Mikrocontroller für Einsteiger & Profis

Die exklusive Wissenssammlung

12 Fachbücher rund um das Thema Mikrocontroller
mit zahlreichen Quellcode- und Programmierbeispielen

FRANZIS

Bibliografische Information der Deutschen Bibliothek

Die Deutsche Bibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte Daten sind im Internet über http://dnb.ddb.de abrufbar.

Autoren: Herbert Bernstein, Michael Hofmann, Felix Irmscher, Dieter Kohtz, Anne König, Manfred König, Fredi Krüger, Irmtraut Meister, Lukas Salzburger, Heinz Schmid, Frank Sichla, Ulli Sommer, Dr. Günter Spanner

ISBN 978-3-645-39016-3

INHALTSÜBERSICHT

Ulli Sommer

Mikrocontroller programmieren mit Bascom Basic

Messen, Steuern, Regeln und Robotertechnik mit den AVR-Controllern

Vorwort

Mikrocontroller zu programmieren wird, wie man in den verschiedensten Foren und Fachzeitschriften beobachten kann, immer populärer. Das liegt daran, dass Mikrocontroller und zusätzliche Peripheriebausteine immer günstiger angeboten werden und an Schulen zunehmend in Mikrocontroller und Computertechnik unterrichtet wird. Schaltungen, die man früher mit mehreren ICs aufbauen musste, können nun in einem einzigen Mikrocontroller untergebracht werden. Das spart Zeit, Geld und Platz auf der Leiterplatte. Dieses Buch baut auf dem beliebten Basic-Compiler BASCOM-AVR mit integrierter Entwicklungsumgebung auf. Er ist für fast alle 8-Bit-AVR- und XMega-Mikrocontroller der Firma Atmel geeignet. BASCOM erfreut sich nicht nur bei Einsteigern, sondern auch bei Entwicklungsprofis, immer größerer Beliebtheit und stellt inzwischen schon fast einen Basic-Standard bei AVRs dar.

Viele Problemstellungen, die früher zeitaufwendig in Assembler oder C gelöst werden mussten, können durch diesen modernen Compiler blitzschnell mit wenigen Befehlen erledigt werden. Beispielsweise genügt ein einziger Befehl, um aus einem I/O-Port eine RS232-Schnittstelle, einen I²C-Bus oder einen Servoanschluss zu machen. Solche Dinge erfordern in anderen Programmiersprachen oft einen enormen Aufwand.

BASCOM erzeugt optimierten Maschinen-Code. Es werden alle AVR-RISC-Controller mit internem RAM der Serien AT90S, ATmega und ATTiny unterstützt. Mit einigen Einschränkungen sind jetzt auch ATTiny-Controller ohne SRAM mit BASCOM-AVR programmierbar. Dazu steht die $TINY-Funktion zur Verfügung.

Aus diesen Gründen ist der BASCOM Basic-Compiler ideal für den Einstieg in die Mikrocontroller-Programmierung geeignet. Er ist trotzdem sehr leistungsfähig und ermöglicht auch optimierte komplexe Software-Entwicklungen mit Profianforderungen. Ein weiterer großer Vorteil ist, dass diese Entwicklungsumgebung in hohem Tempo weiterentwickelt wird und die Updates kostenlos sind. So war BASCOM auch eine der ersten AVR-Entwicklungsumgebungen, die unter Vista und Windows 7 liefen.

Ich wünsche Ihnen viel Spaß beim Lesen und Experimentieren mit diesem Buch!

Waidhaus, Juli 2011

Ulli Sommer

Inhaltsverzeichnis

2 Mikrocontroller-Grundlagen

Bevor man sich näher mit BASCOM beschäftigt, ist es sinnvoll, einen allgemeinen Überblick über Mikrocontroller zu gewinnen. Mikrocontroller werden vor allem im Bereich der Automatisierungs- sowie der Mess-, Steuer- und Regeltechnik eingesetzt. Der Vorteil eines Mikrocontroller-Systems ist es, auf kleinstem Raum energie- und kosteneffizient physikalische Größen messen und interpretieren zu können, um darauf aufbauend Entscheidungen zu treffen und Aktionen durchzuführen.

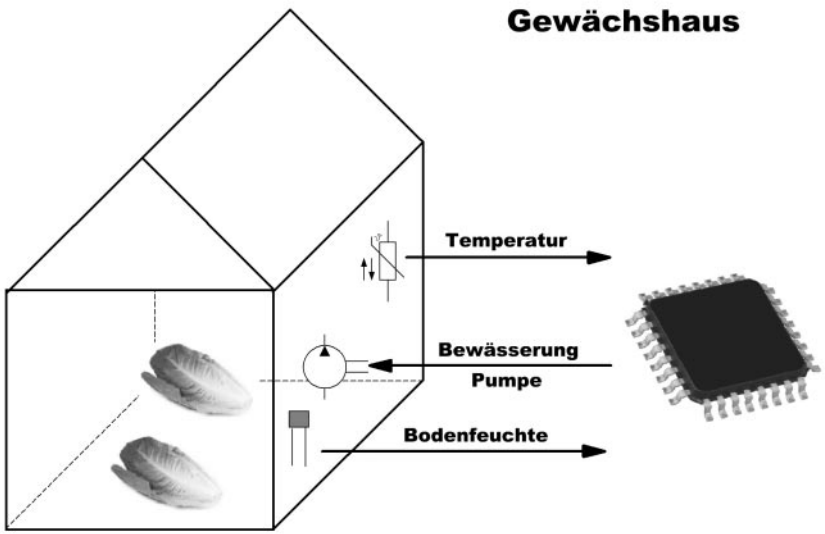

Bild 2.1: Beispiel einer Ein- und Ausgabeverarbeitung anhand eines Gewächshauses.

Das Spektrum möglicher Anwendungen von Mikrocontrollern reicht vom privaten Bereich, z. B. von der Steuerung eines Gewächshauses oder der Hausbeleuchtung, bis hin zum industriellen Bereich, in dem komplette Anlagen mit Mikrocontroller-Systemen gesteuert, gewartet und betrieben werden können. Das Bild zeigt eine typische Datenverarbeitung zur Steuerung einer Bewässerungsanlage eines Gewächshauses. Der Controller nimmt dabei über Sensoren die Messwerte der Umgebungstemperatur und Bodenfeuchte auf. Die Messwerte

werden durch eine digitale Logik im Mikrocontroller (kurz *µC* genannt) interpretiert. Daraufhin wird die Pumpe für die Bewässerung entsprechend angesteuert.

2.1 Aufbau und Funktionsweise

Als vollwertiger Computer im Kleinstformat weist jeder Mikrocontroller, ähnlich einem PC, grundlegende Bausteine auf. Grundbausteine eines jeden Mikrocontrollers sind die CPU, der Arbeitsspeicher (RAM) sowie der Programmspeicher (Flash) und die Peripherie.

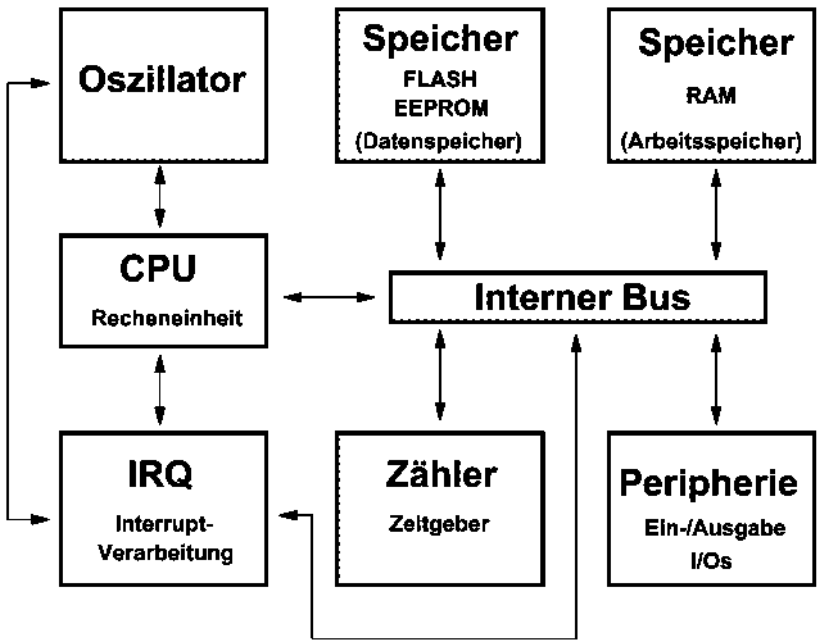

Bild 2.2: Prinzipieller Aufbau eines Mikrocontrollers.

2.2 Die CPU

Die wichtigste Funktionseinheit ist die zentrale Recheneinheit, die CPU (engl. Central Processing Unit). Sie kann als Gehirn des Mikrocontrollers verstanden werden. Dort werden Signale ausgewertet und Befehle und arithmetische Operationen abgearbeitet.

2.3 Arbeits- und Programmspeicher

Arbeits- und Programmspeicher werden in vielen Darstellungen in der Regel logisch getrennt. Das Benutzerprogramm wird dabei meist in einem nichtflüchtigen Flash-Speicher, dem Programmspeicher, abgelegt. Je nach Controllersystem kann man auf (implementieren) Programmspeicher von mehreren Kilo- bis Megabyte zurückgreifen. Bei einigen Systemen ist es darüber hinaus möglich, den Programmspeicher durch externe Flash-Komponenten aufzustocken.

Der Arbeitsspeicher (RAM) dient zur temporären Ablage von Rechen-, Mess- und Steuergrößen. Ziel ist, möglichst schnell auf eine begrenzte Anzahl von Daten zugreifen zu können. Dieser RAM (engl. Random Access Memory) ist in der Regel deutlich kleiner als der Flash-Speicher, dafür aber um ein Vielfaches schneller. Die Werte des RAM werden zur Laufzeit erzeugt und sind, im Gegensatz zum Flash-Speicher, flüchtig. Das heißt, dass nach einem Neustart des Controllers im RAM keine Werte gespeichert sind.

2.4 Peripherie

Peripherie bezeichnet jene Komponenten eines Mikrocontrollers, die nicht durch die CPU und Speicherbausteine abgedeckt werden. Es handelt sich insbesondere um solche Komponenten, die eine Schnittstelle zur Außenwelt darstellen. Digitale Ein- und Ausgänge (kurz: I/O für Input/Output) werden zu den Peripheriebausteinen gezählt. Die meisten Mikrocontroller bieten eine Vielzahl von Ein- und Ausgängen mit verschiedenen Funktionen wie digitale, aber auch analoge Ein- und Ausgänge (ADC und DAC).

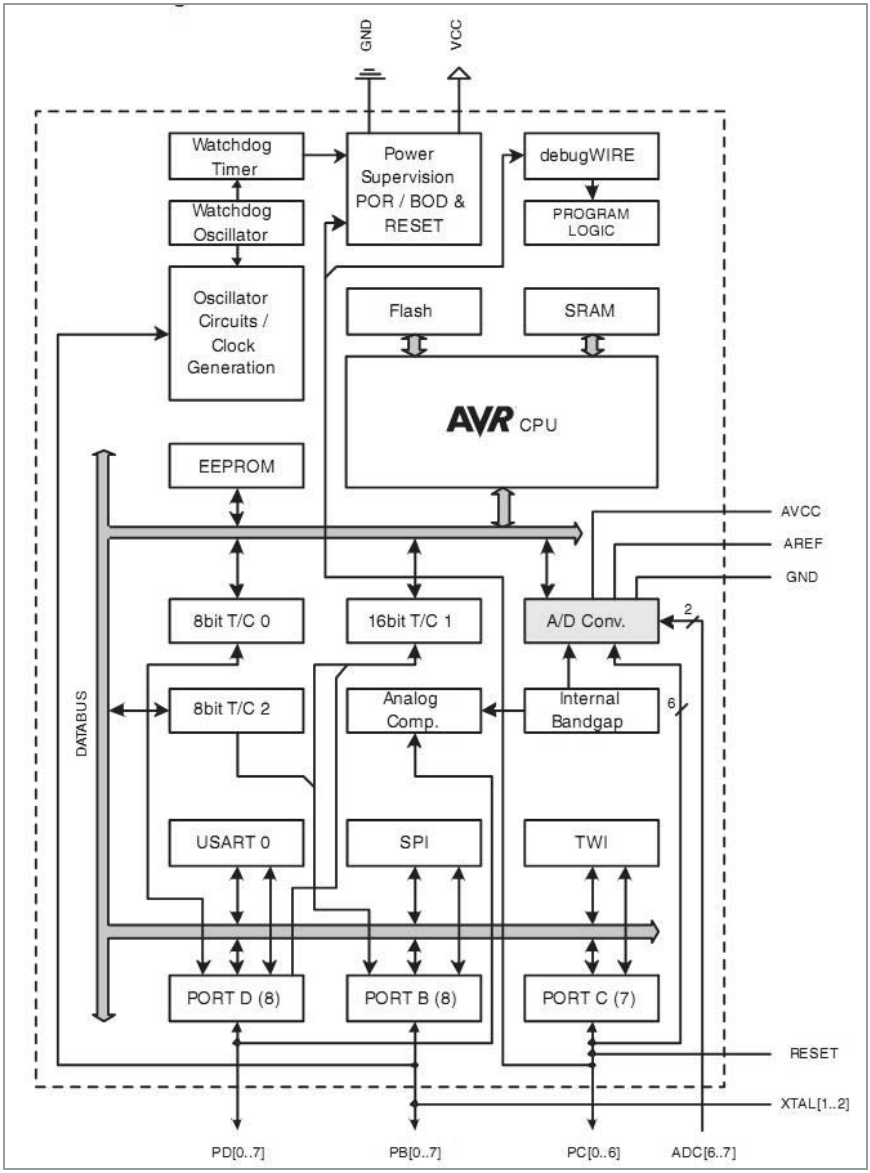

Bild 2.3: ATmega88-Blockschema: Hier sind die internen Strukturen des Controllers gut zu erkennen. (Quelle: Atmel Datenblatt)

3 Mikrocontroller-Programmierung im Allgemeinen

Mit der steigenden Integration von Halbleiterbauteilen wie Mikroprozessoren nahmen Mikrocontroller zunehmend Einzug in die Anwendungsgebiete der Mess-, Steuer- und Regelungstechnik. Aber auch im Hobbybereich wurden sie immer beliebter. Das liegt zum einen daran, dass heute komplexe, meist analoge Schaltungen durch einfachere digitale Mikrocontroller-Schaltungen ersetzt werden. Ein anderer ausschlaggebender Punkt ist heute das unschlagbare Preis-Leistungs-Verhältnis von Mikrocontrollern.

3.1 Was ist ein Programm?

Ein Programm ist die Beschreibung eines Informationsverarbeitungsprozesses. Im Lauf eines solchen Prozesses wird aus einer Menge von variablen oder konstanten Eingangswerten eine Menge von Ausgangswerten berechnet. Die Ausgangswerte sind entweder selbst Ziel der Informationsgewinnung oder dienen mittelbar zur Reaktion auf die Eingangswerte. Neben den eigentlichen Berechnungen kann ein Programm Anweisungen zum Zugriff auf die Hardware des Computers oder zur Steuerung des Programmflusses enthalten. Ein Programm besteht aus mehreren Zeilen sogenannten Quelltextes. Dabei enthält jede Zeile eine oder mehrere Rechen- oder Steueranweisungen. Außer diesen Anweisungen selbst bestimmt ihre Reihenfolge ganz wesentlich die eingangs beschriebene Informationsverarbeitung. Die Ausführung der den Anweisungen entsprechenden Operationen durch den Steuercomputer erfolgt sequenziell (nacheinander). Eine Folge von Programmanweisungen mit einem bestimmten Ziel nennt man auch Algorithmus.

3.2 Programmierung in Basic

Basic ist eine einfach zu erlernende Programmiersprache. Für BASCOM wurde ein Basic-Dialekt, angelehnt an QBASIC und Visual Basic, von Microsoft entwickelt. Dadurch ist es für einen ehemaligen QBASIC- oder Visual-Basic-Programmierer ein Leichtes, auch eine Hardware zu programmieren. Basic ist besonders bei Anfängern beliebt und wird im Bereich der Mikrocontroller-

Anwendungen, aber auch für professionelle Anwendungen eingesetzt. BASCOM erstellt aus dem Basic-Programm einen hochoptimierten Maschinen-Code, der anderen Programmiersprachen, etwa C, nicht nachsteht.

3.3 Konzept von Basic

Der hier betrachtete Basic-Dialekt ist modular aufgebaut. Dafür definiert man ein oder mehrere Unterprogramme, die verschiedenste Befehlsfolgen kapseln. Die Abarbeitung der Befehle erfolgt weitestgehend intuitiv. Das bedeutet, dass man den Controller durch einfache Befehle dazu anweist, bestimmte Aufgaben zu erfüllen. Es gibt insbesondere für Computer, die mit Windows oder Linux arbeiten, auch objektorientierte Basic-Dialekte. Sie sind weniger intuitiv zu bedienen als die hier gezeigte Programmiersprache.

3.4 Vor- und Nachteile von Basic

Ein entscheidender Vorteil von Basic ist ein schneller Einstieg in die Welt der Mikrocontroller. Der hier thematisierte Basic-Dialekt BASCOM ist so ausgelegt, dass er die Hardware optimal und effizient anspricht und der Entwickler trotzdem übersichtlich strukturierten Quellcode schreiben kann.

3.5 Programmierung in Assembler

BASCOM unterstützt neben Basic auch Assembler, sogenannten Inline-Assembler. Es können somit bestimmte Programmabschnitte, dort wo es auf jeden Prozessortakt ankommt oder bestimmte Register angesprochen werden sollen, in Assembler programmiert werden. Ein richtiger Assembler übersetzt ein Assemblerprogramm in der Regel 1:1 in Maschinencode, also in Befehle, die direkt vom Mikroprozessor abgearbeitet und ausgeführt werden können. Der Programmierer muss dabei über fundiertes Wissen über die Systemarchitektur des Mikroprozessors verfügen. Da es eine Fülle von Registern, Befehlen und Adressierungsarten gibt, fällt es insbesondere Anfängern schwer, sich in die Assemblerprogrammierung einzuarbeiten. Hinzu kommt, dass jede Mikroprozessorgeneration und jeder Typ über einen anderen Befehlssatz verfügt. Selbst jemand, der sich gut mit dem Intel-Assembler für PCs auskennt, benötigt eine lange Einarbeitungszeit, um beispielsweise auch für einen ATmega88 ein Assemblerprogramm entwickeln zu können.

Der große Vorteil von Assembler ist, dass man über nahezu jedes Bit und jede Operation frei verfügen kann. Damit können sehr leistungsstarke und dennoch schlanke Programme entwickelt werden, sofern man sich mit den Spezialbefehlen des Mikroprozessortyps auskennt. Manche Anwendungen lassen sich aus Grün-

den der Schnelligkeit nur in Assembler verwirklichen. Heute setzt man Assembler häufig nur in kleinen Programmstücken ein, wo ein Höchstmaß an Leistung gefordert wird. Der Aufruf des Programms erfolgt dann aus einer Hochsprache wie Basic oder C. Auch die Auswertung der Ergebnisse des Assembler-Programms kann von einer Hochsprache übernommen werden.

4 Übersicht über die Atmel-8-Bit-Mikrocontroller

4.1 AT90Sxxx

Die Basic-Line der Atmel AVR-Reihe beinhaltet die ersten AVRs, die produziert wurden und deren Bezeichnung mit AT90S beginnt. Alle Typen wurden mit der Zeit von den beiden Nachfolgereihen ersetzt: ATmega oder ATTiny.

Einige neue AVR-Controller tragen eine mit AT90 ohne »S« beginnende Bezeichnung, haben aber einen »moderneren« Kern. Z. B. sind die Typen AT90PWM2/3 und AT90CAN128 vom Funktionsumfang (interner RC, UART, CAN-Bus etc.) den ATmegas zuzuordnen.

4.2 ATmega

Die ATmega-Mikrocontroller sind ein Teil der AVR-Controllerfamilie. Zusammen mit den ATTiny lösen die ATmega die AT90S-Serie schrittweise ab. In den meisten Fällen gibt es weitgehend pin- und funktionskompatiblen Ersatz für angekündigte Controller (ATmega8 statt AT90S4433, ATmega8515 statt AT90S8515 usw.).

Atmel ATmega-AVRs werden mit aktiviertem internem Taktgeber ausgeliefert. Schließt man eine andere externe Taktquelle an (Quarz, Quarzoszillator o. Ä.), wird diese nicht automatisch genutzt. Zum Aktivieren müssen die Fuse-Bits des Controllers entsprechend eingestellt werden.

ATmegas mit integriertem JTAG-Interface (zzt. solche ab 16 kB Flash-Speicher und mehr als 28 Pins) werden ab Werk mit aktiviertem JTAG-Interface ausgeliefert. Dieses Interface belegt 4 Portpins (z. B. an PORTC beim ATmega16/32), die nicht für eigene Anwendungen genutzt werden können, solange das JTAG-Interface aktiviert ist. Das Interface lässt sich über ein Fuse-Bit (JTAGEN) dauerhaft und über ein Bit (JTD) in dem (oder einem der) MC-Kontrollregister (Datenblatt nach JTD durchsuchen) per Software zur Laufzeit an- und abschalten. Weiteres findet sich im Datenblatt des jeweiligen Controllers in den Abschnitten Memory-Programming (Fuse) und JTAG/ICE (JTD).

Beim ATmega128 ist ab Werk die Mega103-Kompatibilitäts-Fuse gesetzt. Um alle Erweiterungen des Mega128 gegenüber dem Mega103 nutzen zu können, muss diese deaktiviert werden. Die Fuse sorgt außerdem dafür, dass das SRAM in einem anderen Adressbereich liegt.

4.3 ATTiny

Die ATTiny stellen das untere Ende der neuen AVR-Linie von Atmel dar und waren zunächst durch das Fehlen von internem SRAM gekennzeichnet. Mittlerweile gibt es aber so bemerkenswerte Controller wie den ATTiny2313, deren Möglichkeiten und Funktionen den ATmegas in nichts nachstehen.

Ein weiterer Unterschied zu den ATmegas ist der fehlende Hardware-Multiplizierer. Jede Multiplikation muss also in Software ausgeführt werden. Eine Übersicht über die Verfügbarkeit verschiedener Befehle bietet die AVR-Assembler-Befehlsvergleichstabelle.

4.4 XMega

Die XMega-Controller-Reihe von Atmel basiert auf dem erfolgreichen 8-Bit-AVR-Kern, enthält aber einige interessante Neuerungen wie DMA, Crypto-Engine für AES und DES: Einfach Key und Daten in Register schreiben, der Rest läuft automatisch. Das Event-System: ähnlich Interrupts, aber für die Kommunikation zwischen Peripherieeinheiten ohne CPU-Beteiligung.

Die Speicherausstattung ist mit bis zu 256 KB Flash und 16 KB RAM ähnlich wie bei den ATmegas. Dazu kommen bis zu 8 UARTs, mehrere bis zu 2 MS/s (Megasamples pro Sekunde) schnelle 12-Bit-AD- und -DA-Wandler sowie 8 Timer. Für Bastler sind diese vorerst nicht unbedingt geeignet, da keine 5-V-kompatiblen Modelle angekündigt sind (die spezifizierte Betriebsspannung ist 1,6–3,6 V). DIL-Gehäuse sucht man ebenfalls vergeblich.

Tipp: Mehr Informationen gibt es auf der XMega-Webseite von Atmel. Dort finden Sie die aktuellen Informationen und Datenblätter.

Felix Irmscher

Audiospektrum-Analyse mit Mikrocontrollern

Analyseschaltungen mit dem ATmega644 einfach selbst entwickeln

Vorwort

In diesem Buch lernen Sie, ATMEL-Mikrocontroller zur Signalanalyse für Frequenzen im hörbaren Bereich zu nutzen. Der Einstieg in die Assembler-Programmierung von spannenden Anwendungen zur Audiospektrum-Analyse eröffnet sich Ihnen ohne nennenswerten theoretischen Ballast. Sie benötigen keine Kenntnisse in höherer Mathematik!

Um die im Buch aufgeführten Versuche und später auch Ihre eigenen Projekte durchführen zu können, wird der Bau eines komfortablen Experimentiersystems beschrieben.

Damit richtet sich das praxisorientierte Buch an engagierte Hobbyisten und Studierende, die sich mit Signalanalyse und Mikrocontrollertechnik beschäftigen.

Zur Überprüfung der Funktionstüchtigkeit sind auf der beiliegenden CD-ROM auch mehrere Analyse-Tools und Signalgeneratoren für den PC enthalten, die überwiegend in der Programmiersprache LabVIEW verfasst wurden. Da diese Software hierbei eine ideale Ergänzung bietet, ist ihre Darstellung ein weiterer wichtiger Themenschwerpunkt des Buchs. Es wird u. a. beschrieben, wie LabVIEW und ein Lehrbuch zu Lernzwecken sehr kostengünstig erworben werden können. Der Datentransfer zum PC erfolgt über die Serielle Schnittstelle RS-232.

Mikrocontroller zur Signalanalyse zu verwenden, ist eher ungewöhnlich. Sie bieten aber für Frequenzen im hörbaren Bereich zu den nur sehr schwer erhältlichen digitalen Signalprozessoren eine kostengünstige Alternative. Zur Anwendung kommt der *ATMELmega644* im für Experimente praktischen DIL-40-Gehäuse.

Die Programmierung der Mikrocontroller erfolgt zur Geschwindigkeitsoptimierung in Assembler. Diese Programmiersprache hat zu Unrecht das Image, sehr kompliziert zu sein. Es kommen Programmablaufpläne, ausführlich dokumentierte Listings und zusätzliche Erläuterungen zum Einsatz, die das Gegenteil beweisen sollen. Die Anwendung weniger häufig genutzter Funktionen wie z. B. Makros, Puffer und der Watchdog-Timer werden ausführlich beschrieben. Grundkenntnisse in Assembler sollten bereits vorhanden sein.

Die für die Signal-Analyse notwendigen FFT-Algorithmen sind rechenintensiv. Auf dem Mikrocontroller bleiben dann kaum noch Ressourcen für weiterführende Auswertungen. Das ist wohl auch der Grund, warum sich die meisten im Internet verfügbaren Projekte bisher darauf beschränkten, die Machbarkeit der Spektrum-Analyse mithilfe einer kleinen Bargraph-Anzeige zu demonstrieren.

Um leistungsfähigere Schaltungen entwickeln zu können, werden zwei miteinander kommunizierende Mikrocontroller verwendet. Dabei übernimmt der eine die Berechnung der Audiospektren aus den über ein Mikrofon aufgenommenen Signalen. Er wird als *FFT-Baustein* bezeichnet. Der andere, der sogenannte *Analysebaustein,* übernimmt die weiterführenden Berechnungen.

Während der Analysebaustein das letzte Spektrum auswertet, kann der FFT-Baustein bereits neue Daten generieren. Dadurch lässt sich die Arbeitsgeschwindigkeit beträchtlich erhöhen. Für den Anwender hat das zudem den Vorteil, dass er sich nicht mehr mit den komplizierten FFT-Algorithmen beschäftigen muss.

In diesem Buch werden Wege aufgezeigt, wie die Spektren mit dem Analysebaustein weiter ausgewertet werden können. Hierzu steht ein sogenanntes *Basisprogramm* mit vielen nützlichen Routinen zur Verfügung.

Einige Schaltungsbeispiele stellen Anwendungsmöglichkeiten dar. Der Leser wird also letztlich befähigt, mithilfe des vorkonfektionierten FFT-Bausteins und dem Basisprogramm des Analysebausteins eigene Projekte zu realisieren.

Um das Funktionsprinzip der Generierung von Audiospektren bereits vor dem Nachbau des Experimentiersystems nachvollziehen zu können, steht ein einfaches, in LabVIEW erstelltes Simulationsprogramm zur Verfügung.

Mein Dank gilt den Autoren, die mir die Genehmigungen zur Wiedergabe ihrer wertvollen Programme, Schaltungsideen und Beschreibungen ausgesprochen haben. Sie werden im Text namentlich aufgeführt. Besonderer Dank gebührt *National Instruments* und Herrn Backmeyer für die hervorragende Unterstützung bei der Realisierung der LabVIEW-Projekte.

Lübeck, Januar 2013

Felix Irmscher

Inhaltsverzeichnis

1 Theorie zur Fixpunkt-FFT

Meist beginnen Bücher, die sich mit Signalanalyse beschäftigen, mit einer kompakten, mit zahlreichen Formeln gespickten theoretischen Einführung. Leider sind sie für Leser ohne fundiertes Vorwissen in höherer Mathematik kaum nachzuvollziehen. Die Programmierung eigener Anwendungen kann mithilfe dieses Buchs jedoch auch ohne tiefer greifendes Verständnis gelingen. Deshalb beschränkt sich die Einführung nur auf die für eigene Projekte notwendigen Aspekte.

1.1 Grundlagen

Mit den hier vorliegenden Schaltungen können Frequenzanalysen des vom Menschen hörbaren Schalls durchgeführt werden.

In unserem Umfeld überlagert sich eine Vielzahl von Schallereignissen. Mithilfe eines Mikrofons kann der durch den Schall entstehende Schallwechseldruck in eine Wechselspannung umgewandelt werden. Diese wird dann, nach entsprechender Verstärkung und Digitalisierung, durch Mikrocontroller analysiert.

Dabei werden aus dem an der Position des Mikrofons vorliegenden Signalgemisch die Amplituden der einzelnen Frequenzanteile bestimmt.

Diese Komponenten sind die reinen, unterschiedlich lauten Sinusschwingungen, aus denen das Signal aufgebaut ist.

Das hierzu notwendige Verfahren wird als *Fast-Fourier-Transformation* (FFT) bezeichnet.

Bemerkenswert ist, dass der Mathematiker Carl Friedrich Gauß schon im Jahre 1805 hierzu einen ersten Algorithmus entwickelt hatte. 1822 wies er seine sogenannte *Fourierreihe* nach. Eine erhebliche Beschleunigung der aufwendigen Berechnungen gelang James Cooley und John W. Tukey 1965 mit der FFT. Seither ist eine Vielzahl weiterer nochmals verbesserter Algorithmen entstanden.

Die moderne Elektronik ist ohne die reichhaltigen Anwendungsmöglichkeiten der FFT nicht mehr vorstellbar. Als berühmte Beispiele seien hier die Kompressionsalgorithmen für das MP3-Format und die Datenübertragung von DVB-T und WLAN genannt.

Im Alltag begegnet uns die FFT auch häufig bei Hi-Fi-Geräten. Auf den Displays von Radios und Equalizern werden die Frequenzanteile der Musik durch Balkengrafiken grob dargestellt.

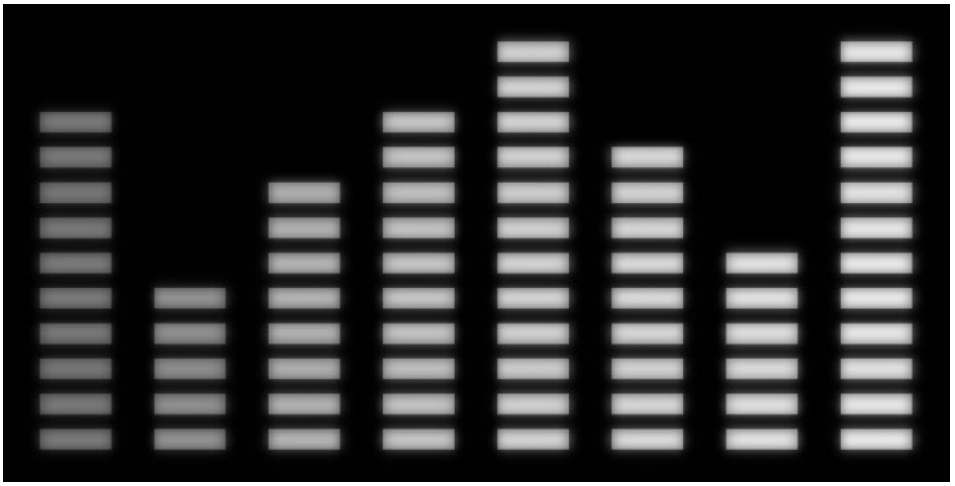

Abb. 1.1: Frequenzanzeige eines HiFi-Geräts

In diesem Beispiel wird das Audiospektrum in Echtzeit in seine Bestandteile zerlegt. Die untere Grenzfrequenz ist bei der FFT stets 0 Hz. Die obere Grenzfrequenz kann (fast) frei gewählt werden. Sie steht im Zusammenhang mit der noch zu beschreibenden Samplingrate. Hier wird angenommen, dass der Balken ganz rechts die für den Menschen gerade noch hörbaren 20 kHz anzeigt.

Da die Frequenzbereiche bei der FFT für jeden Balken gleich groß sein müssen, ergeben sie sich zu 20 kHz / 8 = 2,5 kHz. Der erste Balken zeigt also die gemeinsame Amplitude der Frequenzen von 0 Hz bis 2,5 kHz, der zweite Balken von 2,5 kHz bis 5 kHz usw. Die Anzahl der Balken muss bei der FFT stets eine Potenz zur Basis 2 sein. Weiterhin ist bei dem Beispiel ersichtlich, dass die Amplitude jedes Balkens durch maximal 12 LEDs angezeigt werden kann.

Eine Audiospektrum-Analyse mit so geringen Auflösungen bei den Frequenzbereichen und den Amplituden der Balken bietet natürlich allenfalls einen schönen optischen Effekt. Für ernsthafte Analysen müssen viel genauere Werte ermittelt werden. Dabei ist auch die zeitliche Auflösung, d. h., wie schnell Spektrum-Analysen durchgeführt werden können, von großer Bedeutung.

Da die Erhöhung der Balkenzahl mit immer längeren Berechnungen einhergeht, ist ein Kompromiss bei der Auflösung von Frequenz und Zeit zu finden. Als Grundlage der hier verwendeten FFT-Algorithmen dienen die *GNU*-lizenzierten Programme auf der Internetseite *http://elm-chan.org/works/akilcd/report_e.html*.

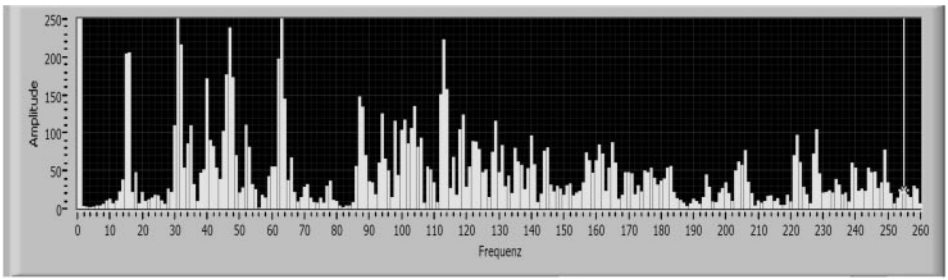

Abb. 1.2: Audiospektrum

Mit dem Experimentiersystem des Buchs lassen sich Audiospektren wie in Abb. 1.2 ermitteln. Im Folgenden sind die Grenzwerte dieser Spektren aufgeführt:

Obere Grenzfrequenz:	4.800 Hz
Untere Grenzfrequenz:	0 Hz
Anzahl der Balken:	256
Frequenzbereich der Balken:	18,75 Hz
Auflösung der Amplituden:	0–254
Geschwindigkeit pro Analyse:	ca. 55,6 ms

Die obere Grenzfrequenz erscheint wahrscheinlich recht niedrig. Sie ist aber für eine Vielzahl von Anwendungen völlig ausreichend. Mikrocontroller sind aufgrund ihrer niedrigen Taktrate, des kleinen SRAM und der geringen ADC-Samplerate nur bedingt für die Signalanalyse geeignet. Bei $2^8 = 256$ Balken wird bereits ein RAM von 4 KByte benötigt, das der verwendete Mikrocontroller *ATMELmega644* gerade noch zur Verfügung stellt. Bei 256 Balken ist die Geschwindigkeit der FFT mit ca. 56 ms noch akzeptabel. Mit $2^9 = 1.024$ Balken würden sich der Speicherplatzbedarf und die Analysedauer verdoppeln.

Um den Frequenzbereich der 256 Balken möglichst schmal zu halten, wurde die obere Grenzfrequenz mit nur 4,8 kHz gewählt. Damit haben die Balken eine Breite von 18,75 Hz (4,8 kHz / 256). Das bedeutet, dass Frequenzen mit einer Genauigkeit von ± 9,375 Hz bestimmt werden können. Durch weiterführende Analysen wird es möglich, Auflösungen von ca. ± 2,5 Hz zu erreichen.

Die Höhe des Balkens ist ein Maß für die Lautstärke seines Frequenzbereichs. Mit Werten von 0 bis 254 wird hierbei eine sehr gute Auflösung möglich. Wie später noch beschrieben, basiert die Balkenhöhe auf logarithmischen Berechnungen. Dadurch können Audiosignale in einem sehr großen Dynamikbereich erfasst werden. Der Wert »255« des 8 Bit umfassenden Balkens wird als Steuerzeichen für die serielle Datenübertragung an den PC benötigt.

1.2 Hochpassfilter

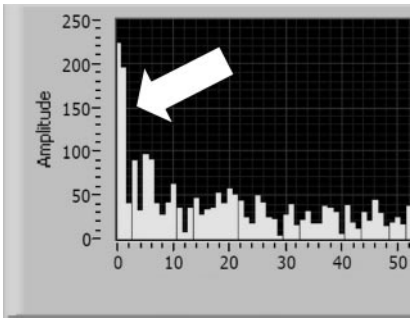

Abb. 1.3: Gleichstromanteil

Am unteren Ende der Audiospektren (bis ca. 50 Hz) finden sich in den Grafiken stets höhere Balken. Diese lassen sich auf den sogenannten Gleichstromanteil und Störungen bei der analogen Schallaufnahme zurückführen (Netzbrummen). Da das verwendete Mikrofon eine untere Grenzfrequenz aufweist, ist es sinnlos, Balken unterhalb dieses Werts zu analysieren. Ein gegen beide Störungen wirksamer Hochpassfilter lässt sich einfach realisieren, indem man die betreffenden Balken bei der anschließenden Analyse nicht mehr verwendet. Um zusätzliche Störungen, die z. B. durch PC-Lüfter auftreten können, zu eliminieren, wird der Hochpass bei vielen der folgenden Experimente vereinfachend auf 300 Hz eingestellt.

1.3 Anti-Aliasing- und Tiefpass-Filter

Die Samplerate für die Digitalisierung des analogen Mikrofonsignals wurde im *ELM-CHAN*-Projekt auf 9,6 kHz festgelegt. Das bedeutet, dass Messwerte fortlaufend in einem zeitlichen Abstand von ca. 104 µs (1 / 9,6 kHz) erhoben werden.

Nach dem sogenannten *Nyquist-Shannon-Abtasttheorem* ist die höchste vom ADC erfassbare analoge Frequenz etwas weniger als die Hälfte der Samplerate. Das bedeutet, dass die Digitalisierung hier nur für Audiosignale unterhalb von 4,8 kHz (9,6 kHz / 2) möglich wird. Die folgenden drei Diagramme zeigen Sinussignale von 4,8 kHz. Die Abtastzeiten der AD-Wandler sind als Punkte dargestellt.

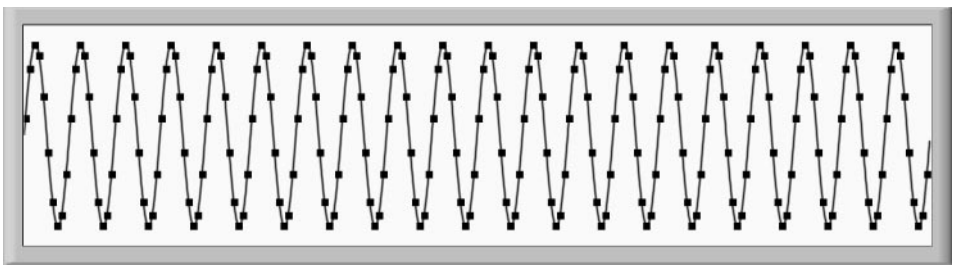

Abb. 1.4: Samplerate >9.600 Hz

In Abb. 1.4 ist die Samplerate mehr als doppelt so hoch wie das Sinussignal. Man kann erkennen, dass das Signal gut mithilfe der Abtastpunkte rekonstruiert werden kann.

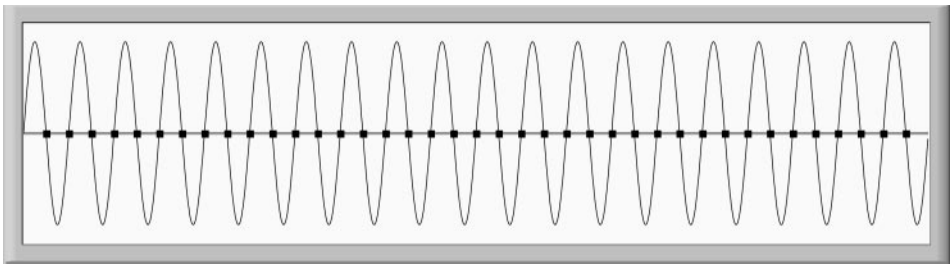

Abb. 1.5: Samplerate = 9.600 Hz

Wenn, wie in Abb. 1.5 dargestellt, die Samplerate genau zweimal so groß wie die Signalfrequenz ist, kann die Frequenzerkennung fehlerhaft werden. In diesem Beispiel bleibt der Sinus für den AD-Wandler sogar unsichtbar.

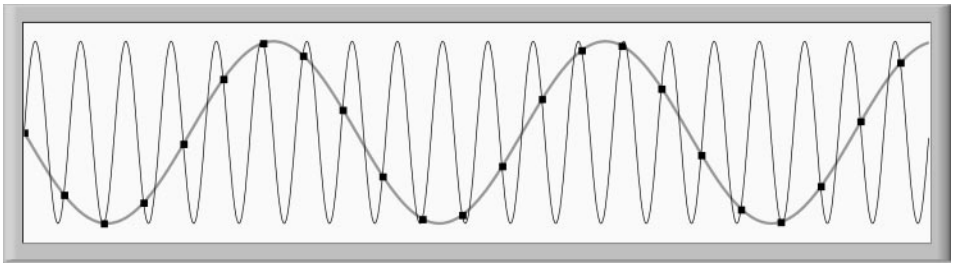

Abb. 1.6: Samplerate < 9.600 Hz

Beträgt die Abtastfrequenz weniger als die Hälfte des zu erfassenden Originalsignals, werden falsche, zu niedrige Frequenzen detektiert (Abb. 1.6).

Methoden, um fehlerhafte Digitalisierungen durch zu geringe Sampleraten zu vermeiden, werden als *Anti-Aliasing* bezeichnet. Die störenden *Alias-Effekte* können mithilfe von Tiefpassfiltern möglichst hoher Ordnung beseitigt werden. In unserem Fall sollte ein Tiefpassfilter verwendet werden, der Frequenzen ab 4,8 kHz herausfiltert. Im *ELM-CHAN*-Projekt wurde hierzu ein sogenannter *elliptischer Filter* achter Ordnung verwendet (*MAX293*). Allgemein gilt der Gebrauch eines Tiefpassfilters zur Vermeidung des *Alias-Effekts* als unumgänglich. Die Beispielschaltungen in diesem Buch liefern allerdings auch ohne Tiefpass gute Ergebnisse. Das lässt sich wie folgt erklären: Die Obergrenze der hier gewonnenen Audiospektren liegt bei 4,8 kHz. Die Spektren der meisten Geräusche in unserer Umgebung haben ihre maximalen Amplituden unterhalb dieser Frequenz. Hinzu kommt, dass das verwendete Elektret-Kondensatormikrofon und die Verstärkerschaltung höhere Frequenzen weniger verstärken und damit auch schon als Tiefpass fungieren.

Da die im *ELM-CHAN*-Projekt genutzte geringe Samplerate mit Bedacht gewählt erscheint, wird sie auch in diesem Buch verwendet. Hiermit können die meisten Töne

unseres Alltags erfasst werden, und die Frequenzbereiche der einzelnen Balken bleiben ausreichend klein. In einem späteren Experiment kann der Alias-Effekt mit einem in der Tonhöhe langsam ansteigenden Sinussignal (sogenanntes *Sweep*-Signal) beeindruckend dargestellt werden (Kapitel 11.1.4).

1.4 Funktionsschema der FFT

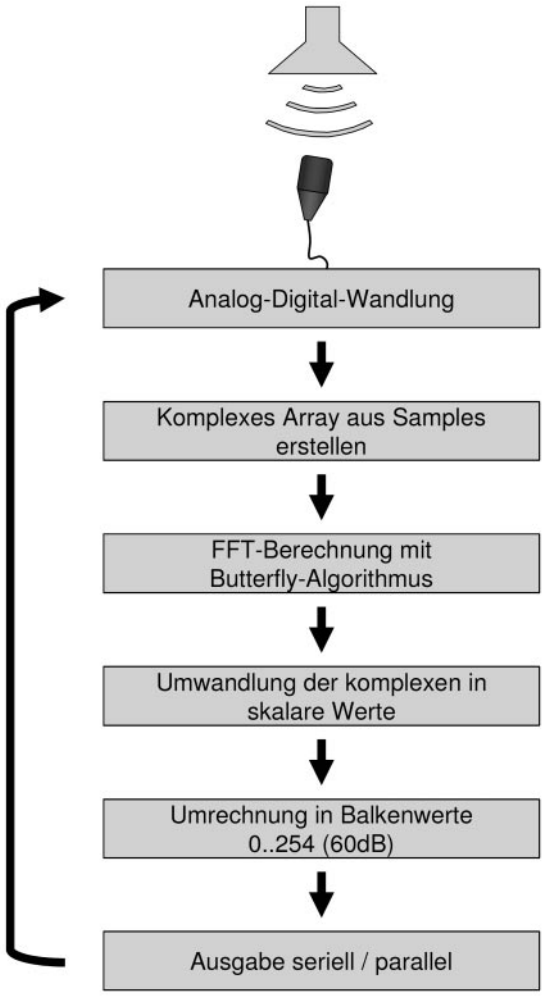

Abb. 1.7: Prinzip der Spektrum-Analyse

Abb. 1.7 zeigt stark vereinfacht die Schritte, die mithilfe des Mikrocontrollers durchgeführt werden müssen, um ein Audiospektrum zu erstellen. Zunächst wird das akustische Signal über ein Mikrofon in eine Wechselspannung umgeformt und ggf. verstärkt und gefiltert. Die Spannung wird dann mit einem ADC-Kanal des Controllers digitalisiert.

Anschließend werden aus den Abtastwerten (Samples) die für die Transformation notwendigen komplexen Zahlen generiert. Diese bestehen aus zwei unabhängigen Komponenten. Sie werden in einem Array für die weiteren Berechnungen sortiert abgespeichert. Danach erfolgt die eigentliche FFT mit dem sogenannten *Butterfly-Algorithmus*. Die besondere Bezeichnung ist durch die Form einer im Datenflussdiagramm des Algorithmus fortlaufend wiederkehrenden Grundstruktur zu erklären, die einem Schmetterling ähnelt.

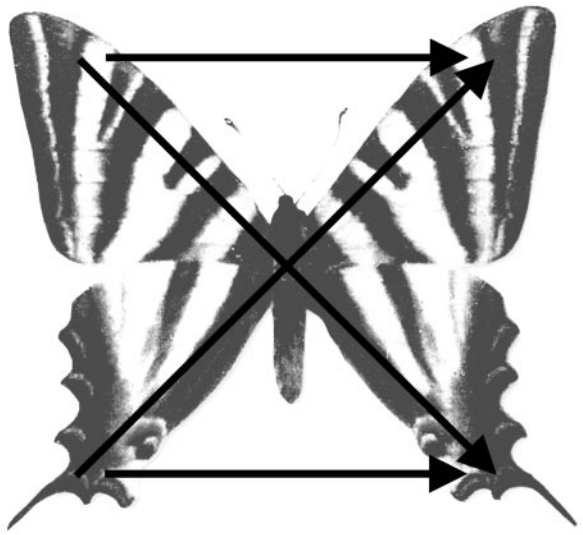

Abb. 1.8: Butterfly-Algorithmus

Die Ergebnisse der Transformation liegen jetzt noch als komplexe Zahlen vor und werden in Skalare, also in Größen, die wieder nur eine Komponente enthalten, zurückverwandelt. Diese Skalare liegen zunächst im 16-Bit-Format vor. Um die Rechengeschwindigkeit der weiteren Berechnungen zu erhöhen und um Speicherplatz zu sparen, erfolgt die Konvertierung in 8-Bit-Werte mit der Einheit Dezibel.

Sämtliche Berechnungen können in einer Endlosschleife wiederholt werden und in Echtzeit Signalverläufe analysieren.

1.5 Weiterführende Literatur

Wer sich näher mit der Fourier-Transformation beschäftigen möchte, wird im Internet und im Buchhandel zahlreich fündig. Nachfolgend werden einige wenige Quellen beschrieben. Außerdem wird eine lohnenswerte Einführung in die Programmiersprache Assembler vorgestellt.

FTT ohne Mathematik erklärt

Auf der Internetseite *http://www.sprut.de/electronic/pic/16bit/dsp/fft/fft.htm* findet der Autor Bredendiek erfrischende und verständliche Worte, um dem Leser die Theorie der FFT auch ohne Formeln näher zu bringen. Dank der Zustimmung des Autors konnte die Erklärung auf der beiliegenden CD-ROM im Ordner *FFT* hinzugefügt werden.

Wikipedia

Für kurze und einfach verständliche naturwissenschaftliche Erklärungen ist Wikipedia fast immer von Nutzen.

Fourier-Transformation für Fußgänger

Wer mathematisches Vorwissen mitbringt, kann mit diesem ausführlichen Werk die Fourier-Transformation »von der Pike auf« verstehen lernen: »Fourier-Transformation für Fußgänger« von Tilman Butz, 7. aktualisierte Auflage 2011; Wiesbaden, Gabler Wissenschaftsverlage, 2011.

Virtuelle Instrumente und Signalverarbeitung

Dieses Buch beschäftigt sich mit der Signalverarbeitung zur Zustandsüberwachung und Fehlerdiagnose von Maschinen. Die Grundlagen der Signalanalyse werden gut verständlich beschrieben und durch zahlreiche, mit LabVIEW-Anwendungen erstellte Diagramme veranschaulicht: »Virtuelle Instrumente und Signalverarbeitung: Zustandsüberwachung und Diagnose von Maschinen mit LabVIEW« von Karl Walter Bonfig und Zhondong Liu, 1. Aufl. Berlin, VDE-Verlag, 2004.

Anfängerkurs zum Erlernen der Assemblersprache von Atmel AVR-Mikroprozessoren

Gerhard Schmidt stellt auf seiner Internetseite eine sehr gute Einführung in Assembler vor. Dieses kleine Nachschlagewerk im PDF-Format bietet sowohl Anfängern als auch Fortgeschrittenen wertvolle Programmiertipps. Es befindet sich – mit freundlicher Genehmigung des Autors – im Ordner *Kurs Assembler* auf der CD-ROM. Alternativ kann die Einführung unter *http://www.avr-asm-tutorial.net/avr_de/index.html* kostenlos heruntergeladen werden

2 Simulationsprogramm in LabVIEW

2.1 Verwendungszweck

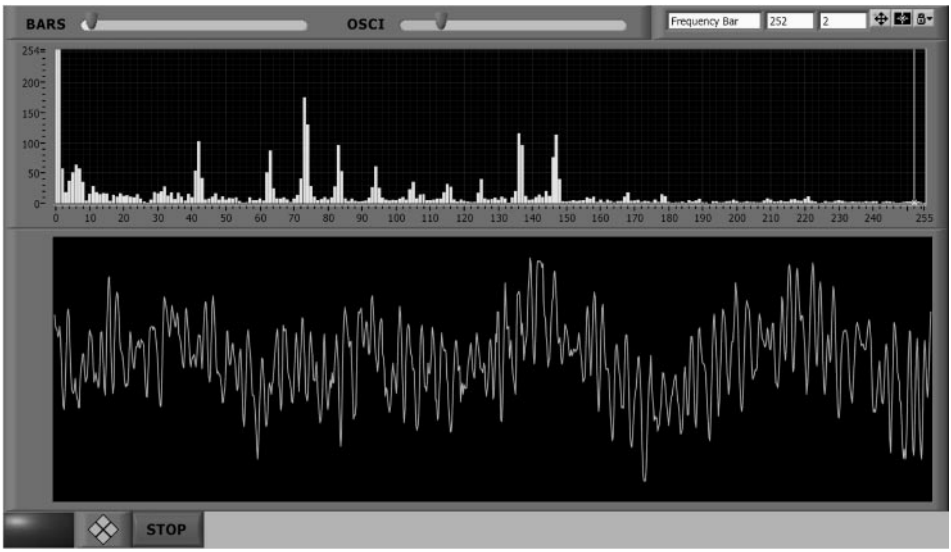

Abb. 2.1: SIMULATOR.vi mit Radiomusik (Front)

Wer sich Wissen autodidaktisch durch entsprechende Literatur aneignet, weiß, wie wichtig Schaltungsbeispiele sind, um den Praxisbezug zu den vermittelten Lerninhalten nachvollziehen zu können. Wahrscheinlich bleibt aber nicht immer die Zeit zum Nachbau der in den Fachbüchern dargestellten Schaltungen. Um auch ohne Lötkolben einen ersten Eindruck über die mit dem FFT-Baustein realisierbaren Audiospektrum-Analysen erhalten zu können, befindet sich auf der beiliegenden CD-ROM im Ordner *LabVIEW* die Anwendung *SIMULATOR.vi.* Es ist jedoch erst nach der Installation der Programmiersprache LabVIEW auf Ihrem PC möglich, dieses und alle weiteren im Buch aufgeführten, mit dieser Sprache erstellten Analyseprogramme und Signalgeneratoren zu öffnen.

2.2 Bedienungsanleitung

Der Simulator dient nicht nur zur Veranschaulichung der Theorie. In begrenztem Umfang kann das Programm auch nützlich für die Entwicklung von neuen Anwendungen sein. Das Audiosignal wird über die Soundkarte des PCs empfangen. Hierzu muss ein Mikrofon angeschlossen werden. Es ist auch möglich, Signale und Tondateien, die mit dem Computer abgespielt werden, mithilfe des sogenannten *Stereo-Mixers* an die Soundkarte zu leiten.

Abb. 2.2: Wahl zwischen Mikrofon und Stereo-Mixer

Abb. 2.2 zeigt das Fenster *Recording Control*, das beispielsweise in *Windows XP* aufgerufen werden kann, wenn in der Taskleiste die Lautstärkeregelung geöffnet wird. Hier wird zur Einstellung Ihrer Soundkarte nur auf die zahlreichen im Internet verfügbaren und an die unterschiedlichen Betriebssysteme angepassten Anleitungen verwiesen. Im oberen Diagramm des Simulators wird stets das aktuelle Audiospektrum für den Bereich von 0 Hz bis 4.800 Hz mithilfe von 256 Frequenzbalken nahezu in Echtzeit angezeigt. Diese Genauigkeit entspricht der, die auch bei der Signalanalyse mithilfe des FFT-Bausteins erreicht wird. Der Simulator zeigt nur Frequenzanteile kleiner oder gleich 4.800 Hz an. Er verhält sich damit so, als ob ihm ein Tiefpassfilter mit extrem hoher Flankensteilheit vorgeschaltet wäre, der alle Frequenzen oberhalb des Durchlassbereichs eliminiert. Deswegen kann der Versuch zum Alias-Effekt auch nur mit dem Experimentiersystem und nicht mit dem Simulator durchgeführt werden (Kapitel 11.1.4).

Die Empfindlichkeit und damit die Balkenhöhe lässt sich mit dem Schieberegler *BARS* variieren. Anders als bei den durch den Mikrocontroller generierten Daten ist es möglich, auch Werte oberhalb von 254 zu erreichen.

Für die genaue Analyse einzelner Balken kann der senkrechte gelbe Cursor mithilfe der Maus verschoben werden. Neben der rechts oben befindlichen schwarz-weißen Anzeige *Frequency Bar* werden die Balkennummer und der Amplitudenwert angezeigt, die der

momentanen Position des Cursors entsprechen. In Abb. 2.1 wird somit Balken 252 markiert. Seine Höhe beträgt 2. Gegebenenfalls ist es notwendig, die Anzeige im oberen Diagramm zu kalibrieren (siehe Kapitel 11.3.3).

Das untere Diagramm zeigt die Wechselspannung, die die Soundkarte aus dem Audiosignal generiert hat. Hier wurde Radiomusik aufgenommen. Es ist gut zu erkennen, dass sich höhere und tiefere Frequenzen überlagern. Die Darstellung ähnelt der eines Oszilloskops. Mit dem Schieberegler *Osci* kann die Empfindlichkeit (Y-Achse) variiert werden. Die Anzeige der Wechselspannung wird dabei automatisch in einer mittleren Y-Position gehalten.

Die Zeitablenkung (X-Achse) ist nicht variabel. Die Voreinstellung ist aber so gewählt, dass sie im relevanten Frequenzbereich von 0 Hz bis 4.800 Hz eine akzeptable Darstellung liefert.

Der Programmcode des Simulators ist in der grafischen Programmiersprache LabVIEW geschrieben. Abb. 2.3 zeigt das sogenannte *Blockdiagramm*, das die einzelnen Anweisungen enthält.

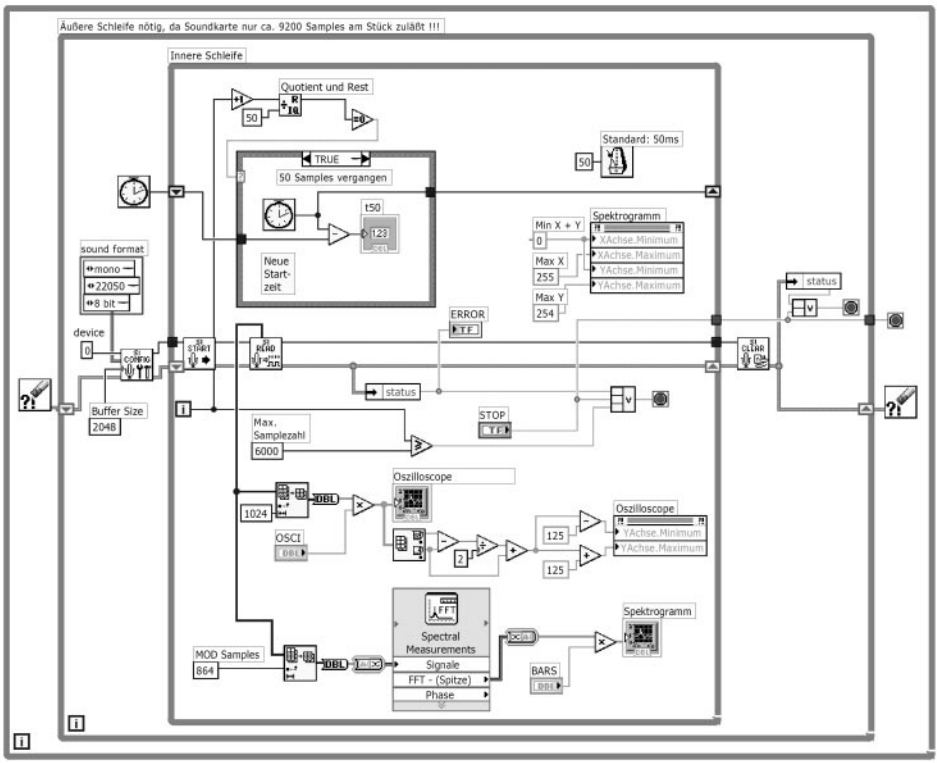

Abb. 2.3: SIMULATOR.vi (Block)

2.3 Soundkarten-Oszilloskope

Mittlerweile gibt es eine Vielzahl von im Internet verfügbaren Soundkarten-Oszilloskopen. Obwohl sie zum Teil einen beachtlichen Funktionsumfang bieten, sind sie noch nicht in der Lage, »echte« Geräte zu ersetzen. Um die Amplituden eines Wechselspannungssignals ermitteln zu können, müssten die Soundkarten-Oszilloskope nämlich möglichst vor jedem Gebrauch mit externen Signalen kalibriert werden, da es durch die Verwendung anderer Programme leicht zu Modifikationen in den Einstellungen der Soundkarte kommen kann, die Fehlmessungen nach sich ziehen. Gleichspannungen können gar nicht gemessen werden. Dennoch erlauben die Soundkarten-Oszilloskope erste Analysen von unbekannten Signalen. Im Folgenden werden die Internetadressen von zwei Oszilloskopen genannt, die zum Zeitpunkt der Drucklegung für den privaten Gebrauch kostenlos heruntergeladen werden können.

Auf der Homepage von Bernd Reiser (*http://dsp4swls.de/*) findet sich das kleine Programm *Oszilight.exe*. Es ist praktischerweise ohne Installation sofort lauffähig. Zu seinem Funktionsumfang gehören neben dem Zweikanal-Oszilloskop auch eine FFT-Darstellung und ein X-/Y-Graph. Zusätzlich kann auf der angegebenen Internetseite ein vielseitiger Signalgenerator *generator.exe* für den hörbaren Frequenzbereich bezogen werden.

Beide Programme werden in dem Buch »Elektronische Schaltungen« von Richard Zierl (1. Aufl. Franzis Verlag, 2007) genau beschrieben.

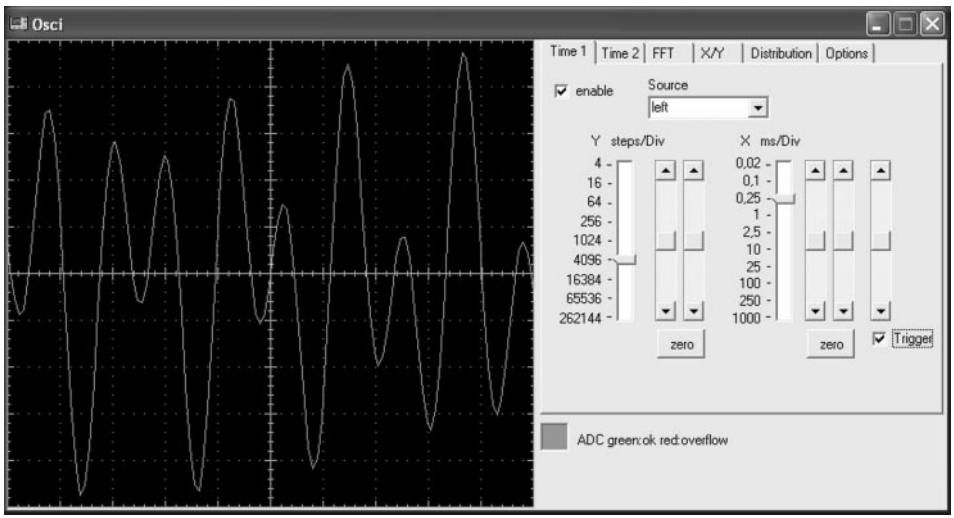

Abb. 2.4: Oszilight.exe

Das sehr aufwendig gestaltete Programm *Scope_140.exe* ist auf der Internetseite *http://www.zeitnitz.de/Christian/* verfügbar. Es wurde mit der auch in diesem Buch verwendeten Programmiersprache LabVIEW erstellt. Neben den oben beschriebenen Funktionen ist auch ein leistungsfähiger Funktionsgenerator integriert. Die Anwendung

kann kostenlos heruntergeladen werden, sie ist aber keine Freeware. Beachten Sie bitte die Lizenzbestimmungen.

Die drei Anwendungen befinden sich auf der CD-ROM im Ordner *Oszi-Signal*.

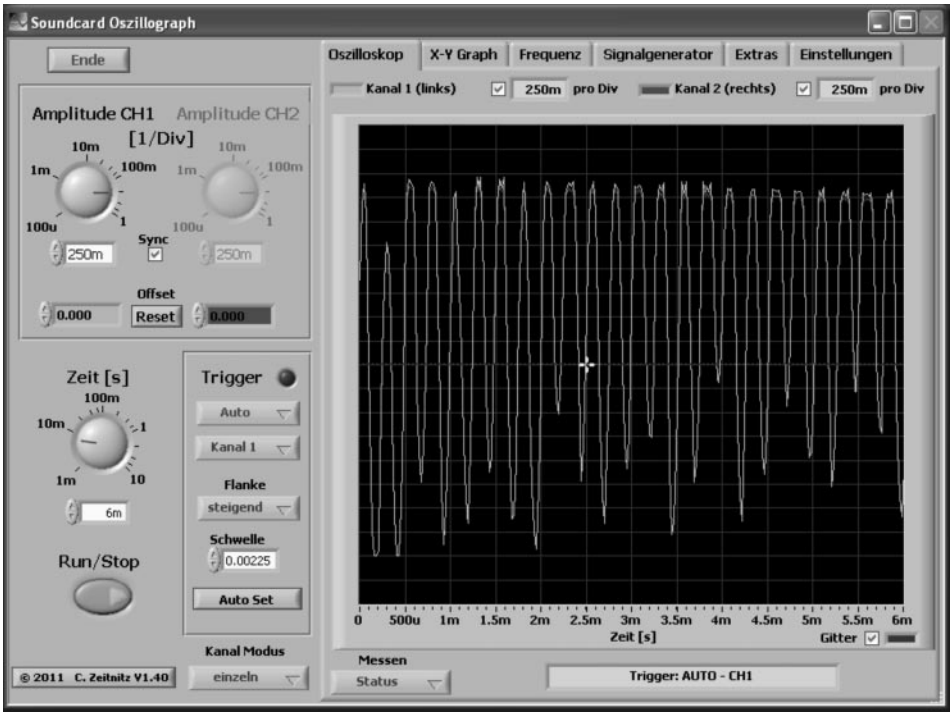

Abb. 2.5: Scope_140.exe

Frank Sichla

Schaltungssammlung
Mikrocontroller und USB

Mehr als 330 erprobte Schaltungen für Labor, Entwicklung und Anwendung

Vorwort

Mikrocontroller (auch als µController, µC oder MCU für Microcontroller Unit bezeichnet) sind aus der modernen Elektronik nicht mehr wegzudenkende digitale Universalbausteine. Sie vereinen einen Prozessor und mindestens einige Peripheriefunktionen auf einem Chip. In den meisten Fällen befinden sich Arbeits- und Programmspeicher ebenfalls (teilweise oder komplett) auf demselben Chip. Ein Mikrocontroller ist somit praktisch ein Einchip-Computersystem.

Heutige Mikrocontroller bieten häufig komplexe Peripheriefunktionen, wie CAN (Controller Area Network), LIN (Local Interconnect Network), USB (Universal Serial Bus), I²C (Inter-Integrated Circuit), SPI (Serial Peripheral Interface), serielle oder Ethernet-Schnittstellen, PWM-Ausgänge (Pulse Width Modulation), LCD-Controller und -Treiber oder Analog/Digital-Wandler.

Die klassische Mikrocontrollerarchitektur war nicht als reines Mikroprozessorsystem gedacht, sondern primär für Steuerungsaufgaben vorgesehen. Hier ist ein Single-Chip-Betrieb ohne externe Speicherbausteine möglich, und der Befehlssatz der CPU hält meist spezielle Befehle für das Steuern einzelner Signalleitungen (mittels sogenannter Bitmanipulationen) bereit. Weiter zeichnen sich diese Controller durch eine kurze Interrupt-Latenzzeit aus. Typische Vertreter dieser Gattung sind der 8051 von Intel sowie der C166 von Siemens (heute Infineon) und der TriCore von Infineon.

Heute kann man oft beobachten, dass Mikroprozessoren nach einiger Zeit als Mikrocontrollervarianten angeboten werden, sodass für ein funktionsfähiges Prozessorsystem oft nur noch ein Quarz und Speicherbausteine nötig sind. Als typische Vertreter dieser Architektur seien der 80186 von Intel (vom 8086 abgeleitet), die XScale-Familie (ARM) sowie ColdFire (MC680xx) von Freescale (vormals Motorola) genannt.

Der 8085 war der Scheideweg, nach dem sich die Bausteine in reine Datenverarbeiter (Mikroprozessoren, wie die 8086-Familie) und Datenübersetzer (Mikrocontroller, wie 8048 und 8051) als Schnittstelle zwischen Hardware und CPU aufteilten.

Anfangs gab es fast ausschließlich ROM-Speicher (Read Only Memory). Die Software muss mit Entwicklungssystemen erstellt werden, die eine Simulation des Mikrocontrollers erlauben und meist über einen In-Circuit Emulator verfügen. Die Software erhält der Chip im Fertigungsprozess („Maskenprogrammierung“).

Ab der zweiten Hälfte der siebziger Jahre wurden EPROMs (Erasable Programmable) verwendet. Der Programmspeicher wurde mithilfe eines Programmiergeräts beschrieben und konnte durch UV-Licht wieder gelöscht werden. OTP-Mikrocomputer (One Time Programmable, einmal programmierbar) besitzen einen EPROM, aber kein Fenster für UV-Licht.

Neue Mikrocontroller nutzen oft einen Flash-Programmspeicher. Da dieser direkt elektrisch löschbar und wiederbeschreibbar ist, entfällt das Keramikgehäuse, und der Speicher kann im Herstellungsprozess komplett getestet werden.

Ein Mikrocontroller kann extern oder intern getaktet werden. Dabei sind maximale Frequenzen von 1 bis über 100 MHz möglich. Die Taktfrequenz ist jedoch kein verlässliches Maß für die tatsächliche Rechengeschwindigkeit. So wird z. B. bei den meisten 8051-Controllern die Frequenz intern durch 12 geteilt (Maschinentakt). Weiter ist von Bedeutung, wie viele Zyklen (Takte) im Schnitt pro Befehl erforderlich sind.

Mikrocontroller-Speichergrößen liegen zwischen 1 KB (1024 Bytes) und 2 MB (2000 KB) für das Programm (interner RAM, Random Access Memory) und zwischen 32 Bytes und 48 KB für die Daten. Wesentlich größere externe Speicher können oft genutzt werden.

Mikrocontroller werden meist in Assembler oder C programmiert. Im Hobbybereich konnten sich darüber hinaus speziell entworfene Sprachen, wie JAL für die PICmicro-Familie von Mikrochip, etablieren.

Assembler nutzt man bei 4- und 8-Bit-Architekturen, um die Ressourcen effektiv auszuschöpfen. Lässt es die Entwicklungsaufgabe zu, die Ressourcen wenig in Anspruch zu nehmen, ermöglicht C ein sehr hardwarenahes Programmieren in kurzer Zeit.

Relativ neu sind Entwicklungswerkzeuge, die den rein grafischen Softwareentwurf erlauben.

Einige Mikrocontroller, wie der 8052AH-BASIC, haben einen Hochspracheninterpreter integriert. Hier genügt ein Rechner mit einem Terminalprogramm zur Programmierung. Solche Lösungen sind im Hobbybereich wie auch in der industriellen Kleinserienfertigung beliebt.

Das Interesse an Mikrocontrollern ist nach wie vor groß. Die vielfältigsten Aufgaben können damit gelöst werden, wobei es darauf ankommt, den optimal geeigneten Typ zu finden. Das ist nicht einfach, eine ganze Reihe von Kriterien ist zu berücksichtigen, und viele Vor- und Nachteile sind gegeneinander abzuwiegen. Die heute eingesetzten Mikrocontroller bieten zudem eine Mischung aus diversen Techniken und Architekturen (nach Harvard oder von Neumann).

Das vorliegende Buch befreit von der anstrengenden Prozedur der Schaltungsfindung. Es orientiert sich nicht an Prozessortypen, sondern an konkreten Einsatzbeispielen und spiegelt anhand praktischer Anwendungsschaltungen die Einsatzmöglichkeiten aktueller Mikroprozessoren wider. Natürlich wird dabei auch angegeben, woher man den Quellcode beziehen kann.

Ähnlich rasant wie die Mikrocontroller hat der USB (Universal Serial Bus) die Elektronikwelt erobert. Auch hier gibt es heute bereits viele interessante Anwendungsschaltungen.

Der USB ist physikalisch sehr einfach aufgebaut (kompakte Stecker, Punkt-zu-Punkt-Verkabelung, wenige Anschlüsse) und bietet die Möglichkeit, Geräte direkt mit Strom zu versorgen. Dabei können große Datenmengen über ein dünnes Kable transportiert werden. Die Schattenseite dieser überaus positiven Aspekte bei der Anwendung ist eine komplizierte Hardwareentwicklung (wegen der hohen Frequenzen und des sehr komplexen Protokolls mit über 300 Seiten). Da ergibt es ebenso wie bei den Mikroprozessoren Sinn, eine anwendungsorientierte Schaltungssammlung anzubieten.

Ich hoffe, das Buch hilft recht vielen Lesern, Zeit zu sparen, neue Möglichkeiten kennenzulernen und zuverlässige Lösungen zu finden.

Ing. Frank Sichla

Inhaltsverzeichnis

1 Grundschaltungen

1.1 Oszillator für PIC16/17

Der Oszillator ist ein wichtiger Teil jedes Mikrocontrollers. Die in *Abb. 1.1* gezeigte Schaltung ist ein typischer Pierce-Oszillator. Dabei wird die Parallelresonanz des Quarzes genutzt. Der Quarz liegt direkt am Eingang eines invertierenden Verstärkers. Die weitere externe Beschaltung beschränkt sich auf zwei Kondensatoren und einen Widerstand. Die Auswahl wird von verschiedenen Faktoren bestimmt.

Der Hersteller gibt die Quarzfrequenz für den Parallelresonanzfall mit einer äußeren Kapazität von 20...32 pF an. Diese hat kaum Einfluss auf die Frequenz, da das L/C-Verhältnis im Quarz-Ersatzschaltbild äußerst gering ist. Je höher man die Taktfrequenz wählt, umso geringer muss die Anlaufzeit des Oszillators sein, und um so höher ist der Stromverbrauch.

Die Werte von C1 und C2 sollten die vom Hersteller genannte Kapazität berücksichtigen. Es sind also möglichst geringe Werte anzustreben, die aber noch sichere Funktion einschließlich schnellem Anschwingen gewährleisten. Für kurze Anlaufzeit sollte C2 größer als C1 sein.

Der Widerstand R_S reduziert die Quarzbelastung. Er darf aber für gute Funktion nicht größer sein als wenige Kiloohm.

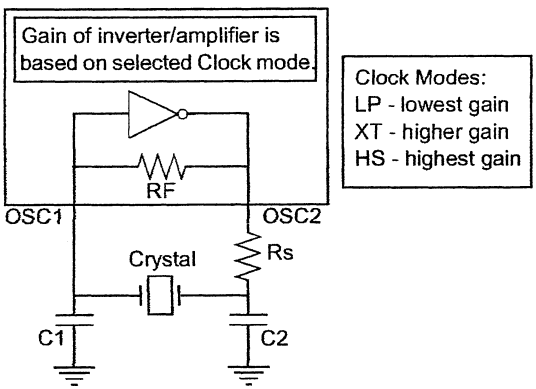

Abb. 1.1: Quelle: Dan Matthews, Basic PIC16/17 Oscillator Design

Das Signal am Ausgang OSCOUT sollte bei guter Sinusform einen Spitze-Spitze-Wert nahe der Betriebsspannung aufweisen.

1.2 RC-Oszillator für PIC16/17

Eine RC-Oszillatorbeschaltung für PICs ist möglich, einfach und gestattet es, die Frequenz durch Änderung des Widerstandswerts einzustellen oder zu steuern. Man benötigt allerdings einen (freien) I/O-Port. Die Beschaltung zeigt *Abb. 1.2*. Die Frequenz wird im Wesentlichen durch C1 und R2 bestimmt. Ist eine Einstellung erwünscht – geringe Frequenz für geringen Stromverbrauch, hohe Frequenz für schnelle Rechenergebnisse – und gelingt sie über R1, sollte man diesen Widerstand nutzen, da er dynamisch an Masse liegt. Eine elektronische Umschaltung gelingt über an Pin OSC1 geschaltete zusätzliche Widerstände, die andererseits an Tristate-Ausgängen liegen (hochohmig: nur R1 wirksam, H-Pegel: R1 und Zusatzwiderstand parallel wirksam, L-Pegel verboten).

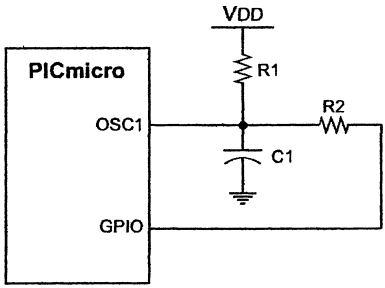

Abb. 1.2: Quelle: Dan Matthews, Basic PIC16/17 Oscillator Design

1.3 RC-Watchdog-Aktivierungsschaltung

Der Watchdog Timer ist ein sehr hilfreicher Bestandteil eines Mikrocontrollers. Er holt ihn beispielsweise aus dem Schlaf-Modus oder sichert den Code. Es gibt aber Applikationen, da stört sein Stromverbrauch. Wenn der Mikrocontroller einen Komparator besitzt, kann man mithilfe einer externen RC-Beschaltung eine Aktivierungsschaltung für den Watchdog realisieren (*Abb. 1.3*). Man fügt die Bauelemente hinzu und programmiert eine Ansprechschwelle. Dann schließt man den Kondensator kurz und geht in den Sleep Mode. Wenn sich der Kondensator über R3 auf die Ansprechschwelle aufgeladen hat, schaltet der Komparator um und generiert ein Interrupt-Signal, welches das Programm aktiviert. Die Aufwach-Prozedur:

- Komparator-Minuseingang als digitalen Ausgang setzen
- 0 schreiben, um Kondensator zu entladen
- Komparatoreingang rücksetzen

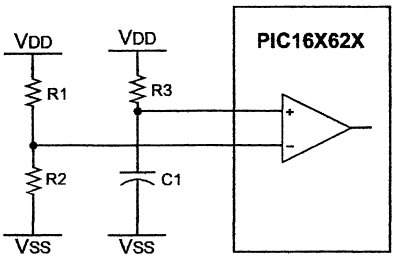

Abb. 1.3: Quelle: Dan Matthews, Basic PIC16/17 Oscillator Design

- Komparator-Zustandsflag lesen, um sie auf 0 zu setzen
- Komparatoreingang aktivieren
- Sleep-Zustand setzen

1.4 Mikrocontroller-Oszillator mit LED-Treiber

Ein Treiber für weiße LEDs besteht im Wesentlichen aus einem Oszillator, einer Ladungspumpe und einer Konstantstromquelle. Ein Beispiel ist der hochintegrierte Baustein LM2791/2. Wenn ein solcher IC im Zusammenhang mit einem Mikrocontroller genutzt werden soll, dann kann man in Erwägung ziehen, aus ihm den Takt für den Controller abzuleiten. Grundlage ist die Tatsache, dass über dem externen Kondensator für die Ladungspumpe eine Dreieck- bis Rechteckspannung auftritt. Wie in *Abb. 1.4* gezeigt, kann man hier über einen Widerstand Gatter anschließen, um den Takt für den Mikrocontroller zu erhöhen (Verdopplung) und mit genügend steilen Flanken anzubieten.

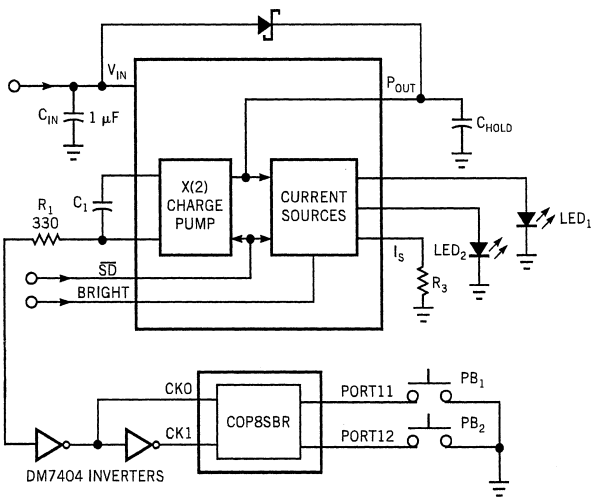

Abb. 1.4: Quelle: Wallace Ly, LED driver provides oscillator for microcontroller, EDN September 16, 2004

1.5 Mikrocontroller visualisiert Hex-Code

Die Schaltung nach *Abb. 1.5* zeigt die prinzipielle Anschaltung von Siebensegment-Anzeigen an einen Mikrocontroller. Auf diese Weise können beispielsweise auch Datenwörter im hexadezimalen System 1-2-4-8 in das Zehnersystem übersetzt und angezeigt werden. Das gelingt ohne weiteren Aufwand (Mikrocontroller) nur für die Ziffern 0...9. Der Mikrocontroller setzt hier den Hex-Code in den BCD-Code um, der auf die Treibereingänge A bis D gelangt. Die Einerstellen liegen an den Ausgängen A0 bis A3, die Zehnerstellen an A4 bis A7.

Das Programm ist unter www.ednmag.com über Search Databases im Software Center als Design Idea 2518 abrufbar.

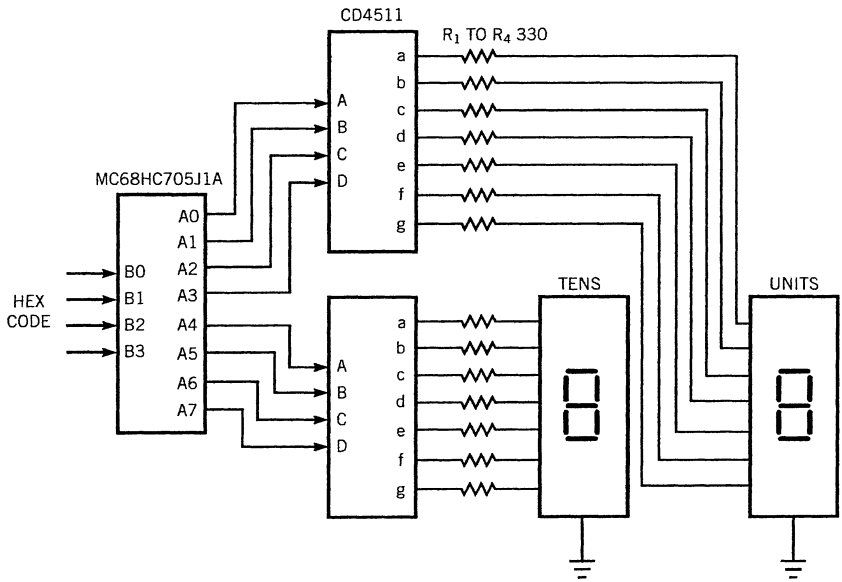

Abb. 1.5: Quelle: Abel Raynus, µC visualizes hex code, EDN April 27, 2000

1.6 Minimalbeschaltung als Basis-Interpreter

In *Abb. 1.6* ist die Minimalbeschaltung für einen sehr kleinen Basic-Interpreter zum begrenzten Debugging sowie für Anzeige und Kontrollzwecke (Monitoring und Control) zu sehen. Der Interpreter nutzt lediglich den On-Chip-RAM des Mikrocontrollers.
Die Software wird in verschiedenen Varianten bereitgestellt. Diese Firmware ist lauffähig auf einem Atmel ATTiny2313 mit 4, 8, 10, 16 oder 20 MHz Taktfrequenz oder auf einem AT90S2313 mit 4-MHz-Takt.

Ein mit diesem Minimal-Interface verbundener und mit dem Interpreter programmierter Chip arbeitet mit einem ASCII-Terminal zusammen. Das Schreiben und Ausführen kleiner Basic-Programme ist ebenso möglich wie das Hardware Debugging. Die Kommunikation mit dem Terminal ist via RS-232 (EIA-232) 9600 Baud, 8 Data Bits, 1 Stop Bit, no parity möglich. Pin 2 (RXD) und Pin 3 (TXD) sollten an ein invertierendes RS-232-Interface angeschlossen werden.

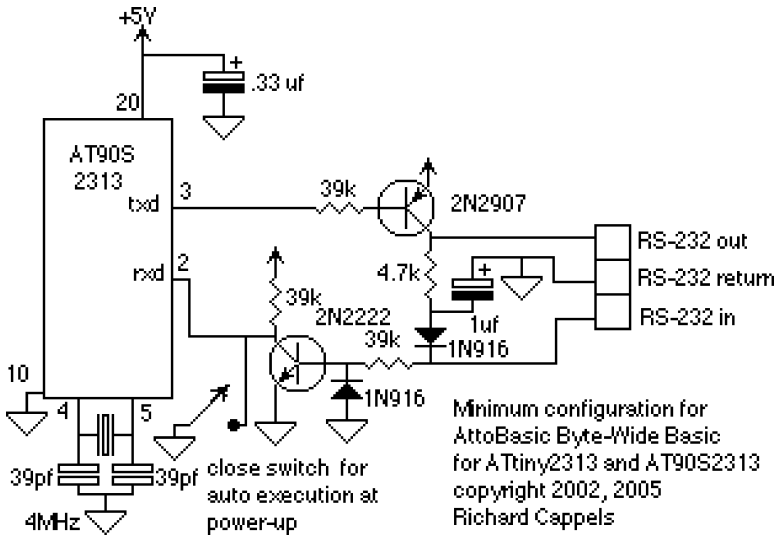

Abb. 1.6: Quelle: Dick Cappels, AttoBasic Byte-Wide Basic for ATTiny2313/AT90S2313, www.projects.cappels.org

1.7 PIC12F675 als Flipflop

Ein PIC wie beispielsweise der 12F675 kann als D- oder JK-Flipflop programmiert werden. Das entsprechende Programm nutzt die flankengetriggerten und statischen Interrupts. Es verwandelt den PIC in ein mit der fallenden Flanke getriggertes D-Flipflop mit L-aktiven Set- und Reset-Eingängen oder in ein oder JK-Flipflop (*Abb. 1.7*).

1.8 Logische Funktionen nachbilden

Beispielsweise mit dem Mikrocontroller 8085 kann man digitale Gatter nachbilden mit Funktionen wie NOT, OR oder AND. Das in *Abb. 1.8* skizzierte System hat einen Auswahlschalter für die Funktion. Jede logische Funktion ist im Prinzip möglich.

Das System hat zwei Eingänge und einen Ausgang und ist von daher einfachst gehalten.

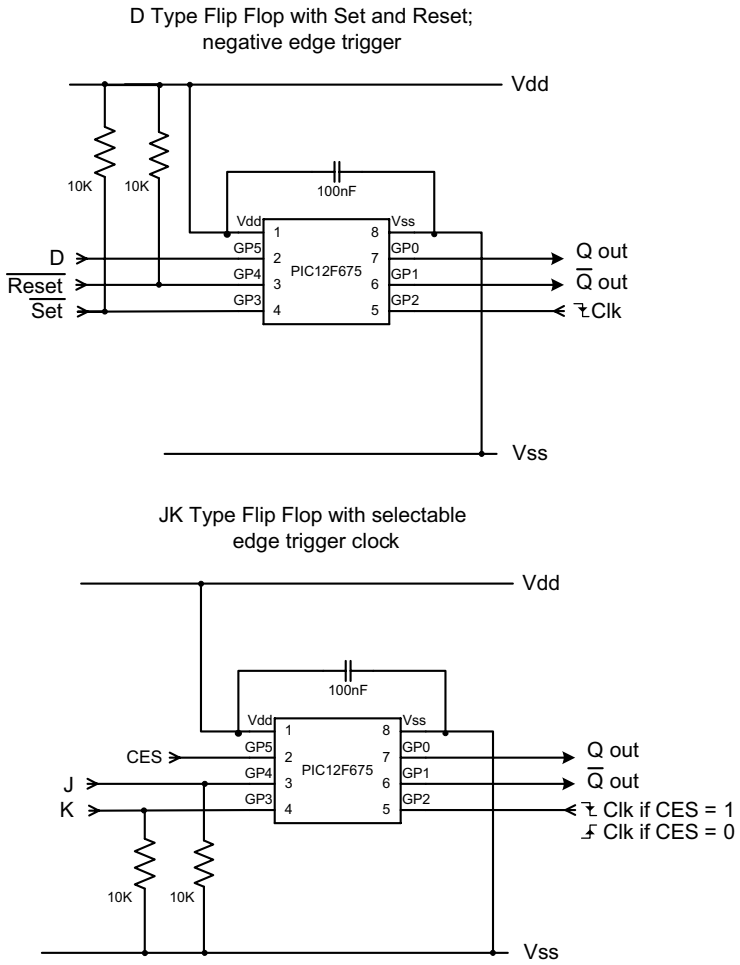

Abb. 1.7: Quelle: 12F675 as Flip Flop, http://picprojects.org.uk

Am Ausgang liegt zur Signalisierung des logischen Zustands eine LED. Weitere LEDs zeigen an, welche Grundfunktion vorliegt.

Der Mikrocontroller agiert mithilfe eines PPI (Programmable Peripheral Interface) vom Typ 8255.

1.9 Einfacher intelligenter A/D-Wandler

Der in *Abb. 1.9* dargestellte PICADC3 ist ein intelligenter 12-Bit-A/D-Konverter mit zusätzlichen digitalen Ein- und Ausgängen. Er basiert auf dem Microchip-Mikrocon-

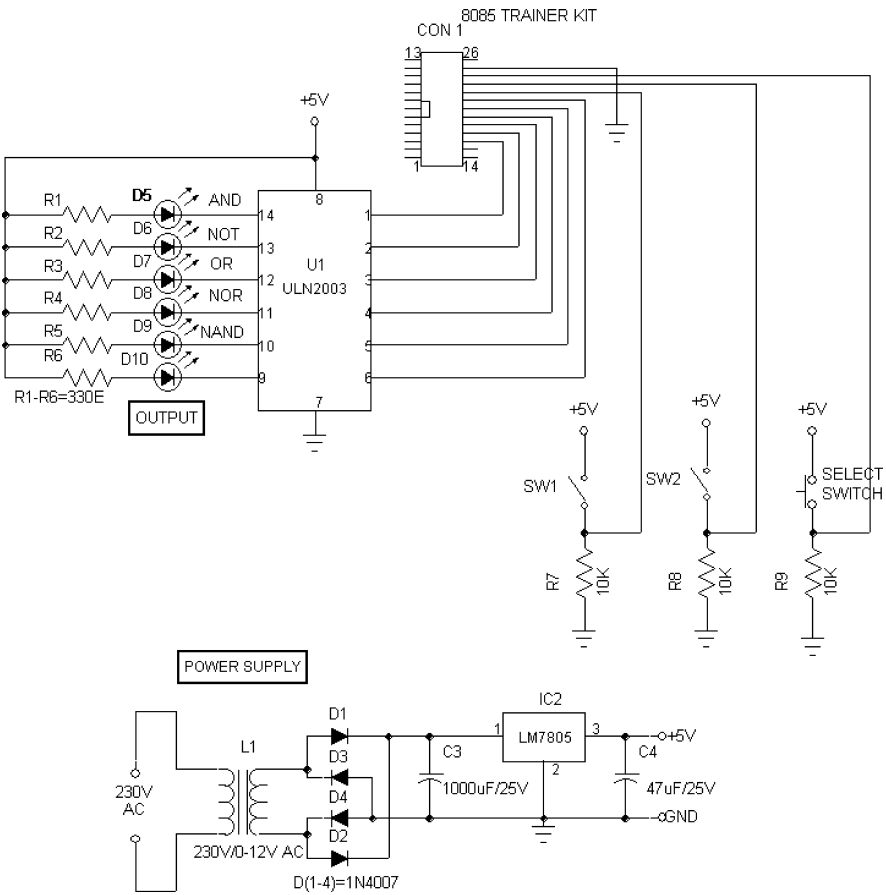

Abb. 1.8: Quelle: J. Binu, Digital Gates Emulator, www.8085projects.info

troller PIC16C774. Es ist möglich, die Firmware für den PIC16F877 mit 10 Bit zu modifizieren.

Der PICADC3 kann an den seriellen Port eines PCs oder unter Nutzung eines RS-232/USB-Konverters an den USB-Port eines PCs angeschlossen werden. Dies ermöglicht eine einfache Optoisolation.

Der PICADC3 benötigt relativ wenig Hardware. Die gezeigte Schaltung enthält nicht alle Signalverarbeitungs-Verstärker.

Die analogen Eingänge sind für maximal 4,096 V ausgelegt. Dabei ist ein niedriger Quellwiderstand erforderlich.

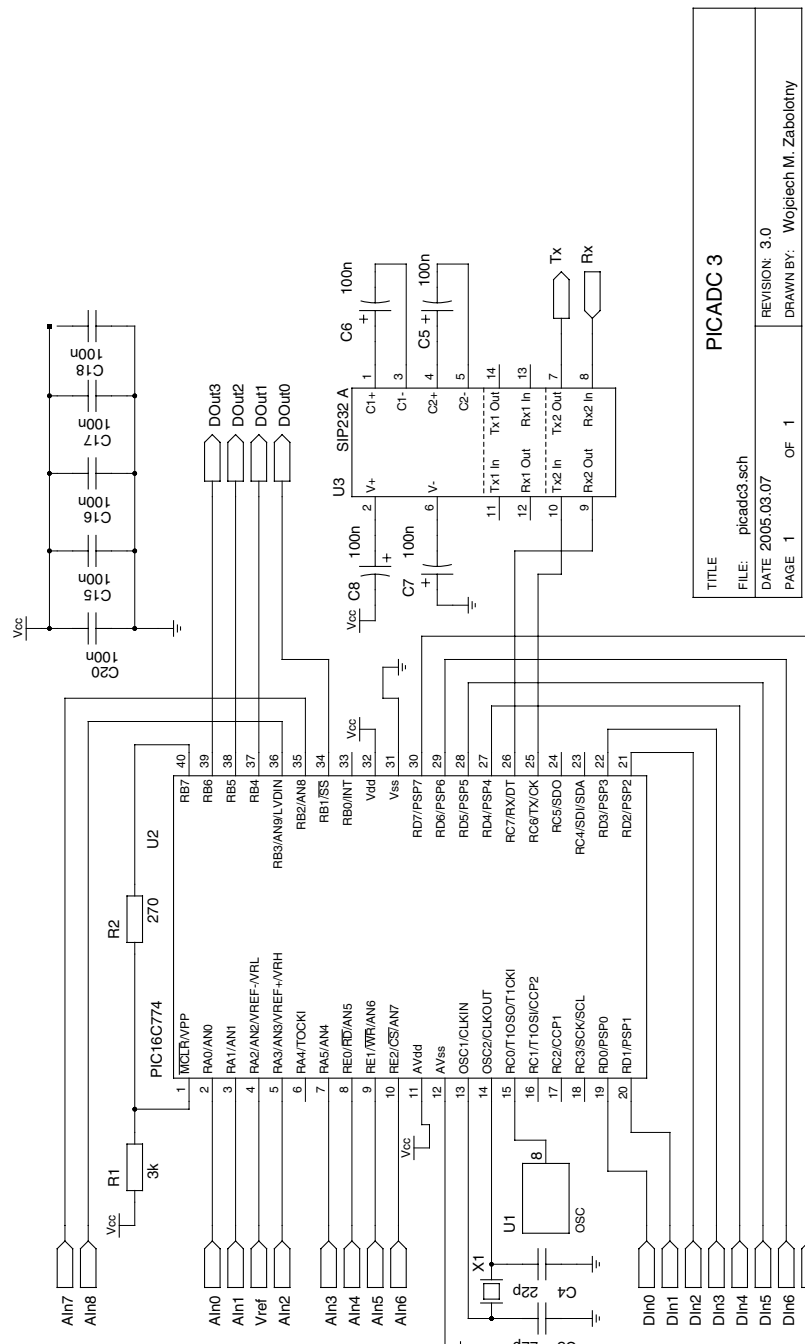

Abb. 1.9: Quelle: Wojciech Zabolotny, PICADC3 – simple intelligent A/D converter, www.ise.pw.edu.pl/

1.10 PIC in Grundbeschaltung

Die Schaltung nach *Abb. 1.10* eignet sich gut zum Einstieg in die Anwendung der PIC-Mikrocontroller. Man kann damit verschiedene Ideen ausprobieren. Benutzt wird der PIC16F84 mit Stabi-IC, Reset und Takt, zwei LEDs und einem Taster. Eine LED wird vom PIC angesteuert, die andere leuchtet beim Betätigen des Tasters. Es genügt ein keramischer Resonator 4 MHz als Quarz-Ersatz.

Der Applikationscode ist recht einfach und bewirkt das Leuchten der LEDs unter bestimmten Bedingungen.

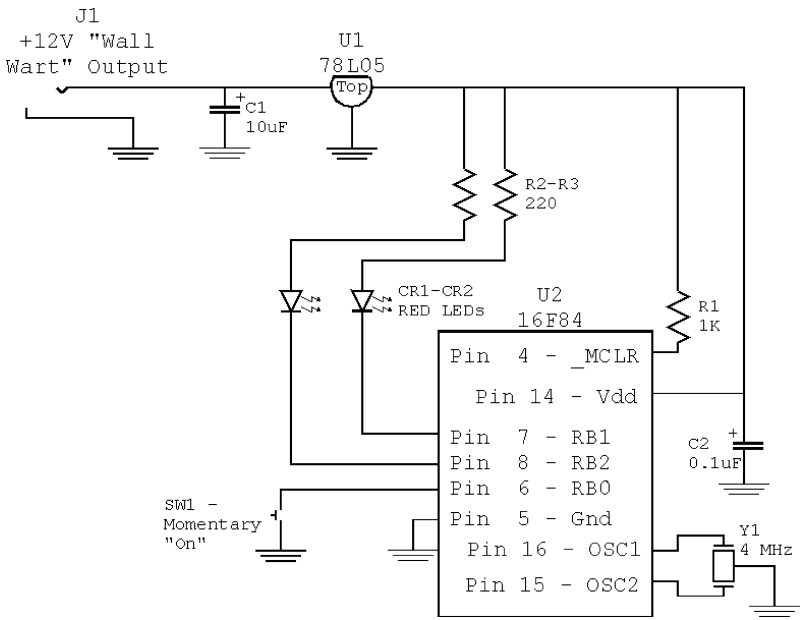

Abb. 1.10: Quelle: Myke Predko, My First PIC Project, www.rentron.com/Myke5.htm

1.11 A/D-Wandler MAX 186 an Mikrocontroller

Der MAX 186 ist ein 12-Bit-Analog/Digital-Wandler mit acht Kanälen. Er wird gemäß *Abb. 1.11* mit vier Leitungen mit einem Mikrocontroller wie der Basis Stamp verbunden.

Eine Einschränkung besteht oft in der maximal verarbeitbaren Eingangsspannung von 4,096 V. Hier hilft ein Spannungsteiler. Um den Einfluss des Eingangswiderstands des MAX 186 zu eliminieren, entwirft man diesen als unbelasteten Spannungsteiler und schaltet einen BiFET-Operationsverstärker als Impedanzwandler nach.

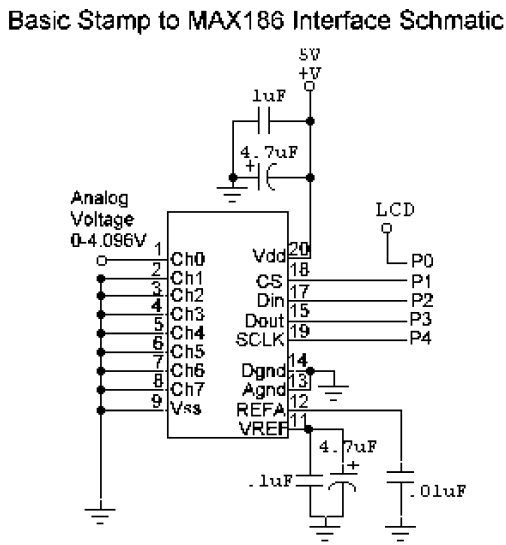

Abb. 1.11: Quelle: Wilson Shaun, MAX 186 to Basic Stamp, http://geocities.com/SiliconValley/Orchard/6633/hstamp.html

1.12 A/D-Wandler MAX 187 an Mikrocontroller

Für viele Aufgaben benötigt man einen einfachen Analog/Digital-Wandler. Der MAX 187 ist ein solcher mit 12 Bit Auflösung. Die Verbindung mit dem Mikrocontroller erfolgt mit nur drei Leitungen, denn ausgangsseitig ist der IC als SPI (Serial Peripheral Interface) ausgelegt – siehe *Abb. 1.12*. Die Leitungen sind Chip Select, Data Serial und Clock.

1.13 Mikrocontroller als LCD-Treiber

Ein LCD-Segment stellt elektrisch eine Kapazität dar. Das Segment wird sichtbar, wenn diese Kapazität auf einen bestimmten Mindestwert aufgeladen ist. Dies ist in *Abb. 1.13* grafisch dargestellt.

Ab diesem Schwellwert nimmt der Kontrast mit steigender Spannung zu, bis er 100 % erreicht hat (*Abb. 1.14*). Somit kann eine Kontrasteinstellung durch Einstellen der Steuerspannung erfolgen.

LCDs werden gemultiplext betrieben. Die Segmente bilden elektrisch eine Matrix (*Abb. 1.15*). Die gemeinsamen Anschlüsse gleicher Segmente COM werden nachein-

ander an eine Spannung gelegt (*Abb. 1.16*, COM1 bis COMn). Diese genügt noch nicht zum Einschalten. Soll ein Segment eingeschaltet werden, erhält der Anschluss SEG eine negative Spannung. Erst die am Segment wirksame Differenz der Spannungen (UD –-US = UD + US) überschreitet die On-Schwelle. Normalerweise sind UD und US betragsgleich. Da Differenzwerte von 5 V völlig genügen, nimmt man ein sogenanntes Level Shifting oder Biasing vor und halbiert die üblichen Betriebsspannungen von z. B. 5 V. In *Abb. 1.17* geschieht das durch die 1-kOhm-Widerstände. Ein PIC16C55 steuert hier ein vierstelliges LCD an. Jedes I/O-Pin steuert den Zustand von zwei Segmenten. Eine Stelle hat einschließlich Dezimalpunkt acht Segmente, daher sind 16 I/O-Pins eingebunden.

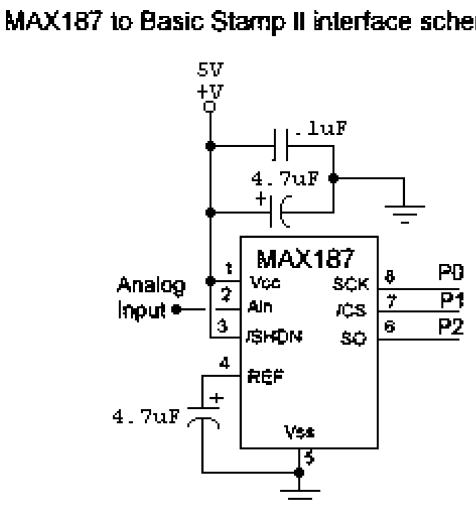

Abb. 1.12: Quelle: Wilson Shaun, MAX 187 to Basic Stamp, http://geocities.com/SiliconValley/Orchard/6633/hstamp.html

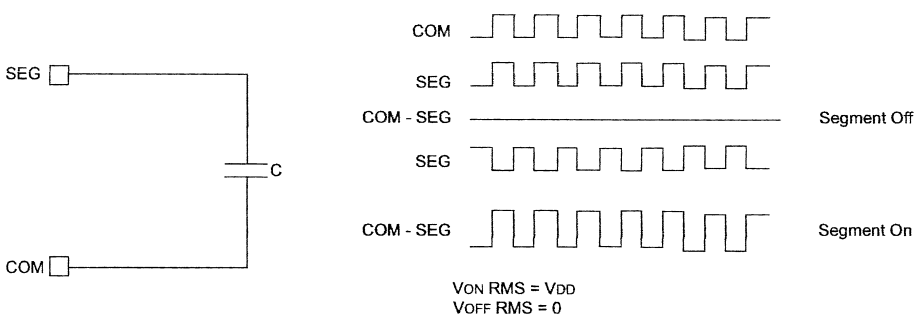

Abb. 1.13: Quelle: Al Lovrich, Using PIC16C5X Microcontrollers as LCD Drivers, Micrichip Application Note 563

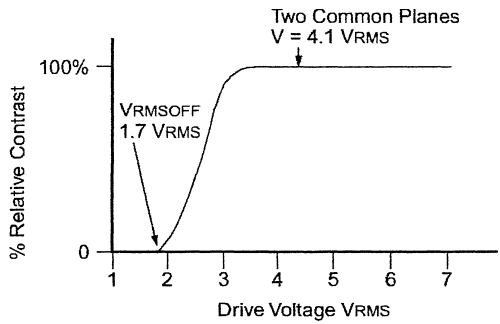

Abb. 1.14: Quelle: Al Lovrich, Using PIC16C5X Microcontrollers as LCD Drivers, Microchip Application Note 563

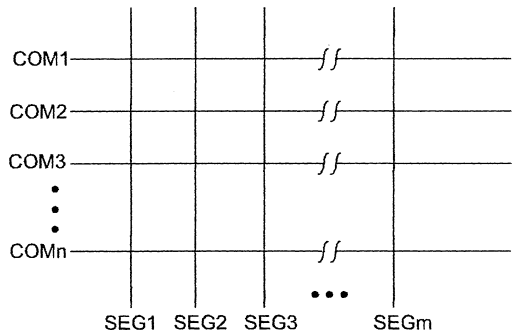

Abb. 1.15: Quelle: Al Lovrich, Using PIC16C5X Microcontrollers as LCD Drivers, Microchip Application Note 563

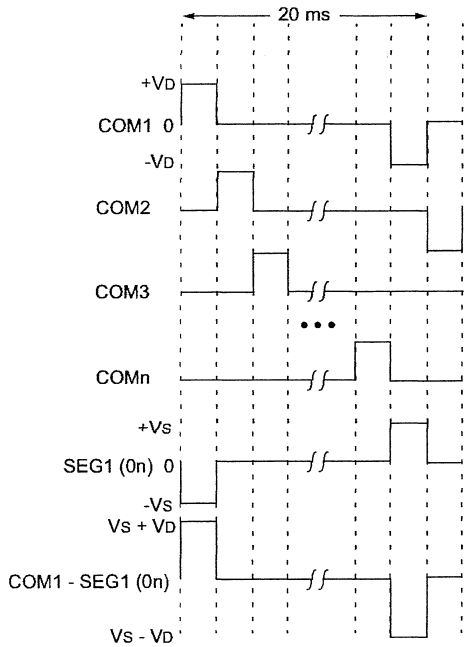

Abb. 1.16: Quelle: Al Lovrich, Using PIC16C5X Microcontrollers as LCD Drivers, Microchip Application Note 563

Heinz Schmid

Roboter mit Mikrocontrollern selbst bauen

Vorwort

Dieses Buch ist für den Einsteiger geschrieben und gibt einen Überblick über die Robotertechnik. Es befasst sich mit der geschichtlichen Entwicklung und den damit befassten Firmen. Auch auf die Mathematik wird kurz hingewiesen. Sie ist die Basis für diese Maschinen mit ihrer teilweise recht komplexen Steuerungs-Software. Auch auf die Wissenschaftler, die die Grundsteine für diese Konzepte legten, wird eingegangen.

Vor allem aber werden die unterschiedlichen Wege aufgezeigt, Zugang zu diesem Bereich zu bekommen – unterschiedliche Wissensniveaus sowie handwerkliche, finanzielle und zeitliche Möglichkeiten werden berücksichtigt.

Das Buch bietet jenen Neues, die sich mit dem Themenkreis *Roboter* bereits auseinandersetzen, und ist dennoch für den Einsteiger eine Hilfe.

Roboter sind Bestandteil unserer Zukunft. Aus dem Industrie-, Spielzeug- und Ausbildungsbereich sind sie längst nicht mehr wegzudenken, im Alltag steht die Entwicklung aber erst am Anfang. In welcher Art und Weise und für welche Verwendungszwecke sie Einzug halten werden, wird von uns auch als Konsumenten mitbestimmt werden.

Faszinierend, lehrreich und interessant zugleich ist dieses Gebiet allemal – und zwar für Alt und Jung!

Ich freue mich über konstruktive Rückmeldungen, Verbesserungsvorschläge und neue Ideen via E-Mail an: *robots@gmx.net*

Heinz Schmid

Eferding, August 2010

Inhaltsverzeichnis

2 Grundlagen der Robotik

Die Robotik ist die Verschmelzung von Mechanik, Elektronik und Informatik. Für diesen neuen Technikbereich gibt es schon seit einiger Zeit das von den Japanern geprägte Kunstwort *Mechatronik*. Aber die Mechatronik umfasst mehr als die Robotik allein und ist für die meisten in der Praxis stehenden Ingenieure mehr Prozess- und Anlagenautomatisierung als reine Robotik.

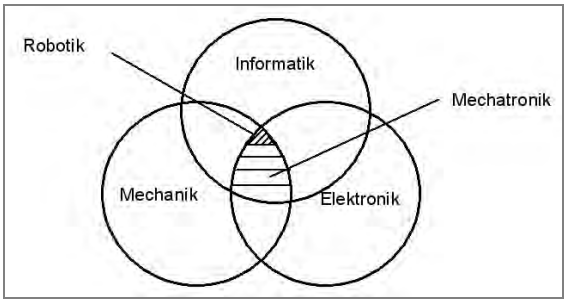

Abb. 2.1: Schnittmenge Mechatronik – Untermenge Robotik

Aus der alten, wenn auch nicht veralteten Ingenieursdisziplin *Maschinenbau*, die früher noch die Unterdisziplin *Elektromaschinenbau* umfasste, ist längst der Bereich *Elektrotechnik* und daraus wiederum die *Elektronik* erwachsen. All diese Disziplinen sind natürlich ohne Physik und Mathematik nicht denkbar.

Die neue Disziplin der Informatik ist hingegen ein Kind der Mathematik. In ihrer reinen Form ist sie nur über die physikalischen Grenzen der Computer an die Physik gebunden.

Durch die neuen Möglichkeiten und die hohe Leistungsfähigkeit der eingesetzten Mikrocomputer und Mikrocontroller ist die Robotik heute auch für Biologen, Soziologen und Neurowissenschaftler ein interessantes Gebiet. Vieles, was die besondere Faszination der Robotik ausmacht, ergibt sich erst durch Anregung, Übernahme und Nachbau der belebten Natur.

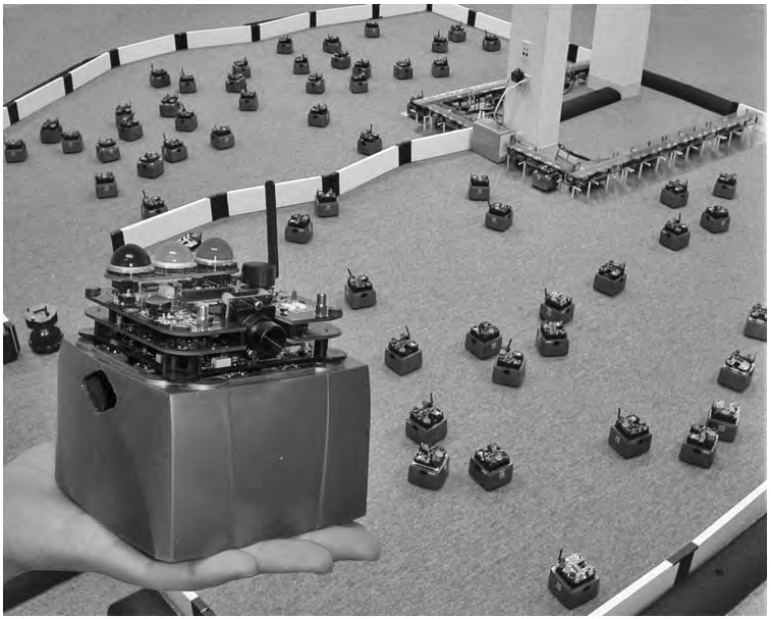

Abb. 2.2: ©MIT Roboterschwarm

Schwarmverhalten, Schwarmintelligenz, der vereinfachte Nachbau der neuronalen Struktur des Gehirns mit elektronischen Bauelementen und deren Simulation am Computer dient derzeit vor allem noch der Mustererkennung. Das gilt auch für die Nachahmung der Bewegungen von Insekten, Fischen und Säugetieren bis hin zum aufrechten Gang des Menschen.

Vom rein Handwerklichen her steht die Robotik natürlich in der Tradition der Feinmechanik und Uhrmacherkunst: Holzbearbeitung, Metallbearbeitung, Kunststoffverarbeitung, Platinenfertigung, Löten, Messen, Prüfen usw. Die Liste der handwerklichen Tätigkeiten ist lang, die der notwendigen Werkzeuge auch.

Simulation am Computer

- reine Robotersimulationen
- quasi Informatik pur, ohne mechanische und elektronische Probleme
- keine leeren Roboterakkus

In ihrem physikalischen Verhalten basiert sie auf den Gesetzen der Mechanik und wird mit mathematischen Gleichungen modelliert, wie z. B. bei einer Roboterfußball-Simulation in der folgenden Abbildung.

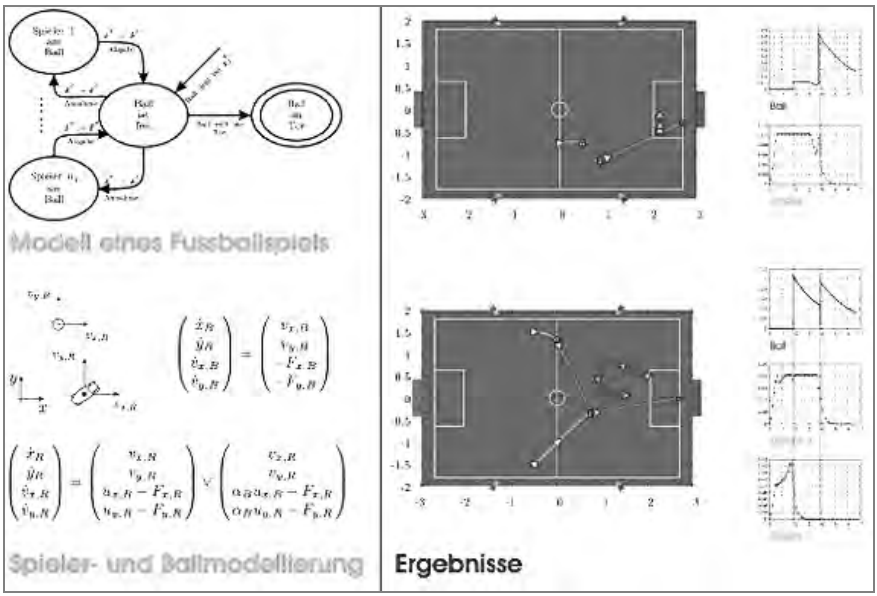

Abb. 2.3: Roboterfußball-Simulation ©TU Darmstadt

Baukästen

Bei den Baukästen ist nur wenig handwerkliches Geschick vonnöten. Es müssen keine Elektronikkenntnisse vorhanden sein. Damit sind sie ein guter zeit-, kosten- und nervensparender Einstieg, nicht nur für Kinder und Jugendliche. Sie werden auch in Fachhochschullabors verwendet. Eigentlich sind sie kein Kinderspielzeug, machen aber Spaß wie eines. Sie sind auch als Einstieg in das Programmieren geeignet und viel spannender, da sich nicht nur auf einem Bildschirm etwas ändert, sondern ein Roboter sich in Bewegung setzt.

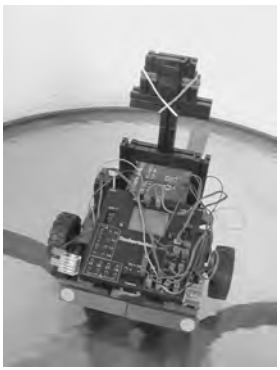

Abb. 2.4 und 2.5: Fischertechnik *ROBO TXC* und *Lego NXT*

Beide in den Abbildungen gezeigten Bausätze können mit einer 2,4-GHz-Micro-Funkkamera erweitert werden, auf die im Kapitel »Bildbearbeitung« noch genauer eingegangen wird. Der NXT verfügt hier zusätzlich über einen Kompasssensor und ein Thermoelement.

Bausätze

Ohne handwerkliche Grundfertigkeiten und Werkzeug wie Schraubenzieher, Seitenschneider, Lötkolben und eine gute Portion Geduld, Geschick und Sorgfalt geht bei Bausätzen nichts. Richtig spannend wird es, wenn es gar nicht oder nicht einwandfrei funktioniert. Die Fehlersuche kann dann beginnen. Ohne Grundkenntnisse der Elektronik und im Messen (Stichwort Multimeter) kommt man schnell an seine Grenzen. Bausätze sind auch als Einstieg in Hardware-nahes Programmieren gut geeignet. Für jüngere Kinder sind sie nur sehr bedingt zu empfehlen.

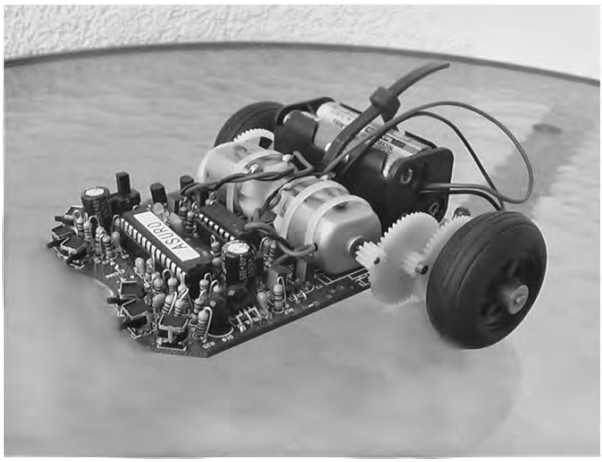

Abb. 2.6: ASURO

(Bezug als Bausatz z. B. über *www.conrad.de*)

Baupläne – Bauanleitungen

Arbeitet man mit Bauanleitungen, müssen alle benötigten Bauteile im Handel besorgt werden. Das erfordert meist das Wälzen dicker Bauteilkataloge der einzelnen Anbieter. Platinen müssen eventuell anhand der Schaltpläne erst selbst entworfen oder die fertigen Layouts auf die Platine übertragen und geätzt werden. Für einfache Schaltungen ist es meist einfacher, sie auf ein Experimentierboard zu stecken oder fix zu verkabeln. Die weitere Vorgangsweise entspricht den Bausätzen.

Die Software ist als Quellcode inklusive der notwendigen Programme meist als CD oder Internetdownload vorhanden. Im weniger günstigen Fall muss er aus einem Buch abgeschrieben werden – in der Hoffnung, dass sich keine Druckfehler eingeschlichen haben. Die Abbildung zeigt einen BEAM-Roboter noch ganz ohne Mikrocontroller und Programmierung.

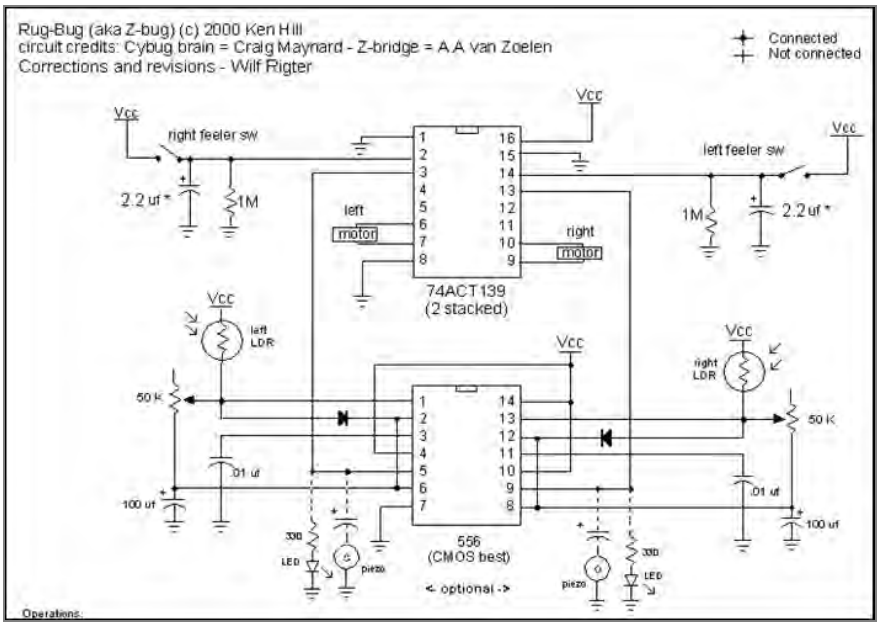

Abb. 2.7: BEAM – Schaltplan ©Ken Hill (Details unter www.acesim.com/beam/rugbug.html)

Königsklasse

Der eigene Hardware-Entwurf ist für Nicht-Elektroniker nur schwer zu verwirklichen. In der Königsklasse schreibt man sich dann natürlich auch die Software selbst und rundet dies mit einer zuverlässigen Mechanik ab. Dies ist für komplexe Roboterkreationen nur im Team und mit Arbeitsteilung zu schaffen. Ohne das geballte Wissen der Robotic Community über das Internet, Bücher und Zeitschriften geht es sowieso nicht.

Wie mag wohl der Schaltplan zur Abbildung aussehen?

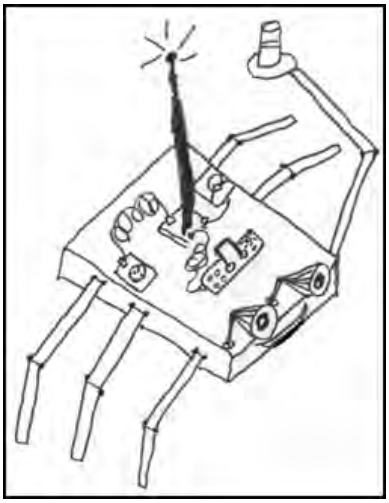

Abb. 2.8: Entwurf von Felix

2.1 Mechanik, Energieversorgung und Aktoren

Das Einfachste ist natürlich, einfach »drauflos zu bauen«: ein Motor, ein Getriebe ein Akku und schon bewegt sich etwas. Wie lange, wie gut und wie effektiv – man wird es sehen. Der sicherere Weg ist allerdings, vorher ein wenig zu skizzieren und zu rechnen. Deswegen werden wir uns zumindest einige der wichtigsten Grundgleichungen aus der Mechanik ansehen, also jenem Gebiet der Physik, das den Technikern zur Anwendung überlassen wurde.

Wir befassen uns mit den Kräften, damit wir Körper verschieben können, und mit den Momenten, damit eine Drehbewegung entsteht.

Natürlich benötigen unsere Roboter auch Energie. Sie muss aus einer anderen Energieform heraus in mechanische Energie und Bewegungsenergie der Maschine umgewandelt werden. Energie kann weder erzeugt noch vernichtet, sondern nur umgewandelt werden.

Bei jeder dieser Umwandlungen gibt es Verluste. Je länger die Umwandlungskette ist, desto mehr Verluste entstehen. Eine autonome Energieversorgung für autonome Roboter wäre eine feine Sache. Aber auch hier sollte man vorher berechnen, welche Solarflächen benötigt werden und was der Roboter wiegt. Akkus werden auf alle Fälle gebraucht, um die Energieversorgung flexibler zu gestalten.

Damit die Motoren zum Getriebe und beide zur gewünschten Geschwindigkeit des Roboters passen, muss wieder gerechnet werden. Bei den Greifern geht's um Reibung, Anpresskräfte und Hebelarme. Laut Schnittmengenbild benötigen wir diese Grundkenntnisse der Mechanik für den Roboterbau. Es werden hier aber

nur einige wichtige Grundbegriffe der Mechanik wiederholt. Eine wirkliche Einführung in die Statik und die Dynamik würde allein schon ein ganzes Buch umfassen.

2.1.1 Kräfte und Momente

Eine Kraft ist eine gerichtete Größe – mathematisch also ein Vektor mit Betrag und Richtung. Ihre Einheit ist N und sie wird als Gerade mit Pfeil dargestellt. Der Pfeil in der Abbildung gibt also die Richtung der Kraft an, die Länge der Geraden steht für den Betrag der Kraft. Je länger die Gerade, desto größer ist die Kraft. Um sie grafisch darstellen zu können, braucht man einen Kraftmaßstab.

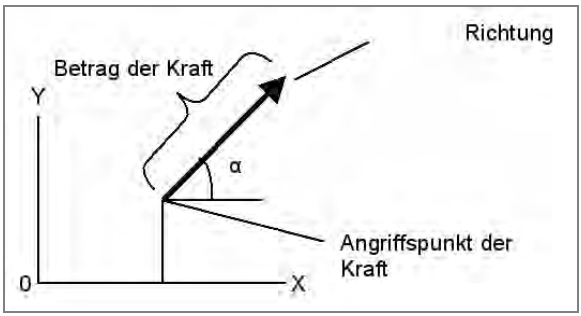

Abb. 2.9: Darstellung einer Kraft in der Ebene im sogenannten Lageplan

Diese Darstellung von Kräften ist natürlich das Ergebnis eines Abstraktionsprozesses. In der Realität sind die Kräfte unsichtbar. Nur ihre Wirkungen sind für uns sichtbar. Gegenstände verformen sich unter dem Einfluss von Kräften, Körper werden beschleunigt oder verzögert. Sie trotzen der Wirkung der Schwerkraft, indem sie durch Reibung gehalten werden, oder sie rutschen ab, weil die Größe der Reibung nicht ausreicht.

Für uns wichtig sind die Gewichtskräfte, die Reibungskräfte und die Beschleunigungskräfte sowie die Lage ihres Angriffspunkts.

Gewichtskraft

Eine Masse ruft im Schwerkraftfeld der Erde eine Gewichtskraft hervor. Auf Mond und Mars hätte diese Gewichtskraft jeweils einen anderen Betrag. Damit ist klar, dass Kraft F N und Masse m kg nicht dasselbe sein können, aber zueinander proportional sind. Es gilt:

$$F_G = m\,g$$

Die Erdbeschleunigung g = 9,81 m/s² ist also der Multiplikator für die Umrechnung der Masse in eine Kraft.

Beschleunigungskraft

Wenn wir einen Körper beschleunigen – aus dem Stillstand oder indem wir seine bestehende Geschwindigkeit v m/s erhöhen –, benötigen wir dazu eine Kraft. Die Beschleunigungskraft der Körper reagiert auf diese Beschleunigung *a m/s²* mit einer Gegenkraft, der Trägheitskraft, die in der Abbildung der Einfachheit halber nicht eingezeichnet wurde. Es gilt:

$$F_B = m\,a$$

Über einen starken Motor erzeugen wir über den Antriebsstrang die Beschleunigungskraft. Ob wir diese allerdings tatsächlich auf die Straße bringen, entscheidet die Reibungskraft.

Reibungskraft

Für die Reibungskraft gilt das Coulombsche Reibungsgesetz:

$$F_R = F_N\,\mu$$

Die Normalkraft F_N *N* (Anpresskraft des Untergrunds) der Reibungskoeffizient μ für die Gleitreibung und μ_0 für die Haftreibung (beide von der Werkstoffkombination Räder – Untergrund abhängig) bestimmen die Größe der Reibungskraft. Haften die Räder am Untergrund, liegt *Haftreibung* vor und wir können eine maximale Kraft übertragen. Rutschen die Räder bereits durch, liegt *Gleitreibung* vor und die maximal übertragbare Kraft ist bereits um einiges kleiner. Es gilt:

$$\mu_0 > \mu$$

In unsere Abbildung sind zwei Reibungskräfte eingezeichnet, d. h. wir hätten einen Allradantrieb. Die Normalkräfte sowie alle anderen wirkenden Kräfte sind hier wieder nicht eingezeichnet worden. Die Reibkräfte am bewegten Körper wirken immer entgegen der Bewegungsrichtung. Bezogen auf den Untergrund wäre das natürlich genau entgegengesetzt.

Der Haftreibungskoeffizent für Luftgummireifen auf trockenem Asphalt liegt bei neuen Reifen bei ≈ 0,85. Für gebrauchte Reifen liegt er bei ≈ 1. Für nassen Asphalt gilt neu 0,65 oder gebraucht 0,5. Ganz klein wird der Wert bei Glatteis, dann liegt er bei ≈ 0,1. Mit speziellen ganz weichen Gummimischungen sind auch Werte von ≈ 1,8 erreichbar.

Der Fahrwiderstandskoeffizient von Fahrzeugen ohne Luftwiderstand, also nur die Lagerreibung und der Rollwiderstand, liegt bei ca. $\mu_w \approx 0{,}025$. Für den Fahrwiderstand ohne Luftwiderstand gilt:

$$F_{RW} = F_N\,\mu_W$$

Damit können wir später die Antriebskraft und damit den erforderlichen Motor abschätzen.

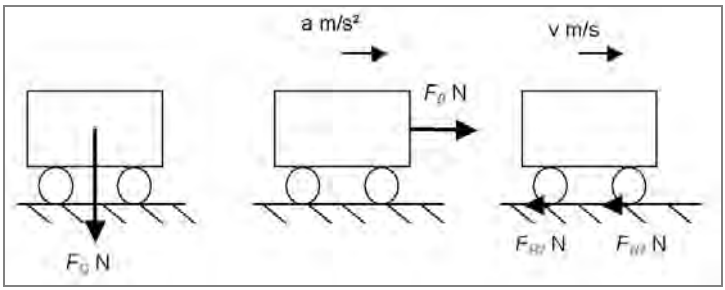

Abb. 2.10: Gewichtskraft, Beschleunigungskraft und Reibungskräfte

Außer den Kräften *N* gibt es in der Statik und Dynamik noch die Momente *Nm*. Vor allem bei den zweibeinigen Laufmaschinen sind diese für die Stabilität der Bewegung ausschlaggebend. Wichtig ist dort die Lage des sogenannten ZMP-Punkts. Bezogen auf diesem *zero moment point* sind alle statischen und dynamischen, also durch die Trägheit hervorgerufenen Momente, gleich Null. Bei langsamem, statischem Gehen muss die senkrechte Projektion dieses Punkts immer innerhalb der Fußsohle der Gehmaschine liegen. Beim dynamischen Gehen liegt dieser Punkt vor der Fußsohle der Maschine. Sie fällt quasi nach vorn in Richtung dieses Punkts.

Die Lage des ZMP ändert sich durch die Bewegung der Maschine laufend. Jede Bewegung des Kopfs, eines Arms oder der Aufprall eines Balls ändern laufend die räumliche Lage des ZMP. Die Regelmechanismen der Gehmaschine müssen in Echtzeit darauf reagieren.

Wenn zwei gleich große, entgegengesetzt wirkende Kräfte nicht auf einer Geraden liegen, entsteht ein Moment *M Nm*. Es ist umso größer, je größer die Kräfte *F N* sind und je weiter sie voneinander entfernt liegen – also je größer ihr Normalabstand oder Hebelsarm x_N m ist.

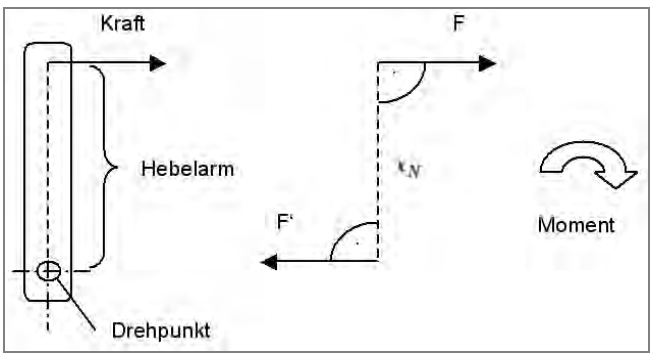

Abb. 2.11: Hebel-Kraftpaar-Moment

Es gilt:

$$M = F\ x_N$$

Die Momente haben dabei auch eine Drehrichtung. Wenn wir nur den zweidimensionalen Fall der x-y-Ebene ansehen, können die Momente im Uhrzeigersinn oder gegen den Uhrzeigersinn drehen. Es ist üblich, im ersten Fall ein negatives und im zweiten Fall dann ein positives Vorzeichen zu vergeben. Wenn sich alle Momente, die auf einen Körper (bezogen auf einen frei wählbaren Bezugspunkt) wirken, gegenseitig aufheben (also zu null werden), kann auch keine Drehbewegung des Körpers erfolgen.

$$\sum M_i = 0$$

Damit kann man auch die Stabilität von Maschinen untersuchen. Momente, die eine Maschine zum Kippen bringen können, werden unter der Bezeichnung *Kippmoment* zusammengefasst, jene, die die Maschine stabilisieren, unter *Standmoment*. Der Bezugspunkt ist aber nicht frei wählbar, sondern der Punkt, um den das System kippen könnte.

$$M_{Kipp} = \sum M_{i,Kipp} \qquad M_{Stand} = \sum M_{i,Stand}$$

Damit das System stabil bleibt, müssen also die Standmomente größer sein als die Kippmomente. Je mehr sie überwiegen, desto stabiler und damit sicherer ist das System.

$$S = \frac{M_{Kipp}}{M_{Stand}} > 1$$

Der Wert *1* ist dabei schon kritisch – ein kleiner Windhauch, und es könnte umfallen.

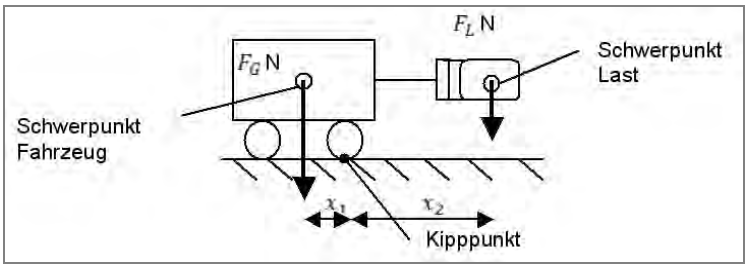

Abb. 2.12: Fahrzeug mit Greifer und Last

Wie sofort ersichtlich ist, gilt:

$$M_{Kipp} = F_L\ x_2 \qquad M_{Stand} = F_G\ x_1$$

Damit können wir also vorab die Stabilität unserer Konstruktion ermitteln.

Unser nächstes Beispiel ist ein Fahrzeug mit Allradantrieb. Bei einem Fahrzeug mit gegebenem Gewicht können wir für einen Allradantrieb mit gegebenem Haftreibungskoeffizienten und Fahrreibungskoeffizienten μ_w die maximal mögliche Beschleunigungskraft ermitteln. Damit könnten wir dann in weiterer Folge den Antrieb dimensionieren.

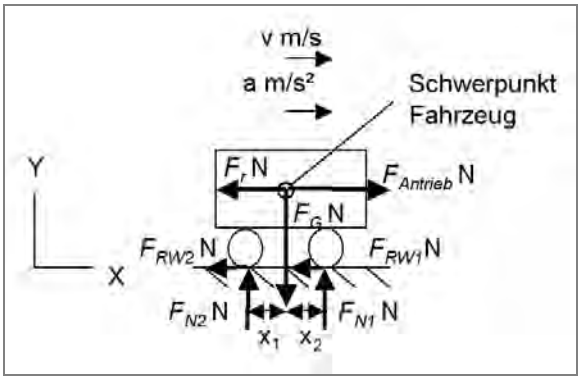

Abb. 2.13: Allradantrieb-Beschleunigung auf einer horizontalen Ebene

Hier ist die Trägheitskraft des Körpers eingezeichnet, um das dynamische Problem auf ein statisches zu reduzieren. In der Statik gilt für das Gleichgewicht immer, dass die Summe aller Kräfte in einer Richtung, also x oder y, gleich Null ist.

In der Schreibweise der Mathematik gilt also:

$$\sum F_x = 0$$

und für die senkrechte Richtung in der eingezeichneten Ebene:

$$\sum F_y = 0$$

Damit folgt für unsere Problemstellung:

$$F_{Antrieb} = F_T + F_{RW1} + F_{RW2} = m\,a + (F_{N2} + F_{N1})\,\mu_W$$

In y-Richtung liegt hier ohnedies nur ein statisches Problem vor, es gilt:

$$F_G = F_{N1} + F_{N2}$$

Damit könnten wir bereits die Zahlenwerte in $F_{Antrieb} = m\,(a + g\,\mu_W)$ einsetzen.

In diesem Fall hätte als Skizze auch ein einfaches Punktmasse-Modell gereicht. Man denkt sich die gesamte Masse im Schwerpunkt vereint und zeichnet dort alle

Kräfte ein. Beim Einzeichnen der Antriebskraft waren wir etwas ungenau. Sie wirkt bei einem Allradantrieb verteilt auf die zwei Räder und liegt auf derselben Wirkungslinie wie die beiden Reibungskräfte. Es gilt also noch zusätzlich die Bedingung gegen das Durchrutschen

$$F_{Ro} \geq \frac{F_{Antrieb}}{2} \text{ für } x_2 = x_1$$

d. h. gleiche Achslasten für die Hinter- und Vorderachse. So, wie wir das gezeichnet haben, könnte die Antriebskraft aber der Rückstoß eines im Schwerpunkt wirkenden Raketenantriebs sein. Es würde dann der Impulssatz gelten und die Haftreibungskräfte wären für die Beschleunigung nicht mehr das limitierende Element. Nur abheben sollte man mit einem solchen Auto ohne Flügel nicht.

Bei einem Hinterradantrieb wäre es dann günstig, den Schwerpunkt nach hinten zu verschieben, um die Normalkraft auf die Antriebsache und damit die maximal übertragbare Reibungskraft zu erhöhen. Aber tückischerweise erhöht diese Konstruktionsweise, wenn man sie auf die Spitze treibt, das Kippmoment. Es wird von der gegen den Uhrzeigersinn drehenden Trägheitskraft besonders bei extremen Beschleunigungen hervorgerufen. Man kennt diese Bilder von den Dragster-Rennen und übermütigen Motorradfahrern.

Wir sehen also, dass mechanische Überlegungen durchaus sinnvoll sind und oft auch spannend oder im Extremfall sogar lebensrettend sein können.

Ein mechanisches Grundbeispiel ist das Fahrzeug mit Hinterradantrieb, das auf einer schiefen Ebene mit konstanter Geschwindigkeit unterwegs ist:

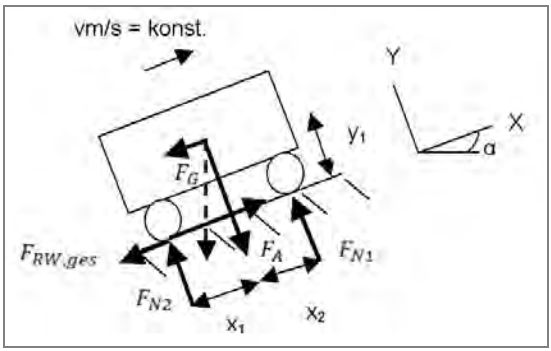

Abb. 2.14: Fahrzeug auf der schiefen Ebene

Um die Gleichgewichtsbedingungen in x- und y-Richtung aufstellen zu können, muss die Gewichtskraft zuerst in diese beiden Richtungen zerlegt werden. Man erhält mit dem Neigungswinkel α der schiefen Ebene

$$F_A = F_{RW,ges} + F_G \sin\alpha$$

und in y-Richtung

$$F_G \cos\alpha = F_{N2} + F_{N1}$$

In diesem Fall brauchen wir dann noch eine Gleichung für das Momentengleichgewicht. Diese setzen wir am besten um den Kontaktpunkt des Hinterrades an, um damit F_{N1} zu berechnen. Es folgt:

$$F_{N1} = \frac{F_{GX} y_1 - F_{Gy} x_1}{x_1 + x_2}$$

Dieser Kontaktpunkt ist auch gleichzeitig der Kipppunkt unseres Fahrzeugs. Damit können wir den maximalen Steigungswinkel für das Fahrzeug bei konstanter Geschwindigkeit ermitteln. Im Fall einer Beschleunigung ist er natürlich entsprechend kleiner.

Außerdem gilt noch die Durchrutschbedingung

$$F_{Ro,max} = F_{N2}\mu_0 \geq F_{A,max}$$

beim Hinterradantrieb.

Als Ergebnis der Berechnungen erhält man dann:

$$\alpha_{max} = \frac{x_1}{\tan \propto y_1} \text{ und } F_A = F_G(\sin \propto + \mu_W \cos\alpha),$$

Dabei muss noch die Durchrutschbedingung berücksichtigt werden. Für einen Kettenantrieb gelten diese beiden Gleichungen ebenfalls. Es gilt dann ca. $\mu_w \approx 0{,}3$ und die Durchrutschbedingung wird zu:

$$F_{Ro,max} = F_G \cos \propto \mu_0 \geq F_{A,max}$$

D. h., die Normalkraft und damit die maximale Haftreibungskraft sind wieder größer.

2.1.2 Energie – Leistung – Wirkungsgrad

Das Thema *Energie* ist eines der heißen Themen des 21. Jahrhunderts. Unsere bisherigen Energieumwandlungssysteme können in der jetzigen Form wegen der Begrenztheit der fossilen Brennstoffvorräte, der Treibhausproblematik und der Bevölkerungsexplosion nicht fortbestehen. Es ist also sinnvoll und leicht zu realisieren, über die Photovoltaik gleich eine weitgehende Unabhängigkeit unserer Roboter vom alten Energiesystem anzustreben.

Zurück zu den Grundlagen der Physik:

- Energie *E J* ist Arbeitsvermögen W *J*

  $$E = W_{Nutz}$$

- Kraft *F N* mal Weg *s m* (wenn beide in die gleiche Richtung weisen)

$$W_{Nutz} = F \cdot s$$

- Moment *M Nm* mal Winkel φ *rad* (wenn beide in die gleiche Drehrichtung weisen)

$$W_{Nutz} = M \cdot \varphi$$

und wird in Joule *J* gemessen.

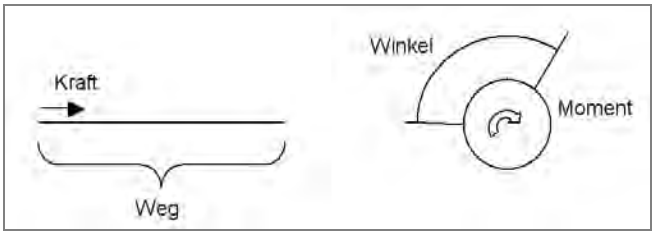

Abb. 2.15: Kraft mal Weg und Moment mal Winkel

Leistung hingegen ist Energie pro Zeiteinheit als Watt. Wer also die gleiche Arbeit in der kürzeren Zeit schafft, hat nach physikalischer Definition mehr geleistet.

$$\mathrm{P} = \frac{\mathrm{E}}{\mathrm{t}}$$

Als Einheitengleichung angeschrieben gilt W = J / s.

Für die mechanische Leistung eines Motors oder Getriebes gilt also

$$P_{mech} = M_t \omega$$

M_t ist das Drehmoment in *Nm* und Ω ist die Winkelgeschwindigkeit in *rad/s*, also dem Winkel, der pro Zeiteinheit überstrichen wird. Wir werden die Winkelgeschwindigkeit dann im Kapitel über die Motoren aus der Drehzahl errechnen.

- Für die elektrische Leistung des Gleichstroms gilt $P_{el} = UI$ mit den Einheiten *W = VA*, der Spannung *U* in Volt *V* und der Stromstärke *I* in Ampere *A*. Die Stromstärke entspricht der Ladung pro Zeiteinheit, also Coulomb pro Sekunde *A = C/s*. (Für die elektrische Energie gilt damit *J = VC*.)

Mehr Leistung bedeutet auch stärkere Motoren und damit auch mehr Akkukapazität und Gewicht oder kürzere Betriebszeiten. Dies ist für eine optimale Roboterkonstruktion meist der falsche Denkansatz. Bei halber Masse benötigt man auch nur die halbe Kraft, um auf die gleiche Beschleunigung zu kommen.

Deshalb versucht die Bionik, aus dem fantastisch optimierten Fundus der Natur und ihrem Artenreichtum zu lernen.

Herbert Bernstein

Mikrocontroller in der Elektronik

Mikrocontroller programmieren und in der Praxis einsetzen

Vorwort

ATMEL hat mehrere leistungsfähige 8-Bit- und 32-Bit-Mikrocontroller in seinem Programm. Für das Werkbuch der Mikrocontroller wurden der ATtiny2313 und der ATtiny26 verwendet. Beide Mikrocontroller verwenden für die Speicherung des Programms einen 2-Kbyte-Flash-Speicher, der über eine „In-System-Programmierung“ (3-Draht-ISP-Schnittstelle) programmiert wird. Die Programmierung übernehmen diese Bausteine selbstständig durch eine interne „High Voltage“ Programmierung.

Mit dem bekannten „digitalen“ 8-Bit-Mikrocontroller ATtiny2313, einem Industriestandard in einem praktischen 20-poligen DIL-Gehäuse, steht für den Elektroniker ein kostengünstiger Mikrocontroller zur Verfügung. Mit diesem Baustein lassen sich zahlreiche Aufgaben aus dem Bereich der Hobbyelektronik und aus der Praxis lösen.

Der „analoge“ 8-Bit-Mikrocontroller ATtiny26, ebenfalls ein Industriestandard in einem praktischen 20-poligen DIL-Gehäuse, hat mehrere 10-Bit-AD-Wandler zur Verfügung. Mit dem AD-Wandler lassen sich zahlreiche Versuche durchführen.

Die Versuche mit dem ATtiny2313 und ATtiny26 sind einfach auf einer Lochrasterplatine aufgebaut und man benötigt für die einzelnen Aufbauten etwa eine Stunde Lötzeit. Die Bauelemente erhalten Sie im Versandhandel.

Die Versuche werden alle im AVR-Assembler programmiert. Entwickler, Studenten und Schüler können auf komfortable Weise die ATMEL-Software für ihre Anwendungen auf dem PC implementieren und testen. Die sehr leistungsfähige Entwicklungsumgebung mit der Bezeichnung „AVR-Studio“ bündelt alle benötigten Funktionen – vom Assembler bis zum Simulator.

Wie bekommt man nun das Programm in Assemblersprache in den Mikrocontroller? Man muss es übersetzen, entweder in Hexadezimal- oder in Binärzahlen. Vor 20 Jahren musste man ein Programm in Assemblersprachen manuell übersetzen, Befehl für Befehl, und dies bezeichnete man als Hand-Assemblierung. Wie im Fall der Umwandlung hexadezimal in binär, ist die Hand-Assemblierung eine Routine-Aufgabe, die uninteressant, sich wiederholend und anfällig für zahllose kleine Irrtümer ist. Verwenden der falschen Zeile, Vertauschen von Ziffern, Auslassen von Befehlen und falsches Lesen der Codes sind nur einige der Fehler, die einem unterlaufen können. Die meisten Mikrocontroller machten früher die Aufgaben noch komplizierter, indem sie verschiedene Befehle mit unterschiedlichen Wortlängen

verwendeten. Manche Befehle sind nur ein Wort lang, während andere eine Länge von zwei oder drei Worten besitzen. Einige Befehle benötigen Daten im zweiten und dritten Wort und andere benötigen Speicheradressen, Registernummern oder andere Informationen.

Die Assemblierung ist eine weitere routinemäßige Aufgabe, die wir einem Mikrocontroller überlassen können. Der Mikrocontroller macht niemals Fehler beim Übersetzen von Codes. Er weiß immer, wieviele Worte und welches Format jeder Befehl benötigt. Ein Programm, das eine derartige Aufgabe ausführt, wird Assembler genannt. Das Assemblerprogramm übersetzt ein Anwenderprogramm oder Quellprogramm, das mit Mnemonics geschrieben wurde, in ein Programm in Maschinensprache (oder Objektprogramm), das der Mikrocontroller ausführen kann. Die Eingabe für den Assembler ist ein Quellprogramm und er gibt ein Objektprogramm aus.

Assembler besitzen eigene Regeln, die man erlernen muss. Diese enthalten die Verwendung bestimmter Markierungen oder Kennzeichen (wie Zwischenräume, Kommata, Strichpunkte oder Doppelpunkte) an den entsprechenden Stellen, korrekte Aussprache, die richtige Steuer-Information und vielleicht auch die ordnungsgemäße Platzierung von Namen und Zahlen. Diese Regeln stellen nur ein kleines Hindernis dar, das man leicht überwinden kann.

Ferner muss man beim Programmieren in Assemblersprache eine sehr detaillierte Kenntnis des verwendeten Mikrocontrollers besitzen. Man muss wissen, welche Register und Befehle der Mikrocontroller hat, wie die Befehle die verschiedenen Register genau beeinflussen, welche Adressierverfahren der Mikrocontroller verwendet und eine Unmenge weiterer Informationen. Keine dieser Informationen ist für die Aufgabe, die der Mikrocontroller letztlich ausführen muss, relevant.

Ich bedanke mich bei meinen Studenten für die vielen Fragen, die viel zu einer besonders eingehenden Darstellung wichtiger und schwieriger Fragen beigetragen haben. Meiner Frau Brigitte danke ich für die Erstellung der Zeichnungen, Herrn Gerhard Zwilling und der Firma ATMEL für die Unterstützung.

Inhalt

1 Hard- und Software für den 8-Bit-Mikrocontroller ATtiny2313

Der Baustein ATtiny2313 ist ein vielseitig verwendbarer 8-Bit-Mikrocontroller von Atmel und wegen seines einfachen Aufbaus und seiner leichten Programmierbarkeit auch bei Hobby-Elektronikern und im physikalischen Schulunterricht weit verbreitet, wenn praktische Übungen und Buchbeispiele kostengünstig zu realisieren sind.

Die Mikrocontroller AVR von Atmel unterteilen sich in folgende Gruppen:

- ATmega: große AVR-Controller mit bis zu 256-Kbyte-Flash-Speicher in 28- bis 100-poligen Gehäusen und mit integriertem Hardware-Multiplizierer.
- ATtiny: kleinere AVR-Controller mit bis zu 16-Kbyte-Flash-Speicher in 6- bis 20-poligen Gehäusen.
- AT90USB: ATmega mit integriertem USB-Controller.
- AT90CAN: ATmega mit CAN-Controller.
- Sondertypen: einige Spezialmodelle, z. B. zur Ansteuerung von Akku-Ladegeräten, LCD-Anzeigen und für Motoren- und Lichtsteuerungen.
- AT90S: veraltete Typen, die „klassischen“ AVRs.

1.1 Merkmale des Mikrocontrollers ATtiny2313

Der Mikrocontroller ATtiny2313 ist in einem 20-poligen DIL-Gehäuse untergebracht und es handelt sich um einen 8-Bit-Mikrocontroller. Im Gegensatz zum 8051 mit seiner CISC-Architektur (Complex Instruction Set Computer) verwendet der Mikrocontroller von Atmel eine RISC-Architektur (Reduced Instruction Set Computer). Die maximale Taktfrequenz liegt bei 12 MHz und hat bei der Abarbeitung der Befehle fast keine Bedeutung.

Bei den Versuchen in diesem Buch wird ohne Quarz beim Mikrocontroller ATtiny2313 gearbeitet. Die Arbeitsfrequenz liegt bei internen 8 MHz, die aber auf 1 MHz heruntergeteilt werden.

Bei dem 8051-Mikrocontroller ist normalerweise ein Quarz bei 12 MHz erforderlich. Die Frequenz wird intern durch 12 geteilt und damit arbeitet ein 8051 eigent-

lich mit 1 MHz. Benötigt ein Befehl beim Mikrocontroller 8051 zwei Taktzyklen, kann er nur 500.000 Befehle pro Sekunde abarbeiten. Ein AVR-Mikrocontroller mit 8 MHz (interne Takterzeugung) arbeitet dagegen fast acht Millionen Befehle pro Sekunde ab.

Der Mikrocontroller ATtiny2313 verwendet eine schnelle und leistungsarme RISC-Architektur in CMOS-Technologie. Der Befehlssatz hat sehr leistungsstarke 120 Befehle und die meisten Befehle lassen sich innerhalb eines Taktzyklus abarbeiten. Bei einer Taktfrequenz von 12 MHz ergeben sich typisch 12 MIPS (Mega Instructions per second) und die Arbeitsweise ist voll statisch. 32 Arbeitsregister mit einem 8-Bit-Format stehen zur Verfügung und davon können sechs Register als drei 16-Bit-Register verwendet werden.

Als interner Programmspeicher stehen 2-Kbyte-EEPROM (Electrically Erasable Programmable Read Only Memory) zur Verfügung und über eine ISP-Schnittstelle (In-System-Programmable) lässt sich das Assemblerprogramm im Mikrocontroller ATtiny2313 abspeichern. Es ist kein teures Programmiergerät erforderlich und die Programmierung kann direkt in der Schaltung durchgeführt werden, wenn man die Hardware entsprechend auslegt. Die Schreib-Löschzyklen liegen in der Größenordnung von 10.000. Der interne Arbeitsspeicher hat 128 Bytes.

Die Programmierung erfolgt über den standardisierten SPI-Bus (Serial Periphere Interface oder auch Microwire bezeichnet). Es handelt sich um ein Bussystem, bestehend aus drei Leitungen für die serielle synchrone Datenübertragung zwischen dem PC und dem AVR-Mikrocontroller. Der Anschluss für den SPI-Bus ist standardisiert, wird aber in den Bauanleitungen in diesem Buch nicht verwendet. Der Bus besteht aus folgenden Leitungen:

- MOSI (Master Out/Slave In) bzw. SDO (Serial Data Out) oder DO
- MISO (Master In/Slave Out) bzw. SDI (Serial Data In) oder DI
- SCK (Shift Clock) oder Schiebetakt

Über den SPI-Bus erfolgt auch die Datenkommunikation zwischen den verschiedenen Mikrocontrollern in einem System. Für den SPI-Bus gibt es kein festgelegtes Protokoll und die Taktfrequenz für den SPI-Bus kann bis 10 MHz betragen. Es gibt verschiedene integrierte Bausteine, die als Slaveeinheiten an dem SPI-Bus betrieben werden können, das reicht von externen Echtzeituhren (RTC = Real Time Clock), mehrstelligen 7-Segment-Anzeigen bis zu Grafikanzeigen mit vorgegebenem Protokoll.

Die Programmierbeispiele in diesem Buch sind in Assembler geschrieben und die dazugehörige Software von Atmel (Studio) ist kostenlos aus dem Internet herunterzuladen. Man lernt hierbei den Aufbau und die Wirkungsweise des Mikrocontrol-

lers besser kennen. Das Programmieren in Assembler ist wesentlich aufwendiger als bei höheren Programmiersprachen wie z. B. C und Basic. Programmieren in Assembler ist jedoch für den Anfänger besonders fehleranfällig und es kann lange dauern, bis das Programm richtig arbeitet. Aus diesem Grund wurden die höheren Programmiersprachen entwickelt. Das Programmieren in C ist besonders einfach, wenn bereits vorhandene Programmblöcke zur Verfügung stehen, die sich miteinander kombinieren lassen.

1.1.1 Anschlüsse des Mikrocontrollers ATtiny2313

Abb. 1.1 zeigt das Anschlussschema des Mikrocontrollers ATtiny2313.

Das 20-polige DIL-Gehäuse, in dem sich der Mikrocontroller ATtiny2313 befindet, ist einfach zu handhaben, wie in diesem Kapitel gezeigt wird. An Pin 10 und Pin 20 schließt man die Betriebsspannung und Masse an. Für die ersten Versuche verwendet man eine Betriebsspannung von U_b = 4,8 V und 0 V (Masse), was einem Batterieblock mit vier Akkumulatoren entspricht.

PortA sind bidirektionale I/O-Schnittstellen mit internen „pull-up"-Widerständen, die sich einzeln ansteuern lassen. Der ATtiny2313 hat drei PA-Anschlüsse für den Reset und für den externen Quarz. Arbeitet PortA als Ausgabeeinheit, kann man diese als Stromsenke oder als Stromquelle programmieren. Arbeiten diese als Eingänge, programmiert man sie als aktive „pull-up"-Widerstände. Schaltet man auf den Reset-Eingang ein 0-Signal, wird der Mikrocontroller zurückgesetzt und die Anschlüsse an PortA sind passiv. In dieser Versuchsanordnung wird immer ohne Quarz gearbeitet und dies ergibt eine Arbeitsfrequenz von etwa 1 MHz, wobei dieser Frequenzwert einen Toleranzbereich um ca. ± 5 % hat.

PortB sind bidirektionale I/O-Schnittstellen mit internen „pull-up"-Widerständen, die sich einzeln ansteuern lassen. Der ATtiny2313 hat acht PB-Anschlüsse. Arbeitet

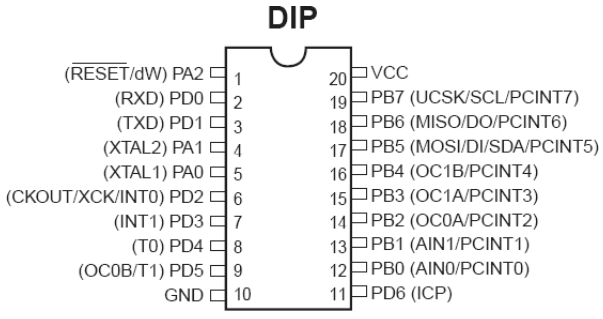

Abb. 1.1: Anschlussschema des Mikrocontrollers ATtiny2313

PortB als Ausgabeeinheit, kann man diese als Stromsenke oder als Stromquelle programmieren. Arbeitet er als Eingabeeinheit, kann man sie als aktive „pull-up"-Widerstände programmieren. Schaltet man auf den Reset-Eingang ein 0-Signal, wird der Mikrocontroller zurückgesetzt und die Anschlüsse an PortB sind passiv. Jeder der PB-Anschlüsse ist separat über das gemeinsame Steuerregister auf seine Aufgabe einzeln zu programmieren.

PortD sind bidirektionale I/O-Schnittstellen mit internen „pull-up"-Widerständen, die sich einzeln ansteuern lassen. Der ATtiny2313 hat sieben PD-Anschlüsse. Arbeitet PortD als Ausgabeeinheit, kann man ihn als Stromsenke oder als Stromquelle programmieren. Arbeitet er als Eingabeeinheit, kann man sie als aktive „pull-up"-Widerstände programmieren. Schaltet man auf den Reset-Eingang ein 0-Signal, wird der Mikrocontroller zurückgesetzt und die Anschlüsse an PortD sind passiv. Jeder der PD-Anschlüsse ist separat über das gemeinsame Steuerregister auf seine Aufgabe einzeln zu programmieren.

Mit einem 0-Signal an Pin 1 (Reset) wird der Mikrocontroller zurückgesetzt.

- PD0: Pin 2 arbeitet als RXD-Eingang (Receiver Data) für den internen USART (Universal-Synchronous-Asynchronous Receiver/Transmitter) und dies ist der Anschluss für den Empfänger der seriellen Schnittstelle. Außerdem dient der Pin als PCINT11 für den Interruptwechsel 2 der Interruptquelle 11.
- PD1: Pin 3 ist der TXD (Transmitter Data) für den internen USART und der Anschluss für den Sender der seriellen Schnittstelle. Außerdem dient der Pin als PCINT12 für den Interruptwechsel 2 der Interruptquelle 12.
- PD2: Pin 4 hat mehrere Funktionen. Für den externen Interrupt INT0 stellt der Pin den Interrupteingang dar. Für den USART (Universal-Synchronous-Asynchronous Receiver/Transmitter) ist der XCK-Ausgang als synchroner Übertragungstakt vorgesehen. Als CKOUT-Ausgang kann man den Systemtakt des ATtiny2313 abgreifen. Außerdem dient der Pin als PCINT13 für den Interruptwechsel 2 der Interruptquelle 13.
- PD3: Dieser Eingang INT1 (Pin 5) ist für den externen Interrupt vorhanden. Außerdem dient der Pin als PCINT14 für den Interruptwechsel 2 der Interruptquelle 14.
- PD4: Für den internen Zeitgeber und Zähler T0 schließt man das externe Taktsignal (Pin 6) an. Außerdem dient der Pin als PCINT15 für den Interruptwechsel 2 der Interruptquelle 15.
- PD5: Pin 9 lässt sich als „OC0B"-Ausgang (Timer/Counter 0 Compare Match B output) programmieren, wenn der Zeitgeber oder Zähler intern verglichen wird, ob der Zählerstand gleich oder ungleich ist. Für den internen Zeitgeber und Zähler T1 schließt man das externe Taktsignal an. Außerdem dient der Pin als PCINT16 für den Interruptwechsel 2 der Interruptquelle 16.

- PD6: Für den internen Zeitgeber und Zähler T1 schließt man an Pin 11 den Fangbereich für das externe Taktsignal an. Außerdem dient der Pin als PCINT17 für den Interruptwechsel 2 der Interruptquelle 17.
- PB0: Pin 12 ist der AIN0-Eingang für den analogen Komparator. Außerdem dient der Pin als PCINT0 für den Interruptwechsel 0 der Interruptquelle 0.
- PB1: Pin 13 ist der AIN1-Eingang für den analogen Komparator. Außerdem dient der Pin als PCINT1 für den Interruptwechsel 0 der Interruptquelle 1.
- PB2: Pin 14 ist der Ausgang OC0A für den internen Zeitgeber/Zähler 0. Außerdem dient der Pin als PCINT2 für den Interruptwechsel 0 der Interruptquelle 2.
- PB3: Pin 15 ist der Ausgang OC1A für den internen Zeitgeber/Zähler 1. Außerdem dient der Pin als PCINT3 für den Interruptwechsel 0 der Interruptquelle 3.
- PB4: Pin 16 ist der Ausgang OC1B für den internen Zeitgeber/Zähler 1. Außerdem dient der Pin als PCINT4 für den Interruptwechsel 0 der Interruptquelle 4.
- PB5: Pin 17 hat drei Funktionen. Für eine 3-Drahtverbindung nach USI (Univeral Serial Interface) arbeitet er als Eingang DE. Für eine 2-Drahtverbindung nach USI arbeitet er als Eingang SDA. Außerdem dient der Pin als PCINT5 für den Interruptwechsel 0 der Interruptquelle 5. Die MISO-Funktion (Master In/Slave Out) ist der Anschluss für die Programmierung.
- PB6: Pin 18 arbeitet als Ausgang DO für eine 3-Drahtverbindung nach USI. Außerdem dient der Pin als PCINT6 für den Interruptwechsel 0 der Interruptquelle 6. Die MOSI-Funktion (Master Out/Slave In) ist der Anschluss für die Programmierung.
- PB7: Pin 19 arbeitet als Taktleitung USCK für eine 3-Drahtverbindung nach USI oder als Taktleitung SCL für eine 3-Drahtverbindung nach USI. Außerdem dient der Pin als PCINT7 für den Interruptwechsel 0 der Interruptquelle 7.

1.1.2 Interner Aufbau des Mikrocontrollers ATtiny2313

Der ATtiny2313 arbeitet als 8-Bit-Mikrocontroller für allgemeine Anwendungen und ist dank seines außerordentlich geringen Bedarfs an zusätzlichen Bausteinen äußerst kostengünstig für kleine Mikrocontrollersysteme. Im Wesentlichen verwendet man diesen Mikrocontroller für digitale Aufgaben, denn es ist kein AD-Wandler vorhanden.

Der ATtiny2313 enthält außer den Funktionen, die der Befehlsausführung dienen, auch noch mehrere Einheiten für die Takterzeugung, die Systembus-Steuerung und die Prioritätsauswahl für die Interruptsteuerung. Der ATtiny2313 überträgt die internen Daten und Befehle auf einen 8-Bit-Datenbus. Der Prozessor des ATtiny2313 erzeugt Steuersignale, die zur Auswahl interner und externer Bausteine und zur Durchführung von Lese- und Schreiboperationen sowie zur Anwahl von I/O-Kanälen verwendet werden können.

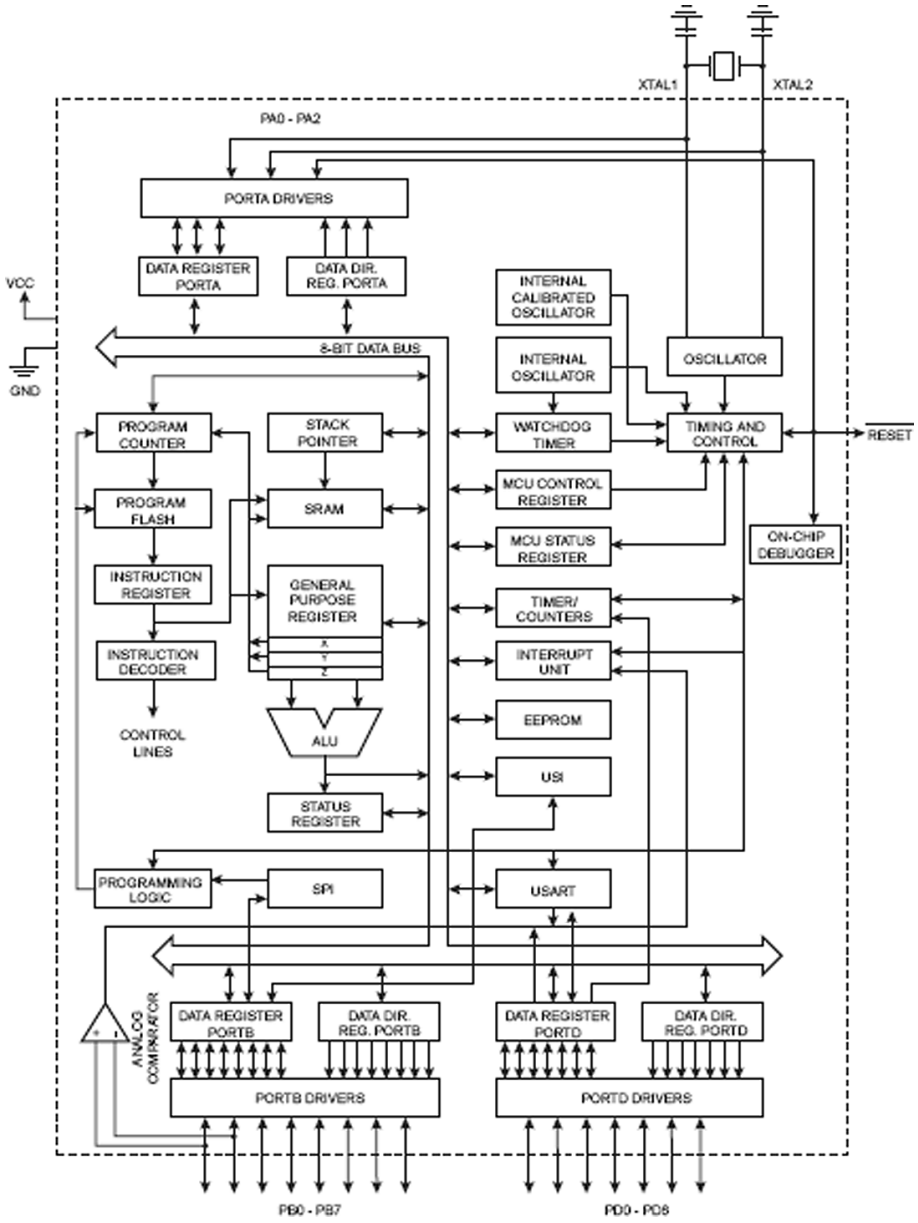

Abb. 1.2: Interner Aufbau des Mikrocontrollers ATtiny2313

Der ATtiny2313 ist mit internen 8-Bit-Registern (General Purpose Register) ausgestattet. Den Registern von R0 (Addr. 0x00) bis R25(Addr. 0x19) folgen die Register R26 (Addr. 0x1A) bis R31 (Addr. 0x1F), die als interne 16-Bit-Register (X, Y und Z) zusammengefasst werden können.

Die Register werden wie folgt unterschieden:

- Der Akkumulator ist das Register aller Akkumulator-Befehle. Dazu gehören arithmetische, logische, Lade- und Speicherbefehle sowie I/O-Anweisungen. Der Akkumulator ist ein 8-Bit-Register und arbeitet in Verbindung mit einem Statusregister, wo die Flags gespeichert werden.
- Der Programmzähler (PC oder Program Counter) zeigt auf den Speicherplatz des nächsten auszuführenden Befehls. Der Programmzähler hat eine 16-Bit-Adresse.
- Die Mehrzweckregister (General Purpose Register) lassen sich als allgemeine Register verwenden, abhängig vom auszuführenden Befehl.
- Der Stackpointer (SP) ist ein besonderer Datenzeiger, der stets auf das Ende des Stack (die Adresse des letzten gültigen Eintrags) zeigt. Es ist ein unteilbares 16-Bit-Register.
- Das Statusregister enthält sechs Flags von je einem Bit, in welchen je eine Zustandsinformation des Prozessors registriert wird und dadurch kann auch die Arbeitsweise des Prozessors gesteuert werden.

1.1.3 Statusregister

Im Statusregister (SREG) werden die Flags gespeichert, wie Tabelle 1.1 zeigt.

Tabelle 1.1: Aufbau des Statusregisters (SREG)

7	6	5	4	3	2	1	0
I	T	H	S	V	N	Z	C

- Bit 7 – I: Global Interrupt Enable

Das Signal ist kein Flag vom Prozessor des Mikrocontrollers, sondern zeigt, ob ein globaler Interrupt gesperrt oder freigegeben ist. Dieses Bit kontrolliert das separate Kontrollregister (MCU Control Register). Hat das Bit ein 0-Signal, kann kein Interrupt eine Funktion auslösen.

- Bit 6 – T: Bit Copy Storage

Das Signal ist kein Flag vom Prozessor des Mikrocontrollers, sondern zeigt, ob ein Kopierbefehl BLD (Bit LoaD) und ein Speicherbefehl BST (Bit STore) ausgeführt worden ist.

- Bit 5 – H: Half Carry Flag

Das Hilfscarry-Flag zeigt einen Überlauf aus dem Bit 3 des Akkumulators an. Dieses Flag wird im Allgemeinen für die BCD-Arithmetik (Arithmetik mit binär codierten Dezimalzahlen) verwendet. Die folgende Addition setzt oder löscht das Hilfs-Carrybit.

Bit-Nr.	7	6	5	4	3	2	1	0
2E =	0	0	1	0	1	1	1	0
+74 =	0	1	1	1	0	1	0	0
A2	1	0	1	0	0	0	1	0

1
↑ Hilfs-Carrybit = 1

Das Hilfs-Carrybit wird durch Additions-, Subtraktions-, Inkrement-, Dekrement- und Vergleichsbefehle verändert.

- Bit 4 – S: Sign Bit

Im Mikrocontroller ATtiny2313 ist grundsätzlich kein Vorzeichen eines Datenbytes festgelegt. Man kann daher ein Byte mit dem numerischen Wert 128 als plus 128 oder aber auch als minus 128 interpretieren. Zur Unterscheidung der beiden Möglichkeiten verwendet man das Bit 7 als Vorzeichen. Hat es den Wert 1, so wird die Zahl im Byte als negativ angesehen (von minus 1 bis minus 128), hat Bit 7 den Wert 0, so wird die Zahl des Bytes als positive Zahl (von 0 bis plus 127) interpretiert. Bei arithmetischen und logischen Operationen wird das Signbit dem Bit 7 des Ergebnisses gleichgesetzt und es kann dann als Bedingungsbit abgefragt werden.

Bit-Nr.	7	6	5	4	3	2	1	0
2E =	0	0	1	0	1	1	1	0
+74 =	0	1	1	1	0	1	0	0
A2	1	0	1	0	0	0	1	0

1
↑ Sign Bit = 1

- Bit 3 – V: Two's Complement Overflow

Dieses Flag wird gesetzt, wenn bei einer 2er-Arithmetik ein Überlauf entstanden ist. Es soll zuerst eine Subtraktion ohne Zweierkomplement durchgeführt werden.

Bit-Nr.	7	6	5	4	3	2	1	0
2E =	0	0	1	0	1	1	1	0
–74 =	0	1	1	1	0	1	0	0
B2 =	1	0	1	0	0	0	1	0

Führt man die Subtraktion im Zweierkomplement aus, wird der Subtrahend komplementiert.

Bit-Nr.	7	6	5	4	3	2	1	0	
2E =	0	0	1	0	1	1	1	0	
–8C	1	0	0	0	1	1	0	0	
	1	0	0	0	1	0	1	1	⇐ Komplement und Addition
	1	0	1	1	0	0	0	1	
1									⇐ Übertrag
	⟶							1	⇐ Korrektur mit Addition
B2 =	1	0	1	1	0	0	1	0	⇐ korrekte Subtraktion
0									
↑ V-Bit = 0									

Dieses Flag wird rückgesetzt, da bei einer Zweierkomplement-Arithmetik kein Überlauf entstanden ist.

- Bit 2 – N: Negatives Flag

Dieses Flag wird gesetzt, wenn bei einer arithmetischen oder logischen Operation ein negatives Ergebnis entstanden ist.

Bit-Nr.	7	6	5	4	3	2	1	0	
2E =	0	0	1	0	1	1	1	0	
–10 =	0	0	0	1	0	0	0	0	
	1	1	1	0	1	1	1	1	⇐ Komplement und Addition
9C =	1	0	0	1	1	1	0	1	
0									⇐ Übertrag
	⟶							1	⇐ Korrektur mit Addition
BF =	1	0	1	1	1	1	1	0	⇐ korrekte Subtraktion
0									
↑ N-Bit = 0									

- Bit 1 – Z: Zero Flag

Dieses Bedingungsbit (Nullbit) wird dann gesetzt, wenn das Ergebnis eines arithmetischen oder logischen Befehls null ist. Das Zerobit wird rückgesetzt, wenn das Ergebnis dieses Befehls ungleich null ist.

Ist das Ergebnis gleich null und das Carrybit gesetzt, so wird das Zerobit ebenfalls gesetzt.

Bit-Nr.	7	6	5	4	3	2	1	0	
	1	0	1	0	0	1	1	1	
	0	1	0	1	1	0	0	1	
	0	0	0	0	0	0	0	0	
1									
↑ Überlauf von Bit 7									⇒ Null-Ergebnis: Zerobit auf 1 gesetzt

- Bit 0 – C: Carry Flag

Das Carrybit C (Überlaufbit oder Übertragsbit) wird durch verschiedene Befehle gesetzt und kann direkt abgefragt werden. Die Operationen, die das Carrybit verändern, sind Addition, Subtraktion, zyklisches Schieben und logische Operationen. So kann z. B. die Addition von zwei 1-Byte-Zahlen einen Überlauf (Carry) an der höchsten Stelle hervorrufen.

Bit-Nr.	7	6	5	4	3	2	1	0
AE =	1	0	1	0	1	1	1	0
+74 =	0	1	1	1	0	1	0	0
122	0	0	1	0	0	0	1	0
	1							

↑ Überlauf = 1, setzt Carrybit = 1

Eine Addition mit Überlauf an der höchsten Stelle setzt das Carrybit. Eine Addition ohne Überlauf setzt das Carrybit zurück. Hier soll noch eigens darauf hingewiesen werden, dass Addition, Subtraktion, zyklisches Schieben und logische Befehle das Carrybit in verschiedenartiger Weise behandeln.

1.1.4 I/O-Einheiten mit Treiber

Die Schnittstellen zwischen den internen und externen Einheiten bilden 3-Bit-Treiber von PORTA, 8-Bit-Treiber von PORTB und 7-Bit-Treiber von PORTD. Diese Treiber arbeiten bidirektional und intern befindet sich das Datenregister bei den Ausgaberegistern, d. h. die Datenregister übernehmen den Datenaustausch und die anderen sind nur für die Datenausgabe zuständig. Jeder Port lässt sich einzeln und separat über das Steuerregister programmieren.

Jede Leitung des Registers besteht aus zwei getrennten Puffern mit drei Zuständen (Tristate), wodurch ein direkter Busanschluss möglich ist und die Bidirektionalität erreicht wird. Auf der einen Seite des Treibers ist der Ausgang des einen Puffers mit dem Eingang eines anderen zusammengeschaltet. An dieser Seite werden externe Teilsysteme angeschlossen, da diese Schnittstelle direkt TTL-kompatibel ist und einen hohen Strom treiben kann.

Alle I/O-Einheiten übernehmen den internen und externen Datenaustausch. Jeder dieser Pins kann mehrere Funktionen ausführen. So lässt sich hier eine externe Prioritätssteuerung für Interruptanforderung durchführen und man hat bis zu 15 verschiedene Möglichkeiten für die Interruptquellen. Es lassen sich 15 Anforderungssignale verarbeiten und der Interruptcontroller bestimmt die höchste Priorität, vergleicht diese mit einem softwaregesteuerten Register für die jeweils ablaufende Programmebene und unterbricht den Mikrocontroller bei

gleichzeitiger Abgabe der Zeigerinformation zum Aufruf des benötigten Service-Programms.

1.1.5 Programmierbarer USART

Der USART ist eine universelle Synchron-/Asynchron-Sender/Empfänger-Einheit für die Datenübertragung in Verbindung mit externer Peripherie. Für den seriellen Sende- und Empfangsbetrieb sind Pin 2 (RXD) und Pin 3 (TXD) vorgesehen. Der ATtiny2313 hat zwei kaskadierte Register im Sender und Empfänger und diese ermöglichen einen schnellen Vollduplexbetrieb. Die Betriebsarten sind der Synchron- und der Asynchronbetrieb.

Der USART übernimmt vom Mikrocontroller die Zeichen in Paralleldarstellung und wandelt sie für das Senden in einen seriellen Datenstrom um. Gleichzeitig kann er einen seriellen Datenstrom empfangen und daraus für den Mikrocontroller die Zeichen in Paralleldarstellung erzeugen. Der USART meldet dem Mikrocontroller, wenn die Interrupteinheit ein neues Zeichen zum Senden annehmen kann oder ein Zeichen für den Mikrocontroller empfangen hat. Der Mikrocontroller kann jederzeit den Zustand der Einheit abfragen. Dabei werden ihm zugleich Aussagen über Datenübertragungsfehler und den Zustand von Steuerschlüssen geliefert.

Die Eigenschaften im Asynchronbetrieb sind:

- Taktfrequenz wahlweise das 1-, 16- oder 64-Fache der Übertragungsgeschwindigkeit
- Störimpulse werden nicht als Anlaufschritte interpretiert
- Länge des Sperrschritts wahlweise 1, 1½ oder 2 Bit
- Erzeugung des BREAK-Zustands
- Automatische Erkennung und Anzeige des BREAK-Zustands
- Übertragungsgeschwindigkeit bis zu 64 kBit/s

Eigenschaften im Synchronbetrieb:

- Wahlweise interne oder externe Zeichensynchronisation
- Automatisches Einschieben von Synchronzeichen
- Übertragungsgeschwindigkeiten bis zu 1 Mbit/s

Der USART ist ein universeller Synchron-/Asynchron-Sender/Empfänger. Wie auch bei anderen Ein-/Ausgabe-Einheiten in Mikrocontrollersystemen wird seine Funktion zur Erzielung größtmöglicher Flexibilität durch die Systemsoftware programmiert. Dies geschieht durch Übergabe von Steuerwörtern (Mode-, Kommando-, SYNC1-, SYNC2-Wort), die in Registern gespeichert werden und bis zum Überschreiben durch neue Ausgaben dort zur Verfügung stehen. Außerdem kann der

Mikrocontroller den Status von Sender und Empfänger durch Einlesen eines Statusworts aus einem Register im Baustein abfragen. Dieses Konzept ist für praktisch jedes heute gebräuchliche serielle Datenübertragungsverfahren geeignet.

Für die serielle Datenübertragung mit einem Mikrocontroller muss der Sender die vom ATtiny2313 in Paralleldarstellung gelieferten Zeichen in den zu sendenden seriellen Datenstrom überführen, und der Empfänger muss aus dem empfangenen seriellen Datenstrom zum Abholen durch den Mikrocontroller Zeichen in Paralleldarstellung erzeugen. Das Umwandeln von Parallel- in Seriendarstellung im Sender erfolgt durch das Sende-Schieberegister, das Umwandeln von Serien- in Paralleldarstellung im Empfänger durch das Empfangs-Schieberegister. Um dem Mikrocontroller mehr Freizügigkeit für den Zeitpunkt zu geben, zu dem er ein Zeichen an den Sender übergibt und ein Zeichen aus dem Empfänger abholt, enthält der Sender außerdem noch ein Sende-Parallelregister und der Empfänger ein Empfangs-Parallelregister. Der Mikrocontroller übergibt seine Zeichen immer an das Sende-Parallelregister, und der Sender holt es von dort in das Sende-Schieberegister, sobald er das vorhergehende Zeichen ausgesendet hat. Entsprechend gelangt das empfangene Zeichen im Empfänger zunächst in das Empfangs-Schieberegister, und der Empfänger übergibt es von dort in das Empfangs-Parallelregister, von wo es der Mikrocontroller abholen kann.

Der Sender muss außer der Parallel-Serien-Wandlung der Zeichen noch Bits oder Zeichen, die nur für das spezielle Übertragungsverfahren benötigt werden, in den seriellen Datenstrom einfügen, und der Empfänger muss diese Bits und Zeichen wieder entfernen. Damit wird der Übertragungsweg „transparent“, d. h. der Anwender braucht sich nicht mit den Besonderheiten des Übertragungsverfahrens zu befassen, sondern tauscht mit Sender und Empfänger nur noch einfache Zeichen aus.

1.1.6 Zähler- und Zeitgebereinheit

Die beiden Zähler- und Zeitgebereinheiten sind programmierbar und arbeiten voneinander unabhängig als 16-Bit-Zähler. Die Einheit ist ein allgemein einsetzbarer Mehrfachzeitgeber, der von der Systemsoftware behandelt wird wie eine Anzahl gewöhnlicher Ein-/Ausgabe-Kanäle.

Die beiden Zähler- und Zeitgebereinheiten lösen eines der in Mikrocontrollersystemen am häufigsten auftretenden Probleme, die programmgesteuerte Erzeugung präziser Verzögerungszeiten. Zur Erfüllung seiner Anforderungen setzt der Programmierer statt den in der System-Software vorgesehenen Zeitschleifen die Zähler- und Zeitgebereinheit ein.

Nachdem der Zähler mit dem gewünschten Anfangswert programmiert wurde, ist er startbereit. Nach Ablauf des Zähldurchlaufs wird ein Zählende-Signal gegeben, das zur Auslösung einer Programmunterbrechung verwendet werden kann. Es ist daraus bereits zu ersehen, dass der Programmieraufwand relativ gering ist und dass z. B. zwei Zeitverzögerungen per Hardware gleichzeitig nebeneinander ablaufen können, indem den einzelnen Zählern verschiedene Unterbrechungsebenen zugewiesen werden.

Andere Zähler-/Zeitgeber-Aufgaben, die normalerweise keine Zeitverzögerungsschaltungen sind, aber in Mikrocontrollersystemen verwendet werden, können ebenfalls von Zähler- und Zeitgebereinheit ausgeführt werden, z. B.:

- Steuerbarer Signalgenerator
- Ereigniszähler
- Binärer Frequenzmultiplizierer
- Echtzeituhr
- Digital einstellbare monostabile Kippstufe
- Komplexer Motor-Steuerbaustein

Die über die Datenbus-Schnittstelle hereinkommenden Daten werden dann in das Steuerwort-Register übernommen. Mit den im Steuerwort-Register gespeicherten Informationen wird die Arbeitsweise jedes Zählers bestimmt sowie die Auswahl der Zählart (dual oder dezimal) und das Laden der Zählerregister gesteuert. In das Steuerwort-Register kann nur eingeschrieben und nicht ausgelesen werden.

Die zwei Zähler sind funktionell vollkommen identisch. Jeder Zähler besteht aus einem vorbelegbaren 16-Bit-Abwärtszähler, der wahlweise für duales oder dezimales Zählen eingestellt werden kann. Die Funktion wird durch die Auswahl des Steuerwortes im Steuerwort-Register bestimmt. Die Zähler arbeiten voneinander unabhängig und können parallel in vollkommen verschiedenen Betriebsarten betrieben werden. Im Format des Steuerwortes sind spezielle Kennzeichen zum Laden des Zähleranfangswertes enthalten, sodass der Aufwand an Systemsoftware möglichst gering bleibt. Der Anwender kann bei Verwendung eines Zählers als Ereigniszähler den Zählerinhalt mit einem einfachen Lesebefehl auslesen. Für das Lesen der Zählerinhalte während des Zählens besitzt der ATtiny2313 ein spezielles Steuerwort und eine Logik, sodass der Eingangstakt nicht gesperrt werden muss.

Die Arbeitsweise der Zähler- und Zeitgebereinheit wird vollständig über die System-Software festgelegt. Der Mikrocontroller muss dazu einige Steuerworte an die Zähler- und Zeitgebereinheit ausgeben. Mit diesen Steuerworten werden die Betriebsart, die Anzahl der zu ladenden Zählerbytes und duales oder dezimales (BCD)-Zählen festgelegt.

Einmal initialisiert, ist die Zähler- und Zeitgebereinheit bereit, jene Zeitgeberaufgaben durchzuführen, die ihr per Software zugewiesen wurde. Der laufende Zählvorgang jedes Zählers ist völlig unabhängig von denen der anderen Zähler. Mit einer Zusatzlogik des Bausteins können die am häufigsten auftretenden Probleme im Zusammenhang mit der Überwachung und Verarbeitung von externen, asynchronen Signalen in einem Mikrocontrollersystem gelöst werden.

Betriebsart 0: Unterbrechungsanforderung bei Zählernulldurchgang

Der Zählerausgang liegt nach dem Ausgeben des Steuerwortes auf 0-Signal. Nachdem der Zähleranfangswert in den ausgewählten Zähler eingeschrieben wurde und der Zählvorgang begonnen hat, bleibt der Zählerausgang weiterhin auf 0-Signal. Bei Erreichen des Zählerendstandes (0) geht der Ausgang auf 1-Signal und bleibt dort solange, bis ein neuer Zähleranfangswert oder ein neues Steuerwort geladen wird. Der Zähler selbst wird auch nach Erreichen des Zählerendstandes weiter dekrementiert.

Wird während des Zählens der Zähler neu geladen, so erfolgt

1. beim Laden des ersten Bytes der Abbruch des laufenden Zählvorgangs,
2. beim Laden des zweiten Bytes der Start mit dem neuen Anfangswert.

Durch ein 0-Signal am GATE-Eingang (ein programmierbarer Anschlusspin von PB oder PD) wird das Zählen gesperrt und durch ein 1-Signal wieder freigegeben.

Betriebsart 1: Programmierbare monostabile Kippstufe

Der Zählerausgang geht nach einer positiven Signalflanke auf den GATE-Eingang (ein programmierbarer Anschlusspin von PB oder PD) und mit der nächsten Periode des Eingangstaktes auf 0-Signal.

Nach Erreichen des Endstandes (0) geht der Zählerausgang wieder auf 1-Signal über. Das Einschreiben eines neuen Zähleranfangswertes zu einem Zeitpunkt, bei dem der Ausgang auf 0-Signal liegt, hat keinen Einfluss auf den laufenden Zählvorgang. Der neue Anfangswert wird erst nach erneutem Triggern übernommen. Der aktuelle Zählerstand ist ohne weiteren Einfluss auf den Zählvorgang selbst jederzeit auslesbar (über einen Lesebefehl des Mikrocontrollers).

Die monostabile Kippstufe ist beliebig oft neu triggerbar, d. h., der Ausgang bleibt nach jeder ansteigenden Flanke des GATE-Eingangssignals (ein programmierbarer Anschlusspin von PB oder PD) auf 0-Signal, bis der Zähler vom Anfangswert bis zum Endwert (0) gezählt hat.

Betriebsart 2: Taktgenerator, Teiler durch N

Der Zählerausgang geht während eines Zählerdurchlaufs für eine Periode des Eingangstaktes auf 0-Signal. Die Anzahl der Taktperioden von einem Ausgangsimpuls zum nächsten ist gleich dem geladenen Zähleranfangswert. Wird zwischen zwei Ausgangsimpulsen der Zähler mit einem neuen Anfangswert geladen, wird erst die laufende Zählperiode beendet, bevor der Zähler mit dem neuen Wert startet.

Mit einem 0-Signal am GATE-Eingang (ein programmierbarer Anschlusspin von PB oder PD) wird am Zählerausgang ein 1-Signal erzeugt. Geht der GATE-Eingang wieder auf 1-Signal über, startet der Zähler erneut mit dem geladenen Anfangswert. Der GATE-Eingang ist daher zum Synchronisieren des Zählers geeignet.

Nachdem das Steuerwort für Betriebsart 2 eingeschrieben wurde, bleibt der Zählerausgang solange auf 1-Signal, bis der Zähleranfangswert vollständig geladen ist. Der Zähler ist also auch über die Software synchronisierbar.

Betriebsart 3: Rechteckgenerator

Diese Betriebsart ist ähnlich wie Betriebsart 2 mit dem Unterschied, dass der Zählerausgang solange auf 1-Signal bleibt, bis der halbe Zähleranfangswert abgezählt wurde (gilt für geradzahlige Startwerte). Anschließend geht der Ausgang auf 0-Signal, bis die andere Hälfte abgezählt ist. Bei ungeradzahligen Zähleranfangswerten N ist der Ausgang für (N + 1)/2 Eingangstakte auf 1-Signal und für (N –1)/2 Takte auf 0-Signal. Wird der Zähler während des Zählens mit einem neuen Anfangswert geladen, wird nach dem nächsten Wechsel des Ausgangssignals mit dem neuen Wert weitergearbeitet.

Betriebsart 4: Softwaregesteuertes Signal (Strobe)

Nach der Ausgabe des Steuerwortes für diese Betriebsart liegt der Zählerausgang auf 1-Signal. Nach dem Laden des Zähleranfangswertes startet der Zähler. Bei Erreichen des Endstandes 0 liegt am Ausgang für eine Taktperiode 0-Signal und anschließend dauerhaft wieder 1-Signal an.

Wird während eines Zählerdurchlaufs ein neuer Zähleranfangswert eingegeben, zählt der Zähler sofort nach beendeter Eingabe mit dem neuen Wert weiter. Wird der Zähler nach Beendigung eines Zählerdurchlaufs neu geladen, startet er ebenfalls nach vollendeter Eingabe mit dem neuen Wert. Mit einem 0-Signal am GATE-Eingang (ein programmierbarer Anschlusspin von PB oder PD) kann das Zählen unterbrochen werden.