einer Ablaufkombination die im Normalbetrieb nie vorkommt. Das gilt auch für die das Ende der Übertragung kennzeichnende Stopp-Condition.

Jeder Datenempfänger quittiert den Empfang mit einem Acknowledge (ACK). Fehlt dieses ACK, bricht der Sender die Übertragung ab. I.d.R. ist dies der Master, der dazu eine Stopp-Condition sendet.

Eine typische Übertragung mit einer 7-Bit Geräteadresse läuft aus Sicht des Masters wie folgt ab:

- Prüfen, ob aktuell keine Datenübertragung läuft.
- Erzeugen einer Start-Condition
- Senden der 7-Bit Adresse und des R/W-Bits (Low für Schreiben)
- Warten bis Empfänger ein ACK sendet
- Senden weiterer Datenbits
- Erzeugen einer Stopp-Condition

Abbildung 78 zeigt diesen Datentransfer aus Sicht des Slaves. Er ist es, der das ACK-Bit auf Low zieht um den Empfang zu bestätigen.

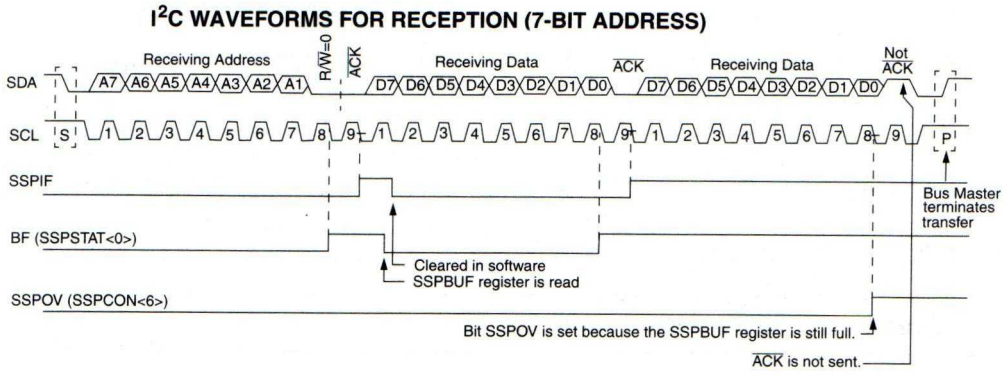


Abbildung 78: Ein Slave empfängt Adresse und Daten

Will der Master Daten vom Slave lesen, erfolgt dies mit folgendem Transfer:

- Prüfen, ob aktuell keine Datenübertragung läuft.
- Erzeugen einer Start-Condition
- Senden der 7-Bit Adresse und des R/W-Bits (High für Lesen)(
- Warten bis Empfänger ein ACK sendet
- Datenleitung umschalten auf Eingang, Slave muss auf Ausgang umschalten
- Empfang der 8 Datenbits
- Master quittiert Empfang mit ACK
- ggf. weitere Daten empfangen und quittieren
- Stopp-Condition erzeugen

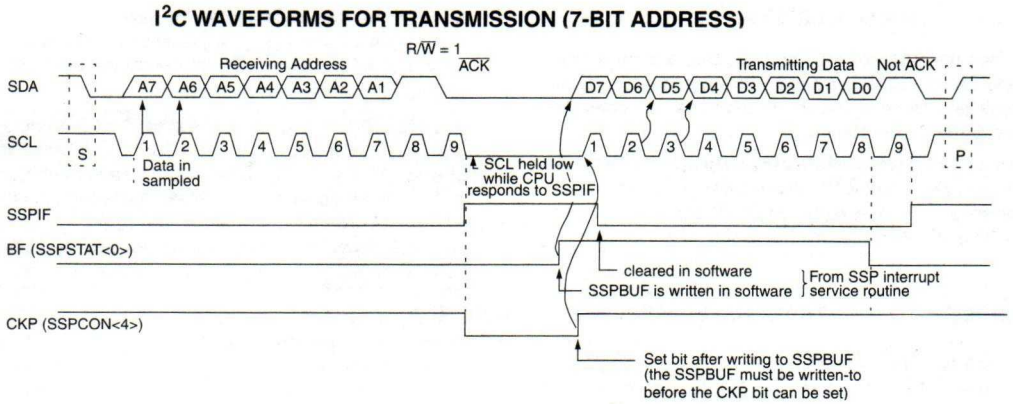


Abbildung 79: Slave wird adressiert und sendet anschließend die Daten

Die Adresse beträgt beim I²C-Bus üblicherweise 7 Bits was einer Anzahl von 128 Adressen entspricht. Man erkannt bald, dass diese Anzahl nicht reichen

würde und man entschloss sich, einen erweiterten Adressbereich zu definieren. Die 10-Bit-Adressierung benötigt zwei Adressblöcke zur Adressierung. Im ersten Block sind eine spezielle Kennung, die zwei höchstwertigen Adressbits und das R/W-Bit untergebracht (11110A_9A_8R/W). Danach folgen die nächsten 8 Adressbits.

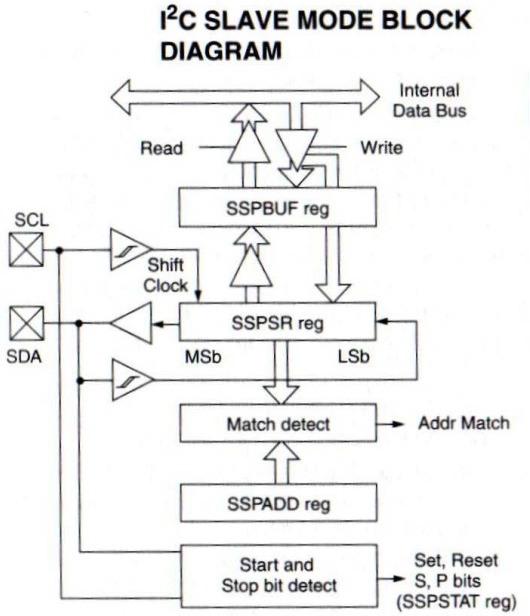


Abbildung 80: I²C-Slave Modul im PIC

1	0	1	0	A2	A1	A0	R/W

Adresse (hex)					A2	A1	A0	R/W
A0 - A1	1	0	1	0	0	0	0	x
A2 - A3	1	0	1	0	0	0	1	x
A4 - A5	1	0	1	0	0	1	0	x
A6 - A7	1	0	1	0	0	1	1	x
A8 - A9	1	0	1	0	1	0	0	x
AA - AB	1	0	1	0	1	0	1	x
AC - AD	1	0	1	0	1	1	0	x
AE - AF	1	0	1	0	1	1	1	x

Daten schreiben

	7 Adr.-bits	R/W	ACK	Adresszeiger	ACK	Daten	ACK	....	ACK/NAK	
S	1 0 1 0 X X X	0	0	ZZZZZZZZ	0	DDDDDDDD	0	.....	0/1	P

Daten lesen

	7 Adr.-bits	R/W	ACK	Adresszeiger	ACK	
S	1 0 1 0 X X X	0	0	ZZZZZZZZ	0	P

	7 Adr.-bits	R/W	ACK	Daten	ACK	Daten	ACK	....	ACK/NAK	
S	1 0 1 0 X X X	1	0	DDDDDDDD	0	DDDDDDDD	0	.....	0/1	P

repeated start conditions

	7 Adr.-bits	R/W	ACK	Adresszeiger	ACK		7 Adr.-bits	R/W	ACK	Daten	ACK	....	ACK/NAK	
S	1 0 1 0 X X X	0	0	ZZZZZZZZ	0	S	1 0 1 0 X X X	1	0	DDDDDDDD	0	.....	0/1	P

10-Bit-Adressierung

	Kennung + 2 Adr.-bits	R/W	ACK	8 Adr.-bits	ACK	Daten	ACK	....	ACK/NAK	
S	1 1 1 1 0 A A	0	0	AAAAAAA	0	DDDDDDDD	0	.....	0/1	P

16.2.3 USB[14]

Die physikalische Ebene auf der USB seine Daten überträgt entspricht dem RS485 Standard. Im 4-adrigen Kabel sind neben der Datenleitung auch die Spannung 5V zum betrieb kleinerer Geräte. Der Strom der pro Anschluss zur Verfügung gestellt wird ist begrenzt. Falls die Leistungsaufnahme größer sein sollte, wird ein eigenes Netzteil benötigt.

Die USB-Schnittstelle ist was ihren Datenaufbau und Administration betrifft ziemlich komplex. Es gibt Mikrocontroller, die eine USB-Schnittstelle integriert haben. Hier muss aber er gesamte Applikationsstack programmiert werden. Dies setzt intime Kenntnisse in diesem Bereich voraus. Alternativen sind der eine oder andere Compiler mit passende Routinen bzw. Befehlen oder man benutzt die Bibliothek der Controllerhersteller.

Man unterscheidet zwischen USB-Host und USB-Device. Als Device werden u.a. Drucker, USB-Speichersticks, Maus usw. bezeichnet.

Device

Die gesamte Funktionalität eines USB-Device lässt sich aber auch nach außen in einen speziellen Baustein auslagern. Die Kommunikation zwischen Mikrocontroller und diesem Schnittstellenbaustein erfolgt i.d.R. mittels seriellem Pro-

14 Der USB ist streng genommen kein Bus sondern eine Schnittstelle mit einer Punkt-zu-Punkt-Verbindung.

tokoll[15] oder über I²C bzw. SPI.

Ein solcher Schnittstellenbaustein ist der FT232R (USB UART IC) von FTDI.

Dieser Baustein verhält sich auf der Mikrocontrollerseite wie eine serielle Schnittstelle, allerdings mit TTL-Pegel. Auf der anderen Seite ist es eine USB-Verbindung. Will man den Mikrocontroller über diesen Baustein mit einem PC verbinden, muss man auf dem PC einen sogenannten virtuellen COM-Port einrichten. Die passenden Treiber liefern die Hersteller dieser Schnittstellenbausteine. Serielle Schnittstellen sind auf einem PC relativ leicht per Hochsprache anzusprechen. In der Regel öffnet man entsprechende Dateien und kann über diese mit der virtuellen Schnittstelle kommunizieren.

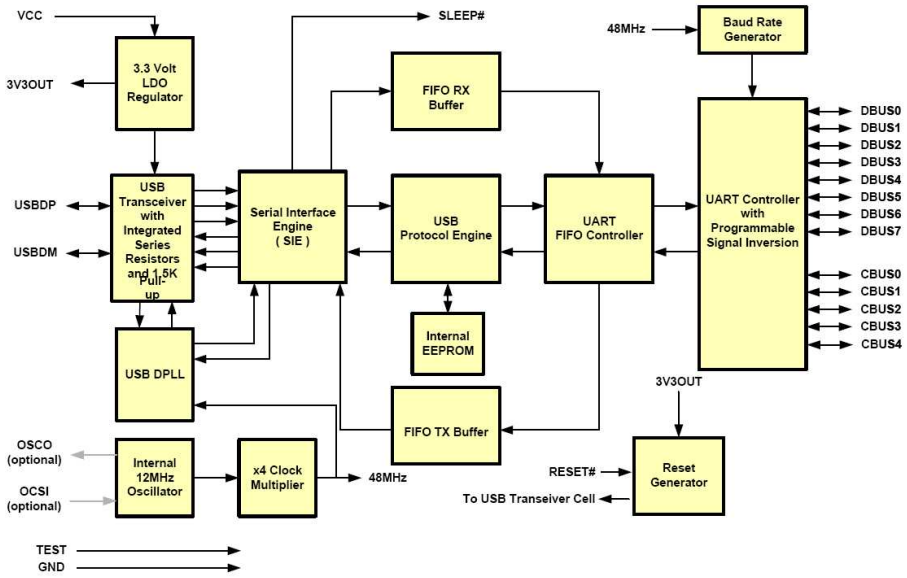


Abbildung 81: Blockschaltbild FT232R

Die Pinbelegung gibt Auskunft, wie dieser Baustein mit dem Mikrocontroller verbunden werden kann (Abbildung 82).

15 Siehe Kapitel 15.1.1 auf Seite 96

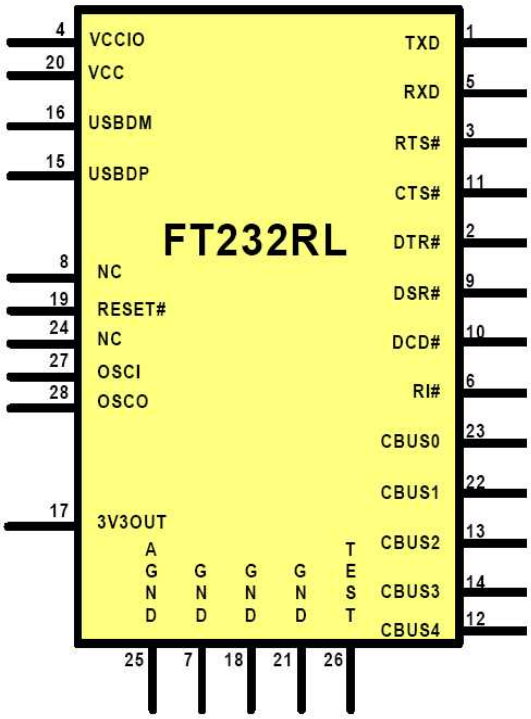


Abbildung 82: Anschlussbelegung des FT232RL

Host

Will man einen USB-Host mit seinem Mikrocontroller aufbauen, weil man beispielsweise Daten in einem USB-Stick so abspeichern will, dass man sie mit jedem PC wieder lesen kann[16], braucht man einen anderen Baustein. Ein solcher ist der Vinculum-II Embeded Dual USB Host Controller.

Auch er wird über eine serielle Verbindung an den Mikrocontroller angeschlossen. Mittels Befehlen lassen sich Dateien anlegen, löschen, öffnen und schließen.

Das Blockschaltbild und eine Befehlsübersicht befindet sich im Anhang 17.6 auf Seite 121.

16 Die Daten müssen als Datei unter einer Dateistruktur wie z.B. FAT abgespeichert werden.

17 Anhang

17.1 Nyquist-Kriterium

Das Nyquist-Kriterium bzw. das Shannon-Theorem besagen, dass ein periodisches Signal mit mindestens der doppelten Eigenfrequenz abgetastet werden muss, damit aus den Abtastwerten das Originalsignal wieder hergestellt werden kann. Dabei sind auch die Oberwellen des Eingangssignals zu berücksichtigen. Falls dies nicht gewährleistet ist, können sogenannte Aliasfrequenzen entstehen. Das sind Signale mit Frequenzen, die im Original nicht vorkommen.

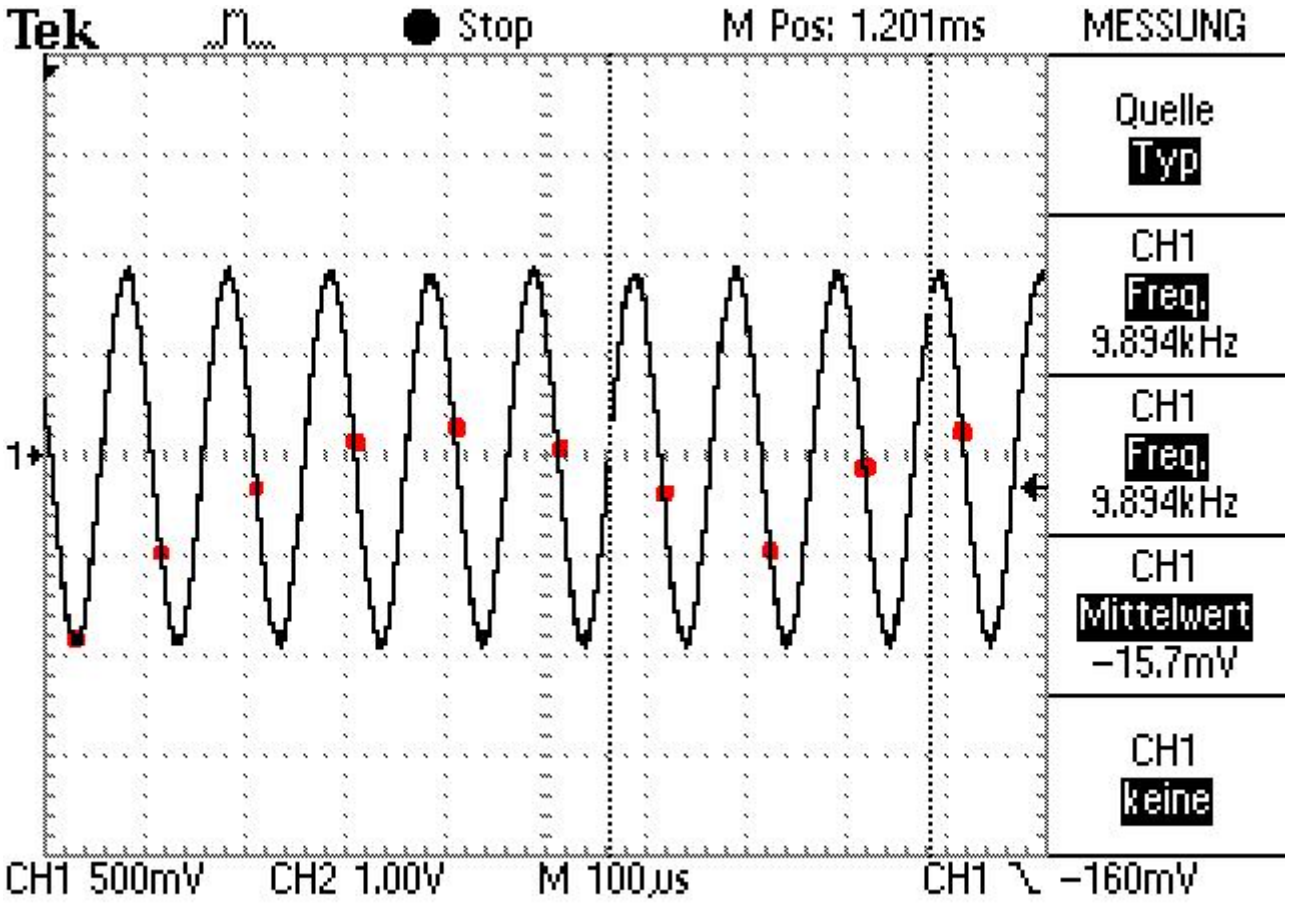


Abbildung 83: Ein Signal wird zu langsam abgetastet

In Abbildung 83 sieht man ein fehlerhaft abgetastetes Signal. Die roten Abtastzeitpunkte ergeben ein neues Signal, das mit dem ursprünglichen nichts gemein hat.

Abhilfe schafft ein Tiefpass, dessen Grenzfrequenz niedrig genug gewählt wird, damit die obigen Bedingungen eingehalten werden können.

17.2 Leitungsterminierung

Unter Leitungsterminierung versteht man den reflexionsfreien Abschluss einer Signalleitung. Ist die Signaldauer sehr viel kürzer als die Signallaufzeit auf den Leitungssegment, kann es zu Reflexionen kommen.
Der Impuls wandert auf der Leitung weg von der Quelle. Kommt er an ein offenes oder kurzgeschlossenes Ende der Leitung, muss der Impuls wieder zurücklaufen. Ist das Ende offen, wird der Impuls mit der gleichen Phasenlage wie der einlaufende reflektiert. Bei einem kurzgeschlossenen Ende wird die Phasenlage des reflektierten Signals um 180° gedreht.
Eine Reflexion wird dann verhindert, wenn der Impuls am Ende der Leitung vollständig in den Empfänger übertragen wird. Das erfolgt indem seine Energie in einem Widerstand (Eingangswiderstand des Empfängers) vollständig in Wärme umgesetzt wird. Die Voraussetzung dazu ist, dass dieser Widerstand mit dem Innenwiderstand des Generators und dem Wellenwiderstand der Leitung übereinstimmt.

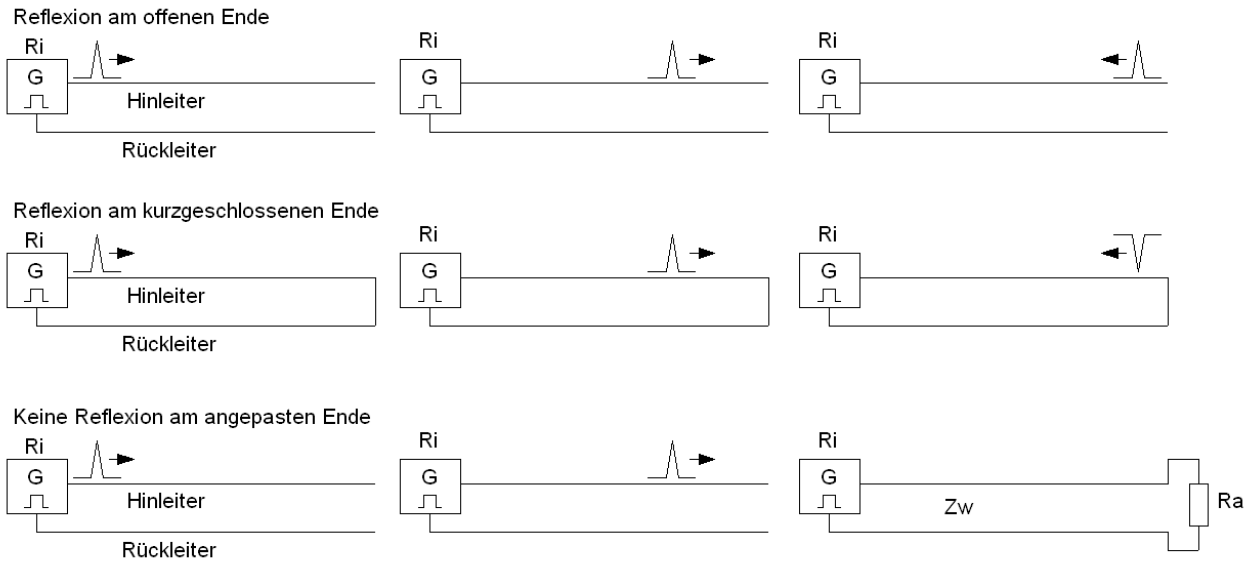


Abbildung 84: Reflexionen bei unterschiedlichen Leitungsabschlüssen

Der Reflexionsfaktor gibt an, wie viel Energie reflektiert wird und lässt sich über die Gleichung:

$$r = \frac{R_a - Z_w}{Z_w + Z_w}$$

ermitteln.

17.3 Speicherelemente

Die Speichertechnologien sind sehr vielfältig. Es folgen deshalb nur Prinzipschaltbilder und eine äußert oberflächliche Beschreibung.

17.3.1 Statischer Speicher (Flip-Flop)

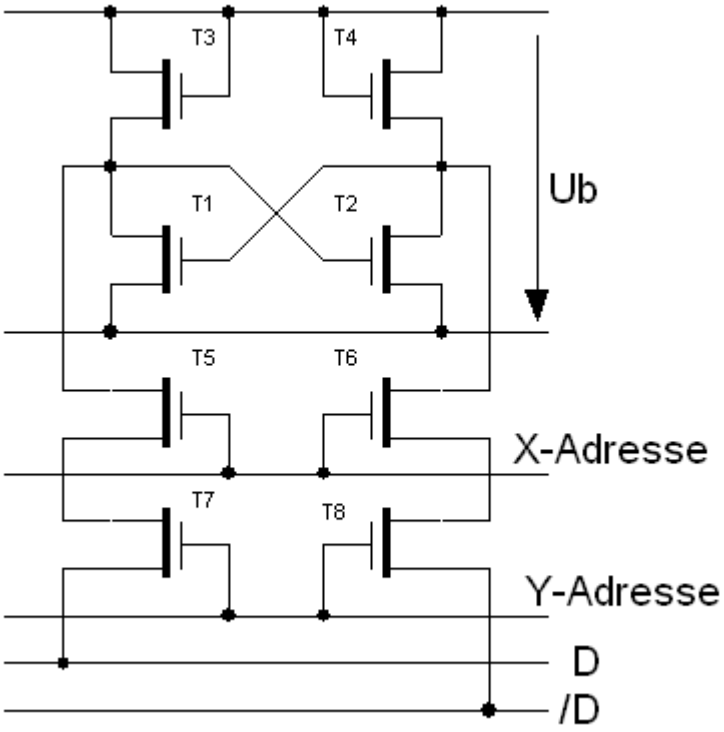


Abbildung 85: Aufbau einer statischen Speicherzelle

Das eigentliche Speicherglied besteht aus T1 und T2. Ist T1 durchgeschaltet, liegt dessen Drain auf 0 Volt und damit auch das Gate von T2. Damit wird T2 gesperrt und an dessen Drain liegt ein High (Betriebsspannung) an. Dies ist auch die Spannung an T1, der aber bereits durchgeschaltet ist. An dem Drain von T1 liegt das Datenbit D, an Drain von T2 das invertierte Datenbit $\overline{D}$ an.

Die Transistoren T3 und T4 sind die Drainwiderstände für T1 und T2.

T5 bis T8 sind als Schalter verschaltet. Sobald an deren Gate ein High anliegt, wird deren Drain-Source-Strecke leitend. Liegt z.B. die X-Adresse auf High (aktiver Zustand), sind T5 und T6 leitend, so dass das Signal D vom Drain von T1 bzw. $\overline{D}$ von T2 auch am Drain von T7 bwz. T8 erscheint. Ist auch die Y-Adressleitung auf High, wird das Datensignal an die Datenleitung D bzw. $\overline{D}$

durchgeleitet.

17.3.2 Dynamischer Speicher

In einer dynamischen Speicherzelle dient eine Kapazität als eigentliches Speicherelement. Der Kondensator C in Abbildung 86 ist eigentlich die Transistor eigene Gate-Source Kapazität. Soll eine 1 in die Speicherzelle geschrieben wer

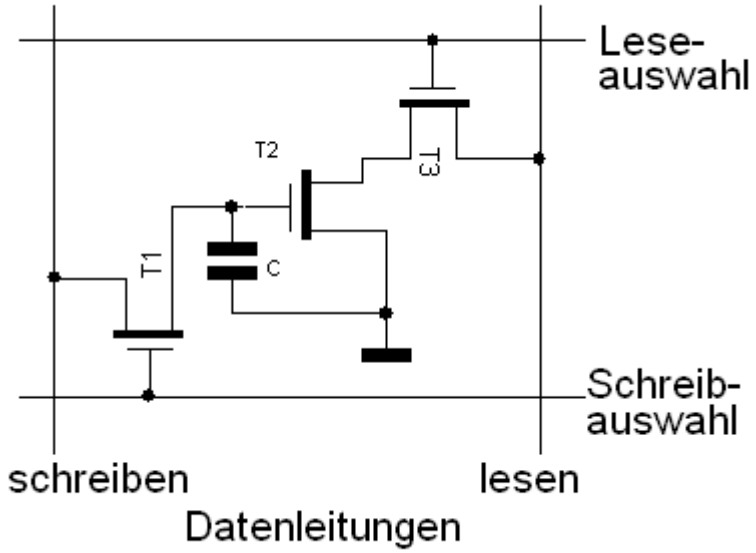


Abbildung 86: Aufbau einer dynamischen Speicherzelle

den, wird die Schreib-Datenleitung auf 1 gesetzt und die Schreibauswahl aktiviert. Um die Zelle auszulesen, wird die Leseauswahl aktiviert. Damit wird der Schalter T3 geöffnet und der Pegel am Drain von T2 auf die Datenleitung 'lesen' durchgeschaltet.

Die Speicherkapazität ist sehr klein und wird durch den Leckstrom kontinuierlich entladen. Damit würde die Zelle ihren gespeicherten Wert verlieren, wenn man die Zelle nicht regelmäßig auffrischen (refresh) würde. Das passiert zu einem Zeitpunkt, an dem keine Speicherzugriff durch die CPU getätigt wird. Dabei ist der Refresh nur das Lesen und Zurückschreiben des Zelleninhaltes. Ein auf 50% abgesunkener Spannungspegel am Kondensator wird immer noch als eine 1 erkannt. Beim Rückschreiben wird jedoch ein 100%-iger Pegel in den Kondensator geladen.

17.3.3 ROM-Zelle

Die ersten ROM-Bausteine waren maskenprogrammiert. Das Programm wurde

somit während der Herstellung des Bausteins mit Hilfe von entsprechenden Masken vorgegeben. Eine Änderung oder eine Programmierung außerhalb der Halbleiterfabrik waren nicht möglich.

17.3.4 PROM-Zelle

Der nächste Schritt war die Möglichkeit den Baustein individuell zu programmieren und das beim Geräteentwickler. Damit konnten massenweise unprogrammierte Speicherbausteine hergestellt werden. Das Programmieren erfolgte in einem speziellen Programmiergerät. Dabei wurden ganz feine metallische Verbindungen mittel einer hohen Programmierspannung (ca. 24V) durchgebrannt. Dieser Vorgang war nicht reversibel und somit ein Umprogrammieren nicht möglich. Dies wurde als One-Time-Programmable (OTP) bezeichnet.

17.3.5 EPROM-Zelle

Um die programmierte Information wieder löschen zu können, müsste eine andere Technologie entwickelt werden. Die neue Speicherzelle besteht aus einem Transistor dessen Gate auf einem hochisolierenden SiO_2 liegt. Im gelöschten Zustand sperrt dieser Transistor und die Information dieser Zelle ist 1 (high). Beim Programmieren wird Ladung auf dieses sogenannte Floating Gate gebracht, so dass der Transistor durchschaltet und eine 0 (low) liefert. Die Hersteller garantieren einen Datenerhalt von mind. 10 Jahre was eine extrem hochohmige Einbettung des Gates in der Siliziumdioxid Schicht bedeutet. Durch Bestrahlung mittels hochenergetischem Licht (UV-Licht) wird der Bereich um das floating Gate niederohmiger und die auf dem Gate liegenden Ladungen können abfließen. Dadurch gelangt der Transistor wieder in den Sperrzustand, der Baustein ist gelöscht. Damit das UV-Licht direkt auf das Silizium gelangen kann, besitzen diese Speicherbausteine ein Quarzfenster genau über dem Die[17].

17.3.6 EEPROM-Zelle

Im Prinzip ist die EEPROM-Zelle wie eine EPROM-Zelle aufgebaut. Lediglich der Löschvorgang erfolgt hier nicht mit UV-Licht sondern über einen zusätzlichen elektrischen Schalter, der die Ladungen vom Gate abfließen lässt.

17 Die nennt man den kleinen Siliziumchip der auf einen Träger gebondet wird.

17.4 TRI-State-Teiber

Neben den beiden Zuständen HIGH und LOW gibt es noch einen dritten, den sogenannten Z-Zustand (z-state) oder TRI-State. Während HIGH und LOW dominante Signale sind, d.h. der Innenwiderstand ist niederohmig, ist der TRI-State ein rezessives Signal mit einem sehr hohen Innenwiderstand. In

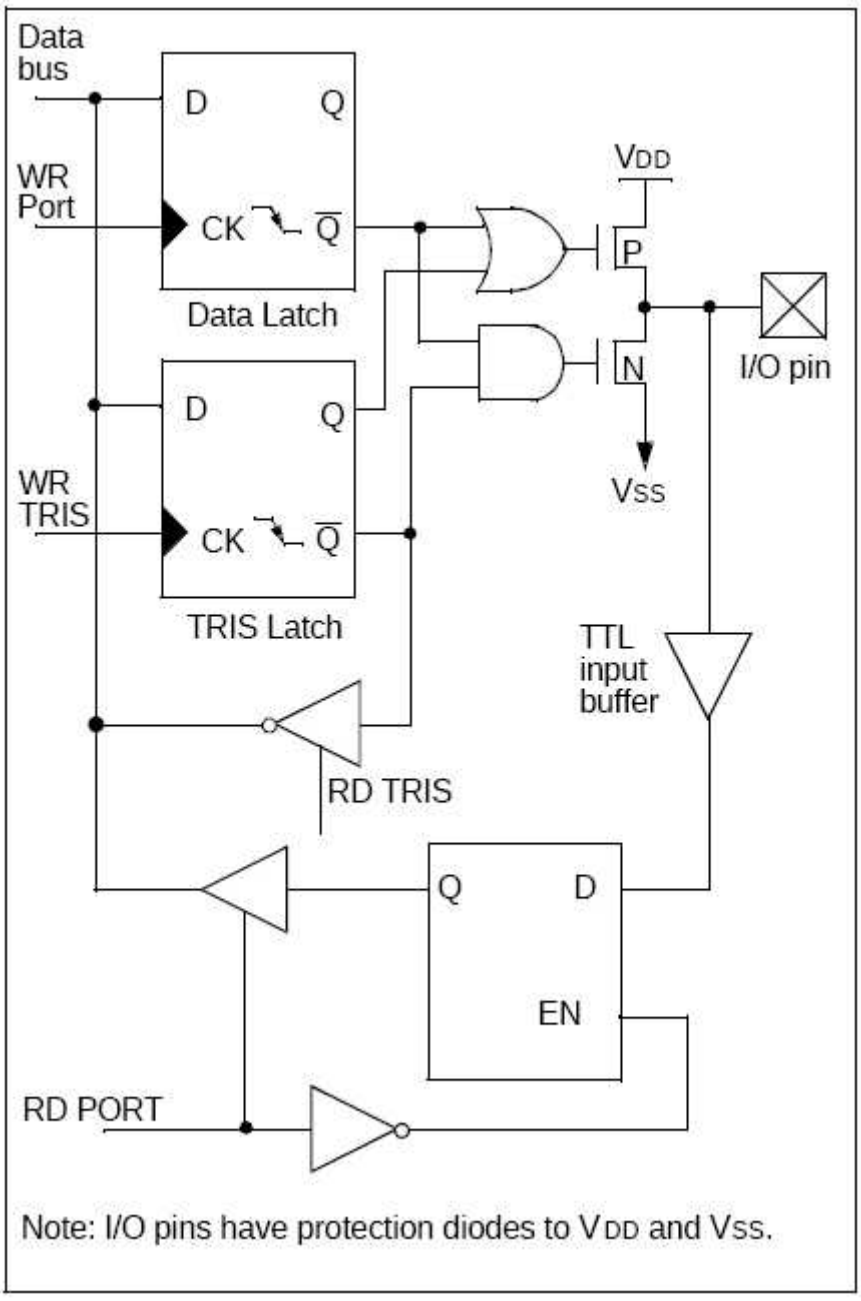


Abbildung 87: Beschaltung eines I/O-Pins beim PIC

Abbildung 87 ist das Blockschaltbild eines I/O-Pins zu sehen. Die beiden Transistoren P und N sind für die HIGH- und LOW-Pegel verantwortlich. Sie schalten zum Einen eine 1 (P-Transistor) oder eine 0 (N-Transistor) auf den I/O-Pin. Die beiden Transistoren sind dann jeweils durchgeschaltet, was zum niederoh-

migen Innenwiderstand führt. Ob einer der Tansistoren überhaupt durchschalten kann, entscheidet was im TRIS-Latch steht. Nur wenn dort eine 0 steht, wird über die Verriegelung mittels UND- und ODER-Gatter, einer der Transistoren geschaltet. Steht dagegen im TRIS-Latch eine 1, kann keiner der beiden Transistoren angesteuert werden. Deshalb ist der Innenwiderstand des I/O-Pins nun hochohmig.

Komponenten, die an einen Bus angeschlossen werden sollen, müssen über TRI-State-Treiber verfügen. Denn durch diesen hochohmigen Zustand ist das Bauteil praktisch vom Bus abgekoppelt (inaktiv), kann jedoch den Datenfluss "mithören".

17.5 SPI-Schnittstelle im PIC-Baustein 16F887

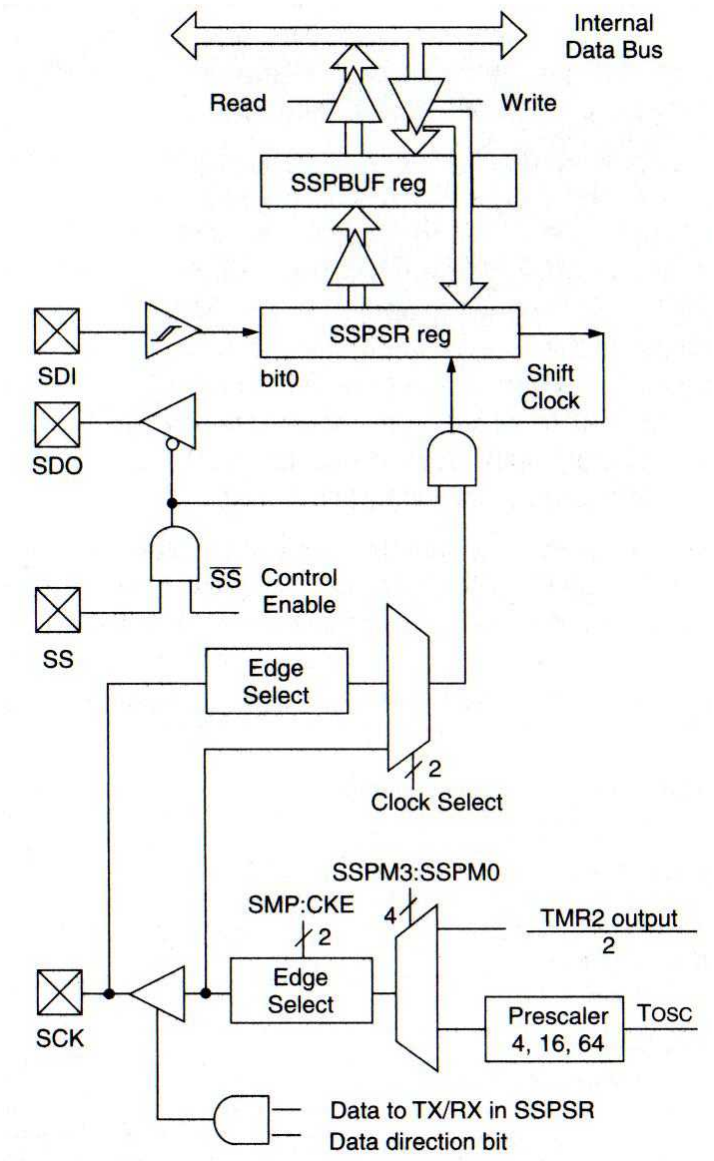


Abbildung 88: Blockschaltbild der SPI-Schnittstelle

17.6 USB Vinculum (Fortsetzung)

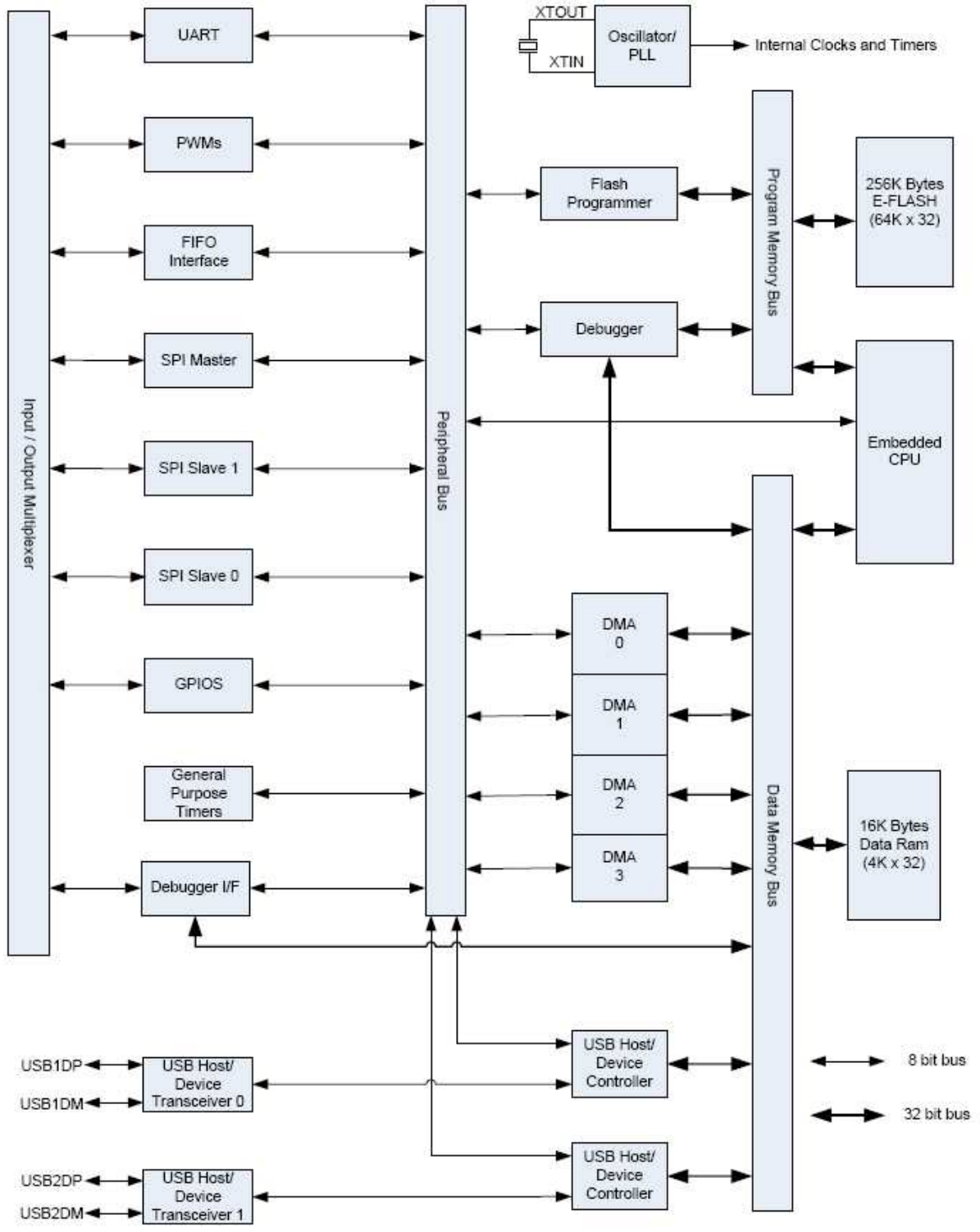


Abbildung 89: Blockschaltbild des Vinculum

Fehlercodes:

Extended Command Set	Short Command Set	Reason
Bad·Command	BC (*42 43 0D*)	Command not recognised.
Command·Failed	CF (*43 46 0D*)	Filename or directory name not found.
Disk·Full	DF (*44 46 0D*)	No free space on disk.
Invalid	FI (*46 49 0D*)	Attempt to open a directory for reading or writing. Attempt to change currently selected directory to a file.
Read·Only	RO (*52 4F 0D*)	Attempt to open a read only file for writing.
File·Open	FO (*46 4F 0D*)	A file is currently open for writing and must be closed before this command can be executed.
Dir·Not·Empty	NE (*4E 45 0D*)	Attempt to delete a directory which is not empty.
Filename·Invalid	FN (*46 4E 0D*)	Firmware invalid or contains disallowed characters
No·Upgrade	NU (*4E 55 0D*)	Firmware Upgrade file not found on disk

Monitor Events

Event	Extended Command Set	Short Command Set
Slave Device inserted in USB Port 1	Device·Detected·P1	DD1 (*44 44 31 0D*)
Slave Device removed from USB Port 1	Device·Removed·P1	DR1 (*44 52 31 0D*)
Slave Device inserted in USB Port 2	Device·Detected·P2	DD2 (*44 44 32 0D*)
Slave Device removed from USB Port 2	Device·Removed·P2	DR2 (*44 52 32 0D*)
Connected to Host Device	Slave·Enabled	SDA (*53 44 41 0D*)
Disconnected from Host Device	Slave·Disabled	SDD (*53 44 44 0D*)

Monitor Konfigurationsbefehle

Extended Command Set	Short Command Set (Hexadecimal Codes)	Function
SCS	*10 0D*	Switches to the shortened command set
ECS	*11 0D*	Switches to the extended command set
IPA	*90 0D*	Monitor commands use ASCII values
IPH	*91 0D*	Monitor commands use binary values
SBD·*divisor*	*14 20 divisor 0D*	Change monitor baud rate

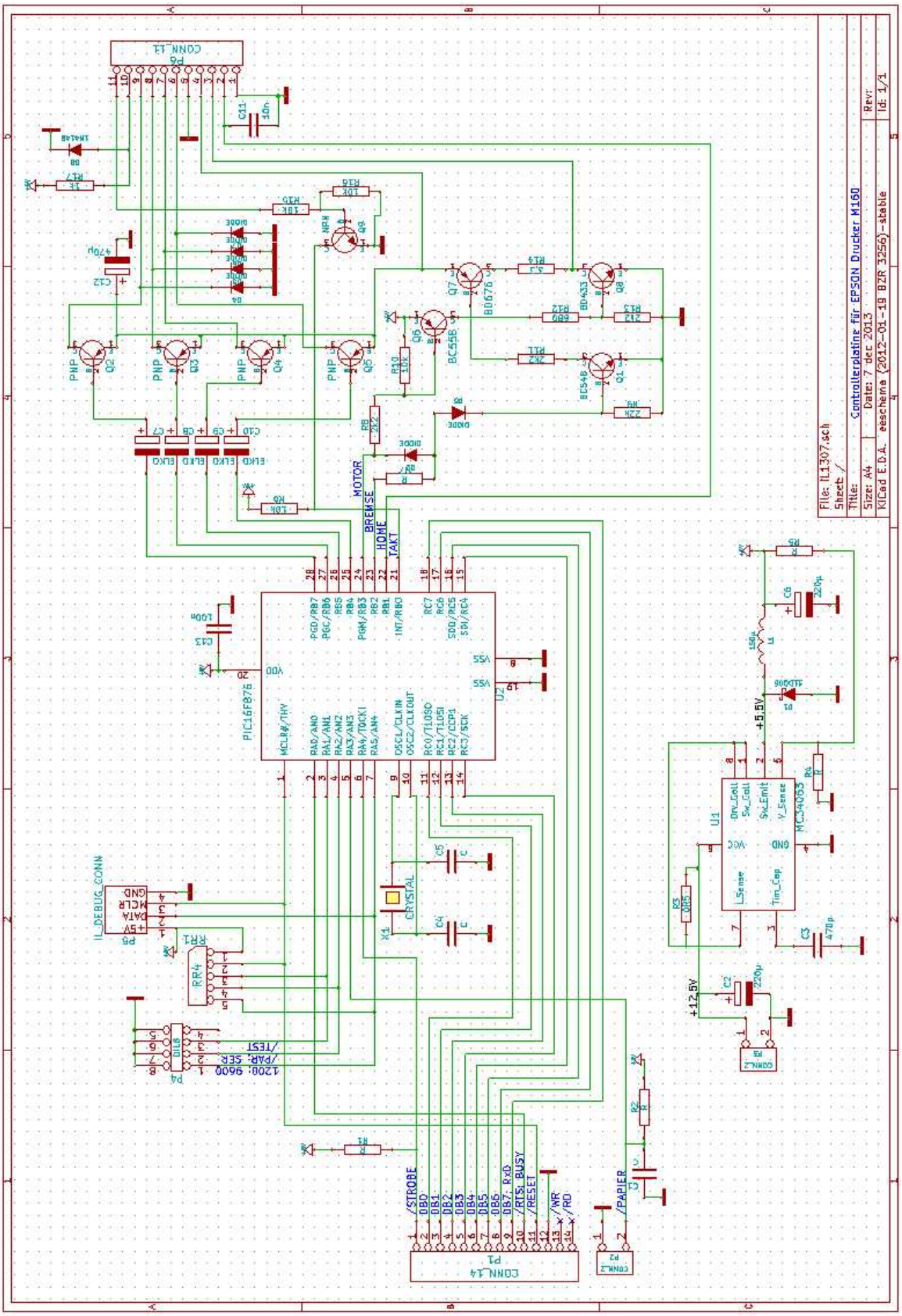
FWV	*13 0D*	Display firmware version
E	*45 0D*	Echo ‘E’ for synchronisation
e	*65 0D*	Echo ‘e’ for synchronisation

Dateibefehle

Extended Command Set	Short Command Set (Hexadecimal Codes)	Function
DIR	*01 0D*	List files in current directory
DIR·*file*	*01 20 file 0D*	List specified file and size
CD·*file*	*02 20 file 0D*	Change current directory
CD·..	*02 20 2E 2E 0D*	Move up one directory level
RD·*file*	*04 20 file 0D*	Reads a whole file
DLD·*file*	*05 20 file 0D*	Delete subdirectory from current directory
MKD·*file*	*06 20 file 0D*	Make a new subdirectory in the current directory
MKD·*file·datetime*	*06 20 file 20 datetime 0D*	Make a new subdirectory in the current directory Also specify a file date and time
DLF·*file*	*07 20 file 0D*	Delete a file
WRF·*dword* data	*08 20 dword 0D data*	Write the number of bytes specified in the 1st parameter to the currently open file
OPW·*file*	*09 20 file 0D*	Open a file for writing or create a new file
OPW·*file·datetime*	*09 20 file 20 datetime 0D*	Open a file for writing or create a new file Also specify a file date and time
CLF·*file*	*0A 20 file 0D*	Close the currently open file
RDF·*dword*	*0B 20 dword 0D*	Read the number of bytes specified in the 1st parameter from the currently open file
REN·*file·file*	*0C 20 file 20 file 0D*	Rename a file or directory
OPR·*file*	*0E 20 file 0D*	Open a file for reading
OPR·*file·date*	*0E 20 file 20 date 0D*	Open a file for reading Also specify a file access date
SEK·*dword¿*	*28 20 dword 0D*	Seek to the byte position specified by the 1st parameter in the currently open file
FS	*12 0D*	Returns the free space available on disk if less than 4GB is free
FSE	*93 0D*	Returns the free space available on disk
IDD	*0F 0D*	Display information about the disk if disk is less than 4GB
IDDE	*94 0D*	Display information about the disk
IDD	*0F 0D*	Display information about the disk
DSN	*2D 0D*	Display disk serial number
DVL	*2E 0D*	Display disk volume label

DIRT·*file*	*2F 20 file 0D*	List specified file and date and time of create, modify and file access

17.7 Schaltbild für Druckeransteuerung



17.8 Elektrische Bauteile

17.8.1 Widerstand

Ein Widerstand dient in den meisten Fällen zur Strombegrenzung. Legt man an einen Widerstand eine Spannung an, fließt ein Strom der umgekehrt proportional zum Widerstandswert ist. Es gilt das Ohmsche Gesetz:

$$I = \frac{U}{R}$$

17.8.2 Kondensator

Ein Kondensators wird als Energiespeicher und zur Trennung von Gleich- und Wechselstromkreisen verwendet.
Bei der Energiespeicherung werden positive Ladungsträger auf die eine, negative Ladungsträger auf die andere Platte gebracht. Diese Ladungen können zu einem späteren Zeitpunkt wieder aus dem Kondensator herausfließen. In diesem Augenblick arbeitet er als Spannungsquelle.
(siehe dazu: Elektronikvorlesung: Gleichrichtung)

In analogen elektronischen Schaltung wird das Nutzsignal einem Gleichstromsignal überlagert um dann verstärkt werden zu können. Für das Gleichstromsignal bildet der Kondensator einen gegen unendlich strebenden Blindwiderstand während für das Wechselstromsignal der Blindwiderstand endlich ist.

$$X_C = -j\frac{1}{\omega C}$$

(siehe dazu: Elektronikvorlesung: Transistorverstärker).

17.8.3 Spule

Die Spule wird zum Einen als Speicherelement für magnetische Energie verwendet. Dabei spielt die Induktion eine wesentliche Rolle. Diese bewirkt, dass beim Einschalten der Strom durch die Spule nicht schlagartig von 0% auf 100% steigen kann. Beim Abschalten entsteht jedoch eine sehr hohe Induktionsspannung. Gleichzeitig dreht sich die Polarität der Spannung um.
(siehe dazu: Elektronikvorlesung: Spule, Freilaufdiode)

Neben der Funktion als Energiespeicher besitzt die Spule auch einen Blindwiderstand. Damit ist auch sie ein Bauteil, dessen Widerstand sich mit der Frequenz ändert.

Zusammen mit einem Kondensator entsteht ein Parallel- oder Reihenschwingkreis, der bei der Resonanzfrequenz einen sehr großen bzw. sehr kleinen Widerstand bildet.

17.9 Elektronische Bauteile

17.9.1 Diode

Eine Diode besteht aus einem n- und einem p-dotierten Halbleitermaterial. An der Schnittstelle dieser beiden Schichten entsteht durch Rekombination eine Sperrschicht. Die Breite dieser Schicht ist materialabhängig. Schließt man eine Spannungsquelle so an die Diode an, dass der positive Pol der Quelle am n-dotierten Material liegt, dann wird die Sperrschicht breiter. Der positive Pol zieht die negativen Ladungen aus dem n-Material an, genauso wie der negative Pol die positiven Ladungsträger aus dem p-Material "heraus" zieht. Die Diode sperrt noch immer.

Dreht man die Spannungsquelle jedoch um, werden die Ladungsträger von beiden Seiten in die Sperrschicht gedrückt. Ist diese äußere Spannung groß genug, verschwindet die Sperrschicht komplett. Die Diode leitet jetzt.

Neben den einfachen Dioden für die Gleichrichtung, gibt es auch Spezialdioden wie z.B. LED, Kapazitätsdiode, Zenerdiode etc.

17.9.2 Transistor (bipolar)

Ein bipolarer Transistor besteht entweder aus npn- oder pnp-dotiertem Halbleitermaterial. Zwischen den zei Übergängen bildet sich durch Rekombination jeweils eine Sperrschicht (Abbildung 90). In diesem Zustand sperrt der Transistor, weil beide Sperrschichten keinen Strom fließen lassen.

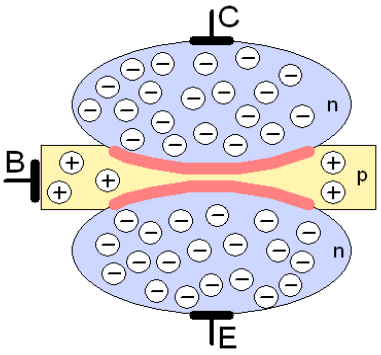


Abbildung 90: Prinzipieller Aufbau eines npn-Transistors

Sobald man die Elektroden Kollektor, Basis und Emitter entsprechend mit Spannungen versorgt, wird die unterer Sperrschicht dünner und bei genügend hoher Spannung zwischen Basis und Emitter auch durchlässiger oder wird komplett abgebaut. Dann fließen mehr oder weniger Elektronen vom Emitter zu Basis. Da die Basis sehr dünn ist und zwischen Kollektor und Emitter ein starkes elektrischen Feld herrscht, wird der größte Teil der Elektronen in den Kollektor "abgesaugt". Mit der Spannung U_{BE} lässt sich auf diese Weise der Kollektorstrom beeinflussen. Der von U_{BE} erzeugte Strom ist um ein vielfaches kleiner als der Kollektorstrom. Aus diesem Grund liegt eine Stromverstärkung vor.

$$B = \frac{I_C}{I_B}$$

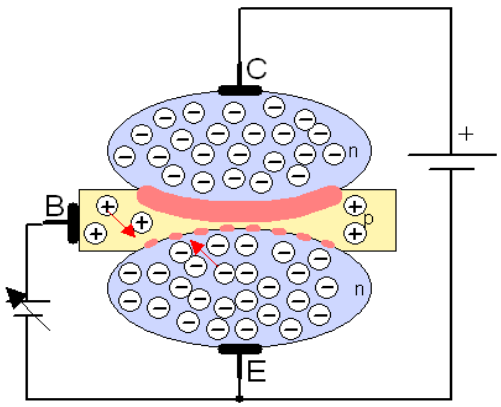


Abbildung 91: Die Spannung an der Basis steuert den Kollektorstrom

17.9.3 Transistor (MOS-FET)

Ein Feldeffekt-Transistor besitzt die Elektroden Drain, Gate und Source und funktioniert auf eine ganz andere Art und Weise als der bipolare Transistor. Beim FET wird mit Hilfe eines elektrischen Feldes die Leitfähigkeit des Halb-

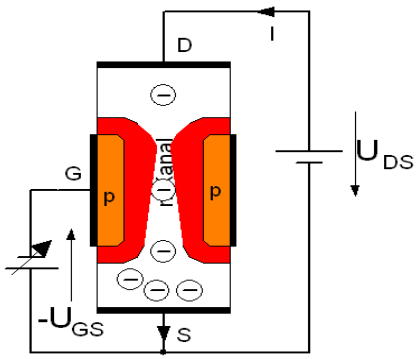


Abbildung 92: Prinzipaufbau eines FETs

leiters beeinflusst. Man spricht von einem "abschnüren" des Kanals. In Abbildung 92 ist das Prinzip eines FETs dargestellt. Je negativer die Spannung am Gate (bezogen auf Source) ist, desto tiefer dringt die Sperrschicht (roter Bereich) in den Kristall ein und desto kleiner wird der Durchlasskanal.

Ein MOS-FET ist in Abbildung 93 dargestellt. Hier ist das Gate durch eine zusätzliche Oxidschicht vom Halbleiter isoliert. Je positiver das Gate gegenüber

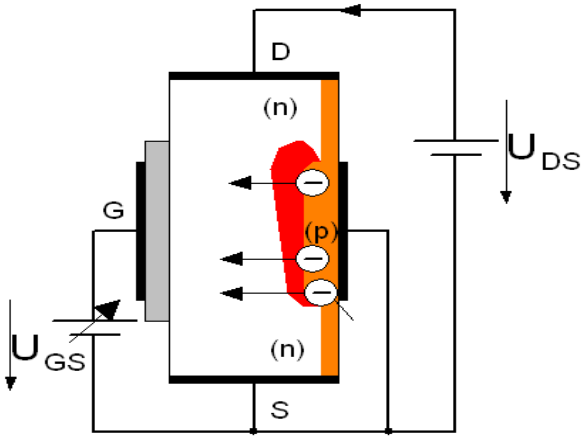


Abbildung 93: Prinzipaufbau eines MOS-FETs

der Source wird, desto mehr negative Ladungsträger sammeln sich zwischen dem Gate und der Gegenelektrode. Auf diese Weise wird die Sperrschicht größer und der leitende Kanal dünner.

17.9.4 Logikgatter

Aus Dioden und / oder Transistoren lassen sich logische Gatter aufbauen. Die einfachsten dieser Gatter sind: UND, ODER und NOT. Aus diesen drei Grundgattern lassen sich weitere einfache Gatter wie z.B. XOR realisieren. Da dieses Gatter in der Digitaltechnik eingesetzt werden, kennen die verwendeten Transistoren nur zwei Zustände: ein- oder ausgeschaltet. Dabei gelten sogenannte Schaltschwellen damit die Logik immer einwandfrei arbeitet. So kann z.B. definiert sein, dass alle Spannungen kleiner 0,7V als logisch Low und alles über 3,7V als logisch High gelten. Damit lassen sich dann logische Verknüpfungen auch mit einfachen Dioden realisieren (Abbildung 94) .

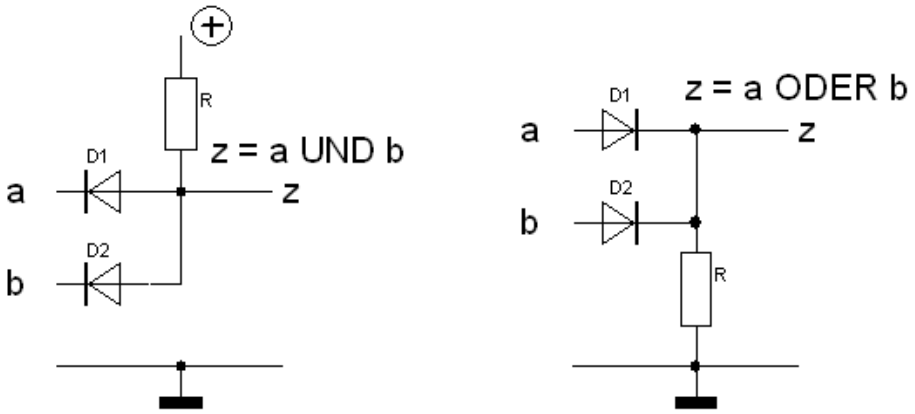


Abbildung 94: UND- und ODER-Gatter aus Dioden aufgebaut

Abbildung 95 zeigt die wichtigsten Gatter.

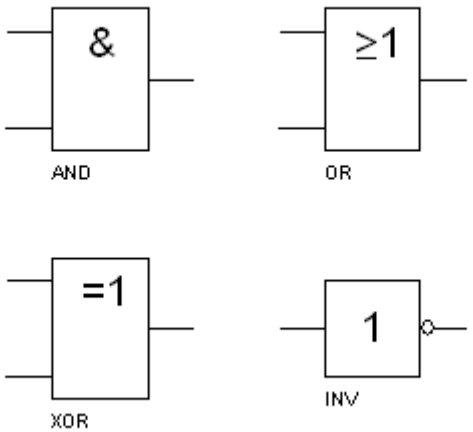


Abbildung 95: Die wichtigsten Gatter

17.9.5 Flip-Flops

Flip-Flops sind bistabile Schaltungen. Sie werden auf einen Zustand gesetzt und bleiben dort solange bis von außen ein Zustandswechsel erzwungen wird. Das einfachste FF ist das RS-FF (Set / Reset). Die Schaltung in Abbildung 96 zeigt

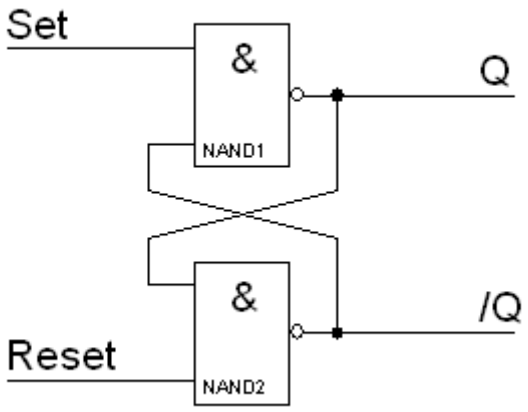


Abbildung 96: Das RS-Flip-Flop

ein solches RS-FF aus NAND-Gattern aufgebaut.
Funktionsweise: Die Eingänge *Set* und *Reset* seinen High[18]. Der Ausgang Q sei auf Low, der Ausgang /Q (≙ $\overline{Q}$) muss dann notwendigerweise auf High sein. Gibt man einen kurzen Low-Impuls auf den Set-Eingang dann ergibt sich am Ausgang Q ein Low. Denn (Set = 0) ∧ (/Q = 1) => Q = 1, /Q = 0.

S	**R**		**Q**	**/Q**
H	H		Q	/Q
L	H		H	L
H	L		L	H
L	L		x	x

x = nicht erlaubt.

In der Rechnertechnik ist das D-FF das wichtigste FF. Auf ihm bauen alle Speicherelemente auf. Es ist ein taktgesteuertes FF. Nur wenn der Takt aktiv ist, wird die Information am D-Eingang übernommen und erscheint am Q-Ausgang. In Abbildung 97 ist ein solches D-FF dargestellt. Es arbeitet hier als sogenanntes Transparent-Latch, d.h. solange der Takt aktiv (High) ist, werden die Daten vom D_Eingang übernommen. Gespeichert wird, wenn der Takt auf Low liegt. In der Rechnertechnik werden vor allem flankengesteuerte D-FF verwen-

18 High-Pegel ist Ruhepegel, Low ist der aktive Pegel (low aktive Logik)

det. Bei diesen wird nur beim Eintreffen einer bestimmte Flanke die Information vom D-Eingang übernommen.

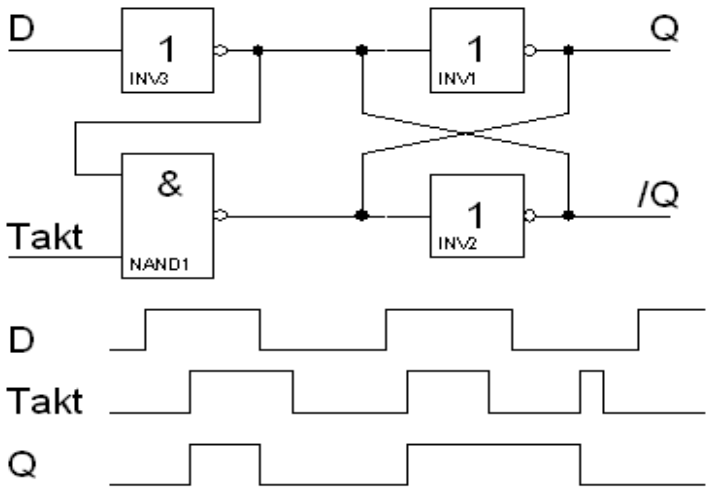


Abbildung 97: Ein pegelgesteuertes D-FF

17.9.6 Speicher

Statische Speicher bestehen im Prinzip aus Flanken gesteuerten D-FFs. Pro Da-

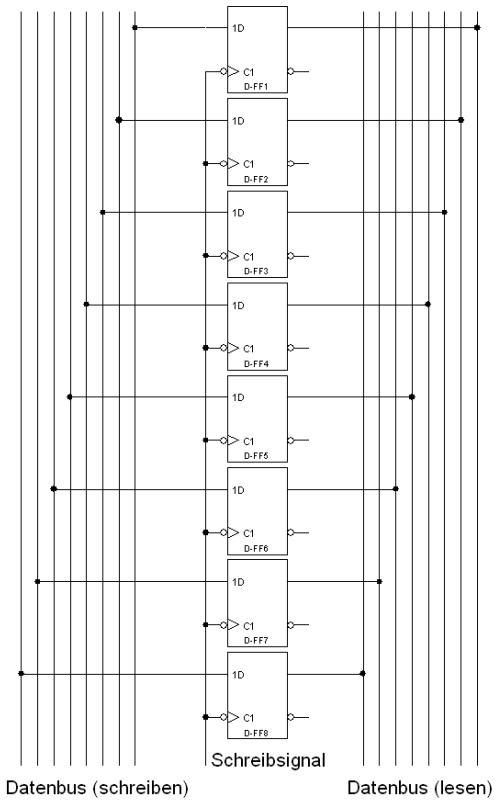


Abbildung 98: 8-Bit Speicheradresse

tenleitung wird ein FF benötigt. Somit sind bei einer Busbreite von 8 Bit pro Speicherstelle 8 FF notwendig. Abbildung 98 zeigt solch eine Speicheradresse für 8-Bit Daten. Sind in einem Speicherbaustein z.B. 2048 Speicheradressen ansprechbar, dann sind insgesamt 16384 D-FFs darin verbaut.

17.9.7 Zähler

Zähler werden i.d.R. aus JK-FF realisiert. In Abbildung 99 ist ein 8-Bit Zähler dargestellt. Er zählt mit jeder fallenden Flanke am Clk-Eingang. Als Besonder-

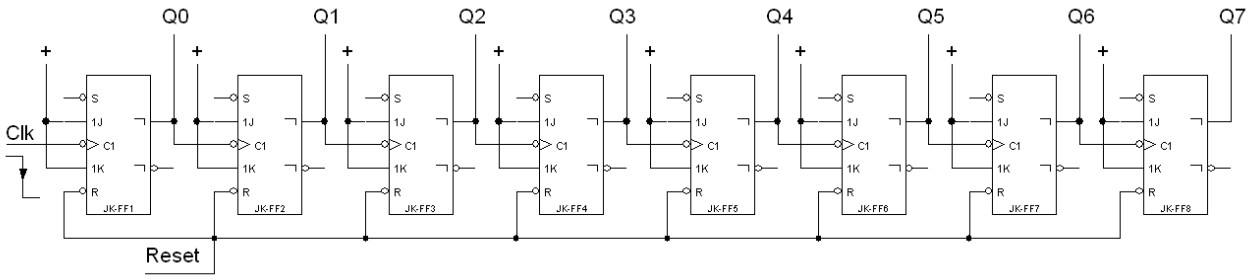


Abbildung 99: 8-Bit Zähler mit Reset-Funktion

heit gibt es hier noch einen Reset. Dieser kann z.B. durch einen Befehl (CLRF TMR0) aktiviert werden um so den Zähler auf 0 zu setzen.

17.9.8 Halb- / Volladdierer

Ein Halbaddierer addiert zwei Eingangsvariablen a und b. Als Ergebnis erscheint die Summe s sowie ein Übertrag c'.

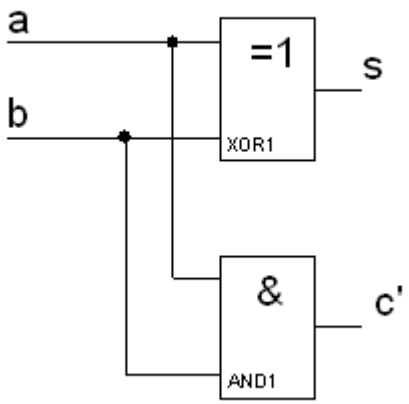


Abbildung 100: Halbaddierer

In der ALU arbeitet u.a. ein Volladdierer um aus zwei Eingangswerten und ggf. einem Übertrag die Summe zu berechnen. Eine Subtraktion wird i.d.R. durch die Addition des 2-er Komplements durchgeführt. Abbildung 101 zeigt einen 1-

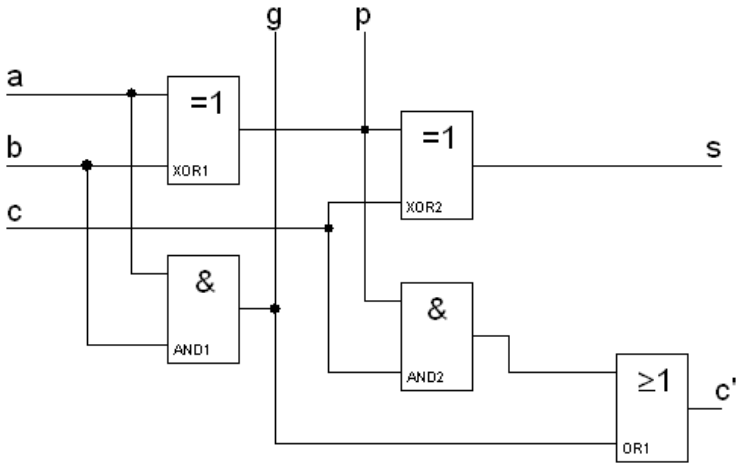


Abbildung 101: 1-Bit Volladdierer

Bit Volladdierer. Er addiert die beiden Eingangsvariablen a und b unter Berücksichtigung des Carryflags c. Das Ergebnis steht bei s zur Verfügung. Einen eventuellen Übertrag steht an c' an. In der Tabelle 5 wird der Zusammenhang deutlich.

a	**b**	**c**		**s**	**c'**
0	0	0		0	0
0	1	0		1	0
1	0	0		1	0
1	1	0		0	1
0	0	1		1	0
0	1	1		0	1
1	0	1		0	1
1	1	1		1	1

Tabelle 5: Wahrheitstabelle eines 1-Bit Volladdierers

Ein 4-Bit Addier- Subtrahierwerk ist in Abbildung 102 dargestellt.

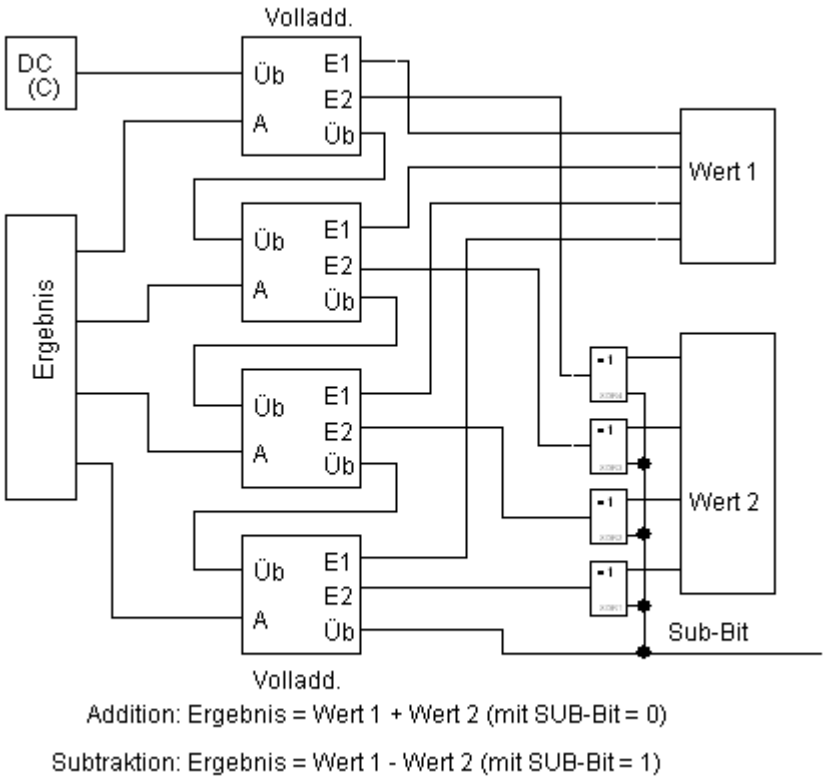


Abbildung 102: 4-Bit Addier- / Subtrahierwerk

17.10 Hexadezimal

Unser Zahlensystem ist ein Stellen bewertetes System. Das bedeutet, wenn der Zeichenvorrat zur Darstellung von Zahlen erschöpft ist, wird einfach eine neue Stelle links von der alten hinzugefügt. Die Wertigkeit der einzelnen Stellen wird im Dezimalsystem mit 10^x berechnet, wobei x die Position der Stelle beginnend bei 0 ist.
Beispiel:
1234 $= 1 * 10^3 + 2 * 10^2 + 3 * 10^1 + 4 * 10^0$

Im Dezimalsystem stehen uns für die 10 einzelnen Werte auch zehn eigene Symbole zur Verfügung. Das Dezimalsystem baut somit auf der Basis 10 auf.

Es existieren neben dem Dezimalsystem viele weiter. Beispielsweise das Binärsystem[19], welches in den digitalen Rechnern verwendet wird. Dieses baut auf der Basis 2 auf, d.h. die Stellenwertigkeit berechnet sich mit 2^x.
Beispiel:
1011 $= 1 * 2^3 + 0 * 2^2 + 1 * 2^1 + 1 * 2^0 = 11_d$
Zur besseren Unterscheidung schreibt man 0x1011 oder 1011_b.

Die Basis 16 gilt für das Hexadezimalsystem. Hier sind pro Stelle 16 einzelne Symbole vorhanden. Diese sind 0, 1, 2, ... 8, 9 und A, B, C, D, E, F. Das A repräsentiert dabei den Wert 10, das B den Wert 11 und so fort bis zu F das für die 15 steht.
Beispiel:
2A5F $= 2 * 16^3 + 10 * 16^2 + 5 * 10^1 + 15 * 10^0 = 10847$
Um Hexadezimalzahlen zu kennzeichnen, wird entweder das Prefix 0x oder der Suffix H verwendet.
0x2A5F 2A5FH

Um vom Hexadezimalsystem zum Binärsystem zu gelangen kann man einfach die entsprechende Hexzahl in deren Binärwert umsetzen. Für das obige Beispiel wäre der Binärwert: $0010\ 1010\ 0101\ 1111_b$.

19 Aber auch ein Dreier- oder Vierersystem ist möglich.

18 Fragen zur Selbstkontrolle

1. Skizzieren Sie die 3 Grundtypen eines Automaten.
2. Zeigen Sie dabei die wesentlichen Eigenschaften und Unterschiede auf.
3. Aus was besteht ein Zustandsdiagramm?
4. Stellen Sie das Zustandsdiagramm einer primitiven Waschmaschine auf. Beschreiben Sie die Abläufe zuerst stichwortartig.
5. Was bedeutet das EVA-Prinzip?
6. Was bedeutet SISD und SIMD? Was ist deren Funktionsweise?
7. Wie ist der Informationsfluss bei einem SISD-Rechner?
8. Was ist eine 1-Adressmaschine? Skizzieren Sie diese.
9. Was ist eine 3-Adressmaschine? Skizzieren Sie diese.
10. Wieso wird der PIC-Mikrocontroller als 1 1/2-Adressmaschine bezeichnet?
11. Was versteht man unter Akkumulator-Architektur?
12. Skizzieren Sie den Registersatz des 8086-Prozessors.
13. Was versteht man unter a) direkter, b) Register direkte, c) indirekter, d) Register indirekte, e) implizite, f) indizierte und f) relative Adressierung?
14. Was ist ein Post-Inkrement? Was ein Pre-Inkrement?
15. Was bedeutet Post-Dekrement? Was ein Pre-Dekrement?
16. Was ist ein Displacement?
17. Was bedeutet Little Endian bzw. Big Endian?
18. Skizzieren Sie den Registersatz des PIC-16F84 Mikrocontrollers.
19. Was ist eine Speicher-Speicher-Architektur?
20. Was ist eine Stack-Architektur?
21. Was bedeutet UPN? Wo wird diese eingesetzt (2 Beispiele)?
22. Was ist der Unterschied zwischen einem Mikroprozessor und einem Mikrocontroller?
23. Skizzieren Sie den Aufbau eines Von-Neumann-Rechners.
24. Welche Aufgaben hat das Steuerwerk?
25. Welche Funktionsgruppen stecken im Rechenwerk?
26. Was zählt alles zum Speicher?
27. Wie werden Informationen mit der Außenwelt ausgetauscht?
28. Erklären Sie die verschiedenen Typen von nur-Lesespeicher.
29. Welche Arten von RAM sind üblicherweise in einem Mikrocontroller vorhanden?
30. Wie sind RAM und ROM intern typischerweise organisiert?
31. Bei einem Flash-Speicher mit blockweiser Programmierung soll ein einzelnes Byte umprogrammiert werden. Wie muss man dies

bewerkstelligen?

32. Nennen Sie die Bussysteme bei einem Von-Neumann-Rechner.
33. Um einen Adressraum von 64k zu bekommen, benötigt man wie viele Adressleitungen?
34. Bei manchen Mikroprozessoren gibt es ein MREQ und eine IORQ. Was hat es damit auf sich ?
35. Welche Aufgabe hat ein Adressdekoder?
36. Stellen Sie eine Wahrheitstabelle für einen Adressdekoder auf. Benutzen Sie dabei mind. vier Speicherbausteine und drei I/O-Bausteine. Die Einteilung obliegt Ihnen.
37. Was ist eine unvollständige Dekodierung und wie entsteht sie?
38. Zeigen Sie am PIC wo sich unvollständige Adressierungen befinden. Wodurch entstehen diese? Sind diese beim PIC sinnvoll?
39. Was unterscheidet die Harvard-Architektur von der Von-Neumann?
40. Welche Vorteile ergeben sich dabei bei dem Aufbau der einzelnen Befehlen?
41. Wie lassen sich Konstantentabellen bei einem a) Harvard-Rechner und b) Von-Neumann-Rechner im ROM ablegen? Wie erfolgt der Zugriff auf diese Tabelle?
42. Nennen Sie die drei Phasen bei der Befehlsausführung. Was passiert in den einzelnen Phasen?
43. Dürfen sich Phasen überlappen? Begründung!
44. Warum muss ein Prozessor beim Einschalten in den Reset-Zustand versetzt werden und was passiert in diesem Zustand?
45. Nennen Sie die wichtigsten unterschiedlichen Befehlstypen.
46. Wie wird das 1-er Komplement und wie das 2-er Komplement gebildet?
47. Wann werden Rotations- und Schiebebefehle als arithmetische und wann als logische Befehle bezeichnet?
48. Was unterscheidet den CALL-Befehl vom GOTO-Befehl?
49. Wie wird der Programmzähler beim PIC modifiziert, wenn ein GOTO-Befehl ausgeführt wird?
50. Auf das PCL-Register im PIC wird ein Wert addiert. Das Ergebnis dieser Operation ist 0x20. Was passiert dadurch?
51. Wie arbeitet der RETLW-Befehl beim PIC?
52. In einem PIC-Programm werden 9 Unterprogramme verschachtelt aufgerufen. Was passiert?
53. Was ist ein Interrupt?
54. Was ist ein Hardware-, was ein Softwareinterrupt?
55. Nennen Sie drei unterschiedliche Interruptmodi.
56. Wie funktioniert die Interruptlogik beim PIC 16F84?

57. Was muss passieren, dass der PIC 16F84 einen Timerinterrupt abarbeitet?
58. In einem PIC 16F84 sind zwei Interruptquellen aktiviert. Was muss in der ISR unbedingt durchgeführt werden?
59. Was ist ein Segment, was eine Page und was eine Bank?
60. Wie funktioniert das Zeroflag?
61. Was unterscheidet das Carryflag im PIC von dem Carryflag in einem anderen Prozessor?
62. Erklären Sie die Arbeitsweise des Carry- und des Overflowflags anhand des Zahlenkreises.
63. Beschreiben Sie die Funktionsweise der Pipeline im PIC 16F84
64. Was bedeutet U-Pipeline und V-Pipeline?
65. Warum sollte eine Pipeline nicht zu groß sein?
66. Was bedeutet Branch Prediction? Wie arbeitet diese?
67. Was ist ein Cache? Erklären Sie, welchen Vorteile es macht, wenn der Cache für Programm und Daten getrennt wird.
68. Was bedeutet Level-1 Cache?
69. Was bedeutet Parität beim Speicher?
70. Was ist DMA? Wo wird DMA eingesetzt.
71. Was bedeutet FPU?
72. Was sind Deskriptoren? Für was werden sie benötigt?
73. Warum kann ein Programm, welches im Protected Modus läuft, nicht feststellen, an welche Speicheradressen es geladen wurde?
74. Was bedeutet Virtuelle 8086 Modus (V86)?
75. Mit welchen Funktionsblöcken interagiert die Northbridge?
76. Mit welchen Funktionsblöcken interagiert die Southbridge?
77. Was und wo ist der Frontsidebus?
78. Welche Aufgabe haben A/D- und D/A-Wandler?
79. Nennen Sie vier verschiedene Wandlungsverfahren bei den A/D-Wandlern.
80. Beschreiben Sie die Arbeitsweise von vier verschiedenen A/D-Wandlern.
81. Was bedeutet: Werte kontinuierlich, Werte diskret, Zeit kontinuierlich und Zeit diskret?
82. Welche Fehler sind beim A/D-Wandler typisch?
83. Wie lässt sich das Werte diskrete Signal eines D/A-Wandlers in ein Werte kontinuierliches Signal umwandeln?
84. Wie ist ein R2R-Netzwerk aufgebaut und wie funktioniert es?
85. Was ist PWM?
86. Warum ist es oft sinnvoll, einem PWM-Ausgang einen Integrierer nach zuschalten?
87. Wozu benötigt man eine Sample & Hold Schaltung?

88. Nennen Sie Vor- und Nachteile von Parallelschnittstellen.
89. Skizzieren Sie das Zeitdiagramm eines 3-Leiter-Handshakes.
90. Was ist eine synchrone bzw. eine asynchrone serielle Schnittstelle?
91. Was ist eine Taktrückgewinnung und wie funktioniert diese?
92. Skizzieren Sie das Zeitdiagramme eines seriellen Datenstroms bestehend aus einem Start- und 2 Stoppbits, 8 Datenbits und einem Paritätsbit. Das Datum ist 0xAA, die Parität ungerade.
93. Wie sind die Spannungspegel der RS232-Schnittstelle definiert?
94. Was unterscheidet die RS232- von der RS422-Schnittstelle?
95. Was sind differentielle Signale? Welchen Vorteil haben diese?
96. Beschreiben Sie die SPI-Schnittstelle.
97. Skizzieren Sie den Anschluss dreier SPI-Slaves an einen Mikrocontroller.
98. Was bedeutet Party-Line, was Daisy-Chain?
99. Was bedeutet SCSI? Wo ist dessen Einsatzgebiet (gewesen)?
100. Was ist GPIB (IEEE-488)?
101. Beschreiben Sie dessen Arbeitsweise.
102. Wo liegt der Unterschied zwischen RS422 und RS485?
103. Was ist der I^2C-Bus?
104. Welche Busarbitrierung wird vom I^2C-Bus verwendet?
105. Wer bedient bei I^2C-Bus die SCL-Leitung?
106. Wer sendet das ACK-Bit bei I^2C-Bus?
107. Wie ist beim I^2C-Bus die Start-Condition definiert?
108. Wie ist beim I^2C-Bus die Stopp-Condition definiert?
109. Wie wird bei I^2C-Bus der Slave-Baustein adressiert?
110. Was bedeutet Nyquist-Kriterium?
111. Wozu benötigt man eine Leitungsterminierung?
112. Was bedeutet TRI-STATE?
113. Skizzieren Sie ein UND- und ein ODER-Gatter mit Hilfe von Dioden.
114. Welchen Dezimalwert repräsentiert die Zahl 0x1234 ?
115. Wie lautet die Binärzahl von 1234H ?

19 Literatur

Als weiterführende und vertiefende Literatur sind folgende Bücher empfehlenswert:

Autor	Titel	Verlag	ISBN
Hoffmann, Dirk W.	Grundlagen der technischen Informatik	Hanser	978-3-446-44867-4
Beierlein, Th. Hagenbruch, O.	Taschenbuch Mikroprozessortechnik	Fachbuchverlag Leipzig	3-446-21049-0
Hellmann, Roland	Rechnerarchitektur	Oldenburg Verlag	978-3-11-049665-9