

Burkhard Kainka

Lernpaket Mikrocontroller

Vorwort

Mikrocontroller sind nichts anderes als vollständige kleine Computer mit Recheneinheit, Speicher, Schnittstellen und allem was sonst noch dazu gehört. Die neuere Entwicklung hat dazu geführt, dass immer mehr in einen kleinen Chip gepackt wurde. Ein achtbeiniges IC wie der im Lernpaket enthaltene ATTINY13 enthält bereits so viele Möglichkeiten, dass es praktisch unmöglich ist alle zu nutzen.

Entwickeln Sie Ihre eigenen Anwendungen und damit praktisch Ihr eigenes Spezial-IC. Sei es eine spezielle Alarmanlage, ein Messgerät oder eine Robotersteuerung, mit den entscheidenden Grundkenntnissen können Sie Ihre Ideen umsetzen. Die im Lernpaket enthaltene Hardware ist zugleich Entwicklungsplattform und Programmiergerät. Sie können also weitere Mikrocontroller programmieren und dann in Ihre Schaltungen einbauen.

Vor nicht allzu langer Zeit war die Arbeit mit Mikrocontrollern eine aufwändige Sache. Da musste man für jeden Versuch ein EPROM programmieren und in die Schaltung einsetzen. Jede Programmänderung machte es erforderlich, den Speicher aus dem Sockel zu nehmen, mit UV-Licht zu löschen, in einem Programmiergerät neu zu programmieren und wieder in die Schaltung einzusetzen. Insbesondere durch die Einführung des in den Mikrocontroller integrierten Flash-Speichers ist dagegen die Programmierung sehr bequem geworden. Nun bleibt der Mikrocontroller in der Schaltung und wird in wenigen Sekunden elektrisch gelöscht und neu programmiert. Da macht es richtig Spaß, ein Programm in vielen kleinen Schritten zu entwickeln und zu testen.

Für die erfolgreiche Arbeit mit dem Lernpaket sind Englischkenntnisse sehr hilfreich, wenn auch nicht unverzichtbar. Es ist nicht mehr zu übersehen, dass Englisch die Sprache der Elektronik ist. Datenblätter zu Mikrocontrollern sind selbst dann in Englisch verfasst, wenn sie von deutschen Herstellern stammen. Auch Hilfedateien und Anleitungen für professionelle Programmierwerkzeuge gibt es nur noch in englischer Sprache. In diesem Handbuch werden jedoch die entscheidenden Begriffe so erläutert, dass Sie die wichtigsten Fakten auch ohne Englischkenntnisse nachvollziehen können.

Inhalt

1	Einleitung	4
1.1	Bauteile	4
1.2	Die Schaltung	5
1.3	Aufbau	7
1.4	Software	9
2	Die ersten Experimente	11
2.1	Überprüfung der Stromversorgung	11
2.2	Initialisierung	12
2.3	Oszillator-Abgleich	14
3	Interface-Funktionen	16
3.1	Nutzung der Ports als Ausgänge	16
3.2	Ports als Eingänge	18
3.3	Spannungsmessung	20
3.4	Lichtmessung mit Fototransistor	24
3.5	Das Oszilloskop	24
3.6	Der PWM-Ausgang	27
3.7	Schaltschwellen	31
3.8	Pullup-Widerstände	32
3.9	Programm-Upload	33
4	Assembler-Grundlagen	36
4.1	Das AVR Studio	36
4.2	Ausgangsports	40
4.3	Bits und Bytes	43
4.4	Ein Blinkprogramm	44
4.5	Unterprogramme	45
4.6	Geschwindigkeitstest	46
4.7	Digitale Eingänge	48
4.8	Die UND-Funktion	50
4.9	Die Oder-Funktion	52
4.10	Das RS-Flipflop	53
4.11	Das D-Flipflop	54
4.12	Das Toggle-Floplop	55
5	Die serielle Schnittstelle	57
5.1	Übertragungsparameter	57
5.2	Daten-Echo	58
5.3	Empfangen und Senden	59

5.3 RS232-Testprogramm	62
5.4 Automatische Baudratenerkennung	63
6 Der Timer/Counter	67
6.1 Zeitmessung	67
6.2 Impulse zählen.....	69
6.3 Timer-Interrupt.....	71
6.4 Minuten-Timer	72
6.5 PWM-Ausgang.....	74
6.6 Der weiche Blinker.....	75
6.7 Frequenzmessung	76
7 Der AD-Wandler	79
7.1 10-Bit-Messung.....	79
7.2 8-Bit-Messung.....	82
7.3 Interne Referenz	83
7.4 Zweipunktregler	83
7.5 Dämmerungsschalter	85
7.6 Alarmanlage	87
8 Datenspeicher	89
8.1 Das RAM.....	89
8.2 Speicheroszilloskop.....	92
8.3 Das EEPROM	94
9 Das Interfaceprogramm.....	97
9.1 Interpreterschleife.....	97
9.2 Ein- und Zweikanal-Oszilloskop.....	99
9.3 Oszillator-Kalibrierung	100
9.4 Der Bootloader	102
10 Bascom-AVR	106
10.1 Blinkprogramm	106
10.2 RS232 und AD-Wandler	108
10.3 Ein Datenlogger.....	111
11 C-Programmierung.....	114
11.1 Win-AVR	114
11.2 Das erste C-Projekt.....	114
12 Ein Programmiertool	119
12.1 ISP-Upload	119
12.2 Fuses.....	120

1 Einleitung

Vor den eigentlichen Experimenten mit dem Mikrocontroller müssen Sie den Bausatz zusammenlöten und die Software installieren. Diese Vorbereitungen sind jedoch schnell erledigt. Falls es zu Problemen beim Aufbau der Platine kommt oder falls etwas kaputt geht, können Sie bei der im Anhang angegebenen Firma Hilfe finden.

1.1 Bauteile

Das Lernpaket enthält einen Platinenbausatz mit allen erforderlichen Bauteilen. Bitte überprüfen Sie den Inhalt und stellen sie sicher, dass Sie die Bauteile korrekt zuordnen.

- 1 Platine
- 1 IC Sockel DIL24
- 1 Mikrocontroller ATtiny13
- 1 DB9-Buchse
- 1 5-V-Spannungsregler 2950
- 2 Dioden 1N4148
- 1 9-poliges Kabel
- 2 keramische Kondensatoren 100 nF (Aufdruck 104)
- 1 Elektrolytkondensator 47 μ F
- 1 Fototransistor (im klaren LED-Gehäuse)
- 1 grüne LED
- 2 Widerstände 100 k Ω (braun, schwarz, schwarz, orange, braun)
- 1 Widerstand 10 k Ω (braun, schwarz, schwarz, rot, braun)
- 2 Widerstände 1 k Ω (braun, schwarz, schwarz, braun, braun)

Nach der Bestückung der Platine bleiben noch fünf Bauteile (Widerstände 1 k Ω und 10 k Ω , Elektrolytkondensator 47 μ F, LED und Fototransistor) übrig, die für Experimente verwendet werden. Bitte sortieren Sie diese Bauteile vor dem Löten der Platine aus.

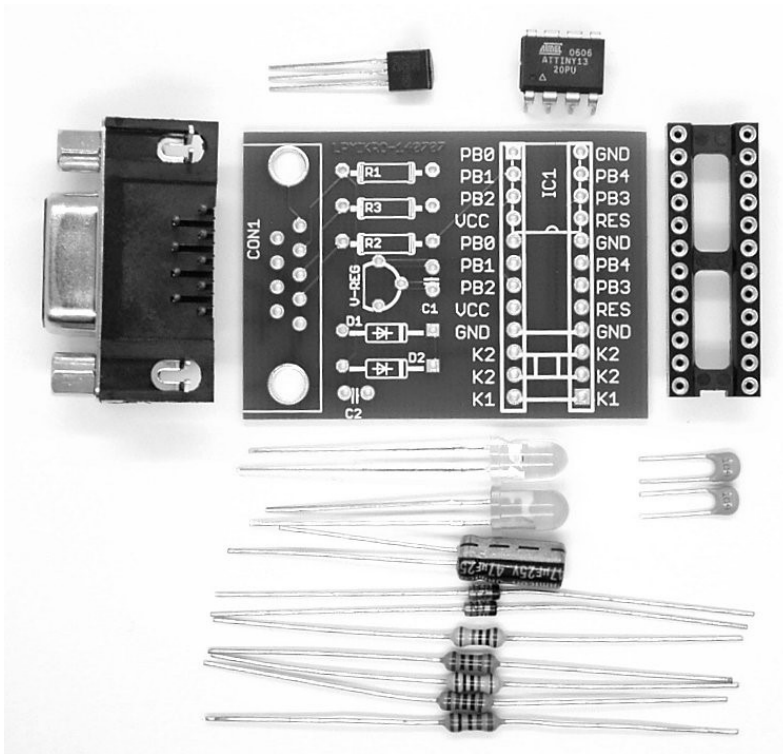


Abb. 1.1 Alle Bauteile im Lernpaket ((Bauteile1.jpg))

1.2 Die Schaltung

Die vollständige Schaltung des Entwicklungssystems sehen Sie in Abb. 1.2. Der Mikrocontroller ATtiny13 ist ein IC im achtbeinigen DIP-Gehäuse. Alle Anschlüsse sind an zusätzliche Sockelbuchsen geführt und können in den Versuchen mit anderen Bauteilen verbunden werden. Acht weitere Anschlüsse des Sockels stellen ein kleines Experimentierfeld dar. Hier können die zusätzlichen losen Bauteile eingesteckt werden. Der insgesamt 24-polige IC-Sockel dient also sowohl zu Aufnahme des Controllers als auch als Stecksystem.

Außerdem kann die Verbindung zwischen PC und Controller auch statisch genutzt werden. Zwei Leitungen dienen dann als Eingänge des Mikrocontrollers. Der PC legt 1- oder 0-Zustände an, die ein Controllerprogramm über die Anschlüsse PB0 und PB2 lesen kann. Umgekehrt kann über PB1 ein Zustand ausgegeben werden, den der PC über CTS liest.

1.3 Aufbau

Vor dem ersten Versuch müssen Sie zunächst die Experimentierplatine zusammenbauen. Grundlegende Elektronik- und Lötkenntnisse werden vorausgesetzt. Verwenden Sie einen passenden LötKolben mit dünner Spitze.

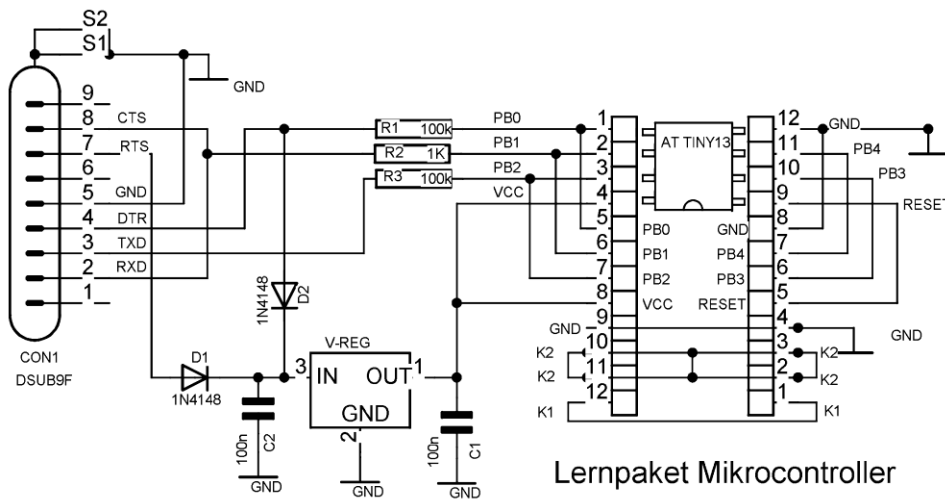


Abb. 1.3 Die Schaltung im Detail ((Schaltbild.tif))

Beginnen Sie mit der DB9-Buchse und achten Sie darauf, sie von der Bestückungsseite einzusetzen. Bei den Dioden kommt es auf die Richtung an, die durch einen schwarzen Ring markiert ist. Die Widerstände dürfen nicht verwechselt werden. Die verwendeten Metallschichtwiderstände sind mit vier Farbringen und einem fünften braunen Ring für die Toleranz 1% gekennzeichnet. Auf der Platine werden zwei Widerstände mit 100 k Ω (braun, schwarz, schwarz, orange, braun) und einer mit 1 k Ω (braun, schwarz, schwarz, braun, braun) verwendet. Für die späteren Versuche bleiben zwei Widerstände übrig: 1 k Ω und 10 k Ω (braun, schwarz, schwarz, rot, braun). Beim Aufbau können Sie sich auch am Foto der

fertigen Platine orientieren (Abb. 1.5). Überlange Drähte der Bauteile werden nach dem Löten abgeschnitten. Heben Sie die Drähte auf, denn sie lassen sich als Drahtbrücken in den Experimenten verwenden. Besonders die Drähte der Dioden sind gut geeignet, weil sie besonders hart und dünn sind.

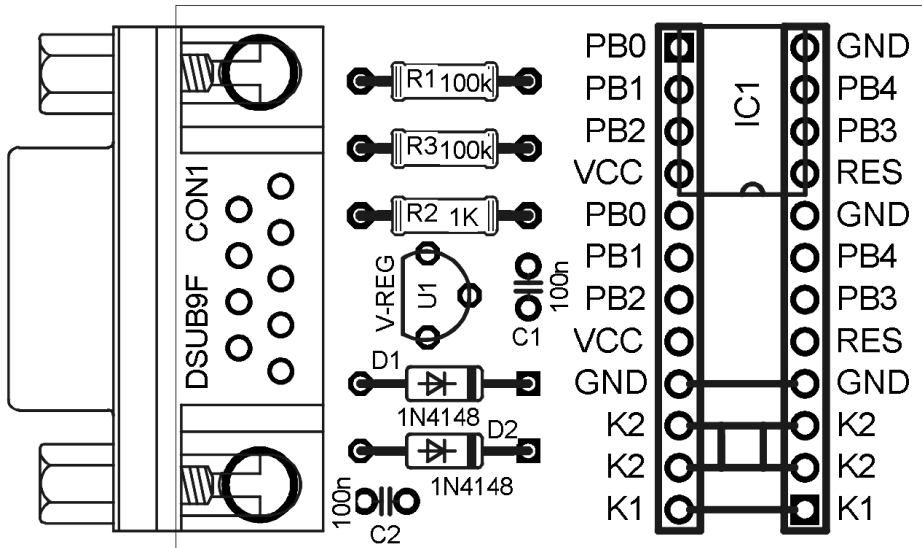


Abb. 1.4 Der Bestückungsplan ((Bestückung.tif))

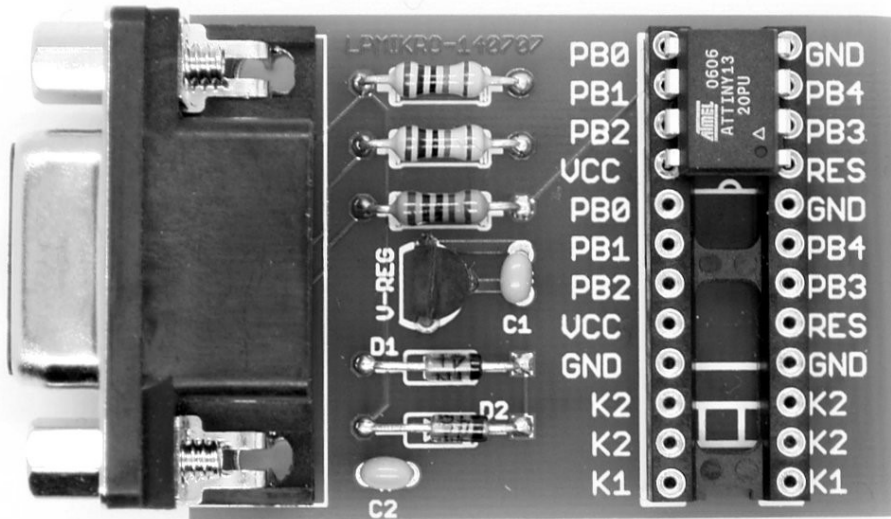


Abb. 1.5 Die fertig bestückte Platine ((Bauteile2.jpg))

1.4 Software

Zur Installation der Software muss das Anwenderprogramm zusammen mit einigen Daten auf die Festplatte kopiert werden. Starten Sie dazu das selbst entpackende Archiv LPmicro_kopieren.exe. Es erzeugt ein Verzeichnis Franzis/LPmikro auf der Festplatte C: und kopiert dort hinein die nötige Software. Die Dateien sollten genau an diesem Ort liegen, weil die Assembler-Projektdateien feste Bezüge zu den Quelltexten enthalten.

Starten Sie das Programm LPmikro.exe. Der Startbildschirm zeigt das Schaltbild der Hardware. Bei der späteren Arbeit können Sie immer wieder einmal auf die Registerkarte „Start“ wechseln um dort nachzusehen, welcher Anschluss wo zu finden ist.

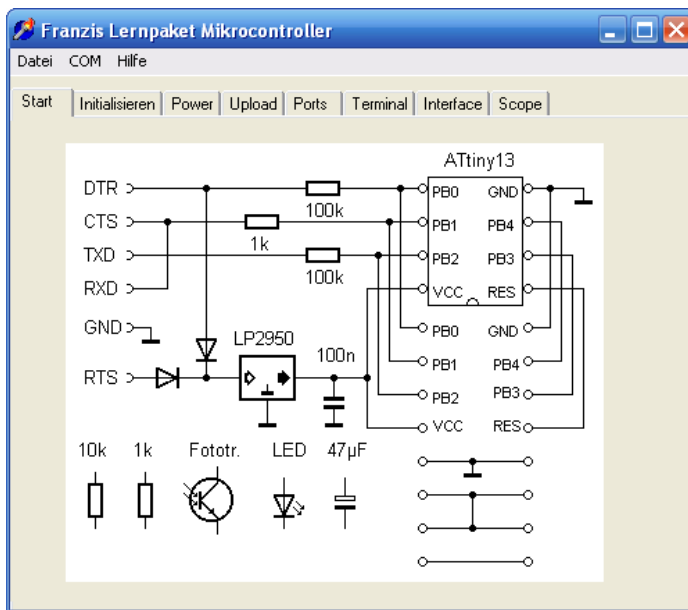


Abb. 1.6 Der Startbildschirm ((Soft1.tif))

Nun muss noch die verwendete serielle Schnittstelle eingestellt werden. Klicken Sie auf COM und dann z.B. auf COM1. Wenn die Schnittstelle vorhanden ist und geöffnet werden kann, erscheint hier ein Häkchen. Beim Beenden des Programms wird die verwendete Schnittstelle in der Datei Lpmikro.ini gespeichert. Bei jedem neuen Start wird diese Einstellung wieder übernommen.

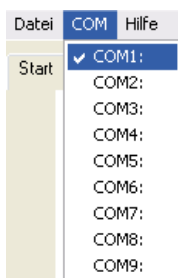


Abb. 1.7 Auswahl der Schnittstelle ((Soft2.tif))

2 Die ersten Experimente

Vor der Entwicklung eigener Programme muss der Mikrocontroller erst einmal initialisiert werden, indem ein passendes Ladogramm übertragen wird. Dieser so genannte Bootloader wird verwendet um Programme bequem und zuverlässig in den Mikrocontroller zu laden. Zusammen mit dem Bootloader wird bei der Initialisierung auch ein Interface-Programm geladen, das den direkten Zugriff auf Ein- und Ausgänge des Mikrocontrollers ermöglicht. Außerdem sind Zusatzfunktionen wie Spannungsmessung und ein Oszilloskop enthalten. Schon vor der ersten eigenen Programmierung können daher zahlreiche Eigenschaften und Möglichkeiten des Mikrocontrollers in praktischen Versuchen ausprobiert werden. Die dahinter stehenden Programmteile werden dann weiter unten genauer behandelt.

2.1 Überprüfung der Stromversorgung

Der erste Test kann noch ohne den eingesetzten Mikrocontroller erfolgen, da es zuerst nur um die Überprüfung der Platine geht. Testen Sie zunächst die Stromversorgung. Wählen Sie dazu die verwendete Schnittstelle aus. Damit wird automatisch die Stromversorgung eingeschaltet. Dahinter verbirgt sich das Einschalten der Leitung RTS an der seriellen Schnittstelle. Verwenden Sie ein Multimeter um die Spannung zu überprüfen. Am Anschluss VCC sollte gegen GND eine Spannung von 5 V gemessen werden.

Alternativ zum Einsatz eines Messgeräts können Sie auch die LED verwenden. Bauen Sie dazu eine Testschaltung mit der LED und einem Vorwiderstand von 1 k Ω auf. Beim Einschalten der Spannung sollte die LED leuchten.

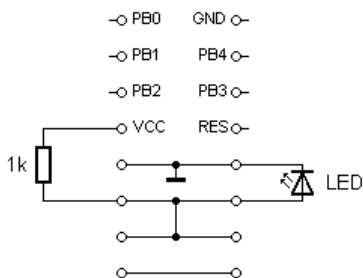


Abb. 2.1 Überprüfung der Betriebsspannung ((Schalt1.gif))

Auf der Registerkarte Power können Sie die Betriebsspannung ein- und ausschalten. Es gibt zwei Leistungsstufen. Mit RTS=1 ist die normale Versorgung eingeschaltet. Mit RTS=1 und DTR=1 steht mehr Strom zur Verfügung, was jedoch in diesem Fall nicht nötig ist. Die

Betriebsspannung am Anschluss VCC bleibt 5 V, die LED leuchtet also mit gleicher Helligkeit.

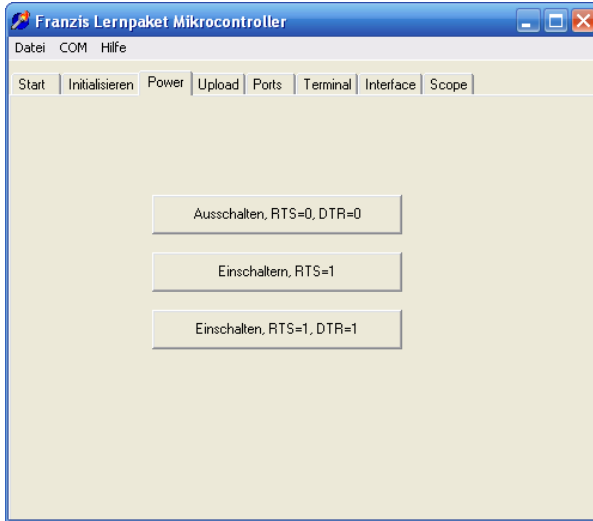


Abb. 2.2 Ein- und Ausschalten der Versorgungsspannung ((Soft3.tif))

Falls Sie keine Versorgungsspannung feststellen können, geht es an die Fehlersuche. Mögliche Ursachen sind ein Irrtum in der verwendeten COM-Schnittstelle oder ein Fehler beim Bestücken der Platine. Überprüfen Sie insbesondere die Einbaurichtung der Dioden.

Wenn die Stromversorgung gesichert ist, können Sie bei abgeschalteter Spannung den Mikrocontroller ATtiny13 einsetzen. Beachten Sie genau die Richtung und Position auf dem Sockel. Der Pin 1 des ICs ist mit einem eingepressten Punkt gekennzeichnet und liegt am Anschluss Reset (RES).

2.2 Initialisierung

Nun soll das Bootprogramm in dem Mikrocontroller geladen werden. Dieser Vorgang verwendet die ISP-Programmierung über die SPI-Schnittstelle im Reset-Zustand des Mikrocontrollers. Programmdaten werden seriell mit einem Taktsignal übertragen.

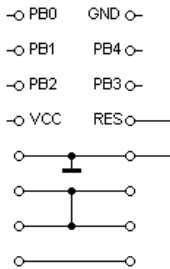


Abb. 2.3 Initialisierung im Reset-Zustand ((Schalt2.gif))

Setzen Sie eine Drahtbrücke zwischen RES und GND. Verwenden Sie dazu eines der Drahtstücke, die beim Bestücken übrig geblieben sind. Starten Sie dann die Initialisierung mit einem Klick auf die Schaltfläche „Bootloader und Fuses“. Nun wird das Programm Init.hex geladen und in das Flash-ROM des Mikrocontrollers gebrannt und der Controller vorbereitet. Bei Erfolg erhalten Sie eine ok-Meldung. Falls ein Fehler angezeigt wird, überprüfen Sie bitte ob der Controller korrekt eingesetzt wurde und die Reset-Brücke vorhanden ist.

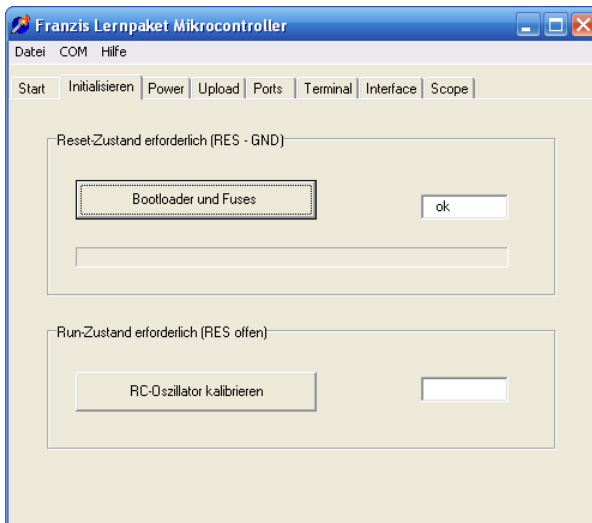


Abb. 2.4 Die Initialisierung ((Soft4.tif))

Der Fortschritt des Ladevorgangs wird mit einem Balken (Progress-Bar) angezeigt und kann einige Sekunden dauern. Falls Sie einen USB/Seriell-Adapter verwenden, kann dieser einmalig notwendige Vorgang mehrere Minuten dauern. Die spätere Selbst-Programmierung des Controllers wird dann wesentlich schneller gehen, sodass bei der Entwicklung und

Erprobung eigener Programme auch mit einem USB-Adapter keine größeren Wartezeiten mehr anfallen.

Das geladene Programm Init.hex umfasst zwei Teile und füllt den vorhandenen Programmspeicher fast vollständig. Insgesamt verfügt der Controller über 1 KB Flash. Das mag als sehr wenig erscheinen. Sie werden jedoch sehen, dass man auch mit wenig Speicher sehr viel erreichen kann. Das obere Viertel des Flash enthält nun den Bootloader. Dieses Programm nimmt Daten von der seriellen Schnittstelle mit 9600 Baud entgegen und programmiert damit den unteren Teil. Das hat zu einem Vorteil, dass der Upload-Vorgang vollautomatisch erfolgen kann und man keine Reset-Brücke mehr zu setzen braucht. Zum anderen ergibt sich insbesondere bei der Verwendung eines USB-Adapters ein Geschwindigkeitsvorteil.

Die genaue Funktion des Bootloaders muss man nicht kennen, um erfolgreich mit dem System zu arbeiten. Sie soll aber schon an dieser Stelle kurz genannt werden, nähere Einzelheiten finden Sie in Kap 9.4. Der ATtiny13 besitzt keinen ausgewiesenen Bootsektor, der nach einem Reset automatisch startet, wie dies z.B. bei den ATmega-Controllern der Fall ist. Deshalb wird von der Software beim Upload eines Programms grundsätzlich am Anfang ein Sprungbefehl auf das Bootprogramm eingefügt. Das Bootprogramm entscheidet dann jeweils je nach Zustand der seriellen Empfangsleitung und eventuell folgender serieller Kommandos ob ein vorhandenes Programm gestartet werden soll oder ein neues Programm geladen werden soll.

Das Bootprogramm und das Interface-Programm verwenden die serielle Schnittstelle mit einer definierten Übertragungsrate von 9600 Baud. Dazu ist es erforderlich, dass der Controller mit einer definierten Taktfrequenz arbeitet. Der ATtiny13 besitzt drei interne RC-Oszillatoren mit 128 kHz, 4,8 MHz und 9,6 MHz. Außerdem kann der Takt durch acht geteilt werden. Das Lernpaket arbeitet mit dem 9,6-MHz-Oszillator und dem Teiler, sodass der Prozessortakt 1,2 MHz beträgt. Damit wird ein geringer Betriebsstrom von unter 1 mA erreicht, der auch unter ungünstigen Bedingungen von der seriellen Schnittstelle problemlos geliefert werden kann. Die Taktrate wird automatisch bei der Initialisierung des Startprogramms eingestellt.

2.3 Oszillator-Abgleich

Ein RC-Oszillator ist grundsätzlich nicht so frequenzgenau wie ein Quarzoszillator. Der interne Oszillator des ATtiny13 verfügt daher über eine Abgleichmöglichkeit. Intern gibt es ein OSCCAL-Register, dessen Inhalt über die genaue Frequenz entscheidet. Bei der Herstellung wurde ein Abgleich durchgeführt, der einen fest programmierten OSCCAL-Wert liefert. Bei jedem Neustart wird das OSCCAL-Register automatisch mit diesem individuellen Wert geladen. Damit erreicht der Oszillator laut Datenblatt eine Abweichung von unter 10%. Tatsächlich ist er meist wesentlich genauer. Dies ist deshalb wichtig, weil die serielle Datenübertragung zwischen PC und Controller auf eine genaue Übertragungsrate angewiesen ist.

Die Genauigkeit des Oszillators lässt sich meist per Software noch verbessern. Entfernen Sie dazu die Reset-Brücke. Klicken Sie auf die Schaltfläche „RC-Oszillator kalibrieren“. Die Kalibrierung erfolgt automatisch durch einen Vergleich mit der bekannten Bitlänge der seriellen Schnittstelle im PC. Der Vorgang wird genauer in Kap. 9.3 erläutert.

3 Interface-Funktionen

Der untere Bereich des Flash enthält nach der Initialisierung ein Interface-Programm, mit dem man Ports, den AD-Wandler, den PWM-Ausgang und ein einfaches Oszilloskop verwenden kann. Diese Software wird im Folgenden verwendet um die grundlegenden Eigenschaften der Hardware zu erproben. Später wird das Interface-Programm dann durch eigene kleine Programme überschrieben. Sie können es aber jederzeit wieder in den Mikrocontroller laden, da es als Datei Interface.hex vorhanden ist.

3.1 Nutzung der Ports als Ausgänge

Der ATtiny13 hat acht Anschlüsse. Neben den Betriebsspannungsanschlüssen GND und VCC und dem Reset-Pin RES stehen fünf frei verwendbare Portanschlüsse zur Verfügung. Bei der Verwendung als Interface werden allerdings zwei Leitungen als Datenleitungen TXD und RXD zur seriellen Kommunikation mit dem PC gebraucht. Damit bleiben noch die drei Leitungen B0, B3 und B4 für sonstige Zwecke übrig. Jeder dieser Eingänge kann als Ausgang oder als Eingang verwendet werden. Außerdem haben sie jeweils noch Sonderfunktionen.

Öffnen Sie die Registerkarte Interface. Klicken Sie die Datenrichtungsbits `ddrb.0`, `ddrb.3` und `ddrb.4` aktiv. Damit sind alle drei Ports als Ausgänge initialisiert. Nun können Sie den jeweiligen Portzustand über die Kästchen `portb.0` bis `portb.4` umschalten. Einschalten liefert eine Spannung von 5 V am entsprechenden Pin, Ausschalten eine Spannung von 0 V. Verwenden Sie ein Voltmeter zur Überprüfung der Zustände. Gleichzeitig wird jeder Anschluss auch als Eingang gelesen. Der gelesene Zustand entspricht dem ausgegebenen Zustand, d.h. Sie können den realen logischen Zustand auch ohne eine Spannungsmessung erkennen.

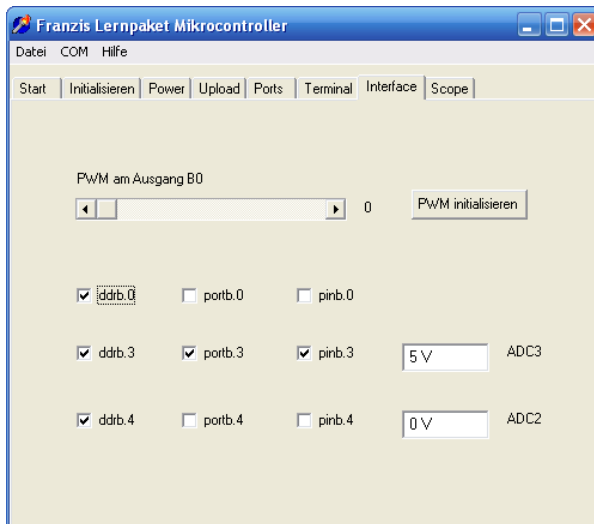


Abb. 3.1 Direkter Zugang zu den Prozessorports ((Soft5.tif))

Schließen Sie eine LED mit Vorwiderstand wie in Abb. 3.2 am Port PB3 an, die Sie dann per Software beliebig ein- und ausschalten können.

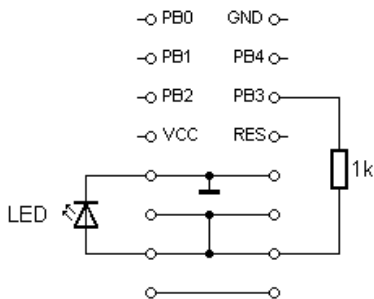


Abb. 3.2 Eine LED an Port B3 ((Schalt3.gif))

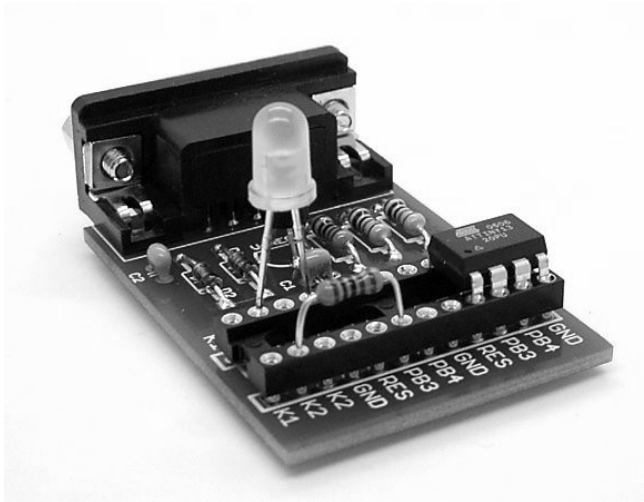


Abb. 3.3 Aufbau der LED-Schaltung ((Aufbau3.jpg))

Gleichzeitig mit der Portfunktion verwendet das Interfaceprogramm die beiden Eingänge B3 und B4 als analoge Eingänge. Sie können also direkt die tatsächliche Spannung am Port ablesen. Bei Belastung mit einer LED ergibt sich aufgrund des Innenwiderstands am Port eine geringfügig kleinere Ausgangsspannung als 5 V. Der digitale Zustand wird aber immer noch als 1 gelesen.

3.2 Ports als Eingänge

Schalten Sie alle `ddrb`-Bits und alle `portb`-Bits aus. Die Ports sind damit hochohmige CMOS-Eingänge. Berühren Sie die Eingänge mit einem Draht oder einem Widerstand. Dabei laden sie sich zufällig auf. Sie können 0 oder 1 sein oder ständig wechseln. Tatsächlich liefern Sie beim Berühren eines Eingangs meist eine 50-Hz-Brummspannung, also einen dauernden Zustandswechsel. Je nach dem zufälligen Zeitpunkt des Loslassens bleibt ein 1- oder ein 0-Zustand stehen, der sich jedoch nach kurzer Zeit allen wieder ändern kann.

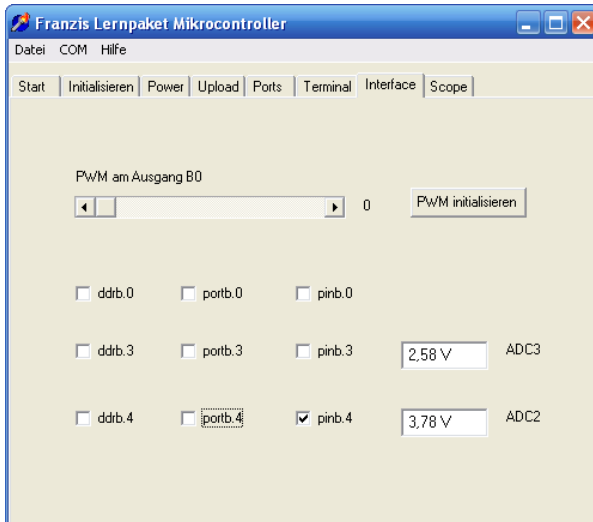


Abb. 3.4 Offene Eingänge ((Soft6.tif))

Allgemein werden offene Eingänge in der Digitaltechnik vermieden, eben weil sie keine definierten Zustände haben. Wenn ein Eingang z.B. verwendet werden soll um einen einfachen Schalter abzufragen, verwendet man zusätzliche Widerstände gegen Masse (pull down) oder gegen die Betriebsspannung (pull up). Sie können Pullup- oder Pulldown-Widerstände simulieren, indem Sie beim Berühren eines Eingangs gleichzeitig VCC oder GND berühren. Ihre Hand dient dann als Widerstand, der eine eindeutige Spannung an den Eingang legt.

Der ATtiny13 enthält aber auch interne Pullup-Widerstände, die sich bei Bedarf einschalten lassen. Dazu muss das jeweilige Port-Bit eingeschaltet werden, während das Datenrichtungsbit low ist. Schalten Sie die Bits portb.3 und portb.4 ein. Es werden dann bei offenem Eingang 1-Zustände zurück gelesen. Die entsprechenden Spannungen am Eingang betragen 5 V.

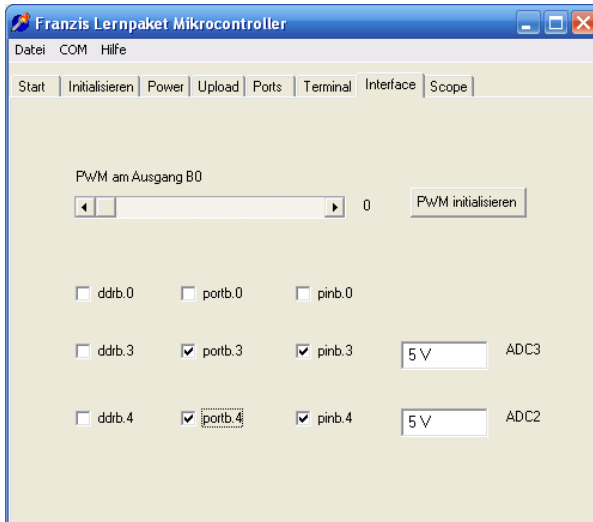


Abb. 3.5 Einschalten der internen Pullup-Widerstände ((Soft7.tif))

In diesem Zustand lassen sich externe Schalter gegen Masse abfragen. Ein geöffneter Schalter liefert 1, ein geschlossener Schalter 0. Verwenden Sie einen Draht nach GND zur Simulation eines geschlossenen Schalters. In diesem Fall können Sie auch den 1-k Ω -Widerstand als leitende Verbindung verwenden, weil die internen Pullups wesentlich hochohmiger sind.

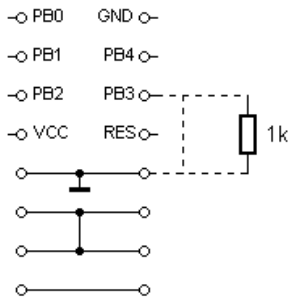


Abb. 3.6 Low-Zustand durch Drahtbrücke oder Widerstand ((Schalt4.gif))

3.3 Spannungsmessung

Die analogen Eingänge des Mikrocontrollers lassen sich auch für allgemeine Spannungsmessungen verwenden. Untersuchen Sie z.B. die Durchlassspannung der LED.

Ein Port wird dazu als Ausgang geschaltet, der zweite als hochohmiger Messeingang. Die LED in Abb. 3.7 wird über den Anschluss PB3 und einen Vorwiderstand von $1\text{ k}\Omega$ eingeschaltet. Der $10\text{-k}\Omega$ -Widerstand dient hier nur als Drahtbrücke zur Messung der LED-Spannung über den Anschluss PB4.

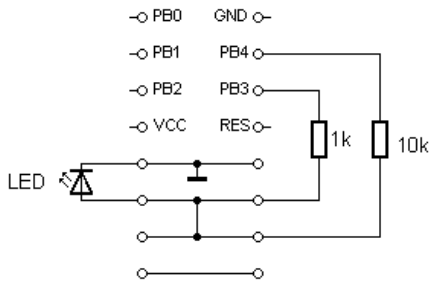


Abb. 3.7 Messung der LED-Spannung ((Schalt5.gif))

Im eingeschalteten Zustand finden Sie eine LED-Spannung von ca. $1,9\text{ V}$. Am Port liegt eine Spannung von $4,9\text{ V}$. Damit beträgt der Spannungsabfall am Vorwiderstand 3 V . Der LED-Strom ist also 3 mA . Gleichzeitig lässt sich der Innenwiderstand des Ausgangsports bestimmen. Bei einem Spannungsabfall von $0,08\text{ V}$ und einem Strom von 3 mA ergibt sich einen On-Widerstand von ca. $30\text{ }\Omega$.

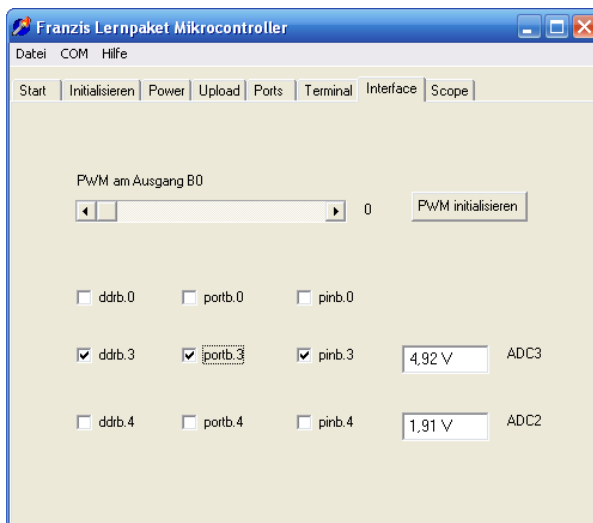


Abb. 3.8 LED-Spannung mit Vorwiderstand $1\text{ k}\Omega$ ((Soft8.tif))

Verwenden Sie nun einen Vorwiderstand von $10\text{ k}\Omega$, indem Sie PB4 als Ausgang und PB3 zur Spannungsmessung verwenden. Die LED leuchtet nur noch schwach, weil der LED-Strom nun nur noch etwa $0,3\text{ mA}$ beträgt. Die Spannung an der LED verringert sich aber nur geringfügig auf $1,8\text{ V}$. Dies ist auf die steile Diodenkennlinie der LED zurückzuführen.

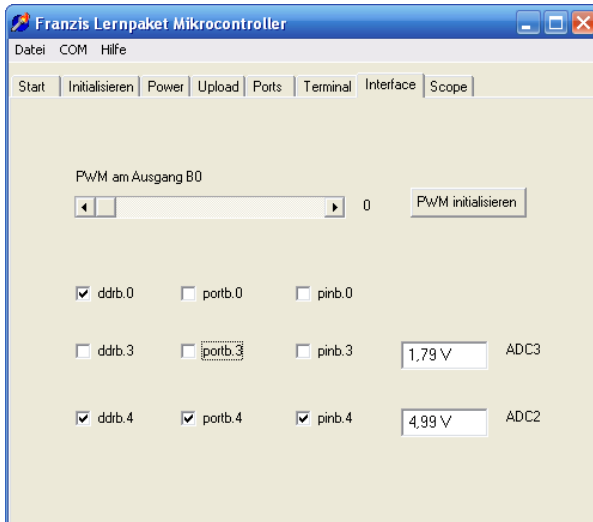


Abb. 3.9 LED-Spannung mit Vorwiderstand $10\text{ k}\Omega$ ((Soft9.tif))

Einen noch geringeren Strom liefert der interne Pullup des Ports. Schalten Sie das Datenrichtungsbit des Ausgangsports aus und das Portbit ein. Ein schwaches Leuchten der LED ist nur noch mit einer Abdunkelung des Umgebungslichts zu erkennen. Die LED-Spannung beträgt aber immer noch über $1,7\text{ V}$.

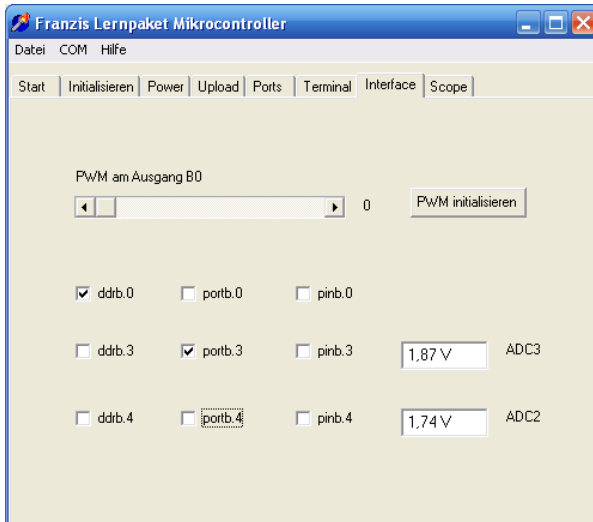


Abb. 3.10 Interner Pullup als Vorwiderstand ((Soft10.tif))

Übrigens ist eine LED zugleich auch eine Fotodiode, die bei Beleuchtung eine Spannung abgibt. Entfernen Sie den Vorwiderstand aus der Schaltung. Bei voller Beleuchtung wird eine Spannung von ca. 1,4 V gemessen. Mit einer Abschattung verringert sich die Spannung. Allerdings geht sie auch bei völliger Dunkelheit nicht auf Null zurück. Das ist auf die Aktivität des AD-Wandlers zurückzuführen, der während der Messung eine geringe Ladungsänderung am Port bewirkt. Das Datenblatt empfiehlt deshalb für genaue Messungen einen geringen Innenwiderstand des Messobjekts, der hier nicht gegeben ist.

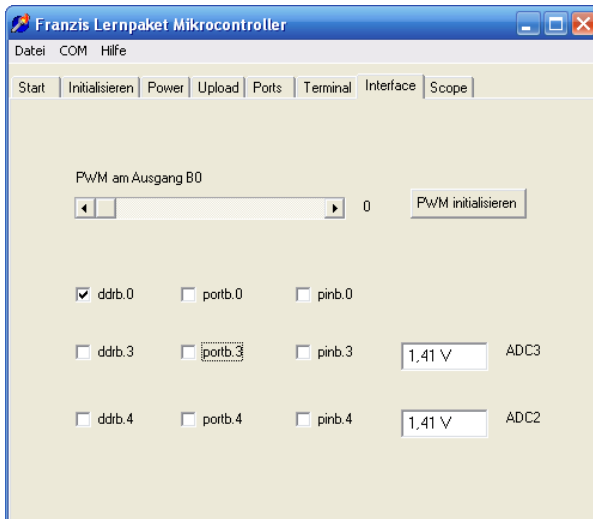


Abb. 3.11 Die LED als Spannungsquelle ((Soft11.tif))

3.4 Lichtmessung mit Fototransistor

Analoge Eingänge werden oft für Messungen an Sensoren eingesetzt. Ein Programm könnte z.B. die Helligkeit überwachen und bei einem bestimmten Grenzwert einen Schaltvorgang auslösen. Die entsprechende Schaltungstechnik können Sie mit dem Interface erproben. Schalten Sie den Fototransistor mit einem zusätzlichen Messwiderstand von 10 k an einen analogen Eingang. Der Spannungsabfall am Widerstand ist dann ein Maß für die Helligkeit.

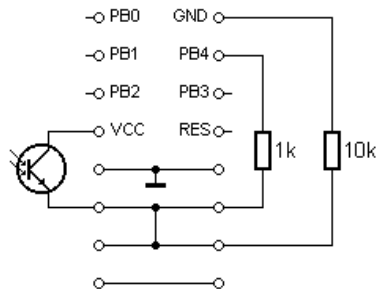


Abb. 3.12 Helligkeitsmessung mit dem Fototransistor ((Schalt6.gif))

Untersuchen Sie die Spannung bei verschiedenen Lichtverhältnissen. Bei künstlicher Beleuchtung können Sie ein Schwanken der Spannung beobachten. Die Helligkeit ändert sich je nach verwendeter Lampe mehr oder weniger stark im 100-Hz-Takt. Da der AD-Wandler bei jeder Messung die Momentanspannung ermittelt und der Zeitpunkt der Messung nicht synchron zum Lichtflackern ist, erhält man eine zufällige Schwankung, die aber insgesamt einen Eindruck von der Modulation des Lichts vermittelt. Die periodischen Schwankungen der Helligkeit müssen beachtet werden, wenn man die Helligkeit in einem Steuerprogramm auswertet. Eine Lichtschranke könnte z.B. unzuverlässige Ausgangszustände liefern, was durch geeignete programmtechnische Maßnahmen vermieden werden kann.

3.5 Das Oszilloskop

Das Interface enthält ein sehr einfaches Speicheroszilloskop, mit dem Änderungen einer Eingangsspannung untersucht werden können. Dazu werden 60 Messwerte in schneller Folge erfasst und im Arbeitsspeicher (RAM) des Controllers abgelegt. Nach der Messung sendet der Controller diese Werte an den PC, wo sie grafisch dargestellt werden.

Wahlweise kann eine Einkanalmessung oder eine Zweikanalmessung mit 30 Messwerten pro Kanal durchgeführt werden. Verwenden Sie die Einkanalmessung zur Untersuchung der Helligkeitsschwankungen einer künstlichen Beleuchtung.

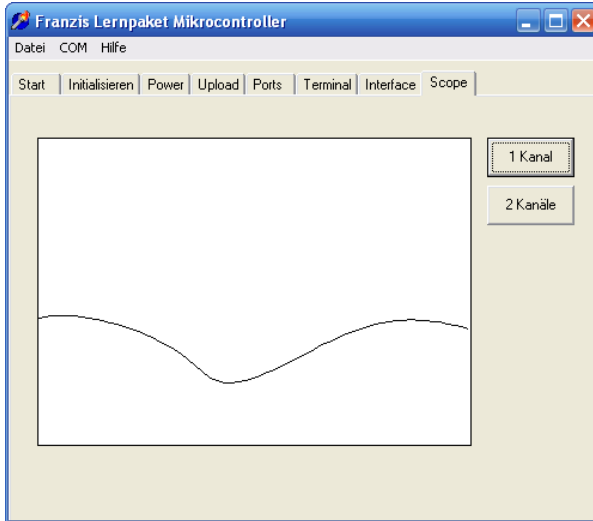


Abb. 3.13 Lampenflackern mit 100 Hz ((Soft13.tif))

Die eigentliche Messung mit dem Oszilloskop dauert nur etwa 12 ms. Sie sehen daher nur einen kleinen Ausschnitt der Helligkeitsschwankungen. Wiederholen Sie die Messung mehrfach um einen Eindruck vom Verlauf der Spannung zu erhalten. Damit können Sie auch schnellere Signale untersuchen. Abb.3.14 zeigt das Oszillogramm der Ausgangsspannung eines Sinusgenerators, der auf 400 Hz eingestellt wurde.

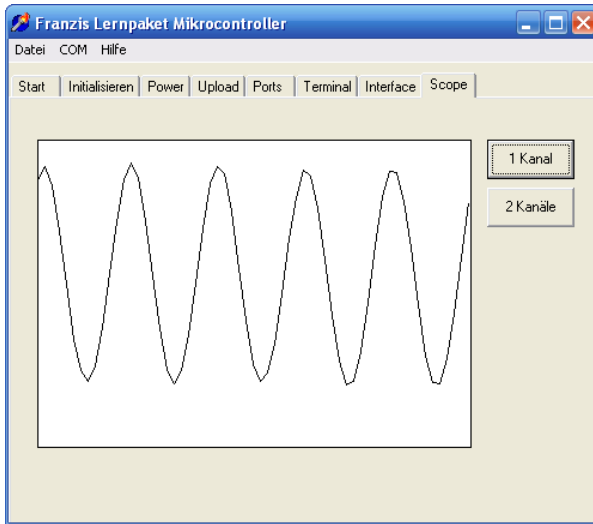


Abb. 3.14 Sinussignal mit 400 Hz ((Soft14.tif))

Untersuchen Sie auch einmal die Brummspannung, die beim Berühren des hochohmigen Eingangs anliegt. Sie liegt im Allgemeinen höher als die Betriebsspannung des Mikrocontrollers. Interne Schutzdioden an den Ports begrenzen die Eingangsspannung bei etwa $-0,5\text{ V}$ und $+5,5\text{ V}$. Man erhält daher eine angenäherte Rechteckspannung. Wenn Sie den Port nicht direkt berühren sondern den Finger nur in die Nähe halten, besteht eine kapazitive Kopplung mit geringerer Amplitude. Verzerrungen des rein sinusförmigen Verlaufs sind oft auf Netzverbraucher wie z.B. Leuchtstofflampen oder Netzteile zurückzuführen.

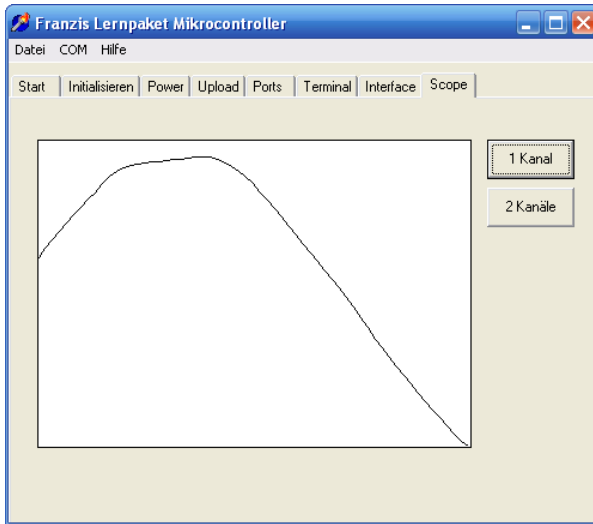


Abb. 3.15 Verzerrungen der Netzspannung ((Soft15.tif))

Das Oszilloskop zeigt, was mit einem kleinen Mikrocontroller wie dem ATtiny13 erreichbar ist. In den folgenden Kapiteln sollen die dazu nötigen Programmtechniken genauer besprochen werden. Sie haben dann die Möglichkeit, die Software nach eigenen Wünschen zu verändern.

3.6 Der PWM-Ausgang

Ein PWM-Signal (Pulsweiten-Modulation) ist ein periodisches Rechtecksignal mit einstellbarem Puls/Pausen-Verhältnis. Ein angeschlossener Verbraucher wird also in schneller Folge ein- und ausgeschaltet und erhält damit eine einstellbare mittlere Spannung bzw. einen mittleren Strom. Die Helligkeit einer angeschlossenen LED (Abb. 3.16) kann in weiten Grenzen verändert werden. Der PWM-Ausgang liegt am Anschluss PB0.

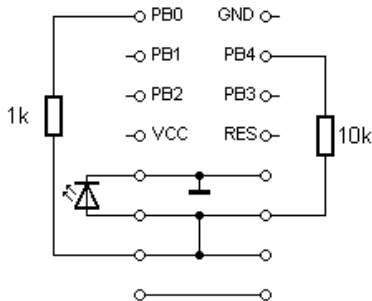


Abb. 3.16 PWM-Steuerung einer LED ((Schalt7.gif))

Klicken Sie auf „PWM initialisieren“. Das PWM-Signal wird eingeschaltet und PB0 in Ausgaberichtung geschaltet. Nun können Sie mit dem PWM-Schieberegler die Helligkeit der LED einstellen.

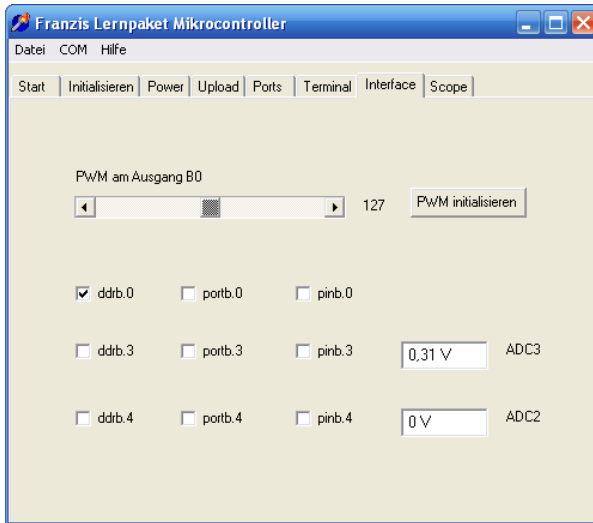


Abb. 3.17 PWM-Ausgabe ((Soft16.tif))

Stellen Sie eine zusätzliche Verbindung zum Eingang PB4 her. So können Sie das PWM-Signal am Oszilloskop betrachten (Abb. 3.18).

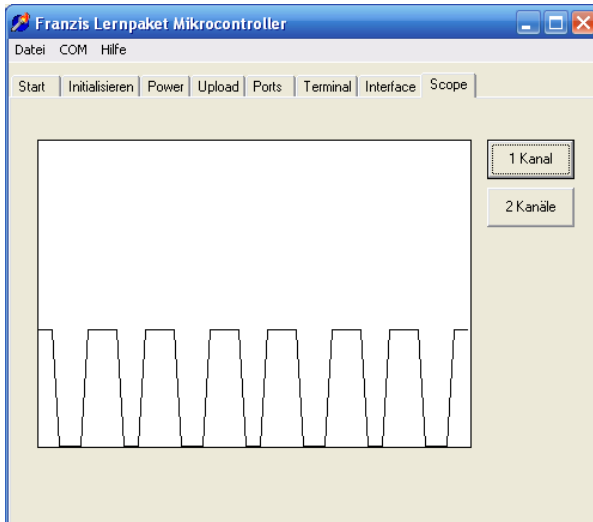


Abb. 3.18 Das PWM-Signal an der LED ((Soft17.tif))

PWM-Ausgänge werden oft zur Ausgabe einer einstellbaren Gleichspannung verwendet. Dazu muss das Signal mit einem Tiefpassfilter geglättet werden. Verwenden Sie den Elektrolytkondensator mit $47\ \mu\text{F}$ und einen Widerstand mit $1\ \text{k}\Omega$ als einfaches Filter. Stellen Sie am PWM-Ausgang eine mittlere Spannung ein. Das Oszilloskop zeigt nur noch eine geringe Restwelligkeit der geglätteten Spannung (Abb. 3.20).

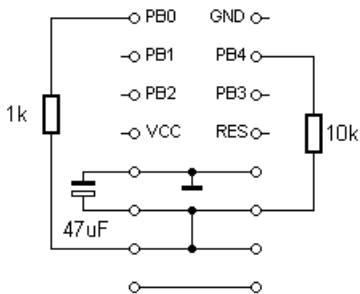


Abb. 3.19 Glätten mit $1\ \text{k}\Omega$ und $47\ \mu\text{F}$ ((Schalt8.gif))

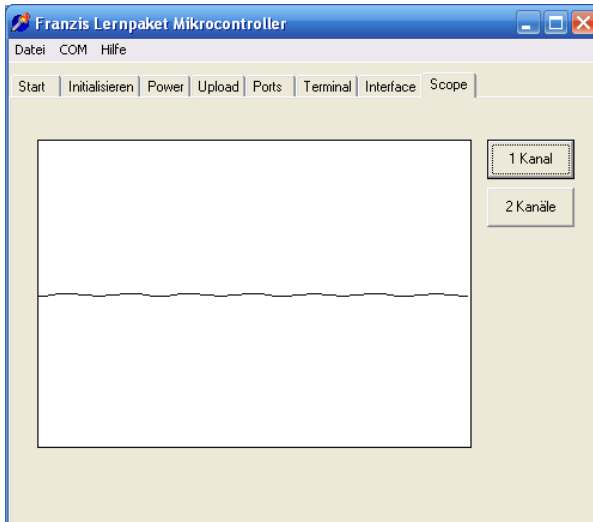


Abb. 3.20 Geringe Restwelligkeit ((Soft18.tif))

Verbessern Sie die Glättung mit einer geringeren Grenzfrequenz des Tiefpassfilters. Mit 10 k Ω und 47 μ F erhalten Sie eine Gleichspannung, bei der Das Oszilloskop keine Restwelligkeit mehr feststellen kann (Abb. 3.21 und Abb. 3.22).

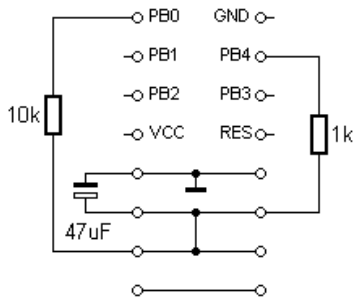


Abb. 3.21 Glätten mit 10 k Ω und 47 μ F ((Schalt9.gif))

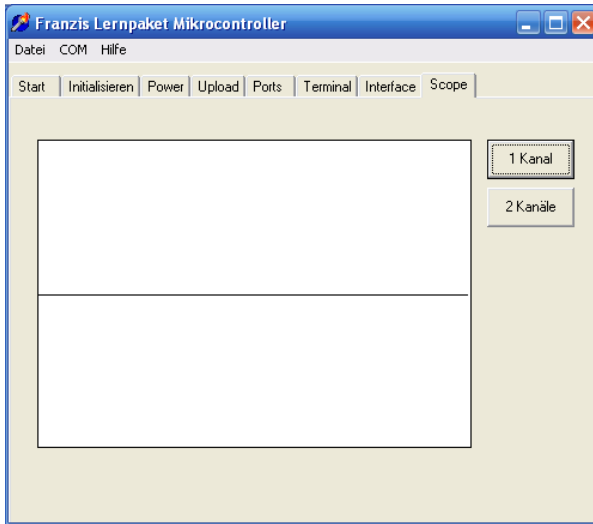


Abb. 3.22 Optimale Glättung ((Soft19.tif))

3.7 Schaltschwellen

Digitale Eingänge unterscheiden nur zwei Zustände, Eins und Null. Laut Datenblatt des ATtiny13 wird bei Spannungen unter 1 V zuverlässig Null gelesen, über 3 V dagegen Eins. Aber auch Zwischenwerte liest der Eingang entweder als Null oder als Eins. Deshalb soll nun die genaue Grenze zwischen beiden Pegeln gesucht werden.

Bauen Sie die Schaltung nach Abb. 3.21 auf. Das PWM-Ausgangssignal wird geglättet und an einen der Eingänge gelegt. Mit dem Interface können Sie nun gleichzeitig die aktuelle Spannung am Eingang und den logischen Pegel ablesen.

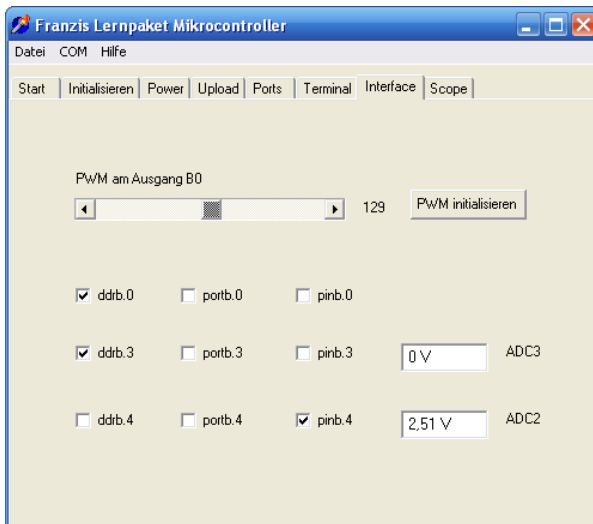


Abb. 3.23 Messen der Eingangs-Schaltschwelle ((Soft20.tif))

Die Messung zeigt, dass die Grenze gerade bei der halben Betriebsspannung, also bei 2,5 V liegt. Allerdings kann es gewisse Exemplarstreuungen geben, sodass es ratsam ist den Bereich zwischen 1 V und 3 V zu vermeiden, wenn eindeutige Pegel gelesen werden müssen.

3.8 Pullup-Widerstände

Die einschaltbaren internen Pullup-Widerstände an jedem Port haben laut Datenblatt einen Widerstand zwischen 20 k Ω und 50 k Ω . Dieser Wert kann sehr einfach überprüft werden. Lassen Sie den Pullup-Strom über einen bekannten Widerstand fließen und bestimmen Sie den Spannungsabfall. Bei einem Messwiderstand von 10 k wurden 1,09 V gemessen. Daraus kann der interne Pullup zu 35,9 k Ω bestimmt werden. Der Wert liegt also mitten im angegebenen Bereich.

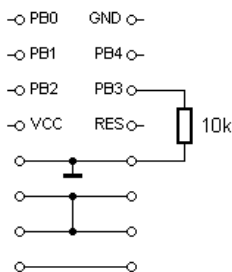


Abb. 3.24 Messung des Pullup-Widerstands ((Schalt19.gif))

Zur Kontrolle können Sie den Kurzschlussstrom am Port mit einem Digitalmultimeter messen. Es wurde ein Strom von 134 μA gefunden. Daraus folgt ein Pullup-Widerstand von 37,3 k Ω , was im Rahmen der Messgenauigkeit mit dem oben gefundenen Ergebnis übereinstimmt.

3.9 Programm-Upload

Nun soll ein Programm in den Mikrocontroller übertragen werden, womit gleichzeitig das Interface-Programm überschrieben wird. Im Gegensatz zur ISP-Programmierung bei der Initialisierung führt der Mikrocontroller nun eine Selbstprogrammierung durch. Daten werden über die serielle Schnittstelle mit 9600 Baud an den Mikrocontroller gesandt, der sie empfängt und in sein Flash-ROM überträgt. Voraussetzung dafür ist natürlich die erfolgreiche Initialisierung, bei der das passende Boot-Programm zusammen mit dem Interfaceprogramm übertragen wurde.

Alle Beispielprogramme aus dem Lernpaket liegen im Quelltext und in ausführbarer Form als Hex-Datei vor. Jedes Beispielprogramm liegt in einem eigenen Verzeichnis. Der Name des Verzeichnisses und der Name der ausführbaren Hex-Datei stimmen überein. Verwenden Sie das Beispiel Blink2.hex aus dem Verzeichnis LPmikro/Blink2.

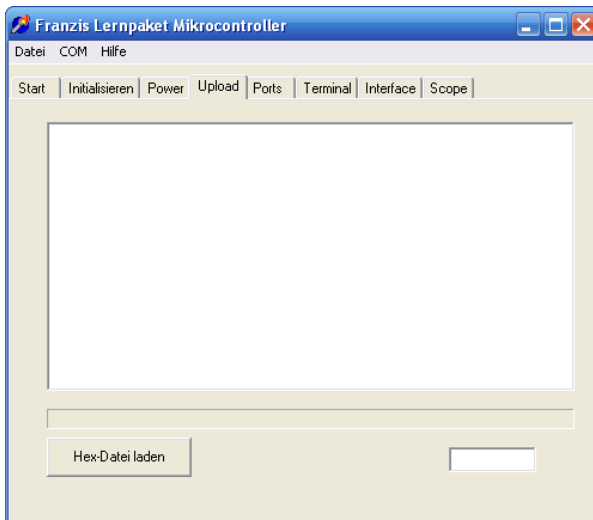


Abb. 3.25 Laden eines Programms ((Soft21.tif))

Öffnen Sie die Registerkarte Upload und klicken sie auf die Schaltfläche „Hex-Datei laden“. Es öffnet sich ein Dateifenster. Wählen Sie die Datei Blink2.hex im Verzeichnis Blink2.

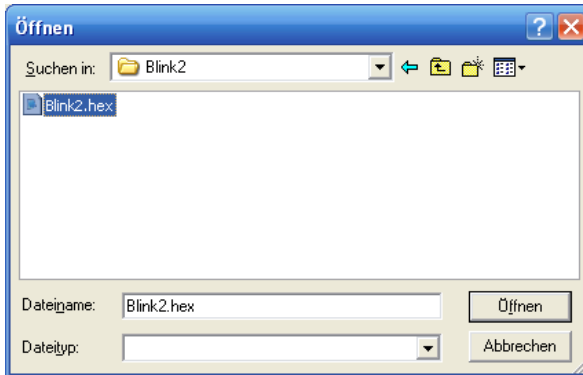


Abb. 3.26 Dateiauswahl ((Soft22.tif))

Das Programm Blink2.hex wird in hexadezimaler Form angezeigt und dann in den Mikrocontroller übertragen. Schon das Hex-Listing zeigt, dass es sich um ein sehr kurzes Programm handelt. Entsprechend schnell ist die Übertragung beendet. Den Verlauf des Vorgangs erkennt man an einer Balkenanzeige. Der Erfolg wird als ok-Meldung angezeigt.

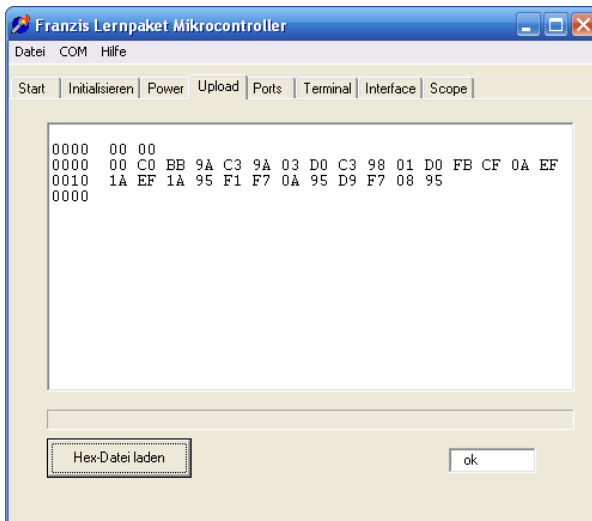


Abb. 3.27 Erfolgreicher Ladevorgang ((Soft23.tif))

Falls eine Fehler-Meldung erscheint, können folgende Punkte überprüft werden: Der RES-Eingang des Mikrocontroller darf nicht angeschlossen sein. Wurde die Initialisierung erfolgreich durchgeführt? Das konnte z.B. daran erkannt werden, dass die Interface-Funktionen wie beschrieben funktionierten.

Passend zum Programm muss eine kleine Schaltung aufgebaut werden. Am Pin PB3 soll eine LED mit Vorwiderstand angeschlossen werden. Der Aufbau entspricht dem in Abb. 3.2 aus Kap. 3.1

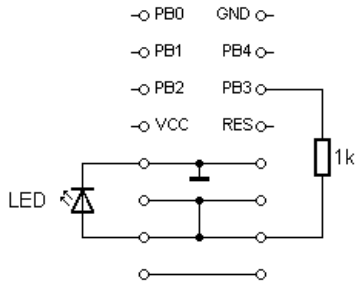


Abb. 3.28 Die blinkende LED ((Schalt3.gif))

Nach der erfolgreichen Übertragung des Programms kommt der spannende Moment. Wenn alles korrekt funktioniert hat, startet das Programm. Nun blinkt die LED. Sie können den Blinker über die Betriebsspannung ein- und ausschalten (Vgl. Kap. 2.1).

4 Assembler-Grundlagen

Ein Programm für den Mikrocontroller besteht aus Zahlen, die der interne Prozessor als Befehle versteht. Dieser Maschinencode könnte direkt als Zahlenfolge geschrieben werden. Das ist aber nicht üblich, weil es den menschlichen Geist überfordert. Stattdessen schreibt man die Befehle als Text, der dann durch eine Software in Maschinencode übersetzt wird. Das Übersetzen nennt man auch assemblieren, die Software dazu Assembler. Die Arbeit mit dem Assembler stellt die unterste Ebene der Programmierung dar, bei der man ganz direkt jedes einzelne Bit beeinflussen kann.

4.1 Das AVR Studio

Zum Programmieren des Mikrocontrollers ATtiny13 wird hier zunächst der Assembler benötigt. Die Firma ATMEL stellt mit dem AVR Studio eine kostenlose Programmierungsumgebung mit Assembler und Simulator bereit, die auch mit dem C-Compiler Win-AVR zusammen arbeitet. Auf der CD finden Sie das AVR Studio 4 (aStudio4b356.exe). Installieren Sie die Oberfläche, indem Sie die Installationsdatei ausführen und den Anweisungen folgen. Aktuellere Versionen auch für neuere AVR-Mikrocontroller gibt es auf der Seite des Herstellers www.atmel.com. Insbesondere die neuere Version 4.12 ist dann erforderlich, wenn Sie das AVR Studio nicht nur für Assembler sondern auch für C-Programme verwenden möchten.

Nach vollständiger Installation können Sie das AVR Studio 4 starten. Zunächst erscheint ein Fenster zur Auswahl eines Projekts oder zum Erstellen eines neuen Projekts.

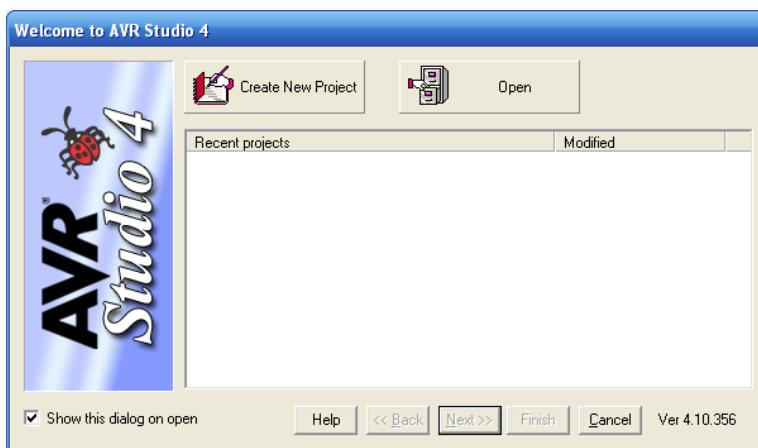


Abb. 4.1 Erzeugen eines neuen Projekts ((AVR1.tif))

Beginnen Sie ein neues Assembler-Projekt mit dem Namen „LED“. Klicken Sie dazu auf „Create New Project“. Die Optionen „Create initial File“ und „Create Folder“ müssen aktiviert sein. Tippen Sie dann im Feld „Project Name“ den Namen „LED“ ein. Damit wird das Feld „Initial File“ mit dem gleichen Namen gefüllt, d.h. Ihr Quelltext heißt nun LED.asm. Das Projekt soll im Verzeichnis C:\Franzis\LPmikro stehen, das bereits die fertigen Projekte des Lernpakets enthält.

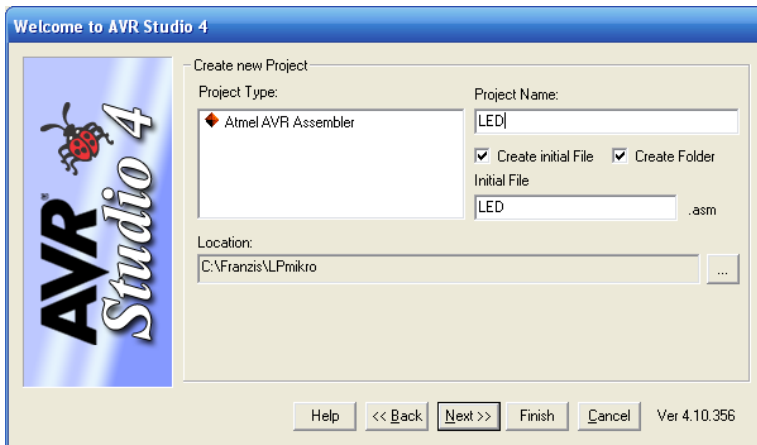


Abb. 4.2 Eingabe des Projektnamens ((AVR2.tif))

Mit Next gelangen Sie ins nächste Fenster, wo der AVR Simulator für den Zielprozessor ATtiny13 ausgewählt wird.

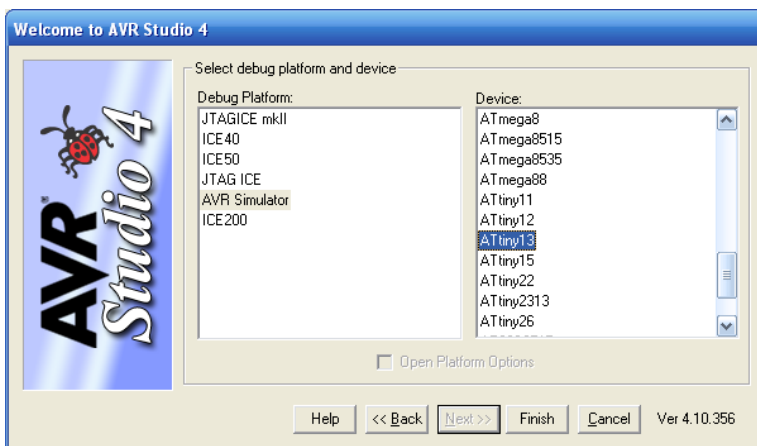


Abb. 4.3 Auswahl des ATtiny13 ((AVR3.tif))

Ein abschließender Klick auf Finish öffnet das neue Projekt. Im Editorfenster erscheint der noch leere Quelltest LED.asm

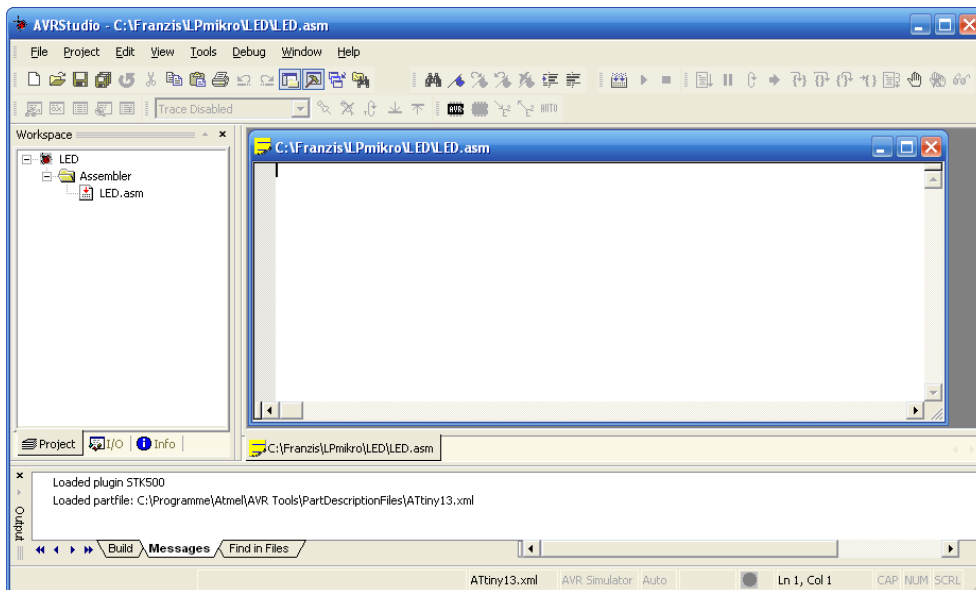


Abb. 4.4 Der Editor ((AVR4.tif))

Tippen Sie nun den folgenden kurzen Programmtext ein. Es wird eine Zieladresse „Anfang“ definiert. Mit dem Befehl „rjmp Anfang“ springt das Programm immer wieder zum Anfang. Es handelt sich also um eine Endlosschleife, die in Mikrocontroller-Projekten oft als Abschluss eines Programms verwendet wird.

```
Anfang:
    rjmp  Anfang
```

Listing 4.1 Das erste Assembler-Programm

Mit Project/Build oder F7 wird das Programm übersetzt. Wenn alles ohne Fehler abläuft, erhalten Sie die Meldung „Assembly complete with no errors“. Im Projektverzeichnis LED befindet sich nun u.a. die Datei LED.hex. Diese könnten Sie in den Mikrocontroller laden, allerdings wird noch keine sinnvolle Funktion erreicht.

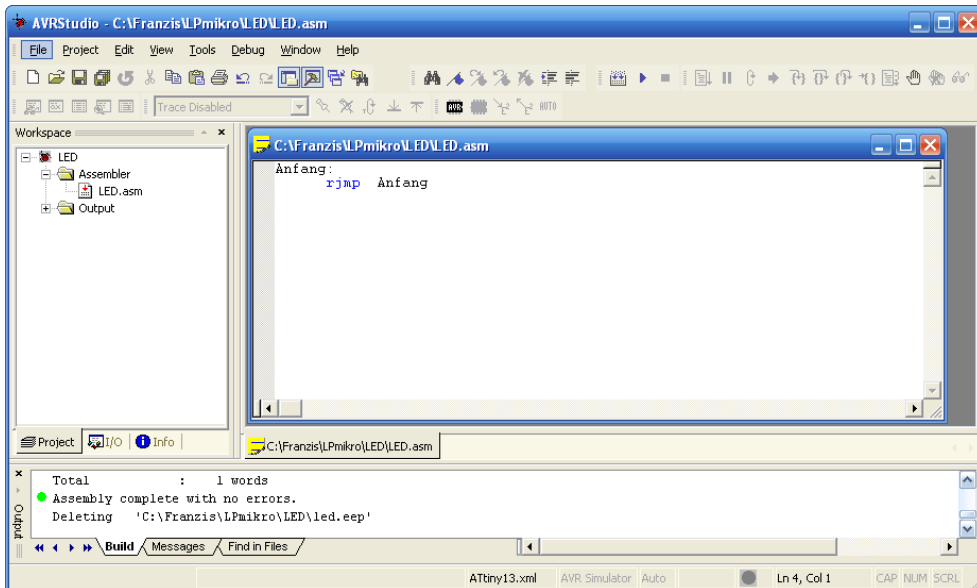


Abb. 4.5 Fehlerfreie Übersetzung ((AVR5.tif))

Falls Sie den Debugger ausprobieren möchten, fügen Sie noch einen nop-Befehl (No Operation, nichts tun) in den Quelltext ein und starten Sie ihn mit Debug /Start Debugging. Das Programm kann dann im Simulator gestartet werden. Beobachten mit Einzelschritten, wie das Programm die beiden Anweisung „nop“ und „rjmp Anfang“ abwechselnd immer wieder ausführt. Das Lernpaket verzichtet aus Platzgründen auf den Einsatz des Debuggers. Sie können aber alle besprochenen Projekte auch im Debugger testen.

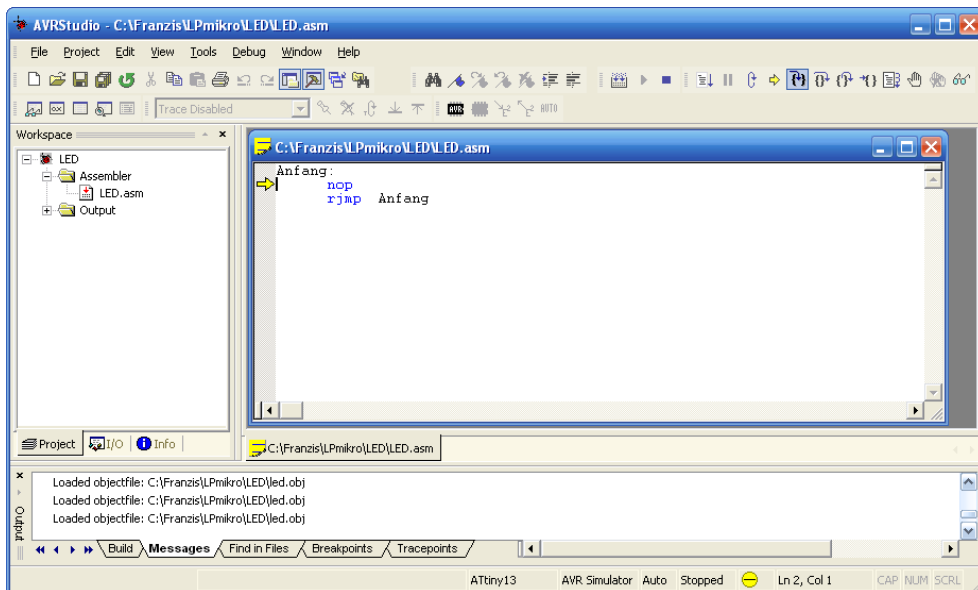


Abb. 4.6 Programmsimulation im Debugger ((AVR6.tif))

4.2 Ausgangsports

Damit sichtbar etwas passiert, soll nun ein Port als Ausgang eingerichtet und eingeschaltet werden. Alle Hardware-Aktionen werden über Register im Controller gesteuert. Das Datenblatt tiny13.pdf befindet sich auf der CD und gibt dazu umfassende Auskunft. Hier geht es um zwei Register, das Datenrichtungsregister DDRB und das Ausgaberegister PORTB.

Damit der Assembler weiß, welches Register an welcher Adresse liegt, muss die Datei "tn13def.inc" in das Projekt eingefügt werden. Dazu dient die Compiler-Anweisung `.include "tn13def.inc"`. Die Datei liegt im Verzeichnis `C:\Programme\Atmel\AVR Tools\AvrAssembler\Appnotes` und wird automatisch gefunden. Sie kann mit einem Editor angesehen werden und enthält neben vielen anderen die folgenden beiden Zeilen:

```
.equ    PORTB = $18
.equ    DDRB  = $17
```

Der Assembler setzt also z.B. die Adresse \$18 (Hexadezimal 18 = dezimal 26) an jeder Stelle ein, die im Listing das Wort PORTB enthält. Allgemein ist Groß- und Kleinschreibung freigestellt.

Erweitern Sie nun den Quelltext LED.asm wie in Listing 2. Übersetzen Sie es dann neu und laden Sie die Datei LED.hex mit der Upload-Funktion in Lpmikro.exe. Am Port PB3 soll eine LED mit Vorwiderstand 1 k Ω angeschlossen sein wie in Abb. 3.2. Sobald das Programm geladen und gestartet ist, leuchtet die LED. Falls es noch Probleme mit dem Editieren oder Assemblieren gibt oder Sie aus anderen Gründen auf ein fertiges Projekt zurückgreifen möchten, können Sie auch die Datei LED2.hex aus dem Projektverzeichnis LED2 verwenden, die schon fertig übersetzt vorliegt.

```
;LED.asm, LED mit Vorwiderstand an PB3

        .include "tn13def.inc"

        rjmp  Anfang
Anfang:
        sbi   ddrb,3           ;Datenrichtungsbit
        sbi   portb,3          ;PB3=1, einschalten
Schleife:
        rjmp  Schleife
```

Listing 4.2 Einschalten des Ports PB3

Das Programm schaltet also den Anschluss PB3 ein. Zum besseren Verständnis sollen die einzelnen Zeilen nun genauer erklärt werden. Am Anfang steht ein Kommentar, der nicht mit übersetzt wird und nur zur besseren Lesbarkeit des Quelltextes dient. Kommentare werden mit einem Semikolon eingeleitet und dürfen an beliebiger Stelle im Text stehen. Danach folgt die schon erwähnte Compiler-Anweisung zur Einbinden der Datei "tn13def.inc".

Das Programm enthält zwei Adressmarken (Label) „Anfang“ und „Schleife“. Der erste echte Assemblerbefehl im Quelltext lautet „rjmp Anfang“. „rjmp“ steht für Relative Jump, also für einen Sprung mit einer relativen Adresse, d.h. mit der Angabe einer Sprungweite. Das Programm verzweigt also nach dem Start zur Adresse „Anfang“. Im Prinzip könnte man diese Zeile entfernen ohne die Funktion des Programms zu verändern. Allerdings ist der erste Sprung wichtig für die korrekte Funktion des Upload-Programms. Es setzt nämlich hier beim Hochladen einen Sprung auf das Bootprogramm ein und vermerkt den wirklichen Anfang des Programms an anderer Stelle. Alle Programme im Lernpaket beginnen deshalb mit einem Sprungbefehl.

Alle Assembler-Befehle sind in der englischen Hilfe zum AVR Studio aufgelistet. Markieren Sie den Befehl „sbi“ mit der Maus und drücken Sie F1. Es erscheint ein Hilfefenster mit einer genauen Definition. Außerdem erhalten Sie einen vollständigen Überblick über alle verfügbaren Assemblerbefehle und ihre Funktion.

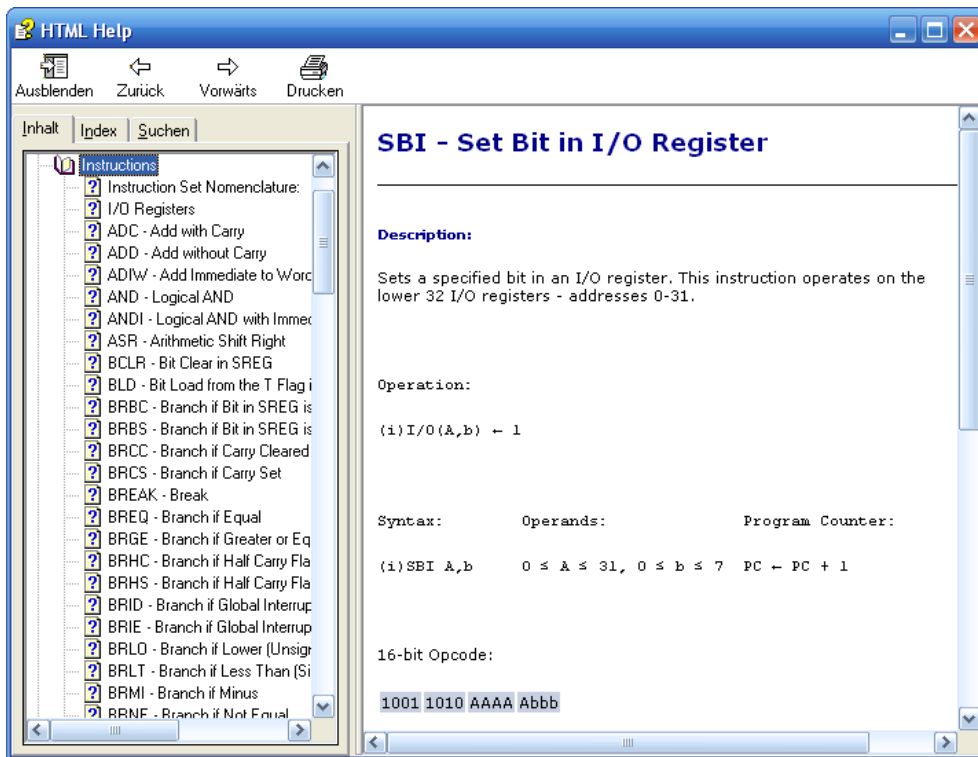


Abb. 4.7 Die Hilfe im AVR Studio ((AVR7.tif))

Der Befehl “SBI - Set Bit in I/O Register” schaltet ein Bit im angegebenen Register ein. Er wird hier zweimal verwendet. Zuerst wird das Bit 3 in DDRB eingeschaltet um die Ausgabe-Datenrichtung einzuschalten, dann das Bit 3 in PORTB um den Portzustand auf Eins zu schalten. Damit wird die LED eingeschaltet. Die grundlegenden Funktionen dieser Bits sind bereits aus Kap. 3.1 bekannt, wo der Controller als Interface verwendet wurde. Die entsprechenden Bits wurden auf der Registerkarte Interface per Mausklick umgeschaltet. Sie können aber nun nicht mehr direkt verwendet werden, weil das Interface-Programm durch das LED-Programm überschrieben wurde.

Am Ende des Programms finden Sie wieder eine Endlosschleife mit einem Sprung zur Label „Schleife“. Dies ist wichtig, um ein definiertes Ende des Programms zu erreichen. Es muss verhindert werden, dass der Controller am Ende noch Befehle ausführt, die als Reste früher geladener Programme im Speicher liegen. Ohne die Schleife am Ende könnten zufällige Aktionen ablaufen, die nicht vorhersagbar sind.

Wenn Sie sehen wollen, was der Assembler aus Ihrem Quelltext gemacht hat, reicht ein Blick in das Programmfenster im Uploadbereich von Lpmikro. Sie sehen hier acht

hexadezimal dargestellte Bytes. Jeweils zwei Bytes gehören zu einem Befehl, weil AVR-Controller die Befehle zusammen mit ihren Argumenten jeweils mit 16 Bit codieren.

```
0000 00 00
0000 00 C0 BB 9A C3 9A FF CF
0000
```

Listing 4.3 Die übertragenen Bytes

Alternativ kann man sich auch die Datei LED.hex in einem Editor anschauen. Sie enthält die selben Bytes zusammen mit einem Rahmen, Adresse und Prüfbyte.

```
:0200000020000FC
:0800000000C0BB9AC39AFFCFB8
:00000001FF
```

Listing 4.4 Inhalt der Datei LED.hex

4.3 Bits und Bytes

Oft ist es nötig, nicht nur ein einzelnes Bit eines Ausgangsports einzuschalten, sondern gleich mehrere. In diesem Fall übergibt man dem Register PORTB acht Bits als ein Byte, also eine Zahl Bereich 0 bis 255. Der Befehl OUT (Store Register to I/O Location, Registerinhalt in IO-Adresse speichern) kopiert den Inhalt eines Registers, das zuvor mit der gewünschten Zahl geladen wurde. Der Mikrocontroller besitzt 32 Register R0 bis R31. Nur die Register R16 bis R31 erlauben das direkte Laden mit einer konstanten Zahl. Dazu dient der Befehl LDI (Load Immediate, direktes Laden). Zahlen können unterschiedlich angegeben werden:

Dezimal	10, 255
Hexadezimal:	0x0a, 0xff
Alternative Schreibweise:	
Hexadezimal:	\$0a, \$ff
Binär:	0b00001010, 0b11111111

Das Programm LED3.asm soll die Ausgänge PB3 und PB4 einschalten. Bit 3 hat die Wertigkeit $2^3=8$, Bit 4 entsprechend $2^4=16$. Die Summe beider Wertigkeiten ist 24. Das Programm muss also die Zahl 24 in DDRB und PORTB schreiben. Alternativ zur Dezimalzahl kann auch die Binärzahl 0b00011000 oder die Hexadezimalzahl 0x18 bzw. \$18 verwendet werden.

```
;LED3.asm, LED mit Vorwiderstand an PB3 oder PB4
```

```

        .include "tn13def.inc"

        rjmp  Anfang
Anfang:
        ldi   r16,0b00011000
        out   ddrb,r16      ;Datenrichtung
        ldi   r16,0x18      ;0x18 = 0b00011000 =24
        out   portb,r16     ;PB3=1 und PB4=1
Schleife:
        rjmp  Schleife

```

Listing 4.5 Einschalten mehrerer Ausgänge

4.4 Ein Blinkprogramm

Als nächste Programmierübung soll die LED am Port PB3 immer wieder ein- und ausgeschaltet werden. Zum Ausschalten wird der neue Befehl CBI (Clear Bit in IO Register, ein Bit im angegebenen Register löschen). Warteschleifen sollen den Vorgang etwas verlangsamen. Man erkennt im Listing 4.6 deutlich die beiden Warteschleifen mit den Sprungmarken Warten1 und Warten2.

;Blink1.asm LED mit Vorwiderstand an PB3

```

        .include "tn13def.inc"

        rjmp  Anfang
Anfang:
        sbi   ddrb,3        ;Datenrichtungsbit
Schleife:
        sbi   portb,3       ;PB3 = 1
        ldi   r16,255       ;r16 laden
Warten1:
        dec   r16           ;r16 - 1
        brne  Warten1
        cbi   portb,3       ;PB3 = 0
        ldi   r16,255       ;r16 laden
Warten2:
        dec   r16           ;r16 -1
        brne  Warten2
        rjmp  Schleife

```

Listing 4.6 Das Programm Blink1.asm

Eine Warteschleife verwendet jeweils ein allgemeines Arbeitsregister (hier Register R16) des Controllers. Zunächst wird es mit einem Startwert geladen. Da bei einem 8-Bit-Mikrocontroller nur Bytes verarbeitet werden können, ist der größte mögliche Wert 255. Achtung, der Befehl LDI (Load Immediate, direktes Laden) ist nicht auf alle Register anwendbar, sondern nur auf die Register R16 bis R31. Mit `ldi r16, 255` wird also die Zahl 255 ins Register R16 geladen. In der Warteschleife zählt der Befehl DEC (Decrement, Verkleinern) jeweils um Eins zurück. Der bedingte Sprungbefehl BRNE (Branch if Not Equal, Verzweige, wenn nicht gleich Null) verzweigt immer wieder zum Anfang der Schleife, solange der Inhalt des Registers noch größer als Null ist. Sobald das Register 255 mal verkleinert wurde und den Wert Null enthält, unterbleibt der Sprung, sodass der auf die Schleife folgende Befehl ausgeführt wird. In diesem Programm gibt es zwei Warteschleifen, jeweils direkt nach dem Einschalten und nach dem Ausschalten des Ports.

Laden Sie das Programm `Blink1.hex` in den Controller. Verwenden Sie eine LED mit Vorwiderstand wie in Abb. 3.2. Die Hoffnung auf ein sichtbares Blinken wird zunächst enttäuscht. Die LED leuchtet dauerhaft, wenn auch etwas schwächer als gewohnt. Tatsächlich blinkt die LED noch viel zu schnell. Wenn Sie jedoch die Platine schnell hin- und her bewegen, können Sie ein Flackern sehen.

4.5 Unterprogramme

Das Blinkprogramm soll mit einer doppelten Warteschleife langsamer laufen. Dabei sollen nicht nur 255 Schleifendurchläufe sondern bis zu $255 \times 255 = \text{ca. } 65000$ Durchläufe verwendet werden. Dazu braucht man zwei Schleifen, die ineinander geschachtelt sind. In der äußeren Schleife wird der Zähler der inneren Schleife immer wieder neu mit seinem Startwert geladen.

Eine solche geschachtelte Schleife wird wieder zweimal gebraucht. In solchen Fällen lagert man Programmteile gern in Unterprogramme aus, um sie nur einmal schreiben zu müssen. Das Unterprogramm wird von beliebigen Stellen aus mit RCALL (RCALL - Relative Call to Subroutine, Relativer Aufruf eines Unterprogramms) angesprungen. Der Controller merkt sich dabei die Adresse, von der aus gesprungen wurde in einem besonderen Speicherbereich, dem so genannten Stapel (Stack, siehe Kap. 8.1). Am Ende des Unterprogramms muss RET (Return from Subroutine, Rücksprung vom Unterprogramm) stehen. Dann wird die Rücksprungsadresse vom Stapel genommen, und das Programm führt die Anweisung nach dem RCALL-Befehl aus.

```
;Blink2.asm Blinker mit Unterprogramm

    .include "tn13def.inc"

rjmp Anfang
Anfang:
    ldi    r16, 0x18    ;PB4 und PB4
```

```

        out    ddrb,r16      ;Datenrichtung
Schleife:
        ldi    r16,8         ;8 = 0x08
        out    portb,r16     ;PB3 = 1, PB4 = 0
        rcall  Warten        ;Unterprogrammaufruf
        ldi    r16,16        ;16 = 0x10
        out    portb,r16     ;PB3 = 0, PB4 = 1
        rcall  Warten        ;Unterprogrammaufruf
rjmp  Schleife

Warten:
        ldi    r16,250
Warten1:                                ;äußere Schleife
        ldi    r17,250
Warten2:                                ;innere Schleife
        dec    r17
        brne   Warten2
        dec    r16
        brne   Warten1
        ret                                ;Rücksprung

```

Listing 4.7 Verwendung eines Unterprogramms

Mit der doppelten Warteschleife tut das Programm genau das, was es soll: Die LED blinkt nun deutlich sichtbar. Sie können mit den Startwerten der inneren und der äußeren Zählschleife experimentieren und das Programm mit unterschiedlichen Geschwindigkeiten laufen lassen. Eine gute Übung ist auch die Erweiterung auf eine dreifach geschachtelte Warteschleife.

In der Schleife setzt das Programm abwechselnd das Bit 3 und das Bit 4 in PORTB. Deshalb blinkt nicht nur der Ausgang PB3, sondern er wechselt sich mit PB4 ab. Es entsteht also ein Gegentaktblinker. Sie können wahlweise den Ausgang PB3 oder PB4 anschließen oder auch eine zweite LED blinken lassen, die allerdings nicht im Material des Lernpakets enthalten ist.

4.6 Geschwindigkeitstest

Die Ausführungsgeschwindigkeit ist letztlich vom internen Taktoszillator des ATtiny13 abhängig. Sie können Rückschlüsse auf die Genauigkeit des Oszillators ziehen, wenn Sie ein Programm mit genau bekanntem Zeitbedarf schreiben. Das Blinkprogramm soll nun so beschleunigt werden, dass genau zehn Prozessortakte für einen vollständigen Durchlauf benötigt werden.

Das Programm in Listing 4.8 erklärt sich selbst. Neu ist nur der Befehl NOP (No Operation, Nichts tun), der genau einen Prozessortakt benötigt. Der Prozessortakt ist bei AVR-Controllern gleich dem Befehlstakt. Die meisten Befehle benötigen also nur einen Takt. Ausnahmen bilden Sprungbefehle und Bitmanipulationen wie SBI und CBI. Die Hilfe gibt für jeden Befehl an, wie viele Taktzyklen verbraucht werden. Sie sind im Listing als Kommentare angegeben. Insgesamt verbraucht die Schleife genau 10 Takte.

```
;Takt.asm  Clock/10 an PB3

        .include "tn13def.inc"

        rjmp Anfang
Anfang:
        sbi  ddrb,3

Loop10:
        sbi  portb,3    ;2
        nop                    ;1
        nop                    ;1
        nop                    ;1
        nop                    ;1
        cbi  portb,3    ;2
        rjmp loop10      ;2
                        ;Summe 10 Takte
```

Listing 4.8 Rechteckausgabe mit 120 kHz

Eine angeschlossene LED zeigt nun etwa halbe Helligkeit, weil die Ausgangsfrequenz sehr hoch ist. Mit einem Oszilloskop oder einem Frequenzzähler kann aber nun die Genauigkeit des internen Oszillators überprüft werden. Wegen der verwendeten Taktrate von 1,2 MHz sollte am Ausgang P3 ein Signal mit 120 kHz erscheinen

Es geht aber auch ohne teure Messgeräte. Schließen Sie ein 20 cm langes Stück Draht als Sendeantenne an. Das Signal ist dann auf einem gewöhnlichen Mittelwellenradio zu empfangen. Möglich ist der Empfang auf allen ungeraden Vielfachen der Ausgangsfrequenz. Da Sie mit einem Takt von 1,2 MHz arbeiten und damit 120 kHz ausgeben, ist das Signal z.B. bei $5 * 120 \text{ kHz} = 600 \text{ kHz}$ zu empfangen.

Da es sich um ein unmoduliertes Signal handelt, hört man meist nichts, erkennt aber an der Feldstärkeanzeige oder an einem geringeren Rauschen das Signal. Falls zufällig ein Mittelwellensender im gleichen Bereich empfangen werden kann, kommt es zu einer Interferenz beider Signale, die man im Normalfall als Pfeifen hört. Hier ist es eher ein etwas verrauschtes Zwitschern, weil der RC-Oszillator nicht die Stabilität eines Quarzoszillators erreicht. Die Genauigkeit ist fast immer besser als 3 %.

4.7 Digitale Eingänge

Ein Portanschluss des Tiny 13 kann sowohl als Ausgangs wie auch als Eingang verwendet werden. Nach dem Einschalten oder einem Reset sind alle Ports zunächst Eingänge mit hohem Eingangswiderstand, die mit CMOS-Eingängen vergleichbar sind. Ein Ausgang muss über das Port-Richtungsregister explizit eingerichtet werden.

Das folgende Programm wertet den Zustand des Eingangs B4 aus und kopiert ihn auf den Ausgang B3. Dazu werden kombinierte Lese- und Sprungbefehle eingesetzt, die speziell für die Auswertung von Zuständen einzelner Hardware-Bits dienen: SBIC (Skip if Bit in I/O Register is Cleared, Überspringe wenn Bit gleich Null) und SBIS (Skip if Bit in I/O Register is Set, Überspringe wenn Bit gleich Eins). Diese Sprungbefehle überspringen jeweils genau einen Befehl, wenn die entsprechende Bedingung wahr ist. Damit ist die Aufgabe einfach zu lösen: Wenn das Eingangsbit Null ist, wird der Befehl zum Hochsetzen des Ausgangs übersprungen und umgekehrt.

```
;Eingang.asm   kopiert Eingang PB4 an PB3

        .include "tn13def.inc"
        rjmp  Anfang

Anfang:
        sbi    ddrb,3           ;Datenrichtungsbit
Schleife:
        sbic   pinb,4
        sbi    portb,3
        sbis   pinb,4
        cbi    portb,3
        rjmp   Schleife
```

Listing 4.9 Auswerten eines Eingangs

Starten Sie dieses Programm mit einer LED an PB3 und berühren Sie den Eingang PB4 mit dem Finger. Dabei wird im Normalfall das immer vorhandene 50-Hz-Wechselfeld der Netzleitungen auf den Eingang gekoppelt. Die Ausgangs-LED erhält dann ebenfalls ein Taktsignal von 50 Hz und ist etwa mit halber Helligkeit an. Wenn Sie den Eingang loslassen, bleibt der zufällige letzte Zustand Eins oder Null stehen. Oft leuchtet dann die LED voll, bevor sie sich von allein ausschaltet, weil die im Eingang gespeicherte Ladung verschwindet.

Wie hochohmig und empfindlich ein offener CMOS-Eingang ist, zeigt der folgende Versuch: Berühren Sie den Eingang nicht, sondern halten Sie Ihre Hand nur nahe an den Mikrocontroller. Heben und senken Sie dann Ihre Schuhe. Je nach Abstand (5 cm bis 20 cm) und Beschaffenheit des Bodens und der Schuhe können Sie damit die LED umschalten.

Bei diesem Versuch entsteht durch elektrische Aufladung ein elektrisches Feld, das den Eingang lädt oder entlädt.

Allgemein sollte man es vermeiden, statische Ladung direkt über elektronische Bauteile abzuleiten. Im Extremfall kann man z.B. nach dem Gang über einen Teppich so weit aufgeladen sein, dass man bei der Berührung einen elektrischen Schlag verspürt und einen kleinen Blitz sehen und hören kann. Eine solche Entladung hat schon so manches Bauteil zerstört. Der ATtiny13 ist jedoch nicht sonderlich empfindlich. Man sollte nur vermeiden, die Anschlüsse direkt nach dem Umhergehen zu berühren. Zur Sicherheit kann man zuerst den über den PC und die Steckdose geerdeten Anschlusskragen des Anschlusssteckers berühren um übergroße Ladungen abzuleiten.

Ein offener CMOS-Eingang ist im Normalfall nicht sinnvoll, weil er einen zufälligen Eingangszustand annehmen kann. Wenn z.B. ein Schalter abgefragt werden soll, ist zwar der geschlossene Zustand eindeutig, der offene aber unbestimmt. Man kann das Programm Eingang.asm nutzen, um einen Schalter abzufragen. Dann ist aber ein zusätzlicher Widerstand sinnvoll, der den Zustand des offenen Eingangs vorgibt. Oft wird dieser Widerstand gegen die positive Versorgungsleitung angeschlossen. Der Eingang wird also im Ruhezustand hochgezogen (engl. pull up).

Bauen Sie einen solchen Widerstand in Ihre Schaltung ein und verwenden Sie eine Drahtbrücke als experimentellen Schalter. Das Ergebnis ist eindeutig: Bei geöffnetem Schalter ist die LED an, bei geschlossenem ist sie aus. Auch die Empfindlichkeit gegenüber externen Signalen ist geringer. Sie können den Eingang berühren, ohne einen Pegelwechsel zu erzeugen.

Pullup-Widerstände für jeden Portanschluss sind bereits im ATtiny13 enthalten. Sie müssen nur eingeschaltet werden. Dazu setzt man das zugehörige Ausgangsbit, lässt aber das Datenrichtungsbit low.

```
;Eingang2.asm   kopiert Eingang PB4 an PB3
```

```
    .include "tn13def.inc"
rjmp Anfang

Anfang:
    sbi    ddrb,3    ;Datenrichtungsbit
    sbi    portb,4    ;Pullup einschalten
Schleife:
    sbic   pinb,4
    sbi    portb,3
    sbis   pinb,4
    cbi    portb,3
    rjmp   Schleife
```

4.8 Die UND-Funktion

Logische Grundfunktionen wie UND, ODER und NICHT sind die Grundlage der digitalen Elektronik. Aus diesen Funktionen lassen sich auch komplexere Schaltungen wie Speicher Zähler und ganze Mikroprozessoren zusammensetzen.

Ein UND-Gatter verknüpft zwei Eingänge mit der logischen AND-Funktion zu einem Ausgang. Nur wenn Eingang 1 und Eingang 2 gesetzt sind wird auch der Ausgang eingeschaltet. Das Programm nach Listing 4.11 verwendet die Eingänge PB0 und PB2 sowie den Ausgang PB1. Diese Anschlüsse sind über die serielle Schnittstelle direkt zugänglich, sodass die Funktion direkt vom PC aus getestet werden kann. Das Programm LPmikro enthält für diesen Zweck die Registerkarte Ports.

```
;And.asm,   PB1 = PB0 AND PB2

        .include "tn13def.inc"

rjmp Anfang
Anfang:
        sbi    ddrb,1      ;Datenrichtungsbit
Schleife:
        clr    r16
        clr    r17
        sbic   pinb,0      ;b.0 in r16 lesen
        ldi    r16,1
        sbic   pinb,2      ;b.2 in r17 lesen
        ldi    r17,1
        and    r16,r17     ;b.0 AND b.2
        breq   Null        ;springe wenn Ergebnis = 0
        sbi    portb,1     ;1 ausgeben
        rjmp   Schleife
Null:
        cbi    portb,1     ;0 ausgeben
        rjmp   Schleife
```

Listing 4.11 Nachbilden der AND-Funktion

Das Programm AND.asm kopiert die Eingangszustände der Ports PB0 und PB1 in die Register R16 und R17. Auf diese kann dann der Assembler-Befehl AND (Logical AND , logische AND-Verknüpfung) angewandt werden. Das Ergebnis 1 oder 0 findet sich im

Register R16. Zugleich wird das Zero-Flag gesetzt oder gelöscht. Der folgende Befehl BREQ (Branch if Equal, Verzweige wenn gleich) führt eine Verzweigung zur angegebenen Adresse aus, wenn das Ergebnis Null ist. Er testet den Inhalt des Zero Flag (Z), ein Bit im Status-Register SREG des Prozessors. Das Z-Flag wird durch Rechenbefehle und Vergleiche gesetzt, wenn das Ergebnis Null ist. In diesem Fall wird PB1 ausgeschaltet, im an deren Fall eingeschaltet.

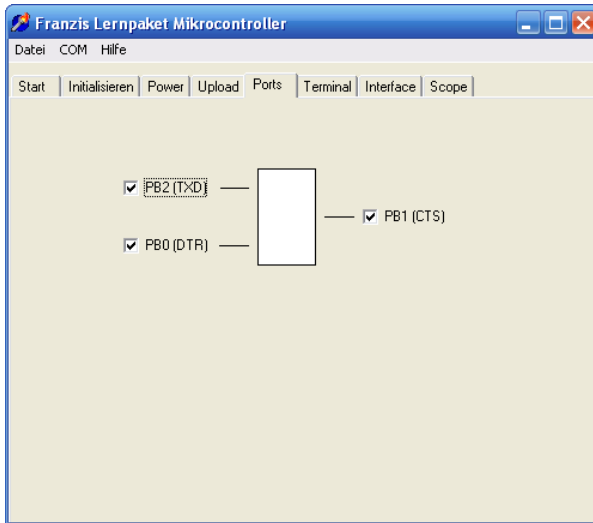


Abb. 4.8 Test der AND-Funktion ((Soft25.tif))

Laden Sie das Programm und testen Sie es auf der Registerkarte Ports. Schalten Sie die Eingänge PB0 und PB2 ein. Dann erscheint auch PB1 als eingeschaltet. Wenn einer der beiden Eingänge aus ist oder wenn beide aus sind, wird auch der Ausgang low.

Die NAND-Funktion kehrt das Ergebnis der AND-Funktion um. Es reicht, zwei Befehle zu vertauschen um eine NAND-Verknüpfung zu realisieren.

```
;Nand.asm,   PB1 = NOT (PB0 AND PB2)

        .include "tn13def.inc"

rjmp Anfang
Anfang:
        sbi    ddrb,1      ;Datenrichtungsbit
Schleife:
        clr    r16
        clr    r17
        sbic   pinb,0      ;b.0 in r16 lesen
```

```

        ldi    r16,1
        sbic   pinb,2      ;b.2 in r17 lesen
        ldi    r17,1
        and    r16,r17     ;b.0 AND b.2
        breq   Eins        ;springe wenn Ergebnis = 0
        cbi    portb,1     ;0 ausgeben
        rjmp   Schleife

Eins:
        sbi    portb,1     ;1 ausgeben
        rjmp   Schleife

```

Listing 4.12 Die NAND-Funktion

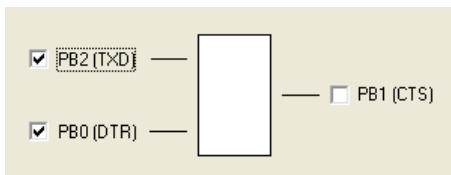


Abb. 4.9 Test der NAND-Funktion ((Soft26.tif))

4.9 Die Oder-Funktion

Mit der ODER-Funktion wird der Ausgang eingeschaltet wenn Eingang 1 oder Eingang 2 oder beide eingeschaltet sind. Das AND-Programm lässt sich leicht entsprechend abwandeln, in dem man den Assembler-Befehl OR (Logical OR, Logische ODER-Verknüpfung) einsetzt.

```

;Or.asm,   PB1 = PB0 OR PB2

        .include "tn13def.inc"

rjmp Anfang
Anfang:
        sbi    ddrb,1      ;Datenrichtungsbit
Schleife:
        clr    r16
        clr    r17
        sbic   pinb,0      ;b.0 in r16 lesen
        ldi    r16,1
        sbic   pinb,2      ;b.2 in r17 lesen
        ldi    r17,1
        or     r16,r17     ;b.0 AND b.2
        breq   Null        ;springe wenn Ergebnis = 0
        sbi    portb,1     ;1 ausgeben

```

```

        rjmp  Schleife
Null:
        cbi   portb,1    ;0 ausgeben
        rjmp  Schleife

```

Listing 4.13 Die ODER-Funktion

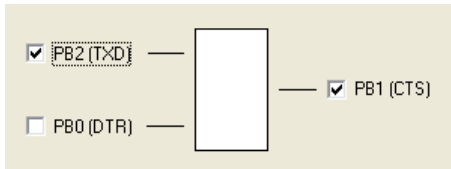


Abb. 4.10 Test der OR-Funktion ((Soft27.tif))

Sicherlich können Sie nun auch die NOR-Funktion problemlos nachbilden. Das Ergebnis finden Sie im Projekt Nor.

4.10 Das RS-Flipflop

Ein Flipflop ist eine rückgekoppelte Schaltung, die zwei stabile Zustände kennt. Ausgangszustände können durch bestimmte Eingangszustände oder -Ereignisse geändert werden.

Das RS-Flipflop besitzt einen Reset-Eingang R und einen Set-Ausgang S. Mit einem Impuls an R wird der Ausgang Q ausgeschaltet, mit einem Impuls an S eingeschaltet. Außerdem gibt es oft noch einen invertierten Ausgang /Q. Je nach Typ des Flipflops können die Impulse positiv oder negativ sein.

Das Programm RSflipflop.asm bildet ein RS-Flipflop mit zwei Eingängen, die auf positive Impulse reagieren. PB0 bildet den Set-Eingang, BP2 den Reset-Eingang. Der Ausgang Q liegt an PB1. Außerdem gibt es noch einen /Q-Ausgang an PB3. Hier kann eine LED angeschlossen werden.

```

;RSflipflop.asm,  RP0 = S,  PB2 = R

        .include "tn13def.inc"

rjmp Anfang
Anfang:
        sbi   ddrb,1      ;PB1
        sbi   ddrb,3      ;PB3

```

```

R:    sbis    pinb,0      ;PB0 = 1?
      rjmp    S
      sbi     portb,1     ;1 an PB1
      cbi     portb,3     ;0 an PB3
      rjmp    R

S:    sbis    pinb,2      ;PB2 = 1?
      rjmp    R
      cbi     portb,1     ;0 an PB1
      sbi     portb,3     ;1 an PB3
      rjmp    S

```

Listing 4.14 Das RS-Flipflop

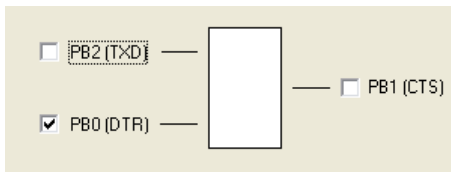


Abb. 4.11 Eingangsimpuls am R-Eingang ((Soft28.tif))

4.11 Das D-Flipflop

Ein D-Flipflop ist die kleinste Einheit eines Datenspeichers. Der Zustand in D wird in Q übernommen, wenn der Clock-Eingang high wird. Bei einem Low-Zustand an Clock bleibt der letzte Zustand erhalten. Im Falle eines flankengetriggerten D-Flipflops wird der Zustand an D immer nur bei einer Low-High-Flanke übernommen. Dagegen ist ein transparentes D-Flipflop offen für neue Zustände, solange Clock high ist.

Das Programm Dflipflop.asm bildet ein transparentes D-Flipflop nach. Solange Clock high ist, wird der Zustand von D an Q übernommen. Zusätzlich gibt es einen invertierten Ausgang /Q zum Anschluss einer LED.

```

;Dflipflop.asm,  PB0 = D,  PB2 = Clock,
;PB1 = Q,  PB3 = /Q

        .include "tn13def.inc"

rjmp Anfang
Anfang:
        sbi     ddrb,1      ;PB1

```

```

        sbi    ddrb,3        ;PB3

Schleife:
        sbis   pinb,2        ;PB2 = 1?
        rjmp  Schleife
        sbic   pinb,0
        sbi    portb,1
        sbis   pinb,0
        cbi    portb,1
        sbic   pinb,0
        cbi    portb,3
        sbis   pinb,0
        sbi    portb,3
        rjmp  Schleife

```

Listing 4.15 Das D-Flipflop

Das Programm fragt zuerst den Clock-Eingang PB2 ab. Solange er low ist, wird zum Anfang der Schleife verzweigt, sodass keine Zustandsänderungen an den Ausgängen stattfinden können. Bei Clock = 1 folgt eine zweifache Abfrage des D-Eingangs PB0. Zuerst wird der Ausgang Q = PB1 behandelt, dann der Ausgang /Q = PB3. Testen Sie das Programm mit der Registerkarte Ports. Solange PB2 eingeschaltet ist, ist das Flipflop offen für Pegelwechsel an PB0. Wenn Sie aber PB2 ausschalten, bleibt der letzte Zustand gespeichert.

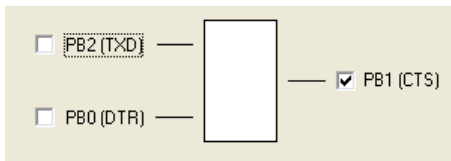


Abb. 4.12 Gespeicherter 1-Zustand ((Soft29.tif))

4.12 Das Toggle-Flipflop

Ein Toggle-Flipflop hat nur einen Eingang und einen Ausgang. Mit jedem Eingangsimpuls wechselt der Ausgangspegel. Das bedeutet zugleich, dass eine Impulsfolge am Eingang zu einem Ausgangssignal mit der halben Frequenz führt.

Das Programm Toggle.asm verwendet ein Unterprogramm WarteAufFlanke zur Erkennung einer positiven Impulsflanke. Im Hauptprogramm wird bei jedem Impuls der Ausgangszustand an PB1 invertiert. Außerdem wird das dazu invertierte Signal an PB3 ausgegeben, wo man eine LED anschließen kann.

```

;Toggle.asm,  PB2 = Clock,
;PB1 = Q, PB3 = /Q

        .include "tn13def.inc"

rjmp Anfang
Anfang:
        sbi    ddrb,1
        sbi    ddrb,3
Schleife:
        rcall  WarteAufFlanke
        cbi    portb,1
        sbi    portb,3
        rcall  WarteAufFlanke
        sbi    portb,1
        cbi    portb,3
        rjmp   Schleife

WarteAufFlanke:
WarteAufLow:
        sbic   pinb,2      ;PB2 = 0?
        rjmp   WarteAufLow
WarteAudHigh:
        sbis   pinb,2      ;PB2 = 1?
        rjmp   WarteAudHigh
        ret

```

Listing 4.16 Das Toggle-Flipflop

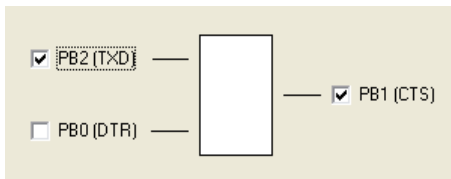


Abb. 4.13 Umschalten mit Impulsen an TXD ((Soft29b.tif))

Das Programm reagiert auf Impulse am Ausgang TXD der seriellen Schnittstelle. Sie können diese Signale mit der Registerkarte Ports erzeugen. Alternativ können Sie auch das Terminal verwenden, wenn TXD in „Ports“ ausgeschaltet ist. Tippen Sie unterschiedliche Zeichen ein. Eine angeschlossene LED blinkt dann. Der Umgang mit dem Terminalprogramm wird im folgenden Abschnitt noch genauer behandelt.

5 Die serielle Schnittstelle

Vom PC kennt man die serielle Schnittstelle, die im Lernpaket Mikrocontroller z.B. als COM1 zur Programmierung des Mikrocontrollers verwendet wird. Aber auch zur Laufzeit eines Programms ist die serielle Schnittstelle nützlich. Man kann den Mikrocontroller z.B. als Messgerät einsetzen und Messwerte zur Anzeige und Auswertung an den PC schicken. Größere Controller wie der ATmega8 besitzen eine serielle Schnittstelle als Hardware-Block. Man benötigt dann aber noch einen Leitungstreiber- und Empfänger, weil die Signale invertiert werden müssen und mit größeren Spannungspegeln arbeiten. Für diesen Zweck wird oft ein Schnittstellenbaustein MAX232 eingesetzt. Hier reichen zwei Widerstände als Leitungsinterface.

5.1 Übertragungsparameter

Beim ATtiny13 gibt es keine in Hardware gegessene serielle Schnittstelle. Man kann sie aber über Software erzeugen. Bei dieser Gelegenheit können die Signale gleich in der erforderlichen Polarität erzeugt werden. Das Interface benötigt dann nicht mehr als zwei Widerstände. Der 100-k Ω -Widerstand von der TXD-Leitung des PCs sorgt in Zusammenarbeit mit den internen Schutzdioden an den Portanschlüssen für einen Begrenzung der +/-12 V-Signale vom PC auf ca. 0V/5V. In umgekehrter Richtung reicht das über 1 k Ω an den PC gesandte Signal mit 5-V-Pegeln aus. Den Widerstand von 1 k Ω könnte man sogar noch weglassen, er dient hier nur zum Schutz gegen versehentlich angeschlossene Fremdspannungen.

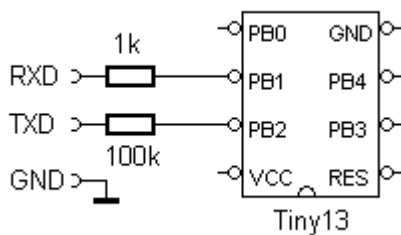


Abb. 5.1 Detailschaltbild zur seriellen Schnittstelle ((Schalt12.gif))

Über die RS232 gesendete Zeichen bestehen aus High- und Low-Zuständen genau definierter Länge. Die Zeichen werden asynchron, also zu beliebigen Anfangszeiten und ohne ein Taktsignal übertragen. Der Empfänger erkennt den Anfang des Zeichens und weiß dann aufgrund der vereinbarten Geschwindigkeit, wann das jeweils nächste Bit zu lesen ist. Das Lernpaket verwendet eine Übertragungsrate von 9600 Baud , d.h. 9600 Bits pro

Sekunde. Ein Bit dauert also $1 \text{ s} / 9600 = 102 \text{ } \mu\text{s}$. Jedes Byte wird durch ein Startbit eingeleitet. Darauf folgen die Datenbits in aufsteigender Folge. Ein Low-Zustand repräsentiert jeweils ein 1-Bit. Am Ende folgen ein oder zwei Stoppbits als Pause bis zum frühesten möglichen Startbit des folgenden Bytes. Die Pause kann aber darüber hinaus beliebig lang sein und muss sich nicht an das Zeitraster halten. Meist sendet man acht Datenbits, ein Stoppbit und kein zusätzliches Parity-Prüfbit. Die Schnittstelleneinstellung lautet dann am PC z.B. COM1:9600,N,8,1.

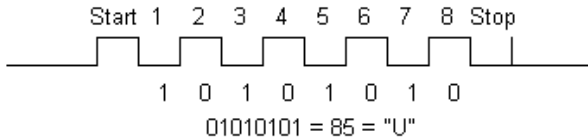


Abb. 5.2 Serielle Datenübertragung von acht Bits ((RS232.gif))

Abb. 5.2 zeigt ein übertragenes Bytes 85, das auch als ASCII-Zeichen U interpretiert werden kann. Sendet man eine Folge dieser Zeichen, entsteht ein Rechtecksignal mit der Frequenz der halben Baudrate.

5.2 Daten-Echo

Das Programm LPmikro enthält ein universelles Terminalprogramm, mit dem Sie Daten senden und empfangen können. In einem ersten kleinen Versuch soll der Mikrocontroller diese Daten unverändert zurücksenden. Dazu reicht es zunächst, den Portzustand der Eingangsleitung zu lesen und die Ausgangsleitung entsprechend zu setzen. Zwischen Eingangs- und Ausgangssignal tritt zwar eine kleine Zeitverzögerung auf, die jedoch bei nicht zu hoher Übertragungsgeschwindigkeit nicht stört.

```
;RxdTxd.asm   kopiert Eingang PB2 nach PB1

        .include "tn13def.inc"
        rjmp Anfang
Anfang:
        sbi    ddrb,1      ;Datenrichtungsbit
Schleife:
        sbic   pinb,2      ;überspringe wenn b.2=0
        sbi    portb,1
        sbis   pinb,2      ;überspringe wenn b.2=1
        cbi    portb,1
        rjmp   Schleife
```

Listing 5.1 RS232-Datenecho

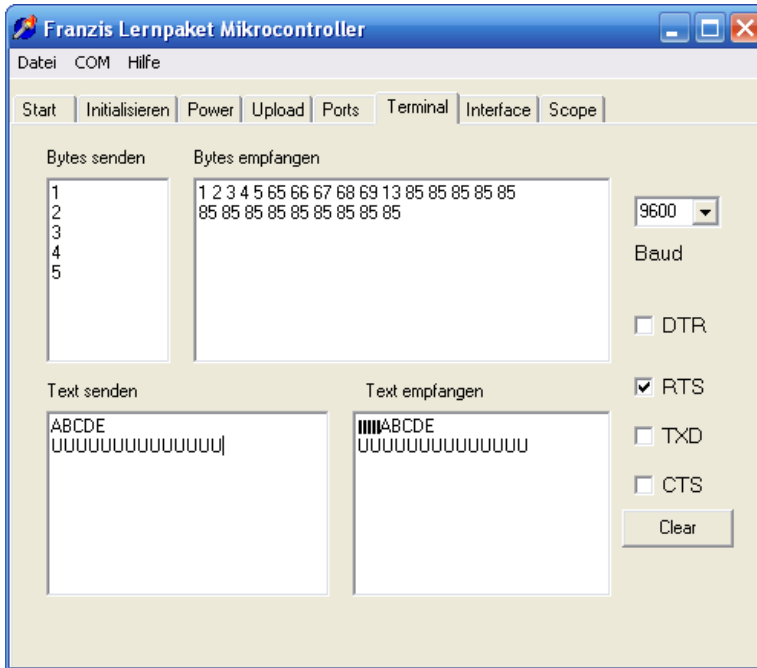


Abb. 5.3 Das Terminalprogramm ((Soft30.tif))

Starten Sie das Terminal in LPmikro und senden Sie einzelne Bytes oder Textzeichen. Bytes werden als Zahl eingegeben und Return abgeschlossen. Textzeichen lassen sich im Textfenster dagegen direkt eintippen. Bei einer Übertragungsrate von 9600 Byte wird jedes Zeichen unverändert empfangen. Die zweifache Darstellung als Text und als Bytes gibt eine Übersicht zur ASCII-Definition.

Testen Sie die Übertragung bei unterschiedlichen Baudraten. Erst bei 115200 Baud kommt es zu einzelnen Übertragungsfehlern. Bei einer Bitlänge von knapp 9 μ s reicht also die zeitliche Auflösung des Assembler-Programms nicht mehr aus. Stellen Sie nach diesen Tests die Übertragungsrate wieder auf 9600 Baud ein.

5.3 Empfangen und Senden

Größere Mikrocontroller aus der AVR-Serie wie der ATmega8 verfügen über einen Hardware-UART (Universal Asynchronous Receiver Transmitter, universeller asynchroner

Empfänger und Sender). Der ATtiny13 enthält eine solche Baugruppe nicht. Deshalb müssen die erforderlichen Funktionen durch Unterprogramme gebildet werden.

Eine Lösung zeigt das Projekt RS232. Hier gibt es ein Unterprogramm RdCOM (Read, Lesen) und ein Unterprogramm WrCOM (Write, Schreiben). Zur Datenübergabe wurde eine Variable A definiert, die das Register R16 belegt. Das Unterprogramm RdCOM empfängt ein Byte und übergibt es in A. Das Unterprogramm WrCOM übernimmt dann das empfangene Byte in A und sendet es zurück.

```
;RS232, Empfangen und Senden mit 9600 Baud bei 1,2 MHz
```

```
.include "tn13def.inc"

.def    A        = r16
.def    Delay    = r17
.def    Count    = r18

;Port B
.equ    TXD      = 1
.equ    RXD      = 2

        rjmp Anfang
Anfang:
        sbi      ddrb,TXD ;Datenrichtung TXD
Schleife:
        rcall    RdCOM
        rcall    WrCOM
        rjmp     Schleife
```

Listing 5.2 Hauptprogramm des Projekts RS232

Die Empfangsroutine RdCOM (Read, Lesen) wartet zunächst auf einen High-Pegel an der Empfangsleitung RXD = PB2. Dann folgt eine Wartezeit von 1,5 Bitlängen (58 Warteschleifen), nach der das erste Datenbit gelesen wird. Das Programm erkennt also den Anfang des Startbits um dann jeweils in der Mitte der Datenbits den Eingangszustand auszuwerten. In der eigentlichen Empfangschleife für die acht Datenbits wird jeweils nur eine Bitlänge gewartet, was mit dem Verzögerungswert Delay = 38 erreicht wird.

Vor der eigentlichen Empfangsschleife mit 8 Durchläufen (Count = 8) wird A gelöscht. Für jedes gelesene 1-Bit wird dann das Bit 7 gesetzt (ori A, 128). Außerdem werden alle Bits schrittweise nach rechts verschoben. Dazu dient der Schiebefehl LSR (Logical Shift Right, bitweise nach rechts schieben). Das zuerst empfangene Bit erscheint daher am Ende an der Stelle 0. Am Ende werden alle Bits mit COM A (One's Complement, bitweise umdrehen) invertiert, weil High-Bits mit Low-Pegeln übertragen werden. Wenn eine Schaltung den

üblichen invertierenden RS232-Leitungstreiber verwendet, müsste die Invertierung entfernt werden.

```
RdCOM:  sbis  pinb,RXD  ;Empfangen
        rjmp RdCOM
        ldi   Delay,58
D1:      dec   Delay
        brne  D1
        ldi   A,0
        ldi   Count,8
L1:      lsr   A
        sbic  pinb,RXD
        ori   A,128
        ldi   Delay, 38
D2:      dec   Delay
        brne  D2
        dec   Count
        brne  L1
        ldi   Delay, 38
D3:      dec   Delay
        brne  D3
        com   A
        ret
```

Listing 5.3 Empfangen eines Byte

Die Senderoutine WrCOM führt den gleichen Vorgang in umgekehrter Richtung aus. Zuerst wird ein Startbit erzeugt. Dann folgt die eigentliche Ausgabeschleife für acht Bits. Hier wird jeweils das Bit 0 in A mit dem Befehl SBRC (Skip if Bit in Register is Cleared, überspringe wenn das Registerbit gelöscht ist) getestet. Je nach Zustand des Bits verzweigt das Programm zu den Adressen ON oder OFF. Am Ende wird noch das Stoppbit mit seinem Low-Pegel angehängt.

```
WrCOM:  sbi    portb,TXD  ;Senden
        ldi   Delay,38
D4:      dec   Delay
        brne  D4
        ldi   Count,8
L2:      sbrc  A,0
        rjmp  OFF
        rjmp  ON
ON:      sbi    portb,TXD
        rjmp  BitD
OFF:     cbi    portb,TXD
        rjmp  BitD
BitD:    ldi   Delay,38
```

```

D5:      dec    Delay
          brne   D5
          lsr    A
          dec    Count
          brne   L2
          cbi    PORTB, TXD
          ldi    Delay, 38
D6:      dec    Delay
          brne   D6
          ret

```

Listing 5.4 Die Senderoutine WrCOM

Testen Sie das Programm mit dem Terminal (vgl. Abb. 5.3). Diesmal ist nur die Übertragungsrate 9600 Baud möglich, alle anderen Baudraten führen zu Fehlern.

5.3 RS232-Testprogramm

Die sichere Übertragung ist von der genauen Einhaltung der Baudrate abhängig, die wiederum an der Genauigkeit des Taktoszillators im ATtiny13 hängt. Sie können die zuverlässige Übertragung testen, indem Sie eine große Anzahl Bytes ohne Pause hintereinander senden. Wenn Sie korrekt empfangen werden, ist alles in Ordnung.

Das Testprogramm erwartet jeweils ein Byte mit der gewünschten Anzahl der zu sendenden Zeichen. Das empfangende Byte wird mit MOV Count2, A (Copy Register, Registerinhalte kopieren) von A in den Zähler Count2 kopiert. Entsprechend oft wird dann die Senderoutine WrCOM mit dem Wert A = 85 aufgerufen.

```
;RS232test1, N Bytes senden
```

```
    .include "tn13def.inc"
```

```
    .def    A      = r16
```

```
    .def    Delay  = r17
```

```
    .def    Count  = r18
```

```
    .def    Count2 = r19
```

```
;Port B
```

```
    .equ    TXD    = 1
```

```
    .equ    RXD    = 2
```

```
        rjmp Anfang
```

```
Anfang:
```

```

        sbi    ddrb, TXD    ;Datenrichtung TXD
Schleife:
        rcall  RdCOM
        mov    Count2, A
Cnt:
        ldi    A, 85
        rcall  WrCOM
        dec    Count2
        brne   Cnt
        rjmp   Schleife

```

Listing 5.5 Test der Schnittstelle

Geben Sie im Terminal z.B. ein Byte 100 ein. Der Mikrocontroller antwortet dann mit 100 Bytes 100 = „U“. Der Datenstrom lässt sich an einem Oszilloskop als Impulsfolge darstellen, aus der ebenfalls die Genauigkeit der Baudrate abgelesen werden kann.

5.4 Automatische Baudratenerkennung

Oft möchte man erreichen, dass ein Mikrocontroller sich automatisch auf die verwendete Baudrate des angeschlossenen Rechners einstellt. Das erfordert eine Messung der Baudrate und eine variable Wartezeit der eigenen Übertragungsroutinen.

Das hier verwendete Programm stellt sich auf Übertragungsraten zwischen 1200 Baud und 19200 Baud ein. Zu Beginn wird eine einzelne Bitlänge gemessen. Der PC muss dazu ein Byte 255 senden, das nur aus einem Startbit mit nachfolgenden Low-Pegeln besteht. Das Unterprogramm Puls führt eine Messung der Pulslänge durch und liefert einen Wert Time, der später als Startwert der Zählschleife für die serielle Übertragung dient. Entsprechend dem übrigen Bedarf an Taktzyklen neben der eigentlichen Wartezeit werden 20 Taktzyklen abgezogen. Zusätzlich wird ein Verzögerungswert Time2 gebildet, der nur halb so groß ist und zur Verzögerung um eine zusätzliche halbe Bitlänge nach dem Empfang eines Startbits dient. Die Division durch Zwei wird durch einen Schiebefehl ROR erreicht.

```

;RS232_AutoBaud, Automatische Erkennung der Baudrate
;Empfangen und Senden mit 1200..19200 Baud bei 1,2 MHz

```

```

        .include "tn13def.inc"

        .def    A          = r16
        .def    Delay      = r17
        .def    Count      = r18
        .def    Time       = r19
        .def    Time2      = r20

```

```

;Port B
.equ    TXD      = 1
.equ    RXD      = 2

    rjmp Anfang

Anfang:
    sbi    ddrb,TXD ;Datenrichtung TXD
    clr    Time

Puls1:
    sbis    pinb,RXD ;Empfangen
    rjmp    Puls1

Puls2:
    inc     Time      ;1
    sbic    pinb,RXD  ;1
    rjmp    Puls2     ;2
    subi    Time, 5    ;-20 Takte
    mov     Time2,Time
    ror     Time2

Schleife:
    rcall   RdCOM
    rcall   WrCOM
    rjmp    Schleife

Wait1:      ;Eine Bitlänge warten
    mov     Delay,Time ;+1

Wait1b:
    nop                      ;1
    dec     Delay            ;1
    brne    Wait1b          ;2
    ret                          ;+4

Wait2:      ;1/2 Bitlänge warten
    mov     Delay,Time2

Wait2b:
    dec     Delay
    brne    Wait2b
    ret

RdCOM:
    sbis    pinb,RXD ;Empfangen
    rjmp    RdCOM
    rcall   Wait1
    rcall   Wait2
    ldi     A,0
    ldi     Count,8

```



```

L1:      lsr      A           ;1
        sbic     pinb,RXD    ;1/2
        ori      A,128       ;2
        rcall    Wait1       ;3 + 5 + 4*Time
        nop      ;1
        nop      ;1
        nop      ;1
        nop      ;1
        dec      Count       ;1
        brne     L1          ;2, ...=19
        rcall    Wait1
        com      A
        ret

WrCOM:   sbi      portb,TXD   ;Senden
        rcall    Wait1
        ldi      Count,8
L2:      sbrc     A,0          ;1
        rjmp     OFF         ;2
        rjmp     ONt
ONt:     sbi      portb,TXD
        rjmp     BitD
OFF:     cbi      portb,TXD   ;2
        rjmp     BitD        ;2
BitD:    rcall    Wait1       ;3 + 5 + 4*Time
        lsr      A           ;1
        dec      Count       ;1
        brne     L2          ;2, ...=19
        cbi      PORTB,TXD
        rcall    Wait1
        ret

```

Listing 5.6 Senden und Empfangen mit variabler Baudrate

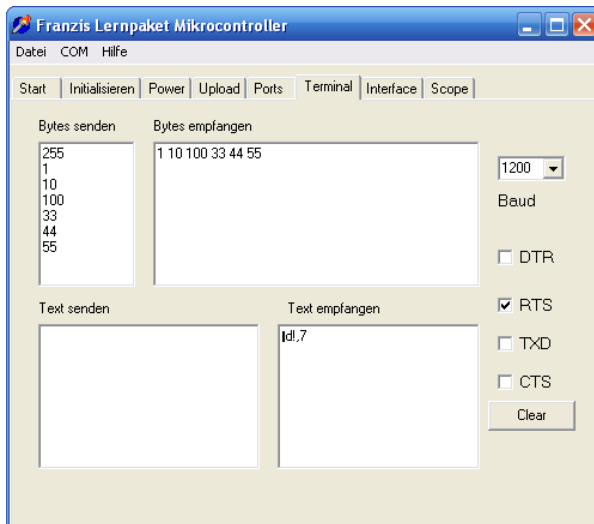


Abb. 5.4 Datenübertragung mit 1200 Baud ((Soft39.tif))

Testen Sie das Programm mit unterschiedlichen Baudraten. Vor jeder Initialisierung mit einem Byte 255 muss ein Reset durchgeführt werden. Schalten Sie dazu die Betriebsspannung über RTS aus und wieder ein. Die Versuche zeigen, dass das Programm zwischen 1200 Baud und 19200 Baud korrekt arbeitet. Bei 39400 Baud treten dagegen einzelne Fehler auf, d.h. die Übertragung ist nicht mehr ausreichend genau.

6 Der Timer/Counter

Der Timer/Counter im ATtiny13 ist ein 8 Bit breiter Vorwärtszähler mit dem Namen Timer0. Er kann auf viele unterschiedliche Arten verwendet werden. Der Zählereingang liegt wahlweise am Anschluss PB2 oder am internen Taktoszillator bzw. an einem Vorteiler hinter dem Taktgenerator. Das Kapitel über den Timer im Datenblatt des ATtiny13 ist recht umfangreich und beschreibt sehr viele Möglichkeiten. Hier sollen die wichtigsten Fälle erprobt werden.

6.1 Zeitmessung

Das eigentliche Zählerregister TCNT0 kann zu jeder Zeit gelesen und beschrieben werden. Entsprechend der Zählerauflösung von 8 Bit beträgt der maximale Zählerstand 255. Danach folgt ein Übertrag auf den Wert Null.

Zum Starten des Zählers muss das Steuerregister TCCR0B initialisiert werden. Die wichtigsten Einstellungen liegen in den unteren drei Bits, sodass Werte zwischen 0 und 7 über den Takteingang entscheiden:

0	Timer/Counter gestoppt
1	Timer ohne Vorteiler
2	Timer mit Vorteiler / 8
3	Timer mit Vorteiler / 64
4	Timer mit Vorteiler / 256
5	Timer mit Vorteiler / 1025
6	Externer Takt, fallende Flanke
7	Externer Takt, steigende Flanke

Der Zeitgeber kann verwendet werden um den Zeitbedarf für die Aussendung eines Byte zu ermitteln. Vor dem Senden muss das Timerregister gelöscht und der Timer gestartet werden. In diesem Beispiel wird der Vorteiler / 8 verwendet, d.h. der Zähler läuft mit einem Takt von $1200 \text{ kHz} / 8 = 150 \text{ kHz}$. Nach dem Senden des Bytes (hier 85) wird der Timer gestoppt, das Timerregister ausgelesen und über die serielle Schnittstelle gesendet. Der gesamte Vorgang wird durch ein beliebiges Byte vom PC gestartet.

```
;Timer0_1.asm, Zeitmessung

.include "tn13def.inc"

.def    A          = r16
```

```

        .def    Delay    = r17
        .def    Count    = r18

;Port B
.equ     TXD      = 1
.equ     RXD      = 2

        rjmp   Anfang

Anfang:
        sbi     ddrb,TXD ;Datenrichtung TXD
Schleife:
        rcall   RdCOM
        ldi     A,0      ;Timer0 löschen
        out     TCNT0,A
        ldi     A,2      ;Start mit Vorteiler / 8
        out     TCCR0B,A
        ldi     A,85
        rcall   WrCOM
        ldi     A,0      ;Timer0 Stop
        out     TCCR0B,A
        in      A,TCNT0
        rcall   WrCOM
        rjmp    Schleife

```

Listing 6.1 Zeitmessung

Senden Sie beliebige Bytes, z.B. immer das Byte 1. Der Mikrocontroller antwortet jeweils mit zwei Bytes. Das erste Antwortbyte ist 85 und wird nur gesendet um den Zeitbedarf zu ermitteln. Das zweite Byte ist die gemessene Zeit. Hier findet sich der Wert 155 oder 156. Bei einem Takt von 150 kHz entspricht das Ergebnis 156 einer Zeit von 1,04 ms.

Tatsächlich benötigen insgesamt zehn Bits (Startbit, acht Datenbits und ein Stoppbit) $10 * 1 \text{ ms} / 9600 = 1,04 \text{ ms}$. Theorie und Praxis passen also gut zusammen. Allerdings muss man bedenken, dass eine Abweichung des Taktgenerators nicht erfasst wird, weil sie sowohl die serielle Ausgabe als auch die Zeitmessung beeinflusst.

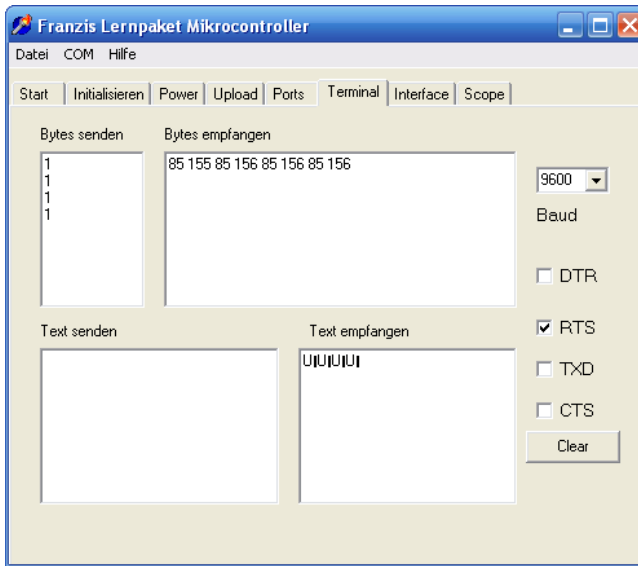


Abb. 6.1 Messung des Zeitbedarfs von WrCOM ((Soft33.tif))

6.2 Impulse zählen

Um den Timer0 als Impulszähler zu nutzen, muss das Register TCCR0B mit dem Wert 6 (fallende Eingangsflanke) oder 7 (steigende Eingangsflanke) initialisiert werden. Der Zählereingang wird dann intern mit dem Pin 7 (PB2) verbunden. Dieser Anschluss dient hier zugleich als Empfangseingang RXD, der von der Sendeleitung TXD des PC angesteuert wird.

Da der Timer völlig autonom arbeitet, spricht nichts dagegen, den Empfang von Daten und das Zählen der zugehörigen Impulse gleichzeitig ablaufen zu lassen. Sendet man z.B. ein Byte 0, besteht dieses nur aus einem einzelnen Impuls, der gezählt wird. Dagegen erzeugt man mit dem schon viel verwendeten Byte 85 zusammen mit dem Startbit genau fünf Impulse. Im Programm Timer0_2.asm wird der Zähler beim Start auf Null gesetzt. Dann werden laufend Bytes empfangen und die zugehörigen Impulse gezählt. Das Programm sendet nach jedem empfangenen Byte den aktuellen Zählerstand zurück.

```
;Timer0_2.asm, Impulse zählen
```

```
.include "tn13def.inc"
```

```
.def A = r16
```

```
.def Delay = r17
```

```

.def    Count    = r18

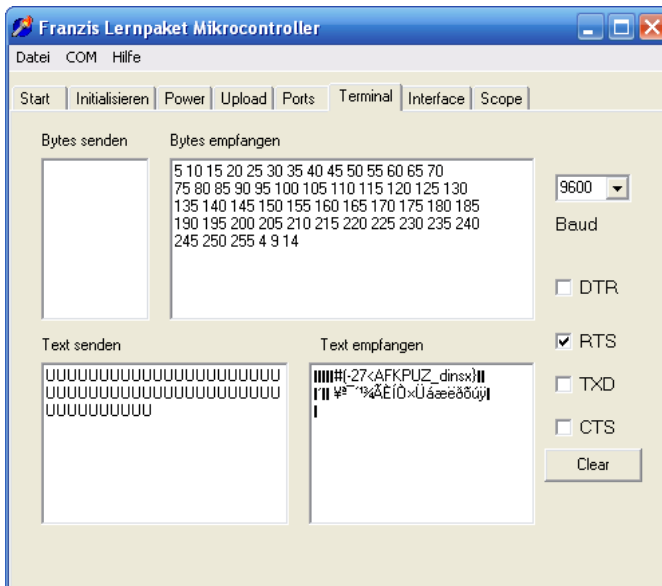
;Port B
.equ    TXD      = 1
.equ    RXD      = 2

        rjmp     Anfang
Anfang:
        sbi      ddrb,TXD    ;Datenrichtung TXD
        ldi      A,0         ;Timer0 löschen
        out      TCNT0,A
        ldi      A,7         ;Impulse an PB2
        out      TCCR0B,A
Schleife:
        rcall    RdCOM
        in       A,TCNT0
        rcall    WrCOM
        rjmp     Schleife

```

Listing 6.2 Ein Impulszähler

Senden Sie z.B. eine Folge von Großbuchstaben U. Wie erwartet erhöht sich der Zähler jeweils um fünf. Bei einem Zählerstand über 255 erfolgt ein Übertrag auf 4, denn $260 = 256 + 4$.



6.3 Timer-Interrupt

Ein Interrupt ist eine automatische Unterbrechung des Programmlaufs, wobei ein vorbereitetes Interrupt-Unterprogramm ausgeführt wird. Hier soll der Timer bei jedem Überlauf das Unterprogramm TIM0_OVF aufrufen. Während ein normales Unterprogramm mit RET abgeschlossen wird, steht am Ende einer Interrupt-Routine ein RETI (Return from Interrupt, Rückkehr aus dem Interrupt).

Der ATtiny13 kennt eine Reihe möglicher Interrupts, die das Programm automatisch zu einer bestimmten Sprungadresse führen. Bei einem Überlauf des Timers verzweigt das Programm zur Adresse 0x0003. Hier muss ein Sprung auf die gewünschte Interruptroutine stehen. Mit .org 0x0003 wird der erforderliche Sprungbefehl an die richtige Stelle gesetzt.

Damit der Interrupt tatsächlich ausgeführt wird, muss im Timer/Counter Interrupt Mask Register TIMSK0 das Bit 1 (TOV0: Timer/Counter0 Overflow Flag) gesetzt werden. Außerdem müssen Interrupts für das gesamte Programm erlaubt werden. Dazu dient der Assemblerbefehl SEI (Set Global Interrupt Flag, setze das globale Interrupt-Flag).

```
;Timer0_3.asm, Timer-Interrupt

.include "tn13def.inc"

.def    A            = r16

        rjmp Anfang
        .org 0x0003
Anfang:  rjmp    TIM0_OVF    ;Timer0 Overflow

        sbi     ddrb,3      ;Datenrichtung PB3
        ldi     A,5         ;Start mit Vorteiler / 1024
        out     TCCR0B,A
        ldi     A,2
        out     TIMSK0,A    ;Timer Interrupt freigeben
        sei     ;Globaler Interrupt frei

Schleife:
        rjmp    Schleife

TIM0_OVF:                                ;Timer Interrupt
        com     A
        out     portb,A
```

reti

Listing 6.3 Ein LED-Blinker mit Timer-Interrupt

Das Programm Timer0_3.asm richtet zunächst den Timer und den Interrupt ein. Dann führt es eine Endlosschleife aus, in der nichts passiert. In regelmäßigen Abständen wird jedoch das Hauptprogramm unterbrochen um TIM0_OVF aufzurufen. In diesem Interrupt-Unterprogramm wird der Zustand am Port B umgeschaltet. An PB3 entsteht damit ein Rechtecksignal, das eine LED zum Blinken bringt.

Die Blinkfrequenz lässt sich aus den Timer-Einstellungen ableiten. Der Vorteiler ist auf 1024 eingestellt. Nach jeweils 256 Impulsen gibt es einen Timer-Überlauf. Die Interruptfrequenz beträgt damit $1200 \text{ kHz} / 1024 / 256 = 4,58 \text{ Hz}$ und ergibt ein gut sichtbares Blinken mit rund 2 Impulsen pro Sekunde.

6.4 Minuten-Timer

Das Programm Timer0_4.asm misst die Zeit in Sekunden und Minuten. Der Timer wird vom Vorteiler mit $1,2 \text{ MHz} / 64$ getaktet. Er zählt diesmal nicht 256 Takte bis zu einem Überlauf, sondern nur 250 Takte. Dazu wird das Timer-Register jedes Mal mit einem neuen Startwert von 6 geladen. Daraus ergibt sich eine Interruptfrequenz von 75 Hz. Nach jeweils 75 Interrupts wird der Sekundenzähler erhöht. Außerdem wird nach jeweils 60 Sekunden der Minutenzähler erhöht.

Im Hauptprogramm kann nun laufend der Minutenzähler abgefragt werden. Mit dem Start des Programms wird der Port PB3 jeweils nach einer Sekunde invertiert. Nach genau drei Minuten soll der Interrupt und damit das LED-Blinken abgeschaltet werden. Dies geschieht, indem der globale Interrupt mit dem Befehl CLI (Clear Global Interrupt Flag, lösche das globale Interrupt-Flag) gesperrt wird. Das Hauptprogramm schaltet dann den LED-Ausgang ganz ein. Das Programm blinkt also drei Minuten lang und zeigt das Ende der Zeit mit einem Dauerleuchten.

In der Abfrageschleife des Hauptprogramms wird laufend der Inhalt des Minutenregisters mit einem konstanten Wert verglichen. Das Ergebnis des Vergleichs führt dann zu einer entsprechenden Verzweigung. In der Interruptroutine werden aber ebenfalls Vergleiche durchgeführt. Prinzipiell weiß man nie, an welcher Stelle das Hauptprogramm unterbrochen wird. Es kann also vorkommen, dass die Unterbrechung genau nach dem Vergleich stattfindet. In dem Fall könnte das Hauptprogramm nach der Rückkehr vom Interrupt ein falsches Ergebnis auswerten. Um dies zu verhindern, muss man das Status-Register SREG beim Start der Interruptroutine sichern und am Ende wiederherstellen. Damit sind alle Flags gesichert, die das Ergebnis von Vergleichen wiedergeben. Zur Sicherung wird hier ein Register verwendet.


```

;Timer0_4.asm, Timer-Interrupt

.include "tn13def.inc"

.def    A      = r16
.def    timer   = r17
.def    timer2  = r18
.def    sek     = r19
.def    min     = r20
.def    sicher  = r21

        rjmp Anfang
        .org 0x0003
Anfang: rjmp TIM0_OVF      ;Timer0 Overflow

        sbi     ddrb,3      ;Datenrichtung PB3
        ldi     A,3         ;Start mit Vorteiler / 64
        out     TCCR0B,A
        ldi     A,2
        out     TIMSK0,A    ;Timer Interrupt freigeben
        clr     timer2
        clr     sek
        clr     min
        sei                      ;Globaler Interrupt frei

Schleife:
        cpi     min,3       ;3 Minuten?
        brlo    Schleife
        cli                      ;Interrupt sperren
        sbi     portb,3     ;LED an
        rjmp    Schleife

TIM0_OVF:                      ;Timer Interrupt
        in      sicher, SREG
        ldi     timer, 6
        out     TCNT0,timer ;250 bis Overflow
        inc     timer2
        cpi     timer2,75
        brlo    TIM0_ende
        clr     timer2
        com     A
        out     portb,A
        inc     sek
        cpi     sek,60
        brlo    TIM0_ende
        inc     min

```

```

        clr     sek
TIM0_ende:
        out     SREG, sicher
        reti

```

Listing 6.4 Der Dreiminuten-Timer

Starten Sie das Programm mit einer an PB3 angeschlossenen LED. Die LED blinkt drei Minuten lang und bleibt dann an.

6.5 PWM-Ausgang

Der Timer des ATtiny13 kennt zahlreiche unterschiedliche Betriebsarten, darunter auch den PWM-Modus. Ein PWM-Signal ist ein Rechtecksignal mit einstellbarem Puls/Pausenverhältnis. Es können zwei unabhängige PWM-Ausgänge PWMA und PWMB erzeugt werden, die an den Ausgängen PB0 (OCOA) und PB1 (OCOB) liegen. Allerdings verwendet die Platine des Lernpakets den Anschluss PB1 für die serielle Schnittstelle. Es bleibt also nur der PWM-Ausgang PWMA an PB0, der sinnvoll genutzt werden kann.

Zur Initialisierung des PWM-Ausgangs muss das Register TCCR0A mit dem Wert 0x83 geladen werden. Außerdem muss der Vorteiler in TCCR0B eingestellt werden. Der aktuelle PWM-Ausgabewert zwischen 0 und 255 schreibt man in das Register OCR0A.

```

;Timer0_5.asm, PWM-Ausgabe

        .include "tn13def.inc"

        .def      A      = r16
        .def      Delay  = r17
        .def      Count  = r18

;Port B
        .equ      TXD    = 1
        .equ      RXD    = 2

        rjmp      Anfang

Anfang:
        sbi       ddrb, TXD    ;Datenrichtung TXD
        sbi       ddrb, 0      ;Datenrichtung PB0
        ldi       A, 0         ;PWM initialisieren
        out       OCR0A, A
        ldi       A, 0x83      ;nur Kanal A
        out       TCCR0A, A

```

```

        ldi    A,2          ;Start mit Vorteiler / 8
        out    TCCR0B, A
Schleife:
        rcall   RdCOM
        out     OCR0A, A
        rjmp    Schleife

```

Listing 6.5 PWM-Ausgabe über das Terminal

Schließen Sie eine LED an PB0 an wie in Abb. 3.2. Geben Sie dann im Terminal beliebige Bytes ein. Die LED-Helligkeit wird auf diese Weise gesteuert. Mit einem Wert von Null ist sie ganz aus, mit 255 wird die maximale Helligkeit erreicht.

6.6 Der weiche *Blinker*

Alle bisherigen Blinkprogramme erzeugten harte Übergänge zwischen An und Aus. Mit dem PWM-Ausgang lassen sich auch weiche Übergänge erzeugen. Die LED-Helligkeit wird kontinuierlich erhöht und verringert. Dazu übergibt das Programm aufsteigende und absteigende Bytes an das Register OCR0A. Die Veränderung des Registers wird durch das Unterprogramm Warten verzögert.

```

;SoftBlinker.asm, PWM-Ausgabe

.include "tn13def.inc"

.def    A      = r16
.def    Delay  = r17
.def    Count  = r18

        rjmp   Anfang
Anfang:
        sbi     ddrb,0      ;Datenrichtung PB0
        ldi     A, 0        ;PWM initialisieren
        out     OCR0A, A
        ldi     A, 0x83     ;nur Kanal A
        out     TCCR0A, A
        ldi     A, 0x02
        out     TCCR0B, A
        ldi     A,2         ;Start mit Vorteiler / 8
        out     TCCR0B,A
Schleife:
        ldi     A,0
        ldi     Count,255
Up:      rcall   Warten

```

```

        out    OCR0A, A
        inc    A
        dec    Count
        brne   Up
        ldi    Count, 255
Down:    rcall  Warten
        out    OCR0A, A
        dec    A
        dec    Count
        brne   Down
        rjmp   Schleife

Warten:
        ldi    Delay, 250
W1:      nop
        nop
        nop
        nop
        nop
        dec    Delay
        brne   W1
        ret

```

Listing 6.6 An- und Abschwollen der Helligkeit

Schließen Sie die LED mit einem Vorwiderstand von 1 k Ω an PB0 an. Mit dem Start des Programms erhalten Sie ein „weiches“ Blinken.

6.7 Frequenzmessung

Die Frequenz eines Rechtecksignals ist definiert als die Anzahl von Schwingungen dividiert durch die Zeit. Deshalb müssen bei einer Frequenzmessung zwei Aufgaben gleichzeitig erledigt werden, nämlich das Zählen der Impulse und eine Zeitmessung. Wenn die Auflösung der Messung ein Hertz betragen soll, muss eine Totzeit von einer Sekunde gewählt werden, d.h. der Impulszähler muss nach genau einer Sekunde angehalten werden.

Das Programm Frequenz.asm löst die Aufgabe der Zeitmessung mit einer Interruptroutine, die in ähnlicher Form schon aus Kap. 6.4 bekannt ist. Das Zählen der Impulse an PB4 übernimmt das Hauptprogramm. Gleichzeitig muss überwacht werden, ob die Messzeit abgelaufen ist.

Für das Zählen der Impulse reicht ein Byte nicht mehr aus, wenn Signale mit Frequenzen über 255 Hz gemessen werden sollen. Deshalb besteht der Zähler aus einem Highbyte und einem Lowbyte. Bei einem Übertrag des Lowbyte wird das Highbyte erhöht. Vor dem

Beginn der Messung werden beide gelöscht, d.h. der Zähler beginnt bei Null. Nach der Messzeit von einer Sekunde werden beide Ergebnisbytes über die serielle Schnittstelle ausgegeben.

```
;Frequenz.asm, Frequenzmessung an PB4

        rjmp  Anfang
        .org  0x0003
        rjmp  TIM0_OVF      ;Timer0 Overflow
Anfang:
        sbi    ddrb,TXD
        sbi    ddrb,3
        ldi    A,3          ;Start mit Vorteiler / 64
        out    TCCR0B,A
        ldi    A,2
        out    TIMSK0,A     ;Timer Interrupt freigeben
Schleife:
        clr    timer2
        clr    sek
        clr    freqlow
        clr    freqhigh
        sei                    ;Interrupt freigeben
Z1:      sbic   pinb,4
        rjmp   Z2
        sbi    portb,3      ;Testsignal
        cpi    sek,1
        brsh   Z4
        rjmp   Z1
Z2:      sbis   pinb,4
        rjmp   Z3
        cbi    portb,3      ;Testsignal
        cpi    sek,1
        brsh   Z4
        rjmp   Z2
Z3:      inc    freqlow
        brne   Z4
        inc    freqhigh
Z4:      cpi    sek,1
        brlo   Z1
Z5:      cli                    ;Interrupt sperren
        mov    A,freqhigh
        rcall  WrCOM
        mov    A,freqlow
        rcall  WrCOM
        rjmp   Schleife
```

```

TIM0_OVF:                                ;Timer Interrupt
    in    sicher, SREG
    ldi    timer, 6
    out    TCNT0, timer    ;250 bis Overflow
    inc    timer2
    cpi    timer2, 75
    brlo   TIM0_ende
    clr    timer2
    inc    sek
TIM0_ende:
    out    SREG, sicher
    reti

```

Listing 6.7 Frequenzmessung mit serieller Ausgabe

Das Programm erzeugt während der Messung ein Testsignal am Ausgang PB3. Verbinden Sie PB3 mit dem Eingang PB4 und starten Sie das Terminal. Im Sekundentakt werden nun jeweils zwei Bytes empfangen: 166 und 224. Die gemessene Anzahl der Impulse ist damit $166 * 256 + 226 = 42722$, d.h. die Frequenz beträgt 42722 Hz.

Trennen Sie die Verbindung zu PB3 und berühren Sie den Eingang PB4 über ein Stück Draht oder einen Widerstand mit dem Finger. In den meisten Fällen wird nun eine Frequenz von 50 Hz angezeigt, d.h. die Ausgabe lautet 0, 50. Sie können mit der anderen Hand ein isoliertes Netzkabel berühren um die 50-Hz-Einstreuung zu vergrößern.

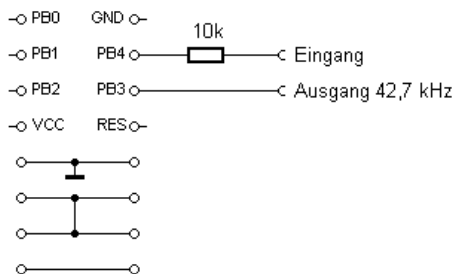


Abb. 6.3 Anschlüsse des Frequenzmessers ((Schalt16.gif))

Über einen Schutzwiderstand von 10 k Ω dürfen nun auch andere Signalquellen angeschlossen werden. Für eine Messung muss das Eingangssignal eine ausreichende Amplitude über etwa 2,5 V aufweisen.

7 Der AD-Wandler

Der Analog-Digitalwandler ist ein wichtiges Bindeglied zwischen analoger und digitaler Elektronik. Analoge Größen wie Spannung, Strom und Helligkeit lassen sich messen und digital weiter verarbeiten. Damit können wichtige Messgeräte bis hin zum Speicheroszilloskop programmiert werden.

7.1 10-Bit-Messung

Der interne AD-Wandler des ATtiny13 setzt eine am Eingang anliegende Spannung in einen Zahlenwert um. Die Auflösung beträgt 10 Bit, d.h. der Ausgangswert liegt im Bereich 0...1023. Das Ergebnis einer Wandlung besteht aus zwei Bytes, die aus dem Register ADCH (Highbyte) und ADCL (Lowbyte) gelesen werden.

Beim Start des Controllers muss der AD-Wandler zuerst einmal initialisiert werden. Dabei muss ein Vorteiler für den Arbeitstakt des Wandlers eingeschaltet werden. Für beste Genauigkeit sollte der AD-Takt im Bereich 50 kHz bis 200 kHz liegen. Die unteren drei Bits des Registers ADCSRA bestimmen den Teilerfaktor. Ein Wert von 3 führt zu einem Teilerfaktor 8, d.h. der AD-Takt beträgt $1200 \text{ kHz} / 8 = 150 \text{ kHz}$. Außerdem muss das Bit 6 (ADEN) im gleichen Register gesetzt werden um den Wandler einzuschalten.

Die eigentliche Messung wird im Unterprogramm RdADC ausgeführt. Am Eingang des Wandlers sitzt ein Multiplexer (Umschalter), der den gewünschten Eingang auswählt. Das Register ADMUX wird mit dem gewünschten Kanal, z.B. 2 oder 3 geladen. Die eigentliche Messung startet, wenn das Bit ADSC im Register ADMUX gesetzt wird. Nach dem Ende der Wandlung setzt der AD-Wandler das Bit ADSC im Register ADCSRA. Das Unterprogramm wartet so lange in einer Abfrageschleife mit Sprüngen zum Label ADrdy. Danach wird sofort eine zweite Messung am selben Eingangskanal gestartet, weil sonst noch das Ergebnis der jeweils letzten Messung ausgelesen würde. Dies ist eine Besonderheit beim ATtiny13 im Vergleich zu Controller der ATmega-Serie. Wenn der Eingangskanal gewechselt werden kann, müssen immer zwei Messungen unmittelbar hintereinander ausgeführt werden. Nach der zweiten Messung wird dann als Highbyte in A und das Lowbyte in B gelesen.

Das Hauptprogramm empfängt jeweils den gewünschten Kanal als ein Byte in A und ruft damit RdADC auf. Dann wird zuerst das Highbyte und danach das Lowbyte zurückgesendet.

```
;ADC1.asm, AD-Wandler abfragen
```

```
.include "tn13def.inc"
```

```

.def    A      = r16
.def    B      = r17
.def    Delay  = r18
.def    Count  = r19

;Port B
.equ    TXD    = 1
.equ    RXD    = 2

        rjmp   Anfang
Anfang:
        sbi     ddrb,TXD    ;Datenrichtung TXD
AdcInit:
        ldi     A,3         ;Clock / 8
        out     ADCSRA,A
        sbi     ADCSRA,ADEN ;AD einschalten
Schleife:
        rcall   RdCOM
        rcall   RdADC
        rcall   WrCOM
        mov     A,B
        rcall   WrCOM
        rjmp    Schleife

RdADC:
        out     ADMUX,A
        sbi     ADCSRA,ADSC    ;Wandlung starten
ADrdy:
        sbic    ADCSRA,ADSC
        rjmp    ADrdy
        sbi     ADCSRA,ADSC
ADrdyb:
        sbic    ADCSRA,ADSC
        rjmp    ADrdyb
        in      B,ADCL
        in      A,ADCH
        ret

```

Listing 7.1 Messungen mit dem AD-Wandler

Testen Sie das Programm mit dem Terminal. Sie können die Kanäle 2 (PB4) und 3 (PB3) messen. Die Kanäle 0 (PB5 = RES) und 1 (PB2 = TXD) können zwar ebenfalls gewählt werden, sie liefern aber keine sinnvollen Werte. An PB 3 und PB4 können jedoch Spannungen angelegt werden, die das Programm messen kann.

Wenn beliebige externe Spannungen gemessen werden sollen, müssen Schutzwiderstände in Reihe zum Eingang gelegt werden. Sie begrenzen den Strom wenn die Messspannung den erlaubten Bereich zwischen 0 V und 5 V überschreitet. Als Schutz gegen Überspannungen bis ca. 30 V eignet sich am besten ein Widerstand von 10 k Ω . Aber auch 1 k Ω reicht fast immer aus, um eine Beschädigung des Eingangs zu verhindern. Vermeiden Sie es nach Möglichkeit, höhere Spannungen als 5 V anzuschließen.

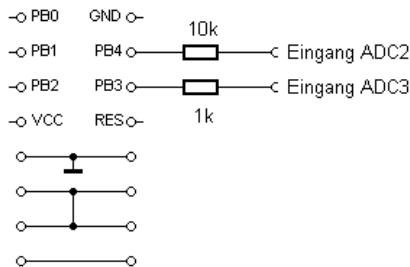


Abb. 7.1 Analoge Eingänge mit Schutzwiderständen ((Schalt17.gif))

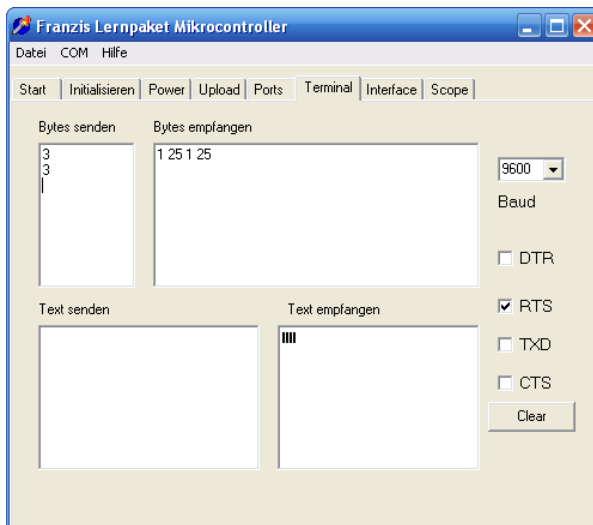


Abb. 7.2 Spannungsmessung an ADC3 ((Soft34.tif))

Senden Sie den Kanal 3. Je nach angeschlossener Eingangsspannung erhalten Sie z.B. das Ergebnis 1, 25 (Abb. 7.2). Es entspricht dem digitalen Wert $1 * 256 + 25 = 281$. Die gemessene Spannung ist dann $5 \text{ V} * 281 / 1023 = 1,37 \text{ V}$.

7.2 8-Bit-Messung

Oft kommt man mit einer Auflösung von 8 Bit aus. Das Ergebnis des AD-Wandlers kann mit dem Bit ADLAR (ADC Left Adjust Result, links justiertes Ergebnis) im Register ADMUX so verschoben werden, dass die oberen acht Bits im Highbyte liegen. Es reicht dann, nur das Highbyte auszulesen und die unteren zwei Bits zu verwerfen. Die Initialisierung des D-Wandlers ändert sich nicht, wurde diesmal aber in das Unterprogramm ADCinit ausgelagert.

```
;ADC2.asm, AD-Wandler abfragen, 8 Bit

.include "tn13def.inc"

.def    A        = r16
.def    Delay    = r18
.def    Count    = r19

;Port B
.equ    TXD      = 1
.equ    RXD      = 2

        rjmp Anfang
Anfang:
        sbi      ddrb,TXD    ;Datenrichtung TXD
        rcall    AdcInit
Schleife:
        rcall    RdCOM
        rcall    RdADC
        rcall    WrCOM
        rjmp     Schleife

AdcInit:
        ldi      A,3          ;Clock / 4
        out      ADCSRA,A
        sbi      ADCSRA,ADEN ;AD einschalten
        ret

RdADC:
        out      ADMUX,A
        sbi      ADMUX,ADLAR  ;Left adjust
        sbi      ADCSRA,ADSC  ;Wandlung starten
ADrdy:
        sbic     ADCSRA,ADSC
        rjmp     ADrdy
        sbi      ADCSRA,ADSC
```

```

ADrdyb:
    sbic  ADCSRA,ADSC
    rjmp  ADrdyb
    in    A,ADCH
    ret

```

Listing 7.2 Messung mit der Auflösung 8 Bit

Geben Sie im Terminal z.B. den Kanal 3 an. Sie erhalten nur ein Ergebnisbyte. Ein Ergebnis 70 bedeutet z.B. eine Spannung von $5 \text{ V} * 70 / 255 = 1,37 \text{ V}$.

7.3 Interne Referenz

Der AD-Wandler verwendet wahlweise eine externe oder interne Spannungsreferenz. Bisher wurde die externe Referenz $VCC = 5 \text{ V}$ benutzt. Mit dem Bit REFS0 (Reference Selection Bit, interne Referenz einschalten) im Register ADMUX wählen Sie die interne Referenz-Spannungsquelle von 1,1 V. Das Projekt ADC3.asm zeigt die Verwendung der internen Referenz für eine 8-Bit-Messung. Der Messbereich reicht nun bis 1,1 V.

```

RdADC:
    out    ADMUX,A
    sbi    ADMUX,ADLAR    ;Left adjust
    sbi    ADMUX,REFS0    ;1,1 V Referenz
    sbi    ADCSRA,ADSC    ;Wandlung starten
ADrdy:
    sbic   ADCSRA,ADSC
    rjmp   ADrdy
    sbi    ADCSRA,ADSC
ADrdyb:
    sbic   ADCSRA,ADSC
    rjmp   ADrdyb
    in     A,ADCH
    ret

```

Listing 7.3 Verwendung der internen Referenz

7.4 Zweipunktregler

Jeder Regelkreis führt einen Vergleich von Istwert und Sollwert durch, um eine Stellgröße zu verändern. Zweipunktregler kennen nur zwei Ausgangszustände, An und Aus. Für einfache Temperaturregelungen verwendet man oft Thermostaten. Bei zu niedriger

Temperatur wird eine Heizung eingeschaltet, bei zu hoher Temperatur wird sie ausgeschaltet.

Ein einfaches Modell eines Zweipunktreglers lässt mit einem RC-Glied aufbauen. Die Spannung am Kondensator wird gemessen und mit einem Sollwert verglichen. In Abhängigkeit von diesem Vergleich wird ein Ausgang ein- oder ausgeschaltet. Der Kondensator lädt sich also weiter auf oder entlädt sich etwas. Die Spannung kann eng an ihrem Sollwert gehalten werden.

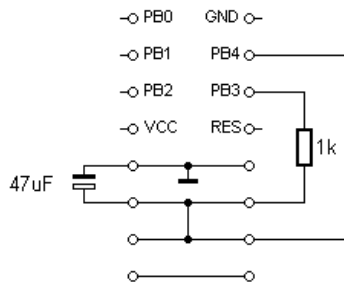


Abb. 7.3 Ein Regelkreis ((Schalt13.gif))

Der eigentliche Regelkreis im Programm Regler1.asm verwendet einen Vergleich zwischen dem Messwert A und dem Sollwert in B = 157. Der Vergleich erfolgt mit dem Assemblerbefehl CP (Compare, Vergleich). Der folgende Befehl BRLO (Branch if Lower, verzweige wenn kleiner) wertet den Vergleich aus.

```
;Regler1.asm, Zweipunktregler 3,0 V

.include "tn13def.inc"

.def    A      = r16
.def    B      = r17
.def    Delay  = r18
.def    Count  = r19

;Port B
.equ    TXD    = 1
.equ    RXD    = 2

    rjmp Anfang
Anfang:
    sbi      ddrb,3    ;Datenrichtung PB4
    rcall   AdcInit
Schleife:
    ldi      A,2        ;ADC2 an PB4
```

```

rcall RdADC
ldi B,153 ;3V/5V*255=153
cp A,B
brlo PB3ein
PB3aus:
cbi portb,3
rjmp Schleife
PB3ein:
sbi portb,3
rjmp Schleife

```

Listing 7.4 Regelung der Kondensatorspannung

Am Elko wurde eine Spannung von 3,01 V gemessen. Der Regler arbeitet also sehr gut. Der Erfolg ist durch die hohe Geschwindigkeit des Reglers bedingt. Die Welligkeit der Ausgangsspannung ist gering. Auch ohne ein Messgerät kann man sich davon überzeugen, dass am Kondensator eine konstante Spannung steht. Eine LED mit Vorwiderstand zeigt kein sichtbares Flackern.

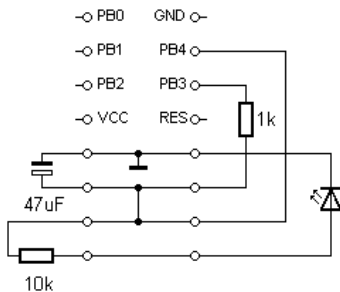


Abb. 7.4 LED an der geregelten Spannung ((Schalt14.gif))

7.5 Dämmerungsschalter

Ein Dämmerungsschalter wertet die Umgebungshelligkeit aus, um eine Lampe bei einsetzender Dämmerung einzuschalten. Es handelt sich im Prinzip um einen Zweipunktregler. Allerdings wird hier eine Hysterese benötigt, also ein gewisser Abstand zwischen dem Einschalt- und dem Ausschaltpunkt. Ohne Hysterese könnte etwas Licht der eingeschalteten Lampe auf den Lichtsensor fallen und sie sofort wieder ausschalten. Außerdem sollen geringen Schwankungen oder ein gewisses Flackern künstlicher Beleuchtungen nicht zu einem Umschalten führen. Dieselbe Funktion erfüllt auch eine Lichtschranke. Bei Abschattung eines Lichtsensors soll ein Signal ausgegeben werden.

7.6 Alarmanlage

Die optische Alarmanlage verwendet die selbe Schaltung wie der Dämmerungsschalter (vgl. Abb. 7.4). Es wird die Änderung der Helligkeit ausgewertet. Immer wenn die Beleuchtung sich schnell ändert, wird über die LED ein Alarm ausgegeben. Dabei reicht schon eine geringe Abschattung, um eine Person in der Nähe des Sensors zu erkennen.

Die Besonderheit des Programms liegt darin, dass die Schaltschwelle automatisch an die Umgebungshelligkeit angepasst wird. Natürlich vorkommende langsame Änderungen der Helligkeit lösen keinen Alarm aus. Erst wenn der Vergleich mit der zuletzt gemessenen Helligkeit eine über einem Grenzwert liegende Verdunkelung zeigt, leuchtet die LED.

```
;Alarm.asm, Optische Alarmanlage
    rjmp Anfang
Anfang:
    sbi     ddrb,3      ;Datenrichtung PB4
    sbi     ddrb,TXD
    rcall   AdcInit
    ldi     A,2          ;ADC2 an PB4
    rcall   RdADC
    mov     B,A
    rjmp    Alarm
Schleife:
    ldi     A,2          ;Neue Messung
    rcall   RdADC
    cp      A,B
    brsh    Ende
    sub     B,A          ;Alt - Neu
    cpi     B,2          ;2 Stufen dunkler?
    brsh    Alarm
Ende:
    mov     B,A          ;Alt = Neu
    rcall   Pause2
    rjmp    Schleife
Alarm:
    sbi     portb,3
    rcall   Pause2
    cbi     portb,3
    rjmp    Ende

Pause:
    ldi     Delay,250
P1:     nop
        dec     Delay
        brne    P1
```

```
        ret

Pause2:
        ldi    Count,250
P2:     rcall  Pause
        dec    Count
        brne   P2
        ret
```

Listing 7.6 Die optische Alarmanlage

8 Datenspeicher

Der ATtiny13 besitzt außer seinen Arbeitsregistern noch 64 Byte RAM und 64 Byte EEPROM. Das ist zwar nicht sehr viel, aber man viele sinnvolle Anwendungen finden, für die der Speicher reicht. Während das RAM seinen Inhalt nur bei anliegender Betriebsspannung behält, bleiben Daten im EEPROM auch nach dem Ausschalten erhalten.

8.1 Das RAM

Das interne RAM beginnt an der Adresse $0x60 = 96$ (dezimal). Es wird mit dem Befehl LD (Load Indirect from data space to Register using Index X, Laden von der Adresse im X-Register) gelesen. Zum Schreiben verwendet man entsprechend den Befehl ST (Store Indirect From Register to data space using Index X, Speichern in die Adresse im X-Register). Das X-Register ist dabei ein 16-Bit-Register und belegt die Byte-Register R26 und R27. In der Datei `tn13def.inc` sind die High- und Lowbytes der Register X, Y und Z definiert.

```
.def    XL      = r26          ; X pointer low
.def    XH      = r27          ; X pointer high
.def    YL      = r28          ; Y pointer low
.def    YH      = r29          ; Y pointer high
.def    ZL      = r30          ; Z pointer low
.def    ZH      = r31          ; Z pointer high
```

Das Programm `RAM.asm` liest zuerst 64 Bytes aus dem RAM und sendet sie an den PC. Vor der Leseschleife wird XL mit 96 und XH mit Null geladen. Damit steht der Adresszeiger am RAM-Anfang. In der Schleife wird mit LD A,X+ jeweils ein Byte an der Adresse X in A kopiert und danach X um Eins erhöht. Das Programm wartet auf ein beliebiges Byte vom Terminal und sendet dann 64 Bytes zurück.

```
;RAM, Lesen und Schreiben von 64 Bytes

        rjmp    Anfang

Anfang:
        sbi      ddrb, TXD ;Datenrichtung TXD

Schleife:
        rcall    RdCOM
        ldi      XL, 96
        ldi      XH, 0
        ldi      Count2, 64
```

Lesen:

```
ld      A,X+
rcall   WrCOM
dec     Count2
brne    Lesen      ; 64 mal
```

```
ldi     XL,96
ldi     XH,0
ldi     Count2,64
```

Schreiben:

```
mov     A,XL
st      X+,A
dec     Count2
brne    Schreiben ; 64 mal
rjmp    Schleife
```

Listing 8.1 Lesen und Schreiben im RAM

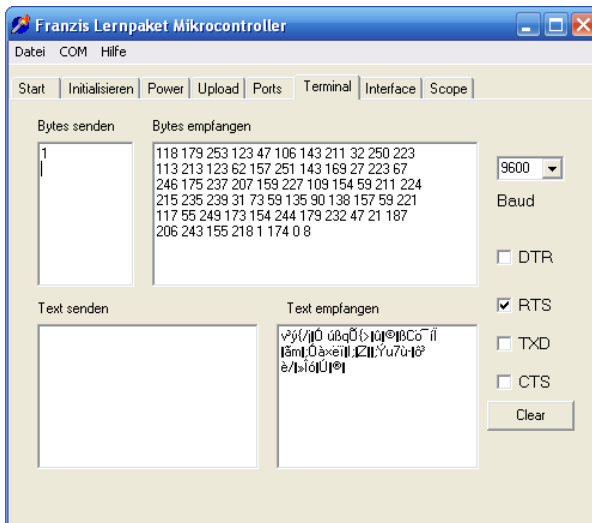


Abb. 8.1 64 zufällige Bytes im RAM ((Soft35.tif))

Senden Sie im Terminal ein beliebiges Byte. Das Programm antwortet mit 64 Bytes. Weil das RAM nach dem Einschalten des Controllers noch nicht beschrieben wurde, enthält es zufällige Speicherinhalte.

Nach dem Lesen wird jedoch das komplette RAM mit einer aufsteigenden Zahlenfolge beschrieben. Der Befehl ST X+, A kopiert jeweils ein Byte aus A in die RAM-Adresse X und erhöht danach X um Eins. Hier wird die Adresse selbst in A kopiert und dann

gespeichert. Beim zweiten Auslesen findet man also keine zufälligen Zahlen mehr sondern eine Zahlenreihe 96, 97, 98 usw.

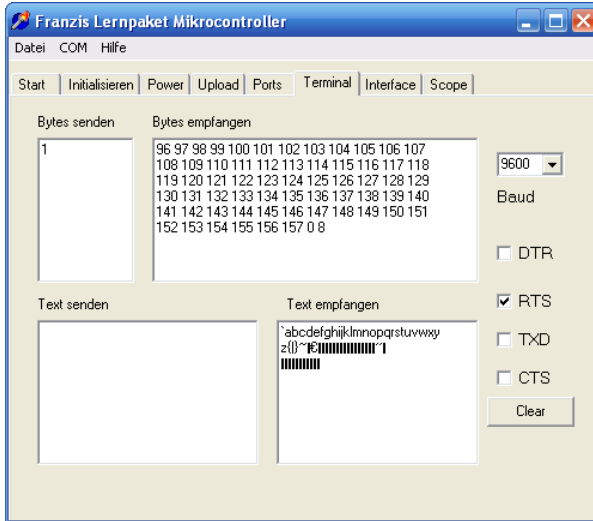


Abb. 8.2 Das beschriebene RAM ((Soft36.tif))

Das gesamte RAM bis auf die beiden letzten Adressen enthält beim nächsten Auslesen tatsächlich die erwarteten Zahlen. An den Adressen 158 und 159 finden sich jedoch die Bytes 00 und 08. Dies ist eine Rücksprungadresse, die beim Aufruf eines Unterprogramms gespeichert wurde. Man kann es sogar noch genauer erkennen: Es handelt sich um das Unterprogramm WrCOM, das von der Adresse 0x0008 aus aufgerufen wurde. Tatsächlich findet man den Aufruf Rcall WrCOM als achten Befehl des Programms.

Im RAM wird ein Stack (Stapel) gebildet, der von oben nach unten wächst und alle erforderlichen Rücksprungadressen speichert. In diesem Programm gibt es zwar zwei Unterprogramme. Aber sie werden nacheinander benutzt und benötigen daher nur den Speicherplatz für eine Adresse. Bis zu 62 Bytes wären also noch für eigene Anwendungen frei.

Wenn Sie aus einem Unterprogramm heraus ein zweites Unterprogramm aufrufen, wächst der Stack um zwei Bytes nach unten. Sie können dies testen, indem Sie z.B. am Ende der Empfangsroutine RdCOM einen weiteren Unterprogrammaufruf einfügen. Verwenden Sie dazu ein neues Unterprogramm Sub2, das nur aus einem RET besteht.

```
RdCOM:  sbis  pinb,RXD  ;Empfangen
        rjmp  RdCOM
```

```

        ldi    Delay,58
...
        com    A
        rcall  Sub2
        ret

Sub2:    ret

```

Listing 8.2 Geschachtelte Unterprogramme

8.2 Speicheroszilloskop

Das Speicheroszilloskop führt eine schnelle Serienmessung durch und speichert die Messwerte im RAM. Danach werden die Daten an den PC übertragen. Das PC-Programm LPmikro enthält eine grafische Darstellung der Messwerte. Entsprechend der Speichergröße des ATtiny13 werden 60 Messzyklen verwendet. Zusammen mit einer Messung am Start müssen 61 Bytes gespeichert werden. Es wird der Eingang ADC2 (PB4) bei einer Auflösung von 8 Bit und mit einer Referenz von 5 V verwendet.

```

;Scope.asm, Speicheroszilloskop

        rjmp   Anfang

Anfang:
        sbi     ddrb,TXD    ;Datenrichtung TXD
        rcall  AdcInit

Schleife:
        rcall  RdCOM
        ldi     XL,96
        ldi     XH,0
        ldi     Count2,61

Schreiben:
        ldi     A,2          ;Kanal 2, PB4
        rcall  RdADC
        st      X+,A
        dec     Count2
        brne    Schreiben ;61 mal

        ldi     XL,96
        ldi     XH,0
        ldi     Count2,61

Lesen:
        ld      A,X+
        rcall  WrCOM

```

```

        dec     Count2
        brne    Lesen      ;614 mal
        rjmp    Schleife

AdcInit:
        ldi     A,7         ;Clock / 128
        out     ADCSRA,A
        sbi     ADCSRA,ADEN ;AD einschalten
        ret

```

Listing 8.3 Das Speicheroszilloskop

Zum Test des Programms können Sie im Terminal ein beliebiges Byte senden. Der Mikrocontroller führt dann die Messung durch und antwortet mit 61 Bytes. Alternativ können Sie die Daten auch mit der Scope-Funktion darstellen. Klicken Sie auf „1 Kanal“. Das Programm sendet dann ein Byte und empfängt 61 Bytes.

Hier wurde der AD-Wandler mit einem langsamen Takt von 1,2 MHz / 128 betrieben. Dadurch dauert die eigentliche Messung länger. Das Oszilloskop zeigt deshalb mehrere Vollwellen eines 50-Hz-Signals (Abb. 8.3). Experimentieren Sie mit unterschiedlichen Teilerfaktoren in AdcInit. Die schnellste Einstellung ist „ldi A,1 ;Clock / 2“ und erlaubt die Messung an Signalen mit höherer Frequenz.

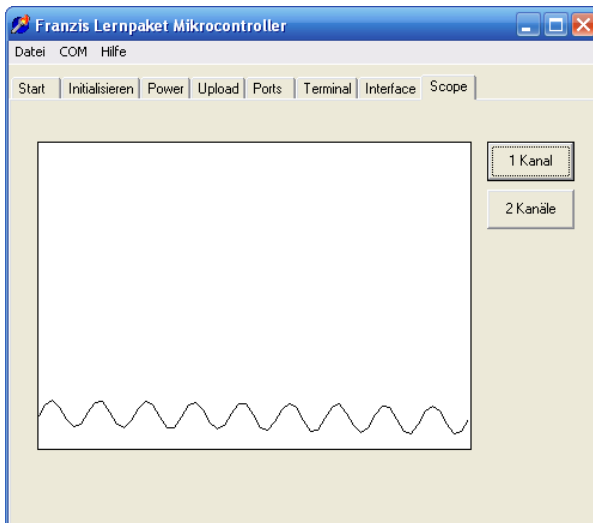


Abb. 8.3 Netzbrummen am offenen Eingang ((Soft37.tif))

8.3 Das EEPROM

Das EEPROM des ATtiny13 liegt nicht im normalen Speicherbereich, sondern stellt eine eigene Einheit mit Adressen ab Null dar. Das Datenblatt erläutert die Details der Ansteuerung, die hier in den Unterprogrammen RdEE (Lesen) und WrEE (Schreiben) umgesetzt wurden. Die aktuelle Adresse wird in einem Register EEadr übergeben. Lese- und Schreibdaten liegen in A.

```
;EEPROM.asm, 60 Bytes lesen und speichern
```

```
    .include "tn13def.inc"

    .def    A      = r16
    .def    Delay  = r17
    .def    Count  = r18
    .def    EEadr  = r19
    .def    EEmode = r20

    ;Port B
    .equ    TXD    = 1
    .equ    RXD    = 2

    rjmp Anfang
Anfang:
    sbi     ddrb,TXD ;Datenrichtung TXD
    ldi     EEadr,0
Schleife1:
    rcall   RdEE
    rcall   WrCOM
    inc     EEadr
    cpi     EEadr,60
    brlo    Schleife1
    ldi     EEadr,0
Schleife2:
    rcall   RdCOM
    rcall   WrEE
    inc     EEadr
    cpi     EEadr,60
    brlo    Schleife2
Ende:      rjmp   Ende

RdEE:      sbic   EECR,EWE
```

```

        rjmp    RdEE
        out     EEAR,EEadr
        sbi     EECR,EERE
        in      A,EEDR
        ret

WrEE:   sbic    EECR,EEWE
        rjmp    WrEE
        ldi     EEmode,0
        out     EECR,EEmode
        out     EEARL,EEadr
        out     EEDR,A
        sbi     EECR,EEMPE
        sbi     EECR,EEPE
        ret

```

Listing 8.4 Auslesen und Beschreiben des EEPROMs

Das Programm liest zunächst 60 Bytes aus dem EEPROM und sendet sie an den PC. Danach können neue Daten z.B. als Text eingegeben werden. Will man die alten Daten nicht verändern, kann einfach die Betriebsspannung mit RTS=0 ausgeschaltet werden. Auch nach langer Zeit ohne Spannung bleiben die gespeicherten Daten sicher erhalten.

Beim Beschreiben des EEPROMs benötigt der Mikrocontroller mehr Strom. Schalten Sie daher das DTR-Bit ein, damit mehr Strom von der seriellen Schnittstelle entnommen werden kann.

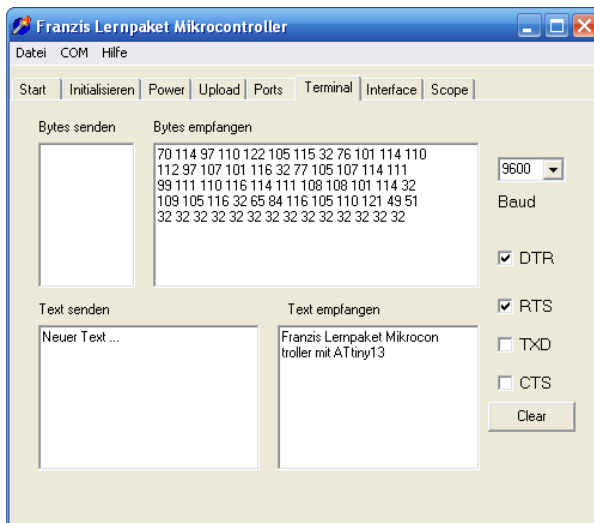


Abb. 8.4 Im EEPROM gespeicherter Text ((Soft38.tif))

Das EEPROM lässt sich z.B. auch für einen Langezeit-Datenlogger einsetzen. Ähnlich wie beim Oszilloskop werden dazu Messwerte erfasst und gespeichert. Eine Lösung wird in Kap. 10.3 gezeigt.

9 Das Interfaceprogramm

Die Verwendung der Interface-Software wurde bereits in Kap. 2 erläutert. Hier soll das dahinter liegende Assemblerprogramm genauer erläutert werden. Es enthält viele Teile, die bereits untersucht wurden und öffnet zahlreiche neue Möglichkeiten. Das Programm kann als Basis für eigene Erweiterungen und Entwicklungen dienen.

9.1 Die Interpreterschleife

Da Interface-Programm besteht im Kern aus einer Interpreterschleife, in der Kommandos vom PC ausgewertet und ausgeführt werden. Am Anfang der Schleife wird ein Byte empfangen und als Kommando in das Register Kom kopiert. Gültige Kommandos sind z.B. die Bytes 1, 16, 17, 18 und 32. Entsprechende Programmabschnitte K1, K16, K17 usw. führen jeweils einen Vergleich mit einer Konstante aus. Wenn ein Kommando erkannt wurde, folgt eine zugehörige Ausgabe oder Eingabe. Im anderen Fall verzweigt das Programm zum nächsten Vergleich.

Die Wahl der Kommandos ist beliebig und kann für jede Anwendung neu definiert werden. Das Protokoll lehnt sich jedoch weitgehend an einen Standard an, der mit dem SIOS-Interface von AK MODUL-BUS festgelegt wurde. Digitale Ausgaben bekommen Kommandos ab 0x10 (16), digitale Eingaben ab 0x20 (32). Analoge Eingaben belegen Kommandos ab 0x30 (48) bzw. 0x38 (56) bei einer erhöhten Auflösung. Ab 0x40 (64) liegen Kommandos für analoge Ausgaben. Das Kommando 1 dient zur Abfrage des aktiven Geräts. Das Programm im ATtiny13 antwortet hier mit einem Byte 100. So kann ein PC-Programm feststellen, ob das Interface reagiert.

Mit einem Byte 16 wird eine Portausgabe eingeleitet. Das Interface empfängt dann noch ein zweites Byte, das den Portzustand enthält. Mit 16, 255 würden z.B. alle Bits am Port B eingeschaltet. Allerdings werden hier nur die erlaubten Bits 0, 3 und 4 eingeschaltet, um die serielle Schnittstelle nicht zu stören. In gleicher Weise kann mit dem Kommando 17 das Datenrichtungsregister verändert werden. Kommando 18 initialisiert die PWM-Ausgabe an PB0. Mit einem Byte 32 fragt der PC den aktuellen Zustand am Port B ab. Alle diese Kommandos können direkt vom Terminal aus gesendet werden. Sie werden aber auch von der Interface-Funktion in LPmikro.exe verwendet.

```
;Interface.asm, Input/Output mit 9600 Baud
```

```
Schleife:
```

```
    rcall  RdCOM
    mov    Kom,A
k1:   cpi    Kom,1
    brne   K16
```

```

        ldi    A,100
        rcall  WrCOM
k16:    cpi    Kom,16
        brne   K17
        rcall  RdCOM
        andi   A,0b00011001
        out    portb,A
k17:    cpi    Kom,17
        brne   K18
        rcall  RdCOM
        andi   A,0b00011011
        ori    A,0b00000010
        out    ddrb,A
k18:    cpi    Kom,18
        brne   K32
        ldi    A,0          ;PWM initialisieren
        out    OCR0A, A
        ldi    A,0x83
        out    TCCR0A, A
        ldi    A,0x02
        out    TCCR0B, A
k32:    cpi    Kom,32
        brne   K48
        in     A,pinb
        rcall  WrCOM
k48:    cpi    Kom,48
        brne   K49
        ldi    A,2
        rcall  AD8Bit
        rcall  WrCOM
k49:    cpi    Kom,49
        brne   K56
        ldi    A,3
        rcall  AD8Bit
        rcall  WrCOM
k56:    cpi    Kom,56
        brne   K57
        ldi    A,2
        rcall  ADCrd
        rcall  WrCOM
        mov    A,B
        rcall  WrCOM
k57:    cpi    Kom,57
        brne   K64
        ldi    A,3
        rcall  ADCrd

```

```

        rcall  WrCOM
        mov   A,B
        rcall  WrCOM
k64:    cpi    Kom,64
        brne  K100
        rcall  RdCOM
        out   OCR0A,A

```

Listing 9.1 Digitale und analoge Aus- und Eingaben

Die Kommandos 48 und 49 fragen den Ad-Wandler im 8-Bit-Modus an den Anschlüssen ADC2 (PB4) und ADC3 (PB3) ab. Die Ergebnisse bestehen jeweils aus einem Antwortbyte. Außerdem gibt es die Kommandos 56 und 57 für die Messung mit erhöhter Genauigkeit und einer Auflösung von 10 Bit. Die Antwort besteht jeweils aus zwei Bytes, wobei zuerst das Highbyte und dann das Lowbyte gesendet wird.

9.2 Ein- und Zweikanal-Oszilloskop

Mit dem Kommando 100 startet man das Einkanal-Oszilloskop, das schon in Kap. 8.2 als einzelnes Programm vorgestellt wurde. Hier gibt es zusätzlich noch eine Zweikanal-Variante mit dem Kommando 101.

```

k101:  cpi    Kom,101    ;Oszi 2 Kanal
        brne  K250
Oszi2: ldi    XL,96
        ldi    XH,0
        ldi    Count2,31
O102:  rcall  ADCrd8BitB4
        st     X+,A
        rcall  ADCrd8BitB3
        st     X+,A
        dec    Count2
        brne  O102      ;31 mal
        ldi    XL,96
        ldi    XH,0
        ldi    Count2,62
O103:  Ld     A,X+
        rcall  WrCOM
        dec    Count2
        brne  O103      ;31 mal

```

Listing 9.2 Das Zweikanal-Oszilloskop

9.3 Oszillator-Kalibrierung

Das Interface-Programm enthält Programmteile zur Kalibrierung des internen Oszillators. Mit dem Kommando 250 können Sie die Genauigkeit direkt messen. Senden Sie mit 9600 Baud das Kommando 250 und ein Byte 255, das nur aus dem Startbit besteht, auf das acht Nullbits folgen. Das Unterprogramm RdRXD misst die Impulslänge und liefert im Idealfall den Wert 25. Sendet man ein Byte 0, dann muss die gemessene Zeit neun mal so lang sein, das Ergebnis sollte also 225 lauten. Erhält man z.B. den Wert 223, beträgt der Fehler etwa 1%.

Mit dem Kommando 251 können Sie den aktuellen Inhalt des OSCAL-Registers lesen, der beim Start mit einem Wert geladen wird, der bei der Herstellung des Controllers kalibriert wurde. Sie lesen hier z.B. den Wert 86.

Mit 252 wird der Inhalt der EEPROM-Adresse 63 gelesen. Wenn Sie bei der Initialisierung die Kalibrierfunktion zum Oszillator verwendet haben, findet sich hier ein Wert, der beim Start des Systems in OSCAL geladen wird. Zur Sicherheit wird allerdings abgefragt, ob das eigene Kalibrierbyte nahe genug am Original liegt, da andernfalls von einem Fehler ausgegangen werden kann. Um die EEPROM-Adresse 63 zu beschreiben, kann das Kommando 253 verwendet werden.

Mit dem Kommando 254 können Sie dem OSCAL-Register direkt einen neuen Wert zuweisen. Wenn der ursprüngliche Wert 86 war, können sie es z.B. mit 87 versuchen. Senden Sie 254, 87, und testen Sie die Genauigkeit des Oszillators mit 250, 0. Falls nun die Impulslänge 225 gemessen wird, ist der neue Kalibrierwert besser als der alte. Programmieren Sie dann mit 253, 87 das EEPROM, um die neue Kalibrierung beim nächsten Start automatisch zu laden.

```
k250: cpi    Kom,250
      brne   K251
      rcall  RdRXD
      rcall  WrCOM

k251: cpi    Kom,251
      brne   K252
      in A,  osccal
      rcall  WrCOM

k252: cpi    Kom,252
      brne   K253
      ldi    EEadr,63
      rcall  RdEE
      rcall  WrCOM

k253: cpi    Kom,253
```

```

        brne    K254
        rcall   RdCOM
        ldi     EEadr,63
        rcall   WrEE

k254:   cpi     Kom,254
        brne    K255
        rcall   RdCOM
        out     osccal,A

k255:   cpi     Kom,255 ;RC-Osc. kalibireren
        brne    SchleifenEnde
        rcall   Cal      ;20 x Byte 0

...

RdRXD:
        sbis    PINB,RXD ;9,6 kBaud, 25/Bit
        rjmp    RdRXD
        ldi     A,0
RXD0:   inc     A          ;1
        nop     ;1
        sbic    PINB,RXD ;2
        rjmp    RXD0      ;1, 5 Takte
        ret

```

Listing 9.4 Kalibrieren des RC-Oszillators

Alternativ zur schwierigen Kalibrierung durch Versuch und Irrtum kann mit dem Kommando 255 eine automatische Kalibrierung durchgeführt werden, die auch bei der Kalibrierung im Initialisieren-Menü des Programms LPmikro.exe verwendet wird. Das Programm erwartet 20 Bytes 0. Mit zwanzig Messungen wird das beste Kalibrierbyte gesucht und ins EEPROM geschrieben. Während der Messung sendet das Programm laufend die Abweichung vom idealen Messwert. Dieses relativ komplexe Programm kann hier aus Platzgründen nicht genauer analysiert werden. Interessierte Leser sollten den Quelltext Interface.asm anschauen.

Beim Start des Programms Interface.asm wird zuerst das Unterprogramm OscKorrektur aufgerufen. Das im EEPROM gespeicherte Kalibrierbyte wird dann übernommen, wenn es weniger als 5 vom vorhandenen OSCAL-Wert abweicht. Damit wird verhindert, dass der Mikrocontroller durch eine fehlgeschlagene Kalibrierung nicht mehr angesprochen werden kann.

```
OscKorrektur:      ;OSCCAL in EEPROM(63)
```

```

        ldi    EEadr,63
        rcall  RdEE
        in     B, osccal
        sub    B,A
        cpi    B,5           ;Abweichung <5?
        brlo   OscCopy
        cpi    B,252         ;Abweichung >-5?
        brsh   OscCopy
        ret
OscCopy:
        rcall  WrCOM
        ldi    EEadr,63
        rcall  RdEE
        out    osccal,A
        ret

```

Listing 9.5 Übernahme des Kalibrierbytes

Ein gleiches Unterprogramm ist auch im Bootloader vorhanden und wird nach einem Reset aufgerufen. Sie können also den Aufruf von OscKorrektur in Interface.asm entfernen. Im Normalfall startet immer zuerst der Bootlader und übernimmt die Oszillator-Kalibrierung. Danach startet das Anwenderprogramm schon mit dem genauen Oszillator.

9.4 Der Bootloader

Das Programm Init.asm enthält das komplette Interface-Programm und den Bootloader. Der Bootloader beginnt im oberen Viertel des Flash ab Adresse \$0180. Der erste Sprungbefehl am Anfang des Programms weist auf den Bootloader. Falls kein neues Programm geladen werden soll, verzweigt der Bootloader zur Adresse \$017E. Dort steht ein Sprung auf den Anfang des Interface-Programms bei Adresse \$0010. Alle Adressen beziehen sich auf 16-Bit-Worte, d.h. der verfügbare Speicher von einem Kilobyte endet nicht bei Adresse \$03FF, sondern bei Adresse \$01FF, denn 1024 Bytes entsprechen 512 Words. Die Adressen in den HEX-Files beziehen sich dagegen auf Bytes.

Das Programm Lpmikro verwendet den Bootloader beim seriellen Upload. Vor dem Hochladen werden die Sprungadressen ausgetauscht. Normalerweise steht an der Adresse Null ein Sprung auf den Anfang des Programms. Er wird durch einen Sprung auf den Bootloader ersetzt. An die Adresse \$017E setzt das Programm den Sprung auf das Anwenderprogramm. Der Anwender bemerkt nicht, dass bei jedem Start zuerst einmal das Bootprogramm angesprungen wird.

Dieser Umweg ist nötig, damit ohne den Reset-Zustand und über die normale serielle Datenübertragung ein Programm nachgeladen und gestartet werden kann. Bei jedem Neustart überprüft der Bootloader, ob ein neues Programm geladen werden soll. Als Signal

dazu wird ein High-Zustand der TXD-Leitung (BREAK-Zustand) am PC verwendet. Der Bootloader kennt eigene Kommandos, mit denen man Programme laden, auslesen und starten kann.

201, Adresse-High, Adresse-Low, 32 Datenbyte: Einen Datenblock schreiben

202, Adresse-High, Adresse-Low: Einen Datenblock auslesen

203: Das geladene Programm starten

```
.org    $0000
        rjmp $0180

.org    $0010
; hier beginnt das Interface

.org    $017E
        rjmp $0010

.equ    PAGESIZEB = PAGESIZE*2
.org    $0180

        .def    spmcsrval    =r28

RESETboot:
        cli
        rcall    OscKorrektur

Breakstart:
        sbic    PINB,RXD
        Rjmp    Brkend
        Rjmp    $017E
Brkend:
        Sbic    PINB,RXD
        Rjmp    Brkend
        sbi     ddrb, TXD
        ldi     A,105
        rcall    WrCOMb

BootKom:
        Rcall    RdCOMb

K201:   mov     Kom,A
        Cpi     Kom,201      ;201, Block schreiben
        Brne    K202
        Rcall    write_page
        Ldi     A,1
        Rcall    WrCOMb
```

```

K202: cpi    Kom,202      ;202, Block lesen
      brne   K203
      rcall  read_page

K203: cpi    Kom,203      ;203, ProgStart
      brne   K204
      in     A,PORTB
      rjmp   $017E
K204: rjmp   BootKom

write_page:
      ldi    Count2,16
      rcall  RdCOMb      ;Adr Hi,Lo vom Host
      mov    ZH,A
      rcall  RdCOMb
      mov    ZL,A
page_erase:
      ldi    spmcsrval,3
      out    SPMCSR, spmcsrval
      spm

wrloop:
      rcall  RdCOMb      ;32 Bytes vom Host
      mov    r0,A        ;Highbyte zuerst
      rcall  RdCOMb
      mov    r1,A
      ldi    spmcsrval,1
      out    SPMCSR, spmcsrval
      spm
      inc    ZL
      inc    ZL
      mov    A,r0
      rcall  WrCOMb
      mov    A,r1
      rcall  WrCOMb
      dec    Count2      ;16 mal
      brne   wrloop
      subi   ZL,32
      ldi    spmcsrval,5
      out    SPMCSR, spmcsrval
      spm
      ret

```

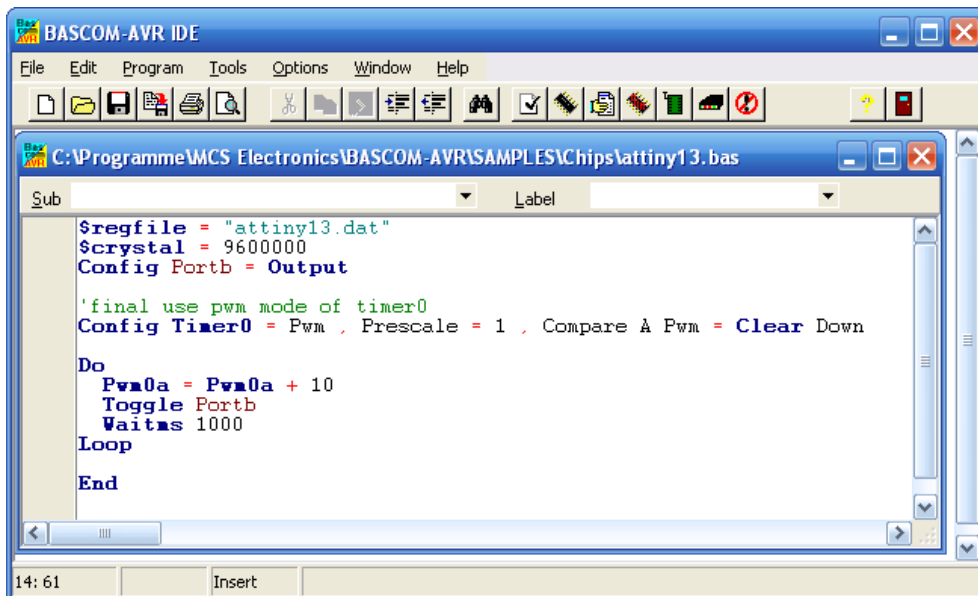

10 Bascom-AVR

Basic-Programme sind meist sehr viel leichter zu lesen als Assemblerprogramme. Zwar ist es wichtig, die entscheidenden Assembler-Grundlagen zu kennen, weil sie sehr nah an der Hardware liegen. Für schnelle und leichte Entwicklungen und die Lösung kleiner Aufgaben wird jedoch oft Basic bevorzugt. Für die AVR-Familie hat sich BASCOM-AVR von MCS-Electronics aus den Niederlanden weitgehend durchgesetzt.

Dieses Kapitel soll nur einen ersten Überblick vermitteln, da Sie die meisten Möglichkeiten der Basic-Programmierung analog zu den vorgestellten Assembler-Programmen erarbeiten können. Eine freie Demoversion von BASCOM-AVR erhalten Sie auf der Homepage des Herstellers www.mcselec.com. Hier wurde die „BASCOM-AVR Demo Version 1.11.8.3“ verwendet. Die freie Software ist auf eine Code-Größe von 4 KB begrenzt, was jedoch wesentlich mehr ist als der ATtiny13 aufnehmen kann. Installieren Sie die Software auf Ihrem PC um die folgenden Versuche durchzuführen.

10.1 Blinkprogramm

Im Verzeichnis Samples/Chips findet man kleine Beispielpprogramme zu den verschiedenen AVR-Chips. Laden Sie das Beispiel attiny13.bas.



The screenshot shows the BASCOM-AVR IDE interface. The title bar reads "BASCOM-AVR IDE". The menu bar includes "File", "Edit", "Program", "Tools", "Options", "Window", and "Help". The toolbar contains various icons for file operations, editing, and programming. The main window displays the code for "attiny13.bas" located at "C:\Programme\MCS Electronics\BASCOM-AVR\SAMPLES\Chips\attiny13.bas". The code is as follows:

```
Sub
$regfile = "attiny13.dat"
$crystal = 9600000
Config Portb = Output

'final use pwm mode of timer0
Config Timer0 = Pwm , Prescale = 1 , Compare A Pwm = Clear Down

Do
  Pwm0a = Pwm0a + 10
  Toggle Portb
  Waitms 1000
Loop

End
```

The status bar at the bottom shows "14: 61" and "Insert".

Abb.10.1 Die Bascom-Programmiersoberfläche ((Bascom1.tif))

Das Beispielprogramm enthält einen Verweis auf das Registerfile „attiny13.dat“. Damit ist die Hardware-Plattform definiert. Hardware-Register wie der in diesem Programm verwendete Port B und der PWM-Ausgang können also ihren Registeradressen zugeordnet werden.

```
$regfile = "attiny13.dat"
$crystal = 9600000
Config Portb = Output

'final use pwm mode of timer0
Config Timer0 = Pwm , Prescale = 1 , Compare A Pwm = Clear
Down

Do
    Pwm0a = Pwm0a + 10
    Toggle Portb
    Waitms 1000
Loop

End
```

Listing 10.1 Demoprogramm in BASCOM-AVR

Unter Options/Output muss das Hex-File angeklickt sein. Kompilieren Sie das Programm mit Program/Compile oder F7. Dabei wird im gleichen Verzeichnis die Datei attiny13.hex erzeugt. Laden Sie das Programm mit der Upload-Funktion in LPmiko in den Controller. Dabei fällt schon auf, dass wesentlich mehr Code erzeugt wurde als mit einem vergleichbaren Assemblerprogramm. Das ist verständlich, weil Basic praktisch aus vielen vorbereiteten universell einsetzbaren Unterprogrammen besteht, die selbst in Assembler entwickelt wurden. Dazu kommt die Verwendung von Variablen und umfangreiche Rechenfähigkeiten.

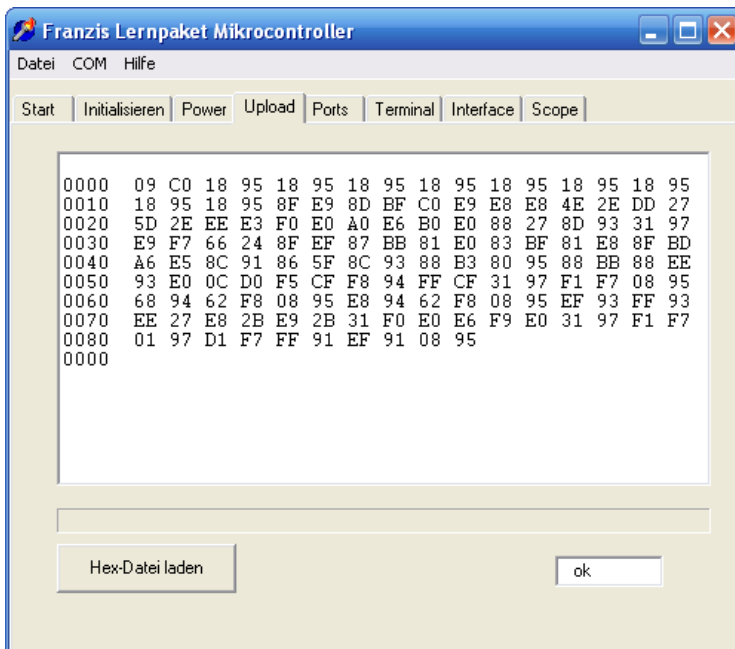


Abb.10.2 Der geladene Programmcode ((Bascom2.tif))

Das geladene Programm funktioniert auf Anhieb. Alle Anschlüsse am Port B wurden als Ausgänge definiert (Config Portb = Output). In einer Schleife werden Die Portzustände immer wieder umgedreht (Toggle Portb). Sie können also an PB3 oder PB4 eine LED mit Vorwiderstand anschließen, die dann langsam blinkt. Außerdem wird der PWM-Ausgang initialisiert und in der Schleife schrittweise erhöht (Pwm0a = Pwm0a + 10). Mit einer LED an PB0 sehen Sie daher einen treppenförmigen Helligkeitsverlauf.

Die Endlosschleife (Do Loop) enthält einen Wartebefehl für 1000 ms (Waitms 1000). Das Programm ist jedoch deutlich langsamer. Das liegt offensichtlich daran, dass die Taktfrequenz mit 9,6 MHz angegeben wurde (\$crystal = 9600000), während sie tatsächlich nur 1,2 MHz beträgt. Passen Sie also diese Zeile an (\$crystal = 1200000), kompilieren Sie erneut und laden Sie das Programm neu. Nun ist deutlich der Sekundentakt erkennbar.

10.2 RS232 und AD-Wandler

Hardware-Funktionen müssen initialisiert werden, wie dies auch von Assemblerprogrammen bekannt ist. Genaue Auskunft gibt die Hilfe in BASCOM-AVR. Markieren Sie z.B. das Schlüsselwort Config und drücken Sie F1. Es öffnet sich die Hilfedatei und gibt einen Überblick, welche anderen Hardwarefunktionen konfiguriert

werden können. Wählen Sie Config Adc. Nun erscheinen detaillierte Informationen zu den Einstellungen des AD-Wandlers. Folgen Sie auch den Links in der Hilfe und schauen Sie die Beispielprogrammteile an. Oft lassen sich aus den Beispielen die entscheidenden Zeilen in das eigene Programm kopieren. So fällt es nicht schwer, den AD-Wandler in BASCOM zu verwenden.

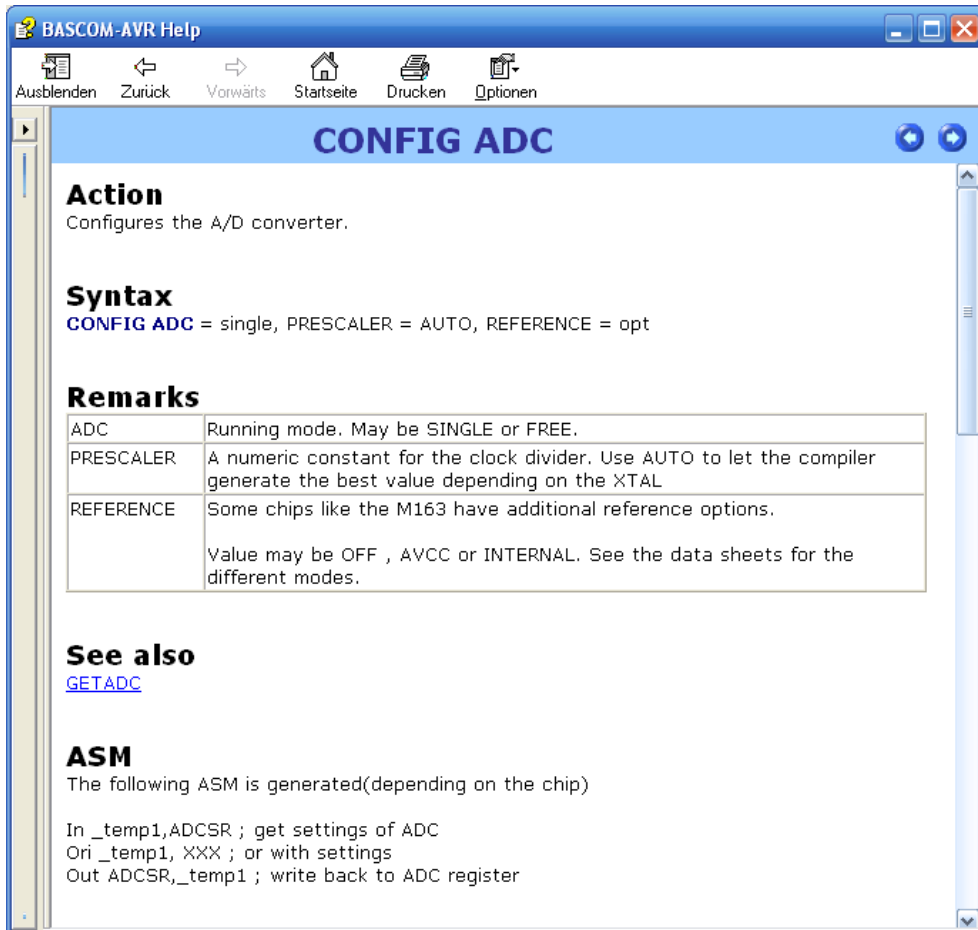


Abb.10.3 Die Hilfe in BASCOM-AVR ((Bascom3.tif))

Auch die serielle Schnittstelle ist sehr einfach zu verwenden. Die größeren AVR-Controller wie der Mega8 besitzen einen Hardware-UART. Dann genügt die Angabe der Übertragungsgeschwindigkeit um die Schnittstelle einzurichten (Baud = 9600). Für den ATtiny13 muss jedoch eine Softwareschnittstelle gebildet werden. Eingabe und Ausgabe werden getrennt definiert. Mit „Open "comb.1:9600,8,n,1,INVERTED" For Output As #1“

erhält man den seriellen Ausgang an PB1 mit 9600 Baud und invertiertem Ausgang, also ohne die Notwendigkeit einen invertierenden Leitungstreiber einzusetzen.

```
$regfile = "attiny13.dat"
$crystal = 1200000
Baud = 9600

Config Adc = Single , Prescaler = Auto
Start Adc

Open "comb.1:9600,8,n,1,INVERTED" For Output As #1
Print #1 , "AD-Wandler"

Dim N As Byte
Do
    N = Getadc(2)
    Print #1 , N
    Waitms 1000
Loop

End
```

Listing 10.2 Senden analoger Messwerte

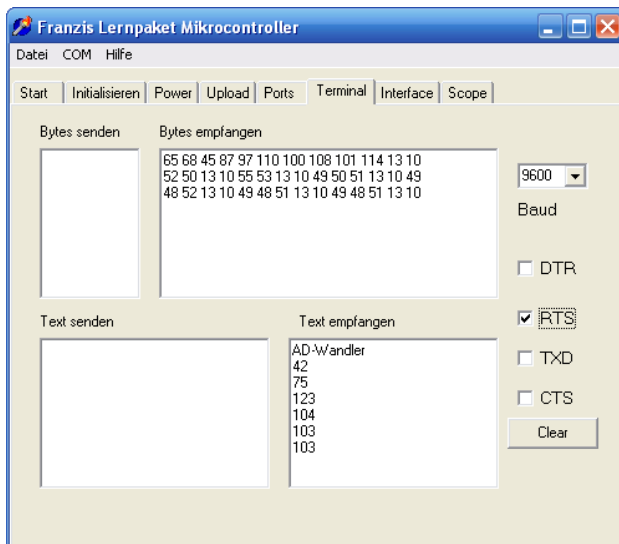


Abb.10.4 Textausgabe analoger Messwerte ((Bascom4.tif))

10.3 Ein Datenlogger

Mit BASCOM-AVR kann auch das EEPROM sehr einfach verwendet werden. Das Programm Logger.bas zeigt die Verwendung in einem Datenlogger. Messwerte an PB4 werden in einem Takt von einer Minute pro Messung erfasst und im EEPROM abgelegt. Das Protokoll ist kompatibel zum Einkanal-Oszilloskop in LPmikro. Dabei wird der Datensatz zuerst ausgelesen und dann neu erzeugt. Das Kommando zum Auslesen startet damit zugleich eine neue Messung.

Der Basic-Befehl GET liest ein Byte von der seriellen Schnittstelle. Genau wie das Assembler-Unterprogramm RdCOM wartet der Befehl auf ein ankommendes Byte und hält damit das Programm so lange auf. Zum Senden eines Byte dient der Befehl PUT.

```
$regfile = "attiny13.dat"
$crystal = 1200000
Baud = 9600

Config Adc = Single , Prescaler = Auto
Start Adc

Open "comb.1:9600,8,n,1,INVERTED" For Output As #1
Open "comb.2:9600,8,n,1,INVERTED" For Input As #2

Dim N As Byte
Dim D As Byte

Do
  Get #2 , D
  For N = 0 To 60
    Readeeprom D , N
    Put #1 , D
  Next N
  For N = 0 To 60
    Wait 60
    D = Getadc(2)
    Writeeprom D , N
  Next N
Loop

End
```

Listing 10.3 Das Programm Logger.bas

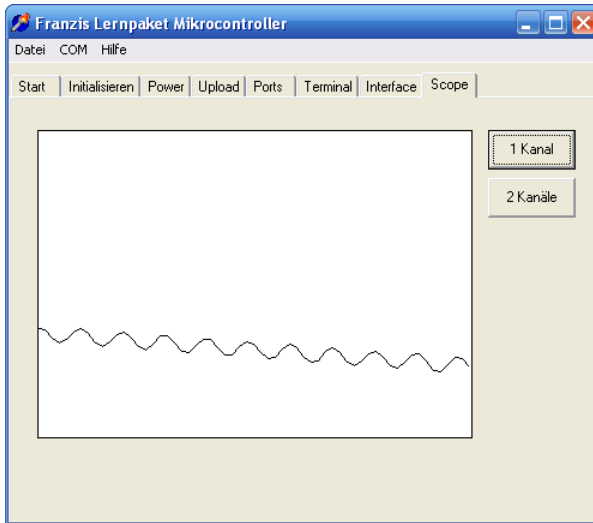


Abb.10.5 Darstellung der Messdaten im Oszilloskop ((Soft44.tif))

Das Programm kann leicht für eine vorgesehene Messdauer verändert werden. Mit dem Befehl `WAIT 60` wartet man vor jeder neuen Messung eine Minute lang. Die Messung in Abb. 10.5 wurde ohne Wartezeit aufgenommen. Aus der aufgenommenen 50-Hz-Schwingung lässt sich der Zeitbedarf für die reine Messung und Speicherung mit 200 ms ermitteln.

Das Beispiel zeigt, wie einfach auch komplexe Hardware-Funktionen in Basic verwenden lassen. Allerdings muss man dabei auf die Länge des erzeugten Programms achten. In diesem Fall werden 41% des Flash im ATtiny13 belegt. Die Grenze liegt bei knapp 75 %, wenn der Bootloader verwendet werden soll.

Ein Datenlogger wird normalerweise offline, also ohne Verbindung zum PC verwendet. Auch das ist mit der Platine machbar. Schließen Sie eine 9-V-Batterie nach Abb. 10.6 direkt an der DB9-Buchse an. In diesem Fall kann zusätzlich ein Taster zum Starten der Messung eingesetzt werden. Ein positiver Impuls hat hier die gleiche Wirkung wie ein vom PC gesendetes Byte.

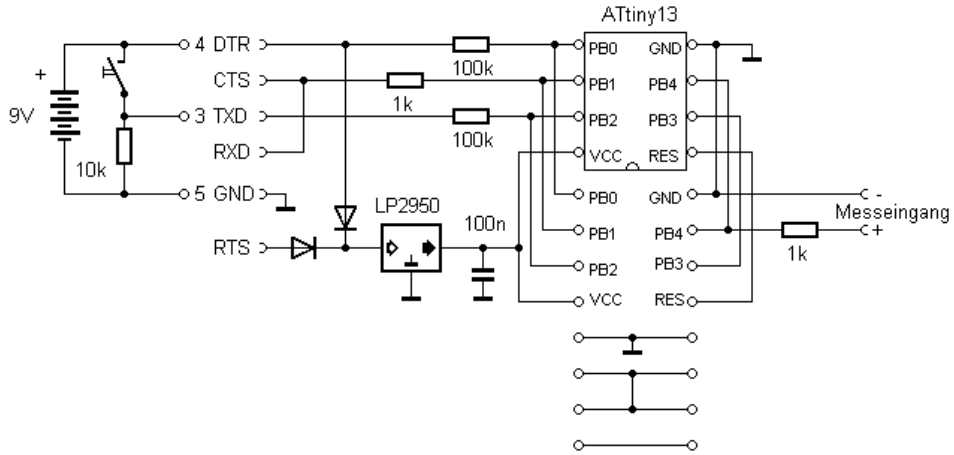


Abb.10.6 Anschluss einer Batterie ((Schalt18.gif))

11 C-Programmierung

Professionelle Programmierer bevorzugen die Programmiersprache C. Damit ist man wesentlich näher an der Hardware und kann ähnlich effektiven Code wie in Assembler entwickeln.

11.1 Win-AVR

Hier wird der C-Cmpiler Win-AVR eingesetzt. WinAVR ist ein Open Source Compiler für die Atmel AVR-Serie. Laden Sie den Compiler WinAVR in der Version 20070122 unter <http://winavr.sourceforge.net/>. Die Installationsdatei WinAVR-20070122-install.exe oder eine neuere Version erlaubt die problemlose Installation auf dem PC. Folgen sie den Anweisungen und installieren Sie damit den Compiler.

Wenn Sie weiterhin die bereits gewohnte Oberfläche des AVR Studio verwenden möchten, muss die vorhandene Software auf die Version 4.12 erneuert werden. Laden Sie die aktuelle Version von der Homepage des Herstellers www.atmel.com oder verwenden Sie die Installationsdatei von der CD. Mit der Installationsdatei aStudio4b460.exe oder einer entsprechenden neueren Datei wird die neue Version auf Ihrem PC installiert.

11.2 Das erste C-Projekt

Starten Sie das AVR Studio in der neuen Version und erzeugen Sie ein neues Projekt. Nun stehen Ihnen zwei Projekttypen zur Verfügung, nämlich Assembler oder C. Wählen Sie den Typ AVR GCC und geben Sie als Projektnamen GCC1 ein.

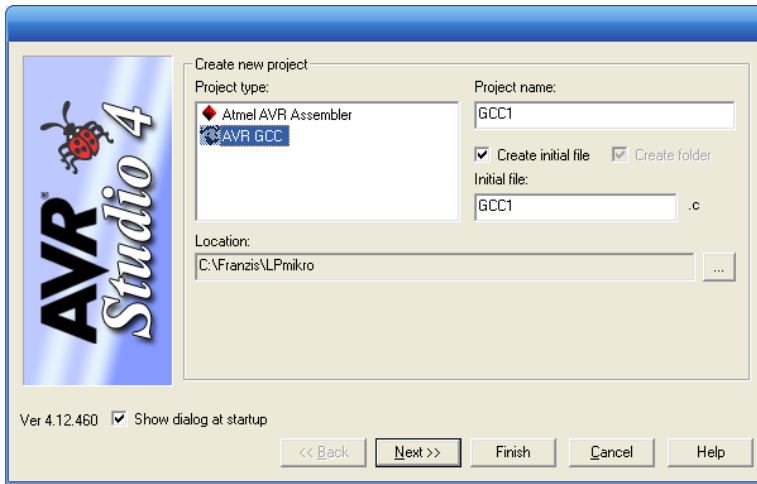


Abb.11.1 Ein neues C-Projekt ((GCC1.tif))

Wie erwartet öffnet sich das Editorfenster mit der noch leeren Datei GCC1.c.

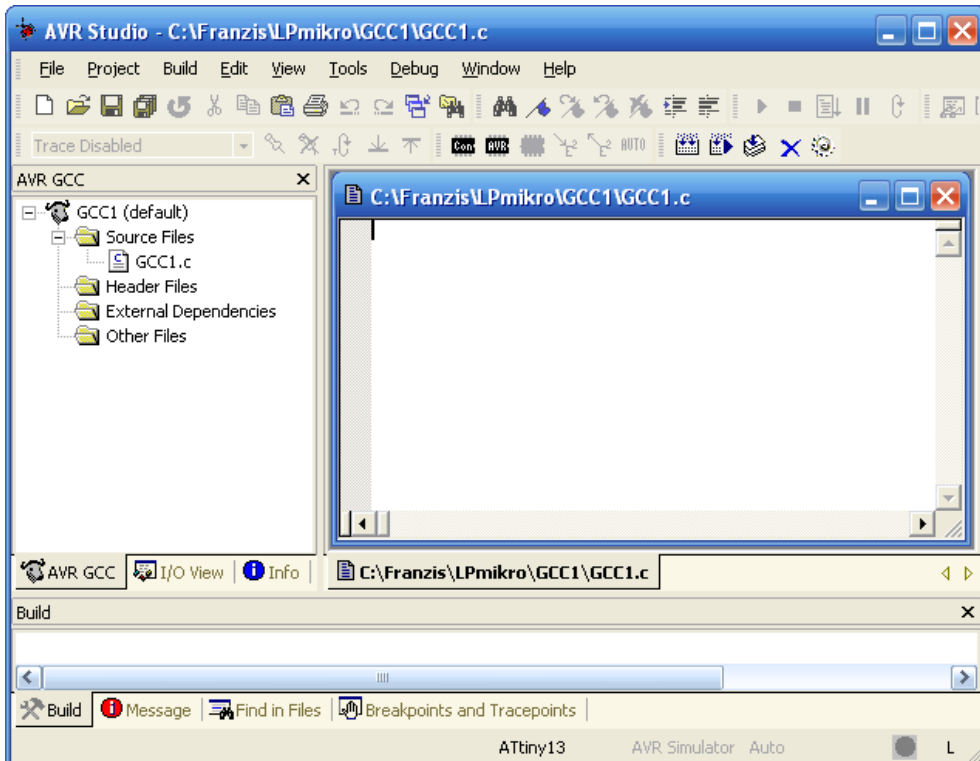


Abb.11.2 Das Editor-Fenster ((GCC2.tif))

Nun müssen noch einige Projekt-Optionen eingestellt werden. Sie erreichen das entsprechende Fenster über Project/Configuration Options. Wichtig ist die Wahl des Zielprozessors attiny13 sowie die Option „Create Hex File“.

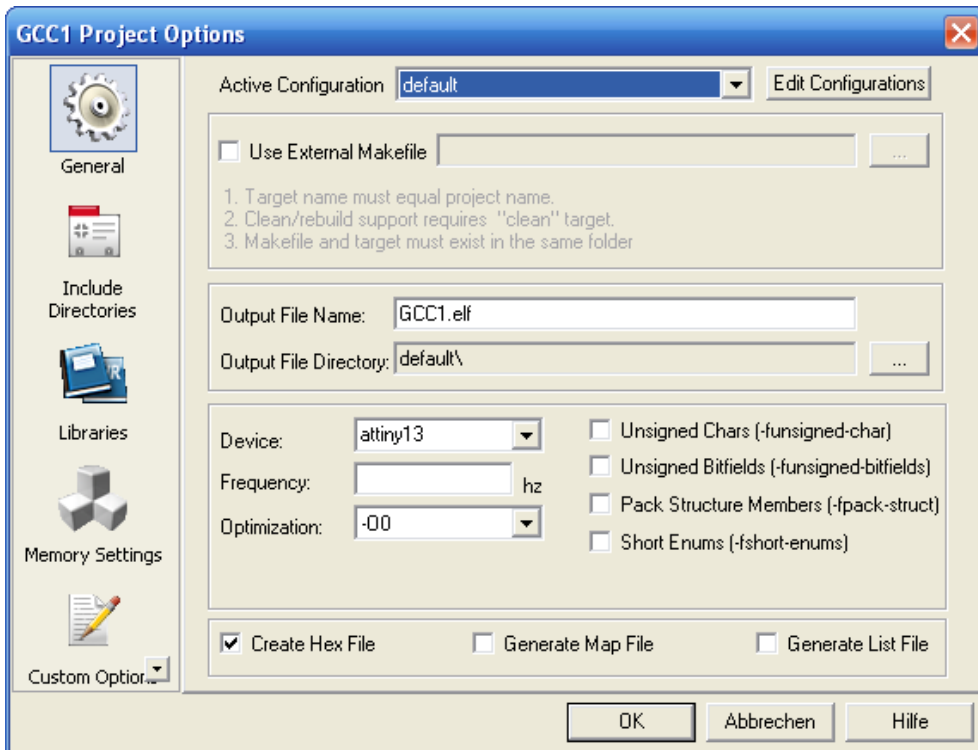


Abb.11.3 Projekt-Optionen ((GCC3.tif))

Tippen Sie dann das folgende Listing in das Editorfenster oder verwenden Sie das vorbereitete Projekt GCC1. Achtung, bei C-Quelltexten muss grundsätzlich die Groß- oder Kleinschreibung genau beachtet werden. Von Assemblerprogrammen sind Sie gewohnt, dass „ddrb“ klein geschrieben werden darf. Nun aber muss dieses Register in Großbuchstaben geschrieben werden, weil es in der Datei avr/io.h so definiert wurde.

```
#include <avr/io.h>

void main (void)
{
    int n;
```

```

DDRB = 0x08;
while (1)
{
    PORTB = 8;
    for (n = 0; n < 30000; n++);
    PORTB = 0;
    for (n = 0; n < 30000; n++);
}
return 1;
}

```

Listing 11.1 Der C-Quelltext

Übersetzen Sie das C-Projekt mit Build/Build. Wenn alles ohne Fehler abläuft erhalten Sie eine Erfolgsmeldung. Außerdem wurde die Liste der im Projekt verwendeten Dateien erheblich erweitert. Sie haben im Quelltext die Header-Datei `avr/io.h` angegeben, in der wichtige Definitionen zum AVR-Controller stehen. Außerdem sind dort mit `#include` weitere Dateien angegeben sind, die wiederum weitere Verweise enthalten. Speziell die Datei `iont13.h` wurde deshalb eingebunden, weil Sie den Zielcontroller ATtiny13 ausgewählt haben. Darin finden sich u.a. die für dieses Projekt wichtigen Einträge „`#define DDRB _SFR_IO8(0x17)`“ und „`#define PORTB _SFR_IO8(0x18)`“.

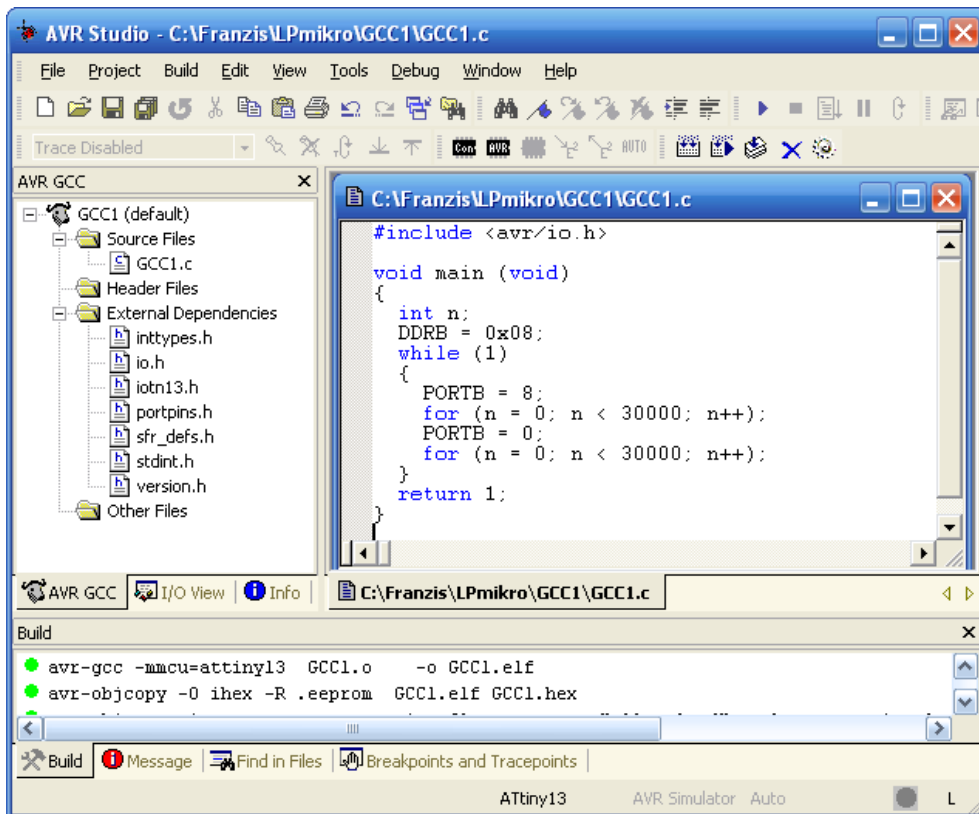


Abb.11.4 Erfolgreiche Kompilierung ((GCC4.tif))

Die kompilierte Datei GCC1.hex finden Sie im Verzeichnis GCC1/default. Laden Sie das übersetzte Programm mit der Upload-Funktion in LPmikro in den Controller. Schließen Sie die LED mit Vorwiderstand an PB3 an. Wie erwartet blinkt die LED.

12 Ein Programmierertool

Im Normalfall werden Programme des Lernpakets seriell über die Upload-Funktion des Programms LPmikro geladen. Dabei ist jedoch die Programmgröße auf 75% des verfügbaren Speicherplatzes begrenzt. Mit der zusätzlichen Software LPmikroISP können Sie dagegen auf den Bootloader verzichten und den gesamten Speicher nutzen. Außerdem können Sie die Fuses des ATtiny13 verändern und andere Taktraten einschalten.

12.1 ISP-Upload

Starten Sie das Programm LPmikroISP und wählen Sie die serielle Schnittstelle (z.B. COM1) aus. Setzen Sie dann die Reset-Brücke und wählen Sie die Registerkarte Program. Ein Klick auf die Schaltfläche Programmieren öffnet ein Dateimenü. Wählen Sie z.B. das Programm Blink2.hex. Die Software lädt das Programm in den Flash-Speicher. Nach dem Entfernen der Reset-Brücke startet das geladene Programm.

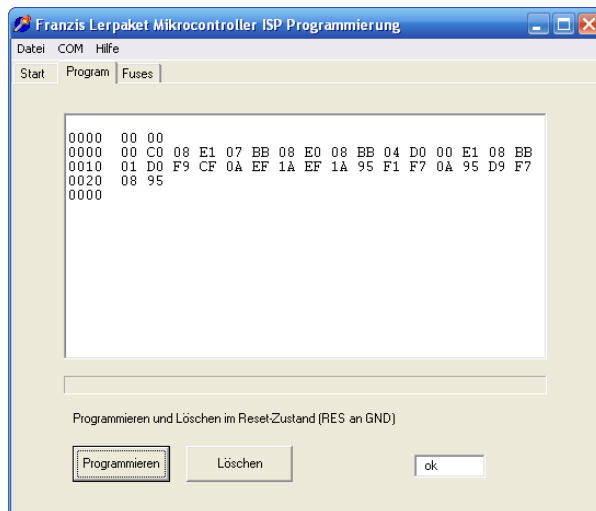


Abb. 12.1 Programmladen im ISP-Modus ((Soft41.tif))

Der Speicher kann mit der Schaltfläche Löschen auch komplett geleert werden. Bitte beachten Sie, dass mit der Verwendung des Programms LPmikroISP der Bootloader aus dem Speicher entfernt wird. Wenn Sie später wieder mit dem seriellen Upload arbeiten wollen, müssen Sie den Mikrocontroller erneut initialisieren. Alternativ können Sie auch mit LPmikroISP das Programm Init.hex laden.

12.2 Fuses

Mit LPMikroISP können Sie auch die Fuses im ATtiny13 verändern, um andere Taktraten einzustellen. Die Veränderung der Fuses ist prinzipiell mit Risiken behaftet. Theoretisch könnte man den Mikrocontroller in einen Zustand versetzen, in dem er nicht wieder programmiert werden kann. Möglich ist es z.B. den internen Oszillator ganz abzuschalten. Dann ist auch die ISP-Programmierung einschließlich der Veränderung der Fuses nicht mehr möglich. Um solche Fehler auszuschließen, sind in LPMikroISP nicht alle Einstellungen erlaubt, sondern es werden die meisten Bits in einer Voreinstellung belassen. Sie können also nur die Taktrate mit erlaubten Voreinstellungen ändern.

Die gewählten Voreinstellungen erkennt man übersichtlich in den Einstellungen des AVR-Studios, wenn das STK500 verwendet wird. Auch ohne die zusätzliche Hardware können Sie das Menü im Offline-Modus benutzen. Die Voreinstellungen stellen die Fuses mit zwei Bytes auf die Werte 0xEB und 0x6A ein. Entscheidend sind die folgenden Bits:

- „Self Programming Enable“ ist wichtig für das Bootprogramm.
- „Brown-out detection level at 2.7 V“ schaltet die Spannungsüberwachung ein.
- „Divide clock bei 8 internally“ teilt den internen Takt durch 8.
- „Int. RC OSC 9.6 MHz“ wählt den Taktoszillator mit 9,6 MHz.

Damit läuft der Mikrocontroller mit der für das Lernpaket gewählten Taktrate von 1,2 MHz. Die eingeschaltete Selbstprogrammierung ist eine wichtige Voraussetzung für das Bootprogramm. Die Spannungsüberwachung ist für einen sicheren Start wichtig. Intern wird ein Reset ausgelöst, solange die Spannung unter 2,7 V liegt.

Ein Brown-Out ist die abgemilderte Form eines Black-Out, also eines Stromausfalls. Wenn die Betriebsspannung eines Mikrocontrollers auch nur für kurze Zeit unter eine kritische Schwelle sinkt, kann dies zu unbestimmten Zuständen führen, die Fehlfunktionen eines Programms verursachen können. Um das zu verhindern, wurden so genannte Spannungswächter entwickelt, die einen solchen Zustand erkennen und den Mikrocontroller in den Reset-Zustand versetzen. Das Programm wird damit neu gestartet. Ein solcher Spannungswächter ist im ATtiny13 bereits eingebaut, im Auslieferungszustand jedoch nicht eingeschaltet. Über die Fuses wird ein Überwachungslevel von 2,7 V eingestellt.

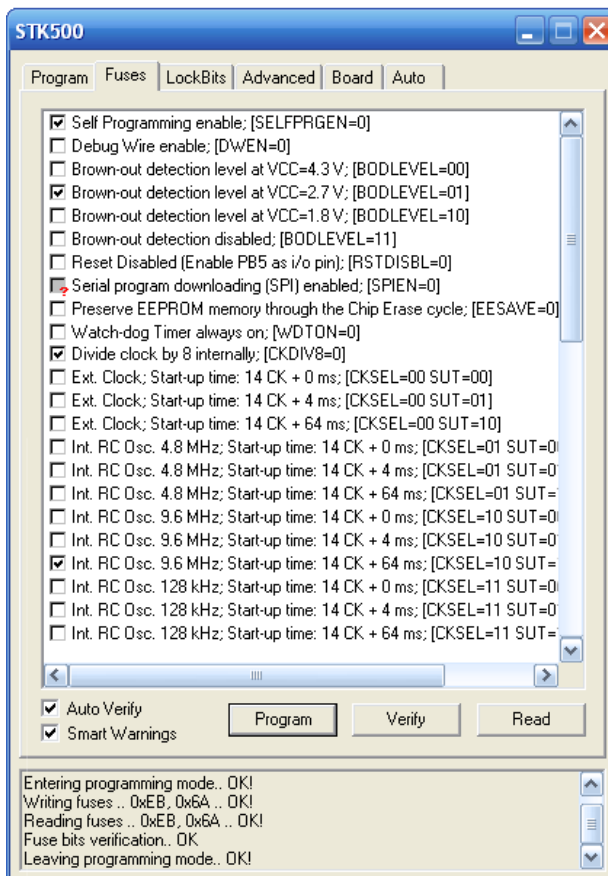


Abb. 12.2 Einstellungen der Fuses im STK500 ((Fuses1.tif))

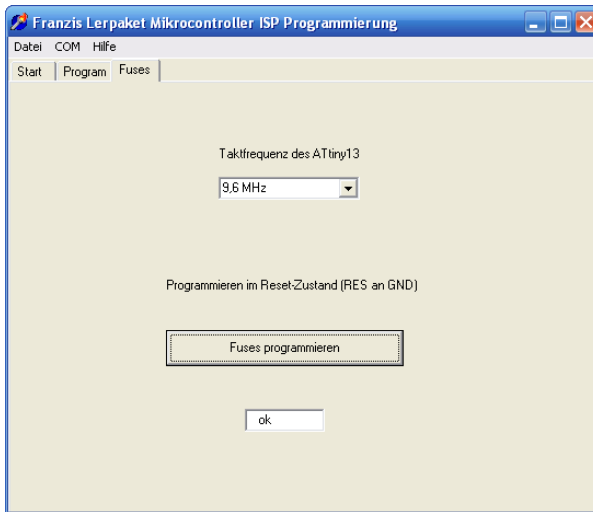


Abb. 12.3 Einstellen der Taktrate ((Soft42.tif))

Setzen Sie die Reset-Brücke und programmieren Sie eine neue Taktrate. Nach dem Freigeben des Reset läuft Ihr Programm mit der veränderten Geschwindigkeit. Wenn Sie zuvor das Programm Blink2.hex geladen haben, ist die Wirkung an der veränderten Blinkgeschwindigkeit erkennbar. Mit 9,6 MHz läuft das Programm acht mal schneller als bisher. Mit 128 kHz / 8 = 16 kHz läuft es extrem langsam. Diese Einstellung könnte sinnvoll sein, wenn es in einer Anwendung auf geringsten Stromverbrauch ankommt.

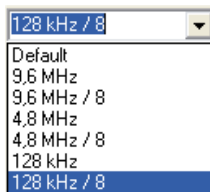


Abb. 12.4 Wählbare Taktraten ((Soft43.tif))

Die Default-Einstellung entspricht der Taktrate $9,6 \text{ MHz} / 8 = 1,2 \text{ MHz}$. Diese Voreinstellung wird auch bei der Initialisierung in Lpmikro.exe verwendet.

Anhang

Bezugsquellen:

Weitere Mikrocontroller ATtiny13 sowie einzelne Bauteile wie Widerstände, LEDs usw. erhalten Sie bei diesen Firmen:

Conrad Electronic
Klaus-Conrad-Straße
92240 Hirschau
www.conrad.de

Reichelt Elektronik
Elektronikring 1
26452 Sande
www.reichelt.de

Die Platine wurde in Zusammenarbeit mit AK MODUL-BUS entwickelt. Falls sich Probleme beim Zusammenbau ergeben oder Ihre Platine nicht funktioniert, können Sie eine fertig aufgebaute und getestete Platine bestellen.

AK MODUL-BUS Computer GmbH
Münsterstr. 45
48477 Hörstel-Riesenbeck
www.ak-modul-bus.de