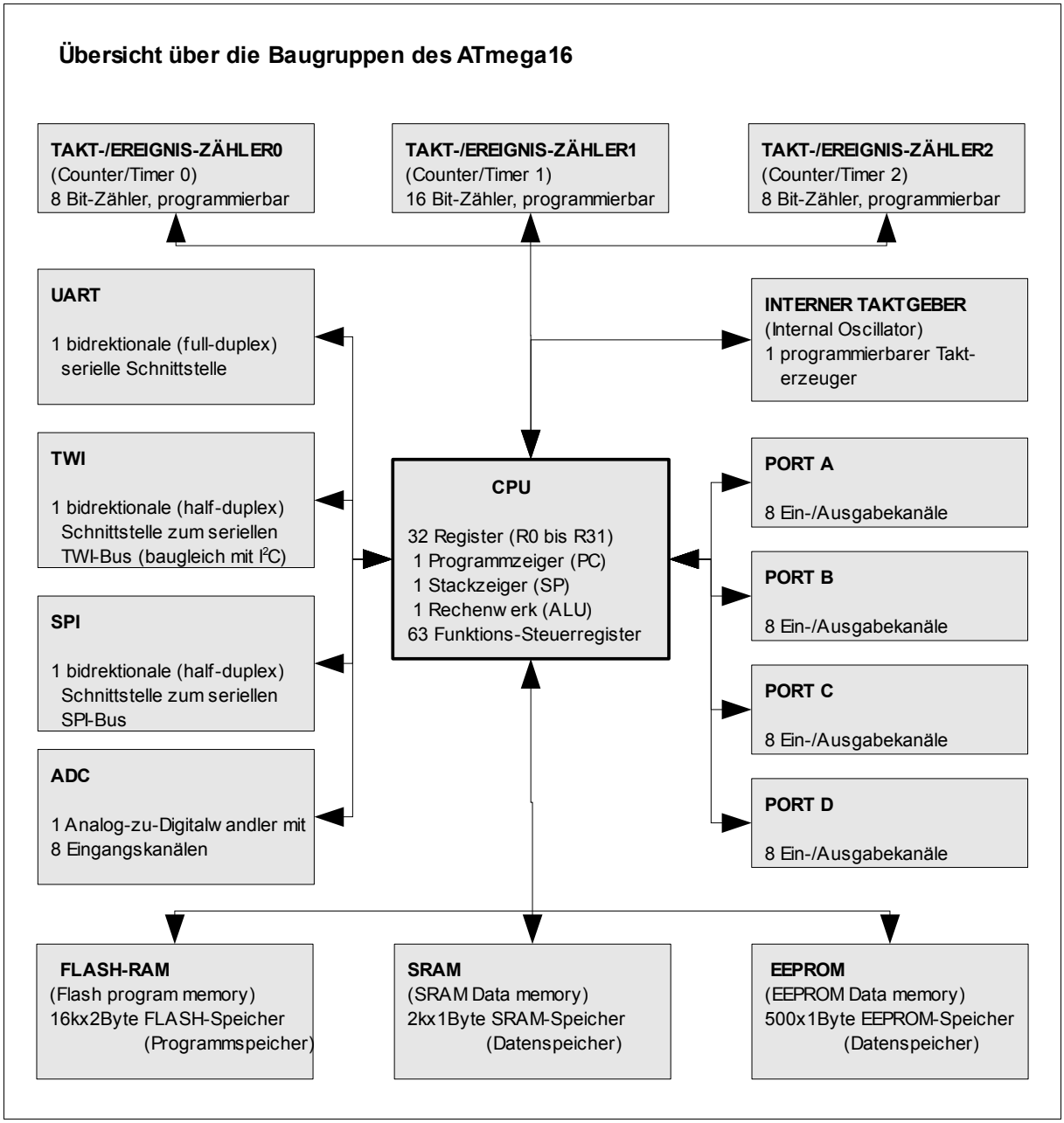
Kapitel 03: Über das Programmieren
Abs02: Ein kleiner Rundgang durch den ATmega16

Die wichtigste Quelle für alle Fragen die die Eigenschaften und Fähigkeiten eines IC ist immer das Datenblatt. Und das nehmen wir jetzt zur Hand, um uns einen Überblick über die Funktionen zu verschaffen, die uns ein ATmega16 bietet.

Das erste wichtige Diagramm ist das auf Seite 2 die „Figure 1-1: Pinout ATmega16A". Da wird nämlich erklärt, welche Funktion jedes Beinchen des IC hat. „Pinout" wird im Deutschen meist mit „Pinbelegung" übersetzt, wobei mit „Pin" „Beinchen" gemeint ist. Ich ziehe die Bezeichnung „Beinchen" oder „Bein" vor, weil bei den ATmegas die Bezeichnung „PIN" noch eine andere Bedeutung haben kann. Weil wir den ATmega16 für unseren Lehrgang in das STK500 einstecken werden, ist die Pinbelegung zunächst noch nicht so interessant, weil das STK500 dafür sorgt, dass sie alle richtig angeschlossen sind (wenn der MC richtig herum in die Fassung eingesteckt ist!). Aber später wird man seine eigenen Platinen bauen und dann muss man die die Funktionen der Beinchen genau beachten.

Die Funktionen sind auf den Seiten 6 und 7 unter Überschrift „2.2 Pin Descriptions" grob beschrieben. Die Einzelheiten finden sich in aller wünschbaren Ausführlichkeit dann jeweils in den Kapiteln 10 bis 22 in Zusammenhang mit der Beschreibung der einzelnen Funktions-Baugruppen, z.B. Ports, UART-, TWI-Schnittstelle, ADC usw.. Diese Beschreibungen werden wir uns in Verbindung mit den Versuchsprogrammen vorknöpfen, in denen wir diese Funktionen benutzen werden.

Hier soll es erst einmal um den inneren Aufbau des Mikrokontroller (MC) gehen. Das Diagramm „Figure 2-1: Block Diagram" auf Seite 4 zeigt zwar alles, was dazu zu sagen ist; es ist aber für den Anfang doch zu detailliert. Das Bild unten ist für den ersten Überblick besser geeignet:



Bussystem
Die zentrale Baugruppe ist die CPU („Cental Processing Unit"). Sie ist mit allen Funktions-Baugruppen und den drei Speichern (SRAM, Flash-RAM und EEPROM) über Datenbusse verbunden; über den fliessen die Daten in beiden Richtungen: Von der CPU zu den anderen Baugruppen und umgekehrt.

CPU
Jedes der 32 Register der CPU kann ein Byte an Daten aufnehmen. Sie bilden das Forum, in dem sich die wesentlichen Aktivitäten des Programms abspielen. Alle Berechnungen und Verwaltungsaufgaben werden hier erledigt, indem man mit den Anweisungen des Programms

steuert, wie die Daten zwischen einem der drei Speicher (SRAM, Flash-RAM oder EEPROM) oder einer Funktionsbaugruppe und diesen Registern hin- und hergeschoben werden. Zwischen den Registern selbst kann man die Daten nicht nur verschieben, sondern man kann sie auch miteinander verrechnen, z.B. addieren, subtrahieren oder multiplizieren. Man kann zwischen ihnen logische Verknüpfungen ausführen lassen, wie ein logisches „Und" oder ein „Oder". Die Ergebnisse dieser mathematischen und logischen Aktivitäten werden jeweils wieder in einem der Register gespeichert.

Interner Taktgeber (Datenblatt Abs. 8.7, Seite 29)
Angetrieben wird der Datenfluss innerhalb und zwischen allen Baugruppen durch den Taktgeber. Den Takt („MC-Takt") kann man wahlweise dem internen Taktgeber beziehen, der sich auf verschiedene Frequenzen einstellen lässt, oder von der Möglichkeit Gebrauch machen, den Takt von einem extern angeschlossenen Taktgeber (Datenblatt, Abs. 8.8, Seite 30) zu übernehmen.

Flash-RAM (Datenblatt Abs. 7.2, Seite 16)
Das Programm, mit dem der Programmierer das Getümmel steuert, wird im FLASH-RAM gespeichert. Der Ausdruck „Flash" bezieht sich auf die Konstruktion der Speicherzellen („Floating Gate ... „, zu Deutsch sinngemäss „Speicherzelle mit schwebender Basiselektrode"). Wie die im Einzelnen aussieht braucht einen Programmierer nicht zu interessieren. Wichtig ist nur, dass diese Art von Speicher die Programmdaten über sehr lange Zeiträume unverändert speichern kann und dass man ihn fast beliebig oft löschen und mit neuen Daten füllen kann. Das Datenblatt gibt an, man könne das Flash-RAM mindestens 10000-mal löschen und neu beschreiben - mir ist es aber noch nie vorgekommen, dass ein ATmega an Flash-Speicherermüdung gestorben ist. Und das, obwohl ich sehr, sehr oft flashe. „Flashen" ist der Ausdruck, der sich für das Löschen und neu Beschreiben des Programmspeichers im Deutschen eingebürgert hat. Als Amateur muss man sich wegen der „begrenzten" 10000 Zyklen keine Beschränkungen beim Flashen auferlegen.

Und wie flasht man ein Programm in den ATmega? Das übernimmt bei uns das STK500. Die Daten, die der Assembler im PC erzeugt, werden über die serielle oder USB-Schnittstelle an den STK500 geliefert. Dort bereitet ein eigener MC die Daten mundgerecht für den eingesteckten ATmega auf, versetzt ihn in Programmierbereitschaft und übergibt ihm die Daten zum Speichern. Die elektrischen Signale, die dafür erforderlich sind, gehen über den ISP-Stecker und das angeschlossene Flachbandkabel zu dem ATmega, der programmiert werden soll. Mehr braucht man für den Anfang über diese Geschichte nicht zu wissen.

SRAM (Datenblatt Abs. 7.3, Seite 17)
Der nächst wichtige Speicherbereich ist das SRAM („Static Random Acess Memory", zu Deutsch „statischer beliebig addressierbarer Lese-/Schreibspeicher"). Das „Static" hebt den Gegensatz zu den dynamischen Speichern („DRAM") hervor. Beim DRAM muss der Speicherinhalt innerhalb einer kurzen Zeit immer wieder aufgefrischt werden, damit er nicht verloren geht. Früher, als man die SRAM-Speicher noch nicht so klein bauen konnte (die brauchen nämlich 2 Transistoren pro gespeichertes Bit) war das DRAM (mit nur einem Transistor pro Bit) der einzige Speichertyp, in dem man mehr als ein Kilobyte Speicherplatz in einem Chip unterbringen konnte.

Im ATmega sind alle Daten im SRAM gespeichert, die der MC immer wieder und schnell zugreifen muss. Das sind die 32 Register, dann der PC („Programmzeiger"), der Stackzeiger („SP"), die Einstellungen, die die Funktionsbaugruppen steuern, der Stack und schliesslich auch alles das, was der Programmierer im Rahmen seines Programms so alles speichern will. Durch diese Speichereinteilung sind die unteren Adressen 0x0000 und 0x005F für den Programmierer nicht frei nutzbar; erst ab 0x0060 kann man speichern, was man will. Die obersten Adressen des SRAMS sind auch nur eingeschränkt nutzbar, weil hier normalerweise der Stack untergebracht ist. Wieviel Speicherplatz er dort beansprucht, hängt von der Art ab, wie das Programm davon Gebrauch macht. Bei den Programmen, die im Rahmen dieses Lehrgangs besprochen werden, belegt der Stack aber nicht mehr als die ca. 128 höchsten Adressen. Der SRAM-Bereich von 0x0060 bis 0x037F darf das Programm also nach Belieben benutzen; das sind 895 Byte. Gemessen an einem PC mit mehreren MegaByte an Arbeitsspeicher ist das fast nichts; für einen MC in einer normalen Anwendung ist es aber reichlich viel Platz.

EEPROM (Datenblatt Abs. 7.7, Seite 18)
Deshalb hat ATMEL den ATmegas den dritten Speicherblock spendiert, das EEPROM („Electrically Eraseable and Programmable Read-Only-Memory", zu Deutsch ungefähr „elektronisch lösch- und beschreibbarer Lesespeicher"). Hier kann man Daten speichern, die man zwar ab und zu verändern können will, die aber nicht mit dem Abschalten der Versorgungsspannung verschwinden sollen. Es sind meistens Daten, deren Werte irgendwie mit den Eigenschaften des Endgerätes zusammenhängen, in die der ATmega eingebaut wird. Dazu zählen, z.B. Parameter, wie Motorkennlinien, Regelparameter usw.. Leider kostet das Lesen und Schreiben von Daten beim EEPROM ziemlich viel Zeit (verglichen mit der Zugriffszeit beim SRAM oder Flash-RAM). Deshalb verwendet man es eher selten.

Timer/Counter allgemein
Das Schöne an den Funktionsbaugruppen ist, dass sie ganz selbstständig funktionieren. Sie laufen alle nebeneinander her, ohne die anderen Baugruppen, vor allem die CPU, irgendwie aufzuhalten; es sei denn, man legt das im Programm ausdrücklich anders fest. Besonders vorteilhaft ist das bei den Takt-/Ereigniszählern. Sie können externe Ereignisse (z.B. die Impulse von einer Lichtschranke) oder die vom Taktgeber erzeugten Impulse zählen (z.B. um einen Zeitraum zu vermessen) und können auch selbst Signale erzeugen, die nach aussen (z.B. zur Motor-Drehzahleinstellung) weitergegeben werden können. Das Beinchen, über das das Signal nach aussen geleitet wird, ist dem Timer jeweils fest zugeordnet. Weil diese Baugruppen sind so vielseitig sind, merkt man beim Programmieren sehr schnell, dass man gar nicht genug davon haben kann: Der ATmega16 hat drei Timer/Counter, die sich in ihren Eigenschaften nur wenig voneinander unterscheiden.

Die intern erzeugten Takte, die die Timer zählen, sind alle vom MC-Takt abgeleitet. Mit dem Vorteiler - jeder Timer hat einen eigenen - hat kann man einstellen, um wieviel die gezählten Takte langsamer sein sollen, als der MC-Takt. Stattdessen kann man (beim ATmega16 nicht bei allen Timern) aber auch ein externes Signal zählen, das sich über ein Beinchen des IC von aussen zum Timer durchgeschaltet werden kann.

Der Zählerstand lässt sich jederzeit abfragen. Die Takte, ob extern oder intern, können die Timer in verschiedenen Betriebsarten verarbeiten, die hier kurz vorgestellt werden (die ausführliche Beschreibung folgt in Zusammenhang mit den entsprechenden Programmen):

„Normal" - der Zähler zählt aufwärts bis er überläuft und fängt dann bei Null wieder von vorn an. Beim Überlauf kann man ein internes Signal erzeugen lassen; nach aussen werden in dieser Betriebsart keine Signale abgegeben.

„Clear Timer On Compare Match" (CTC) - der Zähler zählt aufwärts bis er einen per Programm einstellbaren Wert erreicht, dann wird er Null zurückgesetzt und fängt von Neuem an, aufwärts zu zählen. Beim Gleichstand mit dem eingestellten Wert kann ein internes und ein externes Signal erzeugt werden, das an einem Beinchen ausgegeben wird.

Diese beiden Betriebsarten nimmt man gewöhnlich, um Zeitspannen abzumessen oder um einstellbare Frequenzen zu erzeugen, indem man den Zähltakt der Timer noch weiter herunterteilt.

„Fast PWM Mode" - der Zähler zählt mit dem vorgeteilten MC-Takt aufwärts bis er den Maximalstand erreicht und fängt dann wieder von Null an. Wenn er beim Aufwärtszählen einen vom Programm festgelegten Wert überschreitet, ändert sich das Signal an dem dem Timer zugeordneten Beinchen; zusätzlich lösst sich auch ein internes Signal erzeugen.

„Phase correct PWM Mode" - der Zähler zählt aufwärts bis er den Maximalstand erreicht und zählt dann wieder herunter. Jedesmal, wenn der Zählerstand gleich den vom Programm vorgegebenen Wert erreicht (also beim Auf- wie auch beim Abzählen), ändert sich das Signal am zugeordneten Beinchen.

„Phase and Frequency correct PWM Mode" - der Zähler zählt aufwärts bis er den Maximalstand erreicht und zählt dann wieder herunter. Jedesmal, wenn der Zählerstand gleich den vom Programm vorgegebenen Wert erreicht (also beim Auf- wie auch beim Abzählen), ändert sich das Signal am zugeordneten Beinchen; zusätzlich kann dabei auch ein internes Signal erzeugen lassen.

PWM steht für „Puls Width Modulation" (auf Deutsch „Pulsweiten Modulation" oder „variables Tastverhältnis"). Diese Betriebsart ist bestens geeignet, um extern angeschlossene Geräte mit einem „fast-analogen" Signal zu versorgen. Das können LEDs sein, die gedimmt werden sollen, Strombegrenzungen an Spannungsversorgungen für Motoren usw..

Es gibt noch eine besondere Betriebsart, in der Timer und ADC zusammenarbeiten:

„Input Compare Match" (Datenblatt Abs. 16.6, Seite 72) - hier wandelt der ADC eine externe Spannung (z.B. die vorverstärkte Spannung von einem Mikrophon) in einen Zahlenwert um. Der wird mit einem vom Programm eingestellten Wert verglichen. Der Timer zählt derweil den vorgeteilten MC-Takt. Wenn der Messwert den voreingestellten Wert überschreitet, wird der aktuelle Zählerstand in einen dafür reservierten Speicher geschrieben. Gleichzeitig wird der Zähler wieder auf Null gesetzt und fängt von Neuem an aufwärts zu zählen.

Mit dieser Betriebsart lassen sich sehr bequem die Zeiträume zwischen externen Signalen ausmessen, z.B. die Zeit zwischen Impulsen einer IR-Fernsteuerung oder die zwischen Impulsen eines Fernsteuerempfängers usw..

Timer/Counter 0 (Datenblatt Abs 14, Seite 71)
Beim ATmega16 kann dieser Timer nur von 0 bis 255 zählen. Wie alle Timer beim ATmega hat er einen eigenen einstellbaren Vorteiler

(Stufen 1, 8, 64, 256, 1024), mit dem er den MC-Takt herunterteilt, um den Zähltakt zu erzeugen. Externe Taktquellen kann er nicht verarbeiten. Er erlaubt die Betriebsarten „Normal", „CTC" und zwei PWM-Betriebsarten. Er kann auch ein externes Signal erzeugen, das am Beinchen Nr. 4 ausgegeben wird.

Timer/Counter 1 (Datenblatt Abs 16, Seite 88)
Dieser Timer hat einen 16-bit-Zähler, kann also von 0 bis 65.535 zählen; er hat auch den eigenen, einstellbaren Vorteiler (Stufen 1, 8, 64, 256, 1024). Mit den Betriebsaren „Normal", „CTC", drei PWM-Betriebsarten, dem „Input Compare Match" und der Möglichkeit, externe Taktquellen zu verwenden, ist er besonders vielseitig einsetzbar. Dieses Teil ist ein echtes Goldstück: Dank des 16-Bit-Zählers kann man Frequenzen in Schritten von 1 Promille einstellen. In den PWM-Betriebsarten kann man das Tastverhältnis in 1024 Schritten variieren, d.h. man kann die extern angeschlossenen Geräte sehr feinfühlig verstellen. Deshalb eignet sich dieser Zähler auch hervorragend als Ausgang für Software-Regler. Obendrein hat er zwei Ausgänge (an Beinchen Nr. 18 und 19), die in den Betriebsarten „CTC" und in den PWM-Betriebsarten von einander unabhängige Siganle erzeugen und ausgeben können.

Timer/Counter 2 (Datenblatt Abs. 17, Seite 117)
Dies ist noch ein 8-Bit-Timer, sehr ähnlich dem Timer0: Er hat dieselben Betirebsarten, kann sein Signal über Beinchen Nr. 21 auseben. Sein Vorteiler hat eine andere Stufung als die der der beiden anderen Timer (Stufen 1, 8, 32, 64, 128, 256 und 1024). Und das hat seinen Grund in der wichtigsten Besonderheit dieses Timers: Wenn er kann einen externen Takt asynchron verarbeiten. Bei den anderen beiden Timern werden die externen Taktimpulse erst einmal mit dem MCU-Takt synchron gemacht - deshalb können diese Timer keine externen Frequenz verarbeiten, die schneller sind als der MCU-Takt. - Das ist beim Timer2 anders. An seine externen Eingänge (Beinchen Nr. 28, TOSC1 und Beinchen Nr. 29, TOSC2) kann man auch Quarze mit mehr als der MCT-Taktfrequenz anschliessen, z.B. einen mit 32,768 MHz (also doppelt so schnell wie der maximale MCU-Takt des ATmega16). Das ist der richtige Timer für alle, die unbedingt einen Sekundensynchronen Takt brauchen. Allerdings ist diese Betriebsart mit besonderen Vorkehrungen zu werwenden, die im Datenblatt ausführlich vorgestellt werden (Abs. 17.9, Seite 128).

UART (Datenblatt Abs. 19, Seite 146)
Über die UART („Universal Asynchronous receiver/Transmitter") bietet eine Möglichkeit, mit dem ATmega16 zu kommunizieren. Die Schnittstelle verfügt über einen einstellbaren Baudratengenerator (angeblich bis 250 kBaud); bis 57 KBaud habe ich ihn schon problemlos betrieben. Sehr praktisch ist auch, dass der Empfänger- und der Senderkanal voneinander unabhängig sind. Man kann also auch dann Daten senden, wenn auf der anderen Leitung gerade welche einlaufen („full duplex"-Betrieb). Diese Schnittstelle kann man, mit entsprechender externer Zusatzschaltung (z.B. MAX232 oder FTDI256) entweder über eine serielle PC-Schnittstelle oder über den USB-Bus ansprechen. Sehr nützlich!

TWI (Datenblatt Abs. 20, Seite 175)
Diese Schnittstelle heisst zwar bei Atmel TWI („Two Wire serial Interface") ist aber funktionsidentisch mit der I2C-Schnittstelle von Philips; die andere Namensgebung ist wohl der Tribut an das Lizenzrecht. Jedenfalls kann man die I2C-Bausteine von Philips und anderen Herstellern problemlos an dieser Schnittstelle betreiben. Besonders interessant wird diese Schnittstelle, wenn man sie benutzt, um mehrere ATmegas „miteinander reden" zu lassen. Man kann nämlich jedem ATmega eine eigene Adresse geben, mit der er dann auf dem TWI-Bus gezielt angesprochen werden kann. In den Foren werden über den TWI-Bus sehr unterschiedliche Meinungen vertreten. Man liest, man könne ihn nicht über Distanzen von einem halben Meter hinaus betreiben, er sei zu langsam und was weiss ich noch für Ammenmärchen. Ich kann aus meiner Erfahrung nur sagen, dass sich der TWI-Bus in allen meinen Anwendungen bestens bewährt hat, auch wenn das Kabel 5m lang ist. Das Problem liegt wohl für viele in der Programmierung, obwohl es Atmel einem wirklich leicht macht, indem die ATmegas zu jedem möglichen Zustand des Busses und der Schnittstelle ein Zustandsbyte liefern. Wenn man das sorgfältig auswertet, und ein geeignetes Protokoll verwendet, dann ist das Benutzen der TWI-Schnittstelle das reine Vergnügen.

SPI (Datenblatt Abs. 18, Seite 136)
Dies ist eine sehr einfache Schnittstelle, die gern benutzt wird, um ohne spezialisierte Bauteile (oft wird ein einfacher 74 XX 595 verwendet) und mit nur drei Leitungen Daten mit der Umgebung auszutauschen. Die andere Eigenschaft, die oft zugunsten dieser Schnittstelle in den Foren hervorgehoben wird, ist die hohe Geschwindigkeit. Man kann mit dem Vorteiler Bitraten von bis zum halben MCU-Takt einstellen. Da kommt der TWI-Bus mit seiner Bitrate von gerade einmal 400 kHz natürlich bei weitem nicht mit. Dafür muss aber viel selber programmieren, was beim TWI-Bus schon automatisiert ist; z.B. die Adresserkennung. Ich habe den SPI-Bus bisher nicht verwendet und kann mir kein wirkliches Urteil erlauben.

ADC (Datenblatt Abs. 22, Seite 207)
Der ADC („Analogue to Digital Converter") misst eine Eingangsspannung und stellt den Messwert als Binärzahl dar. Dazu braucht er eine Festspannung, mit der er die Messspannung vergleicht. Man kann sie entweder am Beinchen AREF (Beinchen Nr. 32) extern bereitstellen oder eine intern erzeugte FestSpannung wählen. Die Messspannung kann an einem von 7 Eingängen (Beinchen Nr. 33 bis 40) angelegt werden; der ADC hat einen Multiplexer (eine 1-aus-7 Auswahlschaltung), mit der man programmgesteuert einen der 7 Eingänge auswählen kann. Das ist sehr praktisch, weil man ohne externe Schaltkreise praktisch 7 Messgeräte hat. Zwischen Einigen der Eingänge („differential inputs") kann man sogar, in Grenzen, negative Spannungen messen. Die Messergebnisse können zwischen 8 und 10 Bit haben; d.h. die kleinste noch messbare Spannungsänderung beträgt 1/1023-tel der Vergleichsspannung. Jede Messung dauert normalerweise 14 ADC-Takte, nur die erste Messung, direkt nach dem Einschalten des ADC braucht ca. 39. Zum Einstellen der ADC-Taktfrequenz gibt es wieder einen Vorteiler. Je nach dessen Einstellung, der gewünschten Auflösung (8 oder 10 Bit) und der Höhe der MCU-Taktfrequenz kann man für eine normale Messung (nicht die erste) zwischen 195us (us steht für Mikrosekunden) bis hinunter zu 2us (8Bit Auflösung, 16MHz MCU-Taktfrequenz und Vorteiler 2) veranschlagen. Menschliche Sprache spielt sich zwischen ca. 1000Hz und 10kHz ab; um sie zu digitalisieren, muss man mindestens alle 50us eine Messung machen. Wie man sieht, kann ein ATmega da spielend mithalten. Ich habe den ADC auch schon dazu benutzt, problemlos Ultraschallsignale (40kHz) zu digitalisieren.

Interner Taktgeber (Datenblatt Abs. 8.7, Seite 29)
Der MCU-Takt ist so etwas wie der Herzschlag eines Prozessors. Er gibt die Grundgeschwindigkeit jeder Aktivität vor, die auf dem Chip stattfindet. Im Datenblatt gibt Abs. 8, Seite 24 einen schönen Überblick darüber, wie der der Takt an alle Baugruppen des Chips verteilt wird. Es gibt eine Reihe von Möglichkeiten, einen ATmega mit dem MCU-Takt zu versorgen. Die einfachste ist der interne Taktgeber („calibrated internal RC-oscillator"). Er kann Taktfrequenzen von 1MHz, 2MHz, 4MHz oder 8MHz liefern. Der Vorsatz „RC" bedeutet, dass seine Frequenz von einem auf dem Chip befindlichen Widerstand („R") und einer Kapazität („C") vorgegeben werden. Nun fallen diese Bauteile wegen der Fertigungstoleranzen nicht immer gleich aus. Um das auszugleichen, gibt es die Möglichkeit, sie zu kalibrieren. Bei Auslieferung sind die Grundwerte dieser Kalibrierungs-Bytes schon angepasst. Der interne Taktgeber ist zwar praktisch, hat zwei Nachteile: Zum einen ändern sich der Widerstand und die Kapazität mit der Chiptemperatur. D.h. es ändern sich auch alle davon abgeleiteten Frequenzen, u.a. auch die Baudrate der UART. Es gibt in den Foren Berichte, dass die Veränderungen so gross waren, dass die Kommunikation über die UART gestört bis unterbrochen war. Der zweite Nachteil ist noch heimtückischer. Wenn man nicht gut aufpasst, kann es einem passieren, dass man aus Versehen einen externen Oszillator als Taktquelle auswählt, obwohl gar keiner angeschlossen ist. Der Prozessor bekommt dann gar keinen Takt mehr und bleibt unansprechbar. Man kann ihn also auch nicht mehr umprogrammieren - es sei denn, man kann ihn aus seiner Schaltung herausnehmen (z.B. wenn er in einem Sockel steckt) und in eine Hilfsschaltung verpflanzen, die einen externen Taktgeber bereitstellt. Ansonsten, besonders bei der SMD-Ausführung, hilft nur auslöten, und in die Hilfsschaltung einlöten, umprogrammieren, wieder auslöten und wieder in die Anwendungsschaltung einlöten (und beten, dass er die Tortur übersteht!). - Kurz, es ist sicherer, dem Fehler gleich einen Riegel vorzuschieben, indem man von vornherein eine externe Taktquelle vorsieht. So habe ich's mir jedenfalls angewöhnt.

Port A, Port B, Port C, Port B (Datenblatt Abs. 12, Seite 49)
Dies sind die am meisten gebrauchten Baugruppen des ATmegas, weil sie es ermöglichen, Einsen (Versorgungsspannung) und Nullen (Massepotential) aus der MCU nach aussen weiterzugeben und sie auch umgekehrt von aussen in die MCU einzulesen. Jeder der „Port" hat 8 Kanäle, die programmgesteuert als Aus- oder Eingänge eingestellt werden können. Port A, z.B. benutzt die Beinchen Nr. 33 bis Nr. 40.

So, damit ist der erste Rundgang durch den ATmega16 beendet. Dieser Überblick sollte zeigen, welche reiche Auswahl an Baugruppen und Fähigkeiten der ATmega16 bietet. Damit kann man eine Vielfalt von vielseitigen Anwendungen verwirklichen. Weil es so wenige Beinchen und so viele Baugruppen sind, müssen sich manche von ihnen ein Beinchen teilen. Z.B. kann man Beinchen Nr. 14 und 15 als Ein-Ausgangskanal Nr. 0 und 1 von Port D verwenden. Dann kann man aber die UART nicht benutzen, weil dieselben Beinchen auch als RX-Eingangs- bzw. TX-Ausgangskanal dienen. Man muss sich also für *eine* Nutzung entscheiden. Es spricht aber nichts dagegen, Beinchen Nr. 14 und 15 für die UART zu reservieren, jedoch die Beinchen Nr. 16 bis 21 als die Kanäle 2 bis 7 von Port D und als Ein-Ausgabekanäle zu verwenden. Es ist jedenfalls immer der wichtigste Schritt am Beginn eines Projektes, sich genau zu überlegen, welche Baugruppen man benutzen will und ob dann für die Ein-Ausgabe noch genügende Kanäle (sprich: Beinchen) zur Verfügung stehen.